



ALTAIR

ONLY FORWARD

Altair Accelerator Plus 2024.1.1

Administrator Guide

Contents

Accelerator Plus Administrator Guide	4
System Overview.....	5
Theory of Operation.....	5
Accelerator Plus and Accelerator.....	6
Start Accelerator Plus.....	7
Migrate from a Previous Version.....	10
Accelerator Plus Server and System Configuration.....	12
Start and Stop Accelerator Plus.....	12
Integration with PBS.....	12
Daemons for Accelerator Plus.....	14
Theory of Operation.....	15
vovwxd Configuration.....	17
Using Direct Drive with vovwxd.....	19
VOV Security Keys.....	20
Regulate Access to Accelerator Plus.....	25
Job Status.....	26
Job Submission Policy.....	27
Accelerator Plus Customization.....	30
Customize the wx run Command.....	31
Customize the wx list Command.....	32
Troubleshooting.....	32
Job Placement Policies.....	37
Web Server Configuration.....	37
Web Interface.....	40
Using Accelerator as a Base Scheduler.....	40
Many Accelerator Plus Queues on Many Accelerator Queues.....	41
Jobclasses.....	42
Create Jobclasses.....	42
Use Additional Jobclass Directories.....	43
Define a Default Jobclass.....	44
Reconcile Unused Resources.....	45
Define Jobclasses.....	45
Use Jobclasses.....	47
Resource Management.....	49
Hardware Resources.....	50
Wildcard Tasker Resources.....	54
Resource Mapping.....	55
Resources Representing the Sum of Others.....	57
Commas vs. ORs in Resources.....	57
Frequently Asked Questions and Troubleshooting Tips.....	59
Accelerator Plus Modulation.....	60

Altair Accelerator Plus Agent Preemption.....	61
Legal Notices	62
Intellectual Property Rights Notice.....	63
Technical Support.....	69
Index	71

This guide is written for the Accelerator Plus system administrator who needs to configure and manage one or more Accelerator Plus instances.

This chapter covers the following:

- [System Overview](#) (p. 5)
- [Accelerator Plus Server and System Configuration](#) (p. 12)
- [Jobclasses](#) (p. 42)
- [Resource Management](#) (p. 49)
- [Frequently Asked Questions and Troubleshooting Tips](#) (p. 59)

The administrator is expected to understand UNIX system processes, the dynamics of UNIX interactive shells, shell scripting techniques, and general trouble-shooting concepts. Some knowledge of schedulers is also expected.

For details about the usage and capabilities of using Accelerator Plus, refer to the *Accelerator Plus User Guide*.

Related Documents

Since Accelerator Plus is a derivative of Accelerator, most of the concepts explained in the *Altair Accelerator Administrator Guide* also apply to Accelerator Plus, even though they are not explained in this guide.

The following documents provide additional information that is related to using and configuring Accelerator Plus:

- *Altair Accelerator Installation Guide* to learn how to install Accelerator Plus and the rest of the software
- *Altair Accelerator Plus User Guide* to learn the end-user point of view for Accelerator Plus
- *Altair Accelerator Administrator Guide* to learn all the advanced tricks that also apply to Accelerator and Accelerator Plus
- *VOV Subsystem Reference Guide* contains the reference material about projects, files, registry, etc.

System Overview

Accelerator Plus is a high-performance hierarchical scheduler designed for distributed High Performance Computing (HPC) environments.

Accelerator Plus is based on the patented concepts described in [US Patent 9,658,893](#) about multi-layered resource scheduling.

The current implementation is designed to be run in conjunction with a base scheduler such as Accelerator, or [Altair PBS Professional](#).

With its sub-millisecond latency, Accelerator Plus improves the throughput of difficult workloads, especially those consisting of large numbers of short duration jobs perhaps with complex dependencies, while off-loading the base scheduler. Accelerator Plus allows any user or group to have their own high-performance scheduler without requiring the intervention of the IT department. Since all computing resources are negotiated by means of the base scheduler, Accelerator Plus always obeys all policies established by IT with respect to sharing such resources.

Theory of Operation

During the initial setup, the Accelerator Plus host server (vovserver) establishes a main port for communication and additional ports for web access and read-only access. Afterwards, the vovserver waits for and responds to incoming connection requests from clients.

Clients consist of *regular clients* that request a particular service, *tasker* (server farms) that provide computing resources, and *notify clients* that listen for events.

A fresh instance of Accelerator Plus typically has only one persistent or permanent "tasker", dedicated to launching requests to get more taskers from the underlying base scheduler, depending on the workload.

Regular clients can submit the workload, which consists of one or more jobs, or query data about jobs or system status. When a job is created, it is placed in a Queued state. Queued jobs are sorted into buckets. Jobs that have the same characteristics go in the same bucket.

Each job bucket is analyzed, by an external daemon called `vovwxd`. If a bucket is waiting for hardware resources, then the external daemon issues a request to the underlying base scheduler for resources that match that job bucket. In other words, Accelerator Plus requests from the base scheduler a "tasker" that can run the jobs in a specific bucket. Once the base scheduler grants the request by running a proxy job, the submitted `wx-tasker` connects back to the Accelerator Plus instance advertising the available resources. Jobs from the matching bucket begin executing without any further intervention from the base scheduler. Multiple buckets and multiple jobs from each bucket can be serviced concurrently. With a large base scheduler and a significant workload, thousands of jobs can be run concurrently.

When a job completes, the `wx-tasker` notifies the vovserver. The resources, both tasker-based and central, are recovered, allowing subsequent jobs (queued in the buckets) to be dispatched. When completed, the job status is updated to either VALID or FAILED.

In addition to dispatching jobs and processing their status, the vovserver responds to queries about system and job requests, publishes events to notify clients, and continues to process incoming job requests.

Modes of Operation Examples

Accelerator Plus can be used in many ways. Here are some typical examples.

Single User Mode, Persistent

A Accelerator Plus instance is started on a dedicated compute node using a role account. Another application, for example a Jenkins build server, is used to create the workload. In this scenario, Accelerator Plus is used primarily as an efficient distributed build engine, interfacing with the base scheduler. Multiple Accelerator Plus instances can be deployed concurrently to accelerate multiple flows in the form of execution "lanes." The underlying scheduler is used to balance the resource allocation across the Accelerator Plus instances.

Single User Mode, On-Demand

Similar to the first mode, but now the Accelerator Plus instance itself is also run on the underlying batch system. Upon completion of the workload, the Accelerator Plus instance is halted and all compute resources are returned to the farm. This model is useful for occasional, self-contained resource intensive workloads.

Multi User Mode, Persistent

This mode implements full hierarchical scheduling. The Accelerator Plus instance runs on a dedicated node with a publicly known host name and port number. Multiple Accelerator Plus instances can be used concurrently to provide each team with their own scheduler. While it is possible to allocate Accelerator Plus instances on a per-project basis, the preferred allocation method is on a functional or workload basis. For example, providing a Accelerator Plus instance for each of the Design Verification, Circuit Design and Physical Design teams allows similar work flows to be grouped together on a single Accelerator Plus instance. Commonality of work flow within a Accelerator Plus instance allows more optimal tuning while sharing a common base scheduler.

Accelerator Plus and Accelerator

The two products are essentially the same, but serve rather different usage models.

The following table clarifies the differences between them.

Item	Accelerator	Accelerator Plus	Comments
Manager command to start and stop	<code>ncmgr</code>	<code>wxmgr</code>	Essentially the same command.
User command submit workload	<code>nc</code>	<code>wx</code>	Essentially the same command.
Taskers	Static set of taskers, fully managed: Each tasker monitors load and resources on its host	Elastic set of taskers, determined by workload; each tasker has one slot and is available regardless of load on host	
Licensing of vovserver	<code>server_nc</code>	<code>server_wx</code>	
Licensing of vovtasker	<code>slots_nc</code>	<code>slots_wx</code>	

Item	Accelerator	Accelerator Plus	Comments
Job Resources	-no change-	The resource expression is silently augmented to be <pre>"Bucket : <BucketId> OR (<resource_expression></pre>	In Accelerator Plus, you want to allow precise routing of jobs to the taskers that have been created for that specific bucket. This is for information only. You will not see this change.
Job Placement Policy (JPP)	All JPP are supported: fastest, smallest, largest, slowest, first, ...	Only JPP==first is supported. The JPP of the workload is passed to NC if base scheduler is NC	
Preemption	Full preemption support.	Fully supported but normally not used: job modulation is preferred	
Blackhole Detection	Supported: taskers are allowed to recover	Supported: taskers are dismissed if they appear to be black-holes.	
Reservations (subject to change in future releases)	Only ADMIN can create reservations	USER can create reservations	In Accelerator Plus users are allowed to reserve taskers for specific buckets

Start Accelerator Plus

1. The command to start and stop Accelerator Plus is `wxmgr`, in all ways similar to `ncmgr`. Use the default Accelerator Plus queue name `wx`.


```
% wxmgr start
wxmgr: message: Checking the license...
wxmgr: message: ... the license is good.
wxmgr: message: Starting Accelerator Plus
wxmgr: message:   with name      wx
wxmgr: message:   on host       hostname
wxmgr: message:   in directory  /remote/release/vov/vnc
wxmgr: message:   as user      username
Do you want to proceed? (yes/[no]) > yes
wxmgr: message: Updating config file '/remote/release/vov/...../vwx.tcl'
wxmgr: message: Waiting for server to be ready ...
wxmgr: message: Sanity check...
wxmgr: message: Accelerator Plus wx@hostname is ready.
wxmgr: message: Sanity check...
wxmgr: message: Confirming start of required daemons...
wxmgr: message:
```

```
wxmgr: message: wx@mac01ac is ready. To access the web UI:
wxmgr: message:
wxmgr: message: http://mac01ac:6439 -- Requires login
wxmgr: message:
wxmgr: message: On C-Shell: setenv NC_QUEUE /remote/release/vov/wx/wx.swd/
setup.tcl
wxmgr: message: On Bash: export NC_QUEUE=/remote/release/vov/wx/wx.swd/
setup.tcl
```

2. Try the command below, which is equivalent to `wxmgr start`, but with the explicit values of the working directory, port number and of and the queue name. You will use a command similar to this to start other Accelerator Plus instances.

```
% wxmgr start -queue wx -port 6578 -dir $VOVDIR/../../wx
...output omitted...
```

3. Once the queue has started, you need to be able to connect to it. This is done by setting the environment `NC_QUEUE`.

 **Note:** Same variable used in Accelerator! Not `WX_QUEUE`.

This variable should be set to the `setup.tcl` file that has been created in the server working directory of the new Accelerator Plus instance. These values are shown clearly at the end of the start log (see above).

```
## Example for C-shell users
% setenv NC_QUEUE $VOVDIR/../../wx/wx.swd/setup.tcl
```

4. Check that Accelerator Plus is responding:

```
% wx hosts
# TASKER          LOAD STATUS   JOBS          HB RESERVE    MESSAGE
1 WXLancher      0.00 ready       0/8           43s C=LauncherClass 1y334d Licensed for 8
slots
```

This shows that there is already one vovtasker called "WXLancher" which is reserved for a special class of jobs called "LauncherClass". This tasker is used to issue the requests to the underlying base scheduler on behalf of the users who submit workload.

5. Connect the Accelerator Plus instance to the base scheduler. This can be done most easily by using the utility `vovwxconnect`. In the following example, the new Accelerator Plus instance is connected to the main Accelerator instance called "vnc"

```
% wx cmd vovwxconnect -nc vnc
```

`vovwxconnect` creates the appropriate daemon configuration file (`vovwxd/config.tcl`) and starts the daemon.

6. The security file `wx.swd/security.tcl` determines who may connect to the system and from which host. By default, the owner of the vovserver has ADMIN privilege from all hosts. Check the file `wx.swd/security.tcl` to verify that the login name is correct and add any others needed.
7. Check the file `wx.swd/security.tcl` to verify that the following lines exist:

```
# All users can connect from anywhere
vtk_security + USER +
# You want to be ADMIN
vtk_security YOURNAME ADMIN +
# Root needs to have ADMIN privileges, to permit vovtaskerroot to connect
vtk_security root ADMIN +
# The cadmgrs VovUserGroup needs to have ADMIN privileges
```



```
vtk_security -group cadmgrs ADMIN +
```

8. Check the Accelerator Plus system status with the command:

```
% wx cmd vsi

Vov Server Information - 03/23/2018 17:04:04

wx@mac01ac:6439 | URL: http://mac01ac:6439
-----
Jobs:                1 | Workload:
Files:               2 | - running:                0
Sets:               18 | - queued:                 0
Retraces:           0 | - done:                   0
                   | - failed:                 0
-----
Taskers:             1 | Buckets:                   0
- ready:            1 | Duration:                  0s
Slots:              8 | SchedulerTime:            0.00s
-----
TotalResources:     1,685 | Pid:                       51213
                          | Saved:                     26m41s ago
                          | Size:                      16.00MB
                          | TimeTolerance:             3s
-----
```

9. Submit the first job to Accelerator Plus:

```
# Open the GUI to monitor the job.
% wx gui &
% wx run sleep 30
```

After a few seconds, you will see the job turn from cyan to orange and then eventually green to represent successful execution.

vovwxconnect

Utility to setup a connector for Accelerator Plus to a primary queue.

```
vovwxconnect: Usage Message

DESCRIPTION:
  Utility to setup a connector for Accelerator Plus to a primary queue

OPTIONS:
  -h                -- This help.
  -v                -- Increase verbosity.
  -nc NCQ           -- Connect with specified Accelerator queue.
  -vovdir PATH      -- Specify VOVDIR for the Accelerator queue, if different
                    than that of Accelerator Plus.
  -ncconfigdir     -- Specify NC_CONFIG_DIR for the Accelerator queue, if
                    different than that of Accelerator Plus.
  -lsfemul NCQ     -- Connect with Accelerator but using the LSF emulation
                    interface (only for debugging)
  -dd              -- Connect with Direct Drive
  -pbs             -- Connect with PBS
  -pbsemul NCQ    -- Connect with Accelerator but using the PBS emulation
```

```
interface (only for debugging)
-legacy      -- Connect using legacy daemons vovelasticd vovlsfd.
-show        -- Show what is connected.
-test        -- Validate connections by running a test job.
-nostart     -- Do not start the vovwxd daemon.
-loglevel L  -- Set verbosity level for vovwxd (0-6)
```

EXAMPLES:

```
% vovproject enable MY_WX_PROJECT
% vovwxconnect -nc vnc1 vnc2
% vovwxconnect -nc vnc1 -nc vnc2
% vovwxconnect -nc vnc_test -vovdir /some/path/to/a/vovdir/used/by/vnc_test
% vovwxconnect -pbs
% vovwxconnect -show
% vovwxconnect -test
% vovwxconnect -lsfemul vnc_test
```

Multi-instance vovwxd

There may be situations where a multi-instance vovwxd connection can be useful.

To establish advanced configuration for multiple vovwxd daemons, use the following procedure:

```
% wxmgr start -queue wx -port 6578 -dir $VOVDIR/../../wx
% SWD=$(wx cmd vovserverdir -p .)
% mkdir $SWD/vovwx2d
% wx cmd vovwxconnect -nc vnc -nostart
% vi $SWD/vovwxd/config.tcl
% vi $SWD/vovwx2d/config.tcl
% wx cmd vovdaemonmgr start vovwxd vovwx2d
```

Migrate from a Previous Version

When Accelerator Plus was introduced in version 2016.09, it was based on Tcl daemons called vovelasticd and vovlsfd. This section describes how to migrate from the previous version to the new version based on the binary daemon vovwxd.

1. Assume that the Accelerator Plus instance is called "wx" and the Accelerator instance is called "vnc", the default name. Start with the normal setup:

```
% vovproject enable wx
% vovwxconnect -nc vnc
% vovdaemonmgr show vovwxd
DOWN
% cd `vovserverdir -p vovwxd`
```

2. In the directory `WX.swd/vovwxd` you can find the log file for vovwxd. Look at that file to see if the daemon is up and running.
3. 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.

```
% cp $VOVDIR/etc/autostart/start_wx_daemon.tcl `vovserverdir -p 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.

4. Copy the configuration file template from \$VOVDIR/etc/config/vovwxd/config.tcl into WXSWD/vovwxd.

```
% cp $VOVDIR/etc/config/vovwxd/config.tcl `vovserverdir -p vovwxd/.`
```

5. Copy the batch system driver script from \$VOVDIR/etc/config/vovwxd/vovnc.tcl into WXSWD/vovwxd.

```
% cp $VOVDIR/etc/config/vovwxd/vovnc.tcl `vovserverdir -p vovwxd/.`
```

6. Modify WXSWD/vovwxd/config.tcl to specify the base queue name(s) and ensure the correct driver script is specified:

```
# Fragment of vovwxd/vconfig.tcl
set CONFIG(driver_script) "vovnc.tcl"
set CONFIG(queues) "base_NC_queue_name"
```

7. If multiple NC queues are to be used as the back-end, specify them in space-separated format:

```
# Fragment of vovwxd/config.tcl
set CONFIG(driver_script) "vovnc.tcl"
set CONFIG(queues) "vncl vnc2 vnc3"
```

The vovwxd daemon automatically reads the configuration after changes.

When upgrading Accelerator Plus from version 2016.09, make sure a LauncherTasker is defined. Edit the WXSWD/taskers.tcl file to add a local tasker that will be responsible for submitting taskers. This tasker should be reserved for the LauncherClass class and provide a resource called WXLauncher. For example:

```
# Fragment of WX.swd/taskers.tcl
vtk_tasker_define localhost -name "WXLauncher" -resources "WXLauncher"

#
# Create a reservation for the tasker.
# The syntax is as follows:
# vtk_reservation_create type what quantity start end <OPTIONS>
#
vtk_reservation_create tasker WXLauncher 1 0 forever -jobclass LauncherClass
```

Accelerator Plus Server and System Configuration

Start and Stop Accelerator Plus

The command `wxmgr` is identical to `ncmgr`.

Start Accelerator Plus at System Boot Time on Linux

This step is optional.

The instructions in this section are valid for Linux.

 **Note:** This part of the installation requires root permission.

The Accelerator Plus `vovserver` can be restarted at reboot by installing the proper script in both the `/etc/rc3.d` and `/etc/rc5.d` directories for Linux.

Run the following commands on the host that was selected as the Accelerator Plus `vovserver`.

```
% /bin/su -  
% cp $VOVDIR/etc/boot/S99wx /etc/rc3.d/S99wx  
% chmod 755 /etc/rc3.d/S99wx  
% vi /etc/rc3.d/S99wx  
....  
Edit configurable items as needed.
```

 **Note:** `sudo` should be used where configured. To avoid *Trojan Horse* programs, `su` should always be called using the full path `/bin/su` on Linux.

You can test with:

```
% ./S99wx start
```

```
% ./S99wx stop
```

 **Note:** Re-start the Accelerator Plus server with the command `./S99wx start` after testing the `stop` capability.

Integration with PBS

Accelerator Plus has added the capability to use PBS Professional as its base scheduler. For the 2019.01 release, jobs can be submitted through Accelerator Plus and run on a PBS complex.

This requires a PBS client/server to be installed on the host where Accelerator Plus is running (`qsub` and other commands must be accessible). The PBS hosts must have access to the queue's server working directory (`$$WD`).

In addition, you can connect with Accelerator using the PBS emulation interface. This should only be used for debugging purposes.

Complete the steps below to integrate with PBS Professional.

1. Start a WX queue using the `wxmgr start` command.
2. Configure the `vovwxd` daemon for PBS.
 - a) Use the command `wx cmd vovwxconnect -pbs`.
 - b) Tune your WX configuration by modifying parameters as needed in `$SWD/vovwxd/config.tcl`. Tunable parameters are described in that file. Some useful configuration parameters include: `tasker,max`, `tasker,maxQueuedPerBucket` and `tasker,maxSubsPerBucket`.
3. Once the configuration is complete, you can submit jobs. Additionally there are some PBS parameters that can be specified via job resources:
 - `PBSdest:<destination queue>`: Defines the destination of the job. The destination names a queue, a server, or a queue at a server.
 - `PBSpriority:<priority>` : Defines the priority of the job.
 - `PBSproject:<project>` : Allows a root user or manager to submit a job as another user.
 - `PBSjobname:<name>` : Declares a name for the job.

```
wx run -r PBSdest:myPBSqueue PBSproject:myProjectName -- sleep 0
```

You can also use the `vovwxconnect`, with the `pbsemul NCQ` as a way to debug.

vovwxconnect: Usage Message

DESCRIPTION:

Utility to setup a connector for Accelerator Plus to a primary queue

OPTIONS:

```
-h                -- This help.
-v                -- Increase verbosity.
-nc NCQ           -- Connect with specified Accelerator queue.
-vovdir PATH      -- Specify VOVDIR for the Accelerator queue, if different
                  than that of Accelerator Plus.
-ncconfigdir     -- Specify NC_CONFIG_DIR for the Accelerator queue, if
                  different than that of Accelerator Plus.
-lsfemul NCQ      -- Connect with Accelerator but using the LSF emulation
                  interface (only for debugging)
-dd              -- Connect with Direct Drive
-pbs             -- Connect with PBS
-pbsemul NCQ      -- Connect with Accelerator but using the PBS emulation
                  interface (only for debugging)
-legacy          -- Connect using legacy daemons vovelasticd vovlsfd.
-show           -- Show what is connected.
-test           -- Validate connections by running a test job.
-nostart        -- Do not start the vovwxd daemon.
-loglevel L      -- Set verbosity level for vovwxd (0-6)
```

EXAMPLES:

```
% vovproject enable MY_WX_PROJECT
% vovwxconnect -nc vnc1 vnc2
% vovwxconnect -nc vnc1 -nc vnc2
% vovwxconnect -nc vnc_test -vovdir /some/path/to/a/vovdir/used/by/vnc_test
```

```
% vovwxconnect -pbs
% vovwxconnect -show
% vovwxconnect -test
% vovwxconnect -lsfemul vnc_test
```

Configure Accelerator Plus to Request PBS Resources

You can configure PBSResList to request PBS resources.

Use the steps below:

1. Request PBS resources with the following:

```
wx run -r "PBSResList:select=1:ncpus=1" --sleep 0
```

2. To request multiple resources, use the following:

```
wx run -r "PBSResList:select=2 PBSResList:resource1=1 PBSResList:resource2=abcd"
--sleep 0
```

Daemons for Accelerator Plus

Some key functionality in Accelerator Plus is provided by external daemons.

The functionality is described in the table below.

Daemon	Who needs it?	Description
vovwxd	The binary for running Accelerator Plus on top of Accelerator.	This daemon is required to interface the Accelerator Plus system to an existing Accelerator system.
vovelasticd (obsolete)	Only if you are running Accelerator Plus on top of Accelerator	This daemon is required to interface the Accelerator Plus system to an existing Accelerator system.

The status of the daemons can be viewed at the Daemons page, or can be shown with this command:

```
% wx cmd vovdaemonmgr show
vovresourced          OK
vovdbd                OK
vovwxd                OK
```

Theory of Operation

This section describes the theory of the operation of Accelerator Plus on Accelerator and how to address problems should they occur.

Theory of Operation: Connecting Accelerator Plus to Accelerator

In general, Accelerator Plus connects to a base scheduler with an elastic grid daemon.

Accelerator Plus connects to Accelerator with the daemon `vovwxd`. The run directory for this daemon is `.swd/vovwxd`. The three significant files are `vovwxd.log`, `config.tcl`, and `vovnc.tcl`.

- The `config.tcl` file allows the elastic parameters to be configured.
- A fresh timestamp on the log file confirms that the elastic daemon is running.

 **Note:** It is essential that the elastic daemon is running for jobs to be submitted to the base scheduler.

A change to the `config.tcl` file is automatically picked up by the `vovwxd`. A crashed or halted daemon can be restarted with the command:

```
% wx cmd vovdaemonmgr start
vovresourced          ALREADY RUNNING
vovdbd                ALREADY RUNNING
vovwxd                STARTING...
```

The `vovwxd` daemon watches the Accelerator Plus workload; if some bucket is waiting for hardware or software resources, then the daemon issues a request to the base scheduler for more resources.

When the underlying scheduler dispatches this job to an execution host, the job fires a new `vovtasker` that connects back to the Accelerator Plus, therefore advertising new computing resources that can be used to process the jobs queued in Accelerator Plus.

When this occurs, you will see an extra box appear in the LED monitor of the `vovconsole` (located on the upper bar of the `vovconsole` main window). The taskers started by an elastic daemon appear as additional boxes. The box size decreases as more taskers are added.

Key Notes

- `vovwxd` sends "tasker job requests" to the base scheduler; the actual workload in Accelerator Plus stays in Accelerator Plus and is executed as the base scheduler satisfies the requests from `vovwxd`
- The base scheduler only sees tasker requests. It has no direct visibility into the details of the Accelerator Plus workload, although the tasker's resource request (license, limits, memory, cores etc) precisely matches the underlying workload.
- To Accelerator Plus, it sees a "private" pool of taskers to which it can submit jobs. Being elastic, the pool may grow or shrink.
- 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.

Accelerator Plus taskers terminate when one of two conditions is met:

1. The maximum idle time is exceeded. This is controlled by the parameter "Max Idle Time" represented by the variable `CONFIG(tasker,maxidle)` in `config.tcl`, which is typically 30s

2. The maximum life is exceeded and the tasker is idle. This is controlled by the parameter `CONFIG(tasker,maxlife)` and has a default value of 1w (1 week), but typical values can be 1h or 2h. This allows FairShare policies on the underlying queue to be respected by keeping the maximum job duration in Accelerator to some reasonable range.

When a maximum idle is exceeded, the tasker stops accepting jobs by declaring itself to be "DONE"; the tasker turns blue in the Tasker Monitor status column. After a few seconds the tasker exits and the entry for it disappears from the monitor. At that time, the job representing the request for that tasker will terminate in Accelerator.

When the maximum life is exceeded, the tasker declares itself as suspended (purple) and no new jobs are sent to it, while all jobs currently running on that suspended taskers are allowed to continue running and to terminate normally. The tasker will exit when no more jobs are running on it. This behavior is essential for maintaining adherence to FairShare policies.

A normal Accelerator Plus tasker will go through the following states:

REQUESTED	Request for a new tasker resource has been issued to base scheduler.
NOLIC	Tasker is connecting and is waiting to be given a license.
READY	Tasker is ready to receive a job.
FULL	Tasker is running a job.
OVERLOADED	The load on the tasker is higher than a specified threshold.
SUSPENDED	Tasker is no longer accepting jobs.
PAUSED	Tasker has been paused by the base scheduler.
DONE	Done, about to be deleted.

When Things Go Well

In Accelerator Plus's Tasker Monitor, you will see a reasonable number of taskers listed; most are classed as FULL and some as REQUESTED. Normally, each tasker will have 1 slot and that will be occupied. For freshly created taskers they may go into a NOLIC state for a brief period while they establish their connection, license and capabilities. Some taskers may be SUSPENDED/ purple - they're finishing off the last job they've been allotted before max life kicks in. Some status will be READY/green for a brief period; most will be FULL/yellow (running a job). A few will be red indicating an overload.

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.

vovwxd Configuration

In order to successfully run jobs under Accelerator Plus, submitting to Accelerator, some configuration must be done up front.

Accelerator job classes that will be used by Accelerator Plus must be present in both Accelerator and Accelerator Plus. It is important to note that job classes are processed by Accelerator Plus only, and the tasker job submitted to the base queue will contain the post-processed resources, limits, JPP, etc. as defined by the job class. The tasker job in Accelerator will be annotated with the job class name for traceability reasons.

Resources that are referenced by jobs submitted to Accelerator Plus must be accessible from the downstream Accelerator queues.

```
===== Start sample WXSVD/vovwxd/config.tcl file =====  
  
### vovwxd configuration file  
  
# Specify the driver script to use for interfacing to the back-end batch  
# system. For NC, use "vovnc.tcl".  
set CONFIG(driver_script)          "vovnc.tcl"  
  
# Specify the space-separated list of back-end queues to which WX should  
# submit tasker requests.  
set CONFIG(queues)                 "vnc"  
  
# Specify the VOV named environment to use for the vovwxd daemon.  
# For NC, this will normally be the "BASE" environment.  
set CONFIG(cmd,env)                 "BASE"  
  
# Specify a baseline submission command. Normally not needed.  
set CONFIG(cmd,submit)              ""  
  
# How often should the vovwxd daemon cycle execute ?  
# The value is a VOV time spec and the default is two seconds.  
set CONFIG(refresh)                 5s  
  
# how frequently should we ask for job status from back-end queue ?  
# The value is a VOV time spec and the default value is one minute.  
set CONFIG(jobstat,checkfreq)       5m  
  
# Remove sick taskers that are older than?  
# Value is a VOV time spec, and the default value is five minutes.  
set CONFIG(sick,older)              5m  
  
# What is the maximum number taskers we should start?  
# Should be set to a high value to enable lots of parallelism.  
set CONFIG(tasker,max)              500  
  
# What is the maximum number of queued taskers per bucket that we should allow?  
set CONFIG(tasker,maxQueuedPerBucket) 100  
  
# 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 supersedes this value.  
set CONFIG(tasker,arrayMax)         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 overridden on a per job basis by putting
# a resource on the job that looks similar to the following.
# MAXlife:1w
set CONFIG(tasker,maxlife)          4h

# 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 overridden on a per job basis by putting
# a resource on the job that looks similar to the following.
# MAXidle:2m
set CONFIG(tasker,maxidle)         2s

# 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 vovwxd. For example setting
# this to "MAXlife:1w" will not work as you might expect.
set CONFIG(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 CONFIG(tasker,update)          15s

# Do we want to enable debug messages in the vovtasker log files?
# 0=no; 1=yes; default=0
set CONFIG(tasker,debug)           0

# What level of verbosity should the WX use when writing to its the log file?
# Valid values are 0-7; default=3
set CONFIG(log,level)              6

# What level of verbosity should the vovtasker use when writing to its the log file?
# Valid values are 0-4; default=1
set CONFIG(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 CONFIG(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 CONFIG(client,derate)          50

# What is the name of the launchers directory? (Default: \"../launchers\")
set CONFIG(launchers,dirname)     \"../launchers\"

# Remove launchers that are older than?
# Value is a VOV time spec, and the default value is one hour.
set CONFIG(launchers,older)       1h
```

```
# How many tasker submissions should be done for each
# job bucket, during each refresh cycle ? (Maximum number of submission commands to
# be executed)
# This parameter is a replacement for old (tasker,maxSubsPerBucket)
set CONFIG(launchers,maxLaunchersPerBucketPerCycle) 99

# Percentage of submissions for newly needed taskers
set CONFIG(launchers,quota) 10


# When we should retry requesting taskers on a queue that was not responding?
# Specify a period in vov time format or "0".
# If this parameter is "0", vovwxd dequeues all jobs in the buckets processed for the
# failed queue.
set CONFIG(failedAgentsCooldownPeriod) 0

===== End sample WXSVD/vovwxd/config.tcl file =====
```

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. 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/common/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.

VOV Security Keys

The VOV system supports a public/secret key pair-based security and authentication mechanism called *VOV Security Keys*. VOV security key authentication is the preferred authentication mechanism for REST and is also the basis for highly secure connections between Accelerator and the Event port on Monitor when RDS resource management is activated.

VOV security keys are based on ed25519 public key cryptography. Conceptually, they are like SSH keys:

- You create a public/private key pair.
- You use some other authentication means to connect to the server, and tell the server to trust your public key and associate it with your identity.
- You use your private key as part of an authentication "handshake" with the server, and it verifies your identity using your matching public key.

It's important to remember that you do not need a new public/private key pair for each queue where you intend to use key based authentication. You can create one key pair and set up the public half of your key on all the vovservers you wish to authenticate against.

VOV security keys in no way replace the security rules and policies enforced by `security.tcl` or Access Control Lists. The keys are only a means to authenticate the user, after which Security Levels and Access Control Lists still come into play.

Create a Public/Private Key Pair

Method 1 - Using the VOV CLI

If the host where the REST client will execute has access to the VOV software installation, you can use these VOV CLI commands:

```
% vovproject enable PROJECT
% vovsecurity keygen
```

By default, `vovsecurity keygen` writes the newly generated key pair into `$HOME/.vov/userkey`. If that file already exists, you will be prompted on whether you wish to overwrite the file. Overwriting the file means you will no longer have access to the previous key pair that you may have used to establish key based authentication on other vovserver instances.

Method 2 - Using the Browser

If the client host does not have access to the VOV command line interface on the same local network where the vovserver is running, then this method will work.

Browse to URL `<VOV_URL>/api/v3/apikeys/newkey`

The browser will display a generated random key data in a JSON format. The displayed data shows the new secret key followed by the corresponding public key.

Method 3 - Using Curl

This is equivalent to the prior method, but using the Linux `curl` command to access the target URL:

```
% curl -k -X GET <VOV_URL>/api/v3/apikeys/newkey | jq '.rows[0]'
```

Register Your Public Key on a vovserver Instance

Method 1 - Using the VOV CLI

If you have access to a host on the LAN where the vovserver is running and the VOV CLI commands, the CLI key registration method is convenient. Normally the public VOV security key is stored in `$HOME/.vov/userkey`, and the `vovsecurity getkey` command simply reads the public key. The `vovsecurity addkey` command will prompt for this user's password to authenticate the registration request, which is implemented behind the scenes as an authenticated REST request.

```
% vovproject enable PROJECT
% pubkey=$(vovsecurity getkey)
% vovsecurity addkey -kv $pubkey -kd "My Cool Key"
Enter your password for connecting to PROJECT on HOST: *****
```

To script the above sequence without being interactively prompted for a password, set the `VOV_PASSWORD` environment variable to the value of your password before invoking the `vovsecurity addkey` command.

Method 2 - Using the VOV REST API Interactive Web UI Page

The VOV project's web UI provides a means to register a user's public key at vovserver. To register a public key, follow this procedure:

1. Browse to the VOV REST API Interactive Web UI Page at `<VOV_URL>/html/vovrest.html`
2. Login with the user name that corresponds to the public key being registered.
3. Scroll down to the **apikey**s topic and click on the **POST** button beside the **/apikey Add Key** line.
4. Click **Try it Out** on the right side of the page.
5. Edit the Request Body JSON to add your public key for the "value" keyword, and to add a description string for the "description" keyword.
6. Click **Execute**.

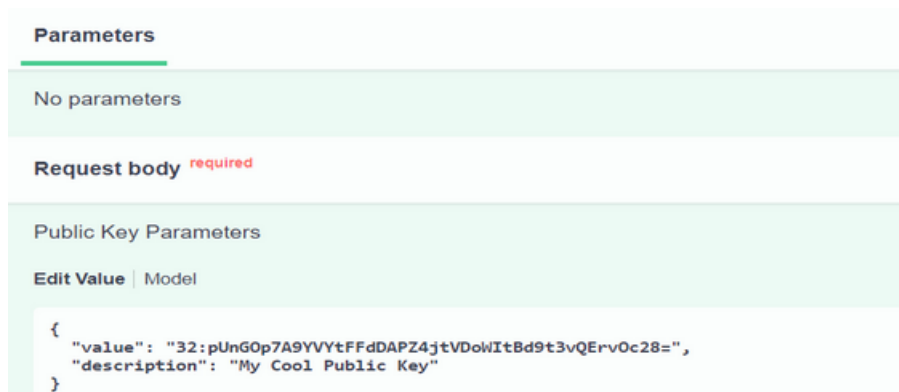


Figure 1:

Listing Your Public Keys on a vovserver Instance

Method 1 - Using the VOV CLI

You can list the public keys that have already been registered on a vovserver instance, using the following command in an enabled shell:

```
% vovsecurity listkeys
user1 32:Ihq6djlX5L9XQzRWRWy0Z2h15fD64JhJoqa7FpDmiyg= Backup key
user1 32:+fmR1SmWValRspQjcLQ7OGX266MZj/m90B5fii3FMTY= 2nd key
```

```
user1 32:wlefJq4QjTm2QgtWTUtD7kG1lnCyUtrKfobW/HPfoD0= 3rd Key
```

In this case, user1 has registered multiple public keys on the same vovserver instance.

Method 2 - Using the VOV REST API Interactive Web UI Page

The VOV project's web UI provides a means to list a user's registered public keys. Follow these steps:

1. Browse to the VOV REST API Interactive Web UI Page at <VOV_URL>/html/vovrest.html
2. Login with the user name that corresponds to the public key being registered.
3. Scroll down to the **apikey**s topic and click on the **GET** button beside the **/apikey List Key** line.
4. Click **Try it Out** on the right side of the page.
5. Click **Execute**.
6. Scroll down to the Response Body sub-window and view the JSON description of the list of registered public keys.

Using Key Based Authentication from a Python-Based REST Client

REST client authentication is typically done through VOV security keys that are generated and properly registered with a vovserver instance and stored in a local file on the client. The key based authentication is handled automatically inside the `submitRequest()` Python REST member function described below.

To perform key based authentication with the REST API, the following must be true:

- The vovserver webport must be non-zero (this is true by default).
- The vovserver webprovider must be set to "internal" (this is the default value).
- The `ssl.enableconfiguration` parameter should be set to 1 (this is the default value)
- A VOV security key pair should be generated and stored on the client host in a file (normally `~/.vov/userkey`), in the following format. This can be done using the `vovsecurity keygen` command or the REST program shown in [Create a Public/Private Key Pair](#) above.

```
% cat ~/.vov/userkey
secret-key = 32:WUhyiwOUBYqonfVR66fNVnHd6sfK9QbK257A8jheYfc=
public-key = 32:NIqRzIOhv0r7GZFTPkL0nQLZX6U6UTihleHhFwlvqnQ=
```

- Key pairs must be registered with a vovserver via the `vovsecurity addkey` command or the REST program shown in [Register Your Public Key on a vovserver Instance](#) above.

The simplest way to program a REST request using key-based authentication is to pass the two required arguments to the `submitRequest()` method function for the `VOVRestV3` class in the `vov_rest_v3.py` module that is included in `$VOVDIR`. Before running this Python program, perform the following setup from an enabled shell:

```
% export VOV_URL=$(vovbrowser)
% cp $VOVDIR/scripts/python/vov_rest_v3.py .
```

This Python program can be run with Python version 3 or higher. The `submitRequest()` function will automatically take care of these prerequisite actions before issuing a REST request:

1. Retrieve the server public key
2. If the first REST call, or if the underlying REST access token is expired, then the user will be authenticated using the secret VOV key from `~/.vov/userkey`. If the secret VOV key is stored in a different file, then add the keyword parameter `authKeyFile=filename` to the argument list in the `submitRequest()` function call.

```
#
# easy_REST_key_101.py
```

```
#
import os, sys, vov_rest_v3
url = os.environ['VOV_URL']
vrest = vov_rest_v3.VOVRestV3()
r = vrest.submitRequest("GET", url + "/api/v3/project/1/project")
print (r)
```

The following example shows an alternative to the above. It explicitly authenticates using the `authorizeWithKey()` method function prior to calling `submitRequest()`. This allows one user impersonate another user whose secret key is known. It also allows explicit passing of the secret key as an argument instead of passing the name of a local file containing the key.

Before running the Python program below, take these preparatory steps:

- Set `VOV_URL` and copy `vov_rest_v3.py` locally:

```
% export VOV_URL=$(vovbrowser)
% cp $VOVDIR/scripts/python/vov_rest_v3.py .
```

- Set the `MY_SECRET_KEY` environment variable to the user secret VOV security key (normally viewed in `~/ .vov/ userkey`)
- Set the `VOV_SERVER_PUBLIC_KEY` environment variable to the value of the server public key that is assigned to a VOV project at creation time. There are three ways to do this:

```
# Method 1
Browse to URL    <VOV_URL>/api/v3/apikeys/serverkey

# Method 2
% curl -k -X GET  <VOV_URL>/api/v3/apikeys/serverkey | jq '.rows[0][0]'

# Method 3
% vovproject enable PROJECT
% vovsecurity getserverkey
```

This example explicitly calls the `authorizeWithKey()` method function to authenticate, followed by the `submitRequest()` function to issue the REST request. This might be convenient if multiple key pairs are registered with the vovserver for this user, and the desired key pair is not currently present in `~/ .vov/userkey`.

```
#
# explicit_REST_key_101.py
#
import os, sys
import vov_rest_v3

url = os.environ['VOV_URL']
secret_key = os.environ['MY_SECRET_KEY']
sp_key = os.environ['VOV_SERVER_PUBLIC_KEY']
user = os.environ['USER']
vrest = vov_rest_v3.VOVRestV3()
vrest.authorizeWithKey(url, secret_key, sp_key, username=user)
r = vrest.submitRequest("GET", url + "/api/v3/project/1/project")
print (r)
```

Advanced Key Handling

The REST client will use the `libsodium` library to construct an encryption message which is a combination of the user's private key and vovserver's public key, which will be sent to REST at the endpoint `/api/v3/token` with `grant_type` set to `apikey` and `apikey` set to the contents of the encrypted message.

If you wish to handle REST sessions yourself, look inside `vov_rest_v3.py` at the `_getEncryptedMessage()` procedure to see how the authentication handshake message is composed using the user's private key and the vovserver's public key. In `_getEncryptedMessage()`, you can see how it decodes the two key strings into raw bytes, creates a PyNaCl Box object using the two keys, and then encrypts the message by calling `Box.encrypt()`. The encrypted message is then base-64 encoded and returned as a string that the vovserver REST API can understand.

```
def _getEncryptedMessage(self, userseckey, serverpubkey, message=None):
    """Generated an encrypted message block using the caller's secret key and the
    server's public key.

    :type userseckey string
    :param userseckey User's secret key. It should be in base64 format with the
    binary length prepended,
        and a colon separating the length and the base64 string.

    :type serverpubkey string
    :param serverpubkey vovserver's public key. It should be in base64 format with
    the binary length prepended,
        and a colon separating the length and the base64 string.
        This can be retrieved from $VOVDIR/local/registry/ as the
    'CLIENT_PUBKEY' entry.

    :type message string
    :param message The message to encode. Optional. If None, a generic default
    message is used. It doesn't
        effect the key authentication process in any way.

    :rtype: string
    :return: A string containing a base64 encoded, then url-encoded buffer.
    """
    idx = userseckey.find(':')
    if idx == -1:
        return ""
    buflen = int(userseckey[:idx])
    userseckey = userseckey[idx + 1:]
    seckeybuf = base64.b64decode(userseckey)
    seckeybuf = seckeybuf[:buflen]

    idx = serverpubkey.find(':')
    if idx == -1:
        return ""
    buflen = int(serverpubkey[:idx])
    serverpubkey = serverpubkey[idx + 1:]
    pubkeybuf = base64.b64decode(serverpubkey)
    pubkeybuf = pubkeybuf[:buflen]

    if message is None:
        message = 'VOV API Key Client Auth ' + str(time.time())
    messageBuf = message.encode("utf-8")
    secretKey = PrivateKey(seckeybuf)
    publicKey = PublicKey(pubkeybuf)

    box = Box(secretKey, publicKey)
    cipher = box.encrypt(plaintext=messageBuf)
    cipherLen = len(cipher)
    encodedCipher = base64.b64encode(cipher)
    cipherStr = str(cipherLen) + ':' + encodedCipher.decode("utf-8")
    return cipherStr
```


The following code snippet demonstrates how to get a REST session token using the `_getEncryptedMessage()` procedure above:

```
url = 'https://hostname:9100/api/v3/token'
clientSecret = ''
postData = {}
postData["grant_type"] = "apikey"
postData["client_id"] = "Python client"
postData["client_secret"] = clientSecret
postData["username"] = "myusername"
postData["apikey"] = self._getEncryptedMessage(userseckey, serverpubkey)
session = requests.Session()
response_obj = session.post(url, timeout=self.connectionTimeout, verify=False,
                            data=postData)
respContent = response_obj.text
respStatus = response_obj.status_code

if (respStatus == 200):
    contentJSON = json.loads(respContent)
    if ("access_token" in contentJSON):
        sToken = contentJSON['access_token']
```

Once you have the session token, you can use it for subsequent REST API requests, similar to previous REST examples which used username/password authentication to get the session token.

Deleting A Key from vovserver

You can delete your own key from vovserver using the `vovsecurity` command:

```
vovsecurity delkey -kv 32:JA0iq1mLbdSuuDKj1PeEoHJX+fYqR/0HxpI5nhgThE8=
```

Users with ADMIN level security access in the project can list and delete keys belonging to other users:

```
> vovsecurity listkeys -a
user1          32:JA0iq1mLbdSuuDKj1PeEoHJX+fYqR/0HxpI5nhgThE8= Primary Key
user2          32:Ihq6djlX5L9XQzRWRWy0Z2hl5fD64JhJoqa7FpDmiyg= Primary Key
user2          32:+fmR1SmWValRspQjcLQ7OGX266MZj/m90B5fii3FMTY= 2nd key
user2          32:wlefJq4QjTm2QgtWTUtD7kG1lnCyUtrKfobW/HPfoD0= 3rd Key
> vovsecurity delkey -kv 32:JA0iq1mLbdSuuDKj1PeEoHJX+fYqR/0HxpI5nhgThE8= -u user1
```

Regulate Access to Accelerator Plus

Like all vovservers, access to Accelerator Plus can be controlled by editing the security file `wx.swd/security.tcl`.

Example: Restricting Accelerator Plus to a Single User

```
# Example of wx.swd/security.tcl file for a Private Accelerator Plus
vtk_security john      ADMIN +
```

After configuring the security file, Accelerator Plus must be reset to apply those the changes:

```
% wxmgr reset
```

Example: Opening up Accelerator Plus to Many Users

To make Accelerator Plus accessible by all users, assign everyone USER privileges from all hosts, as in this example:

```
# This is an example of the wx.swd/security.tcl file.  
# The first + means "everybody"  
# The second + means "from all hosts"  
vtk_security john ADMIN +  
vtk_security + USER +
```


```
% wxmgr reset
```

Job Status

In Accelerator Plus, each job goes through a number of states until completion.

The states are described in the following table:

Status	Color	Description
Idle	BlueViolet	If the node is a job, either it has not been run successfully yet or it needs to be run again, because one of its inputs has been modified since the last time the job was executed. If the node is a file, it is the output of a job that is not Idle.
Queued	Light blue	The job is scheduled to be run. It may be already queued or it will go in the queue as soon as all its inputs are ready.
Running	Orange	The job is currently being retraced; it has been dispatched to one of the taskers. All the outputs of such a job are either RETRACING or RUNNING.
Done	Green	If the node is a job, it has run successfully. If the node is a file, it is up-to-date with respect to all other files and jobs on which it depends.
Failed	Red	The job ran and failed.
Transfer	Cream	The job is being transferred to another cluster and it is not yet running.
Suspended	Pink	The job was running (or retracing) and one of the processes belonging to the job is currently suspended.
Sleeping	Black	Either the job caused an output conflict upon submission (bad dependencies) or the job was not reclaimed by any tasker upon crash recovery.

Status	Color	Description
Withdrawn	Gray	<p>A job has been withdrawn after dispatching, such as by the preemption daemon.</p> <div style="border: 1px solid #ccc; padding: 5px; margin-top: 10px;"> <p> Note: This status occurs rarely and tends to be hard to observe.</p> </div>


The normal sequence for a successful job is **Idle > Queued > Running > Done**

The normal sequence for a failing job is **Idle > Queued > Running > Failed**

Job Submission Policy

The job submission behavior of Accelerator Plus or Accelerator Plus can be controlled by the file `vnc_policy.tcl`, which resides in the vovserver configuration directory.

This file is used to define the procedures that are listed below.

 **Note:** `vnc_policy.tcl` can now reside in `vnc.swd/vnc_policy.tcl` as well as `$VOVDIR/local/vnc_policy.tcl`.

When placed in the configuration directory, it only affects that Accelerator Plus instance. When placed in the 'local' directory, it affects all Accelerator Plus instances.

Procedures for Customizing Job Submission

Procedure	Args	Description								
VncPolicyDefaultPriority	{ user }	Assign the default priority to a job based on the user.								
VncPolicyDefaultResources	{ }	Compute the default resources required by a job.								
VncPolicyGetJobInfo	{ key }	<p>Retrieve job information. Following are the available key values:</p> <table style="width: 100%; border: none;"> <tr> <td style="padding: 5px;">tool</td> <td style="padding: 5px;">Tool or command name, such as hsim</td> </tr> <tr> <td style="padding: 5px;">command</td> <td style="padding: 5px;">Complete command line (without wrapper)</td> </tr> <tr> <td style="padding: 5px;">user</td> <td style="padding: 5px;">Login name submitting the job</td> </tr> <tr> <td style="padding: 5px;">setName</td> <td style="padding: 5px;">Name of the set in which the job is to be placed</td> </tr> </table>	tool	Tool or command name, such as hsim	command	Complete command line (without wrapper)	user	Login name submitting the job	setName	Name of the set in which the job is to be placed
tool	Tool or command name, such as hsim									
command	Complete command line (without wrapper)									
user	Login name submitting the job									
setName	Name of the set in which the job is to be placed									

Procedure	Args	Description
		<p>group The group the submitter requested</p> <p>inputs Inputs to the job</p> <p>outputs Output files of the job</p> <p>mailuser Email address, if notification was requested</p> <p>wrapper Name of the FT wrapper program, such as vw</p> <p>priority,default Default submission priority</p> <p>priority Requested submission priority</p> <p>resources Requested submission resources</p> <p>env Name of requested job run environment</p> <p>xdur Expected duration of job</p>
VncPolicyUserPriority	{ user schedPriority }	Limit the scheduling priority based on the maximum allowed to the user.
VncPolicyUserPriorityExec	{ user execPriority }	Limit the execution priority for a job. By default, this returns the priority that has been passed in.
VncPolicyValidateCommand	{ commandLine }	Make sure that the command line for a job obeys site-specific rules.
VncPolicyValidateEnvironment	{ envName }	Make sure that the environment name for a job obeys site-specific rules.
VncPolicyValidateOptions	{ subCommand argv }	Ensures the arguments obey site-specific rules. Returns a modified list of options. See example below.
VncPolicyValidateResources	{ reslist }	Ensure that the resource list for a job obeys rules defined by the Accelerator Plus administrator.

Example for VncPolicyValidateOptions:

```
proc VncPolicyValidateOptions { subCommand argv } {
  set nargsv []
  set maxAutoKill [VovParseTimeSpec 30d]
  while { $argv ne {} } {
    set arg [shift argv]
```

```
        switch -glob -- $arg {
            "-autokill" {
                lappend nargsv $arg
                set value [VovParseTimeSpec [shift argv]]
                if { $value > $maxAutoKill } {
                    lappend nargsv "30d"
                } else {
                    lappend nargsv $value
                }
            }
            default {
                lappend nargsv $arg
            }
        }
    }
}
return $nargsv
}
```

The VncPolicy* procedures are called at job submission time, and may cause the job entered into the server to have modified resources or priority compared to what the submission requested.

The following is an example for vnc_policy.tcl:

```
# This is an example of vnc_policy.tcl
proc VncPolicyDefaultResources {} {
    global env
    return "$env(VOVARCH) RAM/50"
}

proc VncPolicyValidateResources { resList } {
    #
    # This policy adds a minimum RAM requirement
    # for all submitted jobs.
    #   global VOV_JOB_DESC
    #   if { $VOV_JOB_DESC(tool) == "vovresgrab" } {
    # Do not touch this type of jobs (see vovresreq).
    return $resList
    }

    if [regexp "RAM/" $resList] {
        # Job already has a RAM constraint.
    } else {
        # Add a RAM constraint.
        lappend resList "RAM/100"
    }
    return $resList
}
```

The following is an example using the tool name. This can be used to send jobs of a certain tool to specific hosts. A Tcl catch{ } is used in case someone uses this file with an older version by mistake.

Fragment of \$VOVDIR/local/vnc_policy.tcl:


```
# This is a second example of vnc_policy.tcl
proc VncPolicyDefaultResources {} {
    global env
    return "$env(VOVARCH)"
}

proc VncPolicyValidateResources { resList } {
    #
```

```
# This policy sends tharas jobs to vovtasker hosts offering 'tharas_host'
# and keeps other kinds of jobs off those hosts
#
catch {
    set jtool [VncPolicyGetJobInfo tool]
    if { "$jtool" == "tharas" } {
        lappend_no_dup resList tharas_host
    } else {
        lappend_no_dup resList "!tharas_host"
    }
}
return $resList
}
```

Throttling Job Submission Rate

There is also a way to throttle users who have submitted a number of jobs over a configurable threshold. This was implemented so that users trying to submit too many job in a small time frame can not overload vovserver. The process adds a delay to job submission for users that have gone over that threshold.

 **Note:** Although mentioned in this section because they affect job submission, these values are set in the vovserver configuration file `policy.tcl`.

Procedure	Args	Description
<code>hog.protection.enable</code>	()	The default is 0, which means it is disabled. Add a 1 to enable it.
<code>hog.protection.jobcountth</code>	(N)	N represents the number of jobs which will trigger this threshold. Default is 100000. Min value is 1000. Max value is 999999.
<code>hog.protection.clientdelay</code>	(S)	S is the number of seconds to delay the submit of a user how has triggered this rule. The default is 1 second. Min is 1 second. Max is 600.

Accelerator Plus Customization

Many aspects of Accelerator Plus behavior can be customized.

For example:

- Job submission can be controlled with a layer of policies defined in the `vnc_policy.tcl` file, located either in `wx.swd/vnc_policy.tcl` or in `$VOVDIR/local/vnc_policy.tcl`.
- The Accelerator Plus server behavior can be controlled using the parameters in `wx.swd/policy.tcl`.
- The defaults for `wx run` can be controlled in `$VOVDIR/local/vncrun.config.tcl`. A common use of this file is to disable the automatic check for the validity of the submission working directory.
- The defaults for `wx list` can be controlled in `$VOVDIR/local/vnclist.config.tcl`. A common use of this file is to force list result caching, or to disable listing by job name.
- Some aspects of the GUI can be controlled using `wx.swd/gui.tcl`.

Customize the wx run Command

The `wx run` command has built-in default features that include checking the validity of the run directory, enabling job profiling, etc. This section describes how the Accelerator Plus administrator can use the file `$VOVDIR/local/vncrun.config.tcl` to modify some defaults. This file does not exist by default; it must be created when needed.

Configure Defaults for the 'wx run' Command

The defaults are controlled by slots in the `VOV_JOB_DESC` array variable. The `vncrun.config.tcl` file is loaded after the defaults are set; these defaults can be overridden.

For details about `VOV_JOB_DESC`, refer to [Define Jobclasses](#).

Examples

When submitting a job, the default is to check for a logical name (equivalence) for the filesystem where the run directory is located. This is controlled by the `check,directory` slot.

To change the default to not check the directory, add the following to the `vncrun.config.tcl` file:

```
set VOV_JOB_DESC(check,directory) 0
```

When submitting a job, the default is not collecting profile information, because the data can be large, unless the `-profile` option is used. To make collection of the profile collection the default, add the following to the config file.

```
set VOV_JOB_DESC(profile) 1
```

The example file below shows other commonly-used settings:

```
# Example content of vncrun.config.tcl
set VOV_JOB_DESC(check,directory) 0

# Other settings that may be useful.
# set VOV_JOB_DESC(priority,default) [VncPolicyUserPriority $username]
# set VOV_JOB_DESC(priority,sched) $VOV_JOB_DESC(priority,default)
# set VOV_JOB_DESC(priority,exec) $VOV_JOB_DESC(priority,default)

# set VOV_JOB_DESC(autokill) 0
# set VOV_JOB_DESC(autoforget) 1
# set VOV_JOB_DESC(legalExit) "0"
# set VOV_JOB_DESC(mailuser) ""
# set VOV_JOB_DESC(wrapper) "vw" ;
# set VOV_JOB_DESC(preemptable) 1
# set VOV_JOB_DESC(profile) 0
# set VOV_JOB_DESC(schedule,date) 0
# set VOV_JOB_DESC(xdur) -1
```

Customize the wx list Command

This section describes how the Accelerator Plus administrator can use the file `$VOVDIR/local/vnclist.config.tcl` to modify some defaults for the `wx list` command. This file does not exist by default; it must be created when needed.

Enable List Cache

By default, list results are obtained from the server in real-time. In large-scale workload environments, repeated queries can impact server performance. To reduce this impact, a list result cache can be enabled:

```
set NCLIST(cache,enable) 1
```

Configure List Cache Expiration

If the list cache is enabled, list results will be written to a client-side file, and subsequent list requests will be obtained from this file, up to the cache expiration. The default expiration is 30s from creation. After this time, the cache file will be regenerated upon the next list request. To set the cache expiration to a different value:

```
set NCLIST(cache,timeout,default) 60
```

Disable Listing by Job Name

Another list operation that can affect server performance in a large-scale workload environment is listing by job name. This is due to the need to compare string values across many jobs. Listing by job name can be disabled entirely, this way:

```
# Fragment of $VOVDIR/local/vnclist.config.tcl  
set NCLIST(listbyjobname,enable) 0
```

Troubleshooting

When a problem occurs, first run `vovcheck` and correct the reported errors:

```
% vovcheck  
vovcheck: message: Creating report /usr/tmp/vovcheck.report.15765  
Test: BasicVariables [OK]  
Test: EnvBase [OK]  
Test: FlowTracerLicense [OK]  
Test: FlowTracerPermissions [OK]  
Test: GuiCustomization [OK]  
Test: HostPortConflicts [ERROR]  
Test: Installation [OK]  
Test: OldMakeDefault [OK]  
Test: Rhosts [OK]  
Test: Rsh [OK]  
Test: SecurityPermissions [WARN]  
Test: TaskerRoot [OK]  
Test: WritableLocal [WARN]  
Test: WritableRegistry [OK]  
Test: vovrc [OK]  
vovcheck: message: Detailed report available in /usr/tmp/vovcheck.report.15765
```


The Server Does Not Start

The recommended checkpoints:

- Make sure you have a valid RLM license. Type:

```
% rlmstat -a
```

- Check if the server for your project is already running on the same machine. Do not start a Accelerator Plus project server more than once. For example, you can try:

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

- Check if the server is trying to use a port number that is already used by another vovserver or by 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 thVOV_PORT_NUMBER to specify a known unused port number. The best place to set this variable is in the `setup.tcl` file for the project.
- Check if the server is trying to use an inactive port number that cannot be bound. This can happen when an application, perhaps the server itself, terminates without closing all its sockets.

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 VOV server 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:

1. 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 /opt/rtda/  
latest/linux64/tcl/vtcl/vovresourced.tcl -p acprose  
...
```

2. You can wait for the process to die on its own, or you can kill it, for example with `vovkill`

```
% vovkill pid
```

3. Restart the server.

- You run the server as the Accelerator Plus administrator user. Please check the ownership of the file `security.tcl` in the server configuration directory `vwx.swd`.

The UNIX Taskers Do Not Start

Accelerator Plus normally relies on remote shell execution to start the taskers using either `rsh` or `ssh`.

- If using `rsh` try the following:

```
% rsh host vovarch
```

where *host* is the name of a machine on which there are problems starting a tasker.

This command should return a platform dependent string (such as "linux") and nothing else. Otherwise, there are problems with either the remote execution permission or the shell start-up script.

- If the error message is similar to "Permission denied", check the file `.rhosts` in your home directory. The file should contain a list of host names from which remote execution is allowed. You may have to work with your system administrators to find out if your network configuration allows remote execution.
- If using `ssh`, perform the test above but use `ssh` instead of `rsh`.
- If you get extraneous output from the above command, the problem is probably in your shell start-up script. If you are a C-shell user, check your `~/ .cshrc` file. Following are guidelines for a remote-execution-friendly `.cshrc` file:

- # Echo messages only if the calling shell is interactive. You can test if a shell is interactive by checking the existence of the variable `prompt`, which is defined for interactive shells. Example:

```
# Fragment of .cshrc file.  
if ( $?prompt ) then  
  echo "I am interactive"  
endif
```

- # Many `.cshrc` scripts exit early if they detect a non-interactive shell. It is possible that the scripts exit before sourcing `~/ .vovrc`, which causes Accelerator Plus to not be available in non-interactive shells. Compare the following fragments of `.cshrc` files and make sure the code in your file works properly:

The following example will not work properly for non-interactive shells:

```
if ( $?prompt ) exit  
source ~/ .vovrc
```

This example is correct; source `.vovrc` and then check the `prompt` variable:

```
source ~/ .vovrc  
if ( $?prompt ) exit
```

This example is also correct:

```
if ( $?prompt ) then  
  # Define shell aliases  
  ...  
endif
```

```
source ~/.vovrc
```

Do not apply `exec` to a sub-shell. This will cause the `rsh` command to hang.

```
# Do not do this in a .cshrc file
exec tcsh
```

License Violation

Accelerator Plus is licensed by restricting the number of tasker slots. This is the sum of the tasker slots from both elastic and statically defined taskers (if any).

You can find out the capacity of your license (`wx_slots`) with the following command:

```
% rlmstat -avail
```

The file `$VOVDIR/../../../../vnc/vwx.swd/taskers.tcl` defines the list of static taskers that are managed by the server. Make sure the number of tasker hosts is within the license capability.

Crash Recovery

In the event of a crash or failover, you can find a checklist of what to do at <http://wx-host:wx-port/cgi/sysrecovery.cgi>.

This address can found using the command:

```
wx cmd vovbrowser -url /cgi/sysrecovery.cgi
```

When Things Go Wrong

The following may occur after a major infrastructure event, a problem with submission scripts or a change to the workload or a change to things like Limits.

The most significant symptom is "stuck jobs": most likely `elastic` daemon has stopped. Check the `elastic` daemon log.

 **Note:** Both cases are common Accelerator debug scenarios.

If the timestamp is not fresh, you will need to restart:

- Save off the existing log file (and send that file to Altair for diagnosis).
- Restart: `nc -f $WxQueueName cmd vovautostart`

If elastic daemon is running:

- Check that tasker jobs are being submitted and that they are being executed by the base scheduler. To do this, connect to the correct Accelerator cluster through the web browser, locate the job set called `vovelasticd`. This set contains other sets, one set for each Accelerator Plus session.
- Locate the appropriate set for your Accelerator Plus session. Look at the name of the set.
- If you see only cyan (scheduled) jobs, the problem is that the base scheduler cannot schedule these tasker jobs. You need to debug why these jobs are not being run by the underlying scheduler.
- If the jobs are getting run but keep failing (turning red) then debug and determine the reason for those failures.

When Jobs Are Not Running

You may discover that a tasker job is asking for an impossible resource. This is often due to an error in the resource requirements, or an error in the configuration. You fix the problem but Accelerator Plus continues to not run jobs. In this case, those "impossible" jobs are still seen by the base scheduler (and are therefore not runnable) but also they are seen by the elastic daemon, which assumes that these jobs are runnable and that no new jobs should be submitted: a *live-lock* scenario.

The recommendation here is to dequeue any queued jobs in the base scheduler after changing the problematic resource request. This lets the elastic daemon launch replacement jobs.

A different scenario is tasker jobs being scheduled, dispatched and running in the base but no jobs are getting executed inside the Accelerator Plus session. The symptom is a growing list of green nodes (valid) with a small number of orange (running) and cyan (scheduled) jobs. In this case, the base is dispatching the tasker jobs without a problem, but Accelerator Plus is not making use of them. The results: they execute, waiting for the end user job that never comes until they hit their maximum idle limit.

For this case, we recommend checking the Accelerator Plus session using either a `vovconsole` (`wx -q $WxQueueName cmd vovconsole`) or an Accelerator Plus monitor. If you see no activity in the LED bar (taskers connecting, waiting and then terminating - often yellow-to-green-to-black), most likely there is a problem with the configuration. The taskers that the base scheduler is executing are not connecting back to the desired Accelerator Plus session.

For the configuration, verify that the taskers are being launched with the right parameters and check that the config files are correct:

- If the LED monitor is active, then taskers are connecting and the failure to launch is mostly likely in Accelerator Plus. A common problem is that the job requests a limit or a special resource: the limit must be satisfied in Accelerator Plus and the base scheduler. In this case, elastic daemon tries to detect expandable limits such as `Limit:foo@USER@N` and will set the limit to be unlimited in Accelerator Plus, and then pass the limit request on to the base scheduler where it is honored. Occasionally, this process goes wrong.
- If the limit does not exist or is set to 0 in Accelerator Plus, then the job will not launch. In this state, the job will appear to be queued (cyan color), but you will not see a bucket created for it (`wx mon` can be used to help diagnose that case). To add the missing resources edit `resources.tcl`. In general, set the resource value to "unlimited" in Accelerator Plus; the restrictive value in the base scheduler will still be honored.

What to Check When Jobs are Not Being Submitted

Accelerator Plus reports the tasker as SICK and it has a missing heartbeat. Check in the base scheduler and see that the tasker has been suspended. In this case someone or something has suspended the tasker - both it and its underlying job are suspended (T-state/CTRL-Z). Find out why - it could be a user or RAM sentry - the tasker job may have some property information telling you what. When resumed the tasker will become healthy (i.e. non SICK) and continue normally within the base scheduler and Accelerator Plus.

Everything seems fine but jobs are not being submitted. Check the base scheduler and if you find that the tasker jobs aren't running due to FairShare reasons then it may be possible there's a regression that is using the same FairShare node and subgroup (either from Accelerator Plus or native in the base scheduler). In this case the jobs will be treated in a first-come-first-served basis.

Avoid Suspending Accelerator Plus Taskers in the Base Scheduler

Elastic tasker jobs in the base scheduler should not be subjected to suspension events from users or the preemption system. Accelerator Plus supports a form of preemption called *modulation*. In this case, the base scheduler queue requests the Accelerator Plus tasker to terminate on the next Accelerator Plus job boundary.

Job Placement Policies

One of the few differences between Accelerator Plus and Accelerator is that, while Accelerator supports many placement policies, Accelerator Plus only supports the job placement policy called "first", meaning that a job is dispatched to the first tasker found that is capable of running the job.

You can still specify any JPP as you submit the workload, but that JPP is passed to Accelerator (assuming Accelerator is the base scheduler) while Accelerator Plus itself uses the "first" JPP.

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 appropriate 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	<code>\$SWD/vovnginxd/conf/nginx.conf.template</code>
2.	Site-wide	<code>\$VOVDIR/local/config/vovnginxd/nginx.conf.template</code>
3.	Installation-specific(edits not recommended)	<code>\$VOVDIR/etc/config/vovnginxd/nginx.conf.template</code>

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 webserver support TLS/SSL Protocol communication via "https" - prefixed URLs when configured correctly.

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	<code>\$VOVDIR/local/ssl</code>	<code>wildcard-crt.pem</code> <code>wildcard-key.pem</code>
2.	Host-specific	<code>\$SWD/config/ssl</code>	<code>hostname-crt.pem</code> <code>hostname-key.pem</code>
3.	Host-specific (auto-generated and self-signed)	<code>\$SWD/config/ssl</code>	<code>hostname-self-crt.pem</code> <code>hostname-self-key.pem</code>



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
```

Web Interface

Some features of the Accelerator Plus web interface can be configured by the administrator. Configuration for these items is performed in the `vnc.swd/config/web.cfg` file.

The complete list of customizable items is shown below.

Example Configuration: `$VOVDIR/etc/config/nc/web.cfg`:

```
### NODE VIEWER SETTINGS ###  
# Use a select (drop-down) menu for priority-based retrace controls  
# Set to 0 to disable, set to 1 or comment-out to enable (default)  
# set nodeviewer(retraceSelect) 1  
  
# Use a select (drop-down) menu for preemption controls  
# Set to 0 to disable, set to 1 or comment-out to enable (default)  
# set nodeviewer(preemptSelect) 1
```

Using Accelerator as a Base Scheduler

When interfacing to Accelerator, a minimal (zero) configuration for Accelerator Plus is recommended.

Do not configure each Accelerator Plus instance (resources, limits, etc.), and instead rely on the base scheduler's configuration to implement these policies. This approach allows many Accelerator Plus instances to be employed without a corresponding increase in administration overhead.

A notable exception is with queue-specific job classes: jobclasses need to be present in the Accelerator Plus instance otherwise an error is flagged. To avoid duplication of code, it may be desirable to symlink the base scheduler's jobclass directory to the Accelerator Plus instance. It may be necessary to copy or symlink the Accelerator `vnc_policy.tcl` file to the Accelerator Plus instance, for example: data specific to the job submission environment is required to run the policies.

Also, the jobclasses must be written so that they can be sourced multiple times, (e.g. to prevent duplicate entries in the resource lists), since it is likely that they will be executed once in the Accelerator Plus context and once in the Accelerator context.

The use of preemption within Accelerator Plus is not recommended; it is best to let Accelerator do all necessary preemptions, which will result in [job modulation](#).

FairShare works the same in Accelerator Plus and in Accelerator. You will notice that, in Accelerator, FairShare convergence will be somewhat slower than you would expect, because Accelerator Plus is effectively sending to Accelerator much longer jobs than the jobs in the Accelerator Plus workload, due to the fact that Accelerator sees the "vovtasker" as a single job.

Many Accelerator Plus Queues on Many Accelerator Queues

For the "many to many" model, where multiple Accelerator Plus queues are running over multiple Accelerator queues, you should create separate Accelerator Plus queues, and configure the `vovwxd` daemon for each of them, then specify the `CONFIG(queues)` parameter accordingly for each Accelerator Plus queue.

Jobclasses

Jobclasses provide the following advantages:

- Simplifying the command line for job submission, which can prevent errors and omissions.
- Emulate the concept of *queues*, which is similar to the processes of other batch processing systems. This queue emulation enables additional behaviors such as:
 - # Automatic revocation of resources that have been grabbed by jobs in the jobclass but are not used
 - # Automatic warning and termination of jobs that are stuck: jobs that have been dispatched to a vovtasker but appear to be using no CPU time

A jobclass represents a collection of `wx run` options that are needed to run a type of jobs, such as VCS regression jobs.

Membership in a jobclass can be used to differentiate between jobs in preemption: preempt jobs in a regression jobclass to free up resources for jobs in an interactive jobclass.

If more than one `-C` option is given, the jobclasses are processed left-to-right as the command line is parsed. This method requires great care and planning.

Create Jobclasses

The administrator of Accelerator Plus can define jobclasses using one of the following methods:

- Logged in as ADMIN, click the **Job classes** link in the **Workload** section of the Accelerator Plus main page. This page displays all of the available jobclasses, and allows creating and editing jobclasses, and allows the administrator to designate a default jobclass.
- Directly add Tcl syntax files in the directory `jobclass` under the server working directory, which is typically `$VOVDIR/../../vnc/vnc.swd/jobclass`.

Each file in the `jobclass` directory manipulates the submission parameters defined in the Tcl array `VOV_JOB_DESC` so as to define a jobclass. See the following table for the meanings of the items in this data structure. The variable `classDescription` holds a string used for documentation, i.e. a one-line summary of the jobclass. The variable `classEditable` holds a boolean value that controls whether the jobclass can be edited using the jobclass web UI page.

An optional procedure `initJobClass` can be defined to do any initializations needed for the jobclass to perform correctly. Often, this is used to create any Limit: resources that may be used by the jobclass.

The files in the `jobclass` directory are parsed by `vovresourced` when it starts, and any `initJobClass` procedures are evaluated once.

The jobclass definition files are located using a search path.

The built-in search path is computed by a procedure `VncJobClassSearchPath`, and is shown in the table below.

This procedure adds any directories named by the environment variable `VOV_JOBCLASS_DIRS` to the beginning of the path, analogous to `VOV_ENV_DIR`.

This procedure is defined in `$VOVDIR/tcl/vtcl/vovutils.tcl`. You may change the search path for jobclass files by including a modified definition in `vnc.swd/resources.tcl`, and also in `vnc_policy.tcl`, if used. See example.

Search path for jobclass definitions:

```
#directories named by VOV_JOBCLASS_DIRS
<project>.swd/jobclass
$VOVDIR/local/jobclass
$VOVDIR/etc/jobclass
```

Use Additional Jobclass Directories

If you have an extensive Accelerator Plus setup, you may wish to manage jobclasses in a more-centralized way than placing their definitions into each `vncNNN.swd/jobclass` directory. The following example shows one way to accomplish this.

For this example, we want to implement a system where jobclasses are searched for first in the specific Accelerator queue, then in a site-specific directory, and finally in a global area.

We implement this by using the regular queue-specific directory, and two symlinks called `jobclass_site`, and `jobclass_global` under `vncNNN.swd` that resolve to the site-specific and global directories for jobclasses. You will need to arrange for the jobclass directories to be available and up to date at each site.

Additionally, some generic code is shown that may be dropped into `vncNNN.swd` to automatically compute the value of the `VOV_JOBCLASS_DIRS` environment variable.

Code to compute `VOV_JOBCLASS_DIRS`:

```
# This is file jcdirs.tcl
# NOTE: vovwait4server runs in vtclsh, and sources setup.tcl
# so vovGetProjectFileName may not be used here

# If env-var VNCSWD is set, use its value, it is much simpler
if { [info exists env(VNCSWD)] && [file isdirectory $env(VNCSWD)] } {
    set cfgdir [file join $env(VNCSWD) $env(VOV_PROJECT_NAME).swd]
} else {
    # compute path to the .swd directory
    # setup.tcl needs to be source-able in vtclsh, must use exec
    if { [catch {set sdp [exec vovsh -x {puts [vtk_server_dir -physical]}}] em] } {
        VovError "determining config directory -- $em"
    } else {
        set cfgdir [file join $sdp "$env(VOV_PROJECT_NAME).swd"]
    }
}

set jcdirs {}
foreach dn {jobclass jobclass_site jobclass_global} {
    set tdir [file join $cfgdir $dn]
    if { [file isdirectory $tdir] } {
        lappend jcdirs $tdir
        VovMessage "added jobclass dir '$tdir'" 3
    } else {
        VovMessage "non-existing jobclass dir '$tdir'" 3
    }
}

if { $jcdirs != {} } {
    setenv VOV_JOBCLASS_DIRS [join $jcdirs ":"]
}
```

```
}
```

The `VOV_JOBCLASS_DIRS` need to be set in all Accelerator-related processes. Sourcing the above script from `vncNNN.swd/setup.tcl` will arrange this for `vovserver` and the `vovtaskers`, which also inherit their environment from a shell that has sourced `setup.tcl`. To have it set in the clients, source it from `vnc_policy.tcl`, which is sourced by the top-level `vnc` command. The following example places the drop-in code in `vncNNN.swd/scripts/`:

```
# This is file jcdirs.tcl
# source the drop-in code for VncJobClassSearchPath{}
# compute the value of VOV_JOBCLASS_DIRS
if { [info exists env(VNCSWD)] } {
    set jcsetup $env(VNCSWD)/$env(VOV_PROJECT_NAME).swd/scripts/jcdirs.tcl
}
if { [file readable $jcsetup] } {
    if { [catch {source $jcsetup} smsg] } {
        VovError "jcsetup error -- $smsg"
    } else {
        VovMessage "jcsetup OK" 3
    }
} else {
    VovError "jcsetup not found -- $jcsetup"
}
```

Define a Default Jobclass

When defined, a default jobclass is evaluated for each job as it is submitted before any other `wx run` options are parsed. The default jobclass should be simple and limited to actions such as supplying basic values for `RAM/` and `CORES/`.

When a default jobclass is in effect, the values it establishes may be changed if a jobclass is later named by the `-C` option of the `wx run` command.

There are two methods to designate a jobclass as the default:

- Via the Jobclass page web page.
- Setting a property from the CLI.

The default jobclass is determined by the value of the property `NC_DEFAULT_JOBCLASS` attached to the trace (`VovId=1`). You can use the utility `vovprop` to set this property. In the following example, `normal` is set as the default jobclass.

```
% wx cmd vovprop SET -text 1 "NC_DEFAULT_JOBCLASS" "normal"
```

View the default jobclass from the CLI by:

```
% wx cmd vovprop GET 1 NC_DEFAULT_JOBCLASS
```

Jobclass Definitions Examples

Example of job class "SHORT"

```
# This is file short.tcl
set classDescription "Jobs taking less than 30s"
set classEditable true; # Allow editing via web UI
set VOV_JOB_DESC(xdur) 30
```

```
set VOV_JOB_DESC(autokill)      1
set VOV_JOB_DESC(priority,sched) 8
set VOV_JOB_DESC(env)          "BASE+D(VOV_LIMIT_cputime=30)"

proc initJobClass {} {}
```

Example of jobclass "INTERACTIVE"

```
# This is file interactive.tcl
set classDescription      "Interactive Jobs"
set classEditable         false; # Disallow editing via web UI

set VOV_JOB_DESC(resources)  ""

# Make the environment unique for each interactive job,
# so that multiple submissions of the same command in the same
# directory will result in multiple jobs
set VOV_JOB_DESC(env) "BASE+D(uniqify=[clock seconds])"

set VOV_JOB_DESC(priority,sched) 9
set VOV_JOB_DESC(interactive,useXdisplay) 1

# We want Ctrl-C and similar commands to be handled by the remote host.
set VOV_JOB_DESC(interactive,flag) "tty_remote"

# We do not want any wrapper for interactive jobs,
# to allow stdout and stdin to go directly to the TTY.
set VOV_JOB_DESC(wrapper)  ""
```

Reconcile Unused Resources

The use of license revocation is not recommended with Accelerator Plus.

Define Jobclasses

This VOV_JOB_DESC data structure is an associative array that describes the characteristics of a job. The array has a number of slots that hold the values describing the job. The following table shows the array slot fields.

Field in Array	Description
autokill	Set the autokill flag (option <code>-kill</code>)
check,directory	Set it to 0 to disable checking of canonicalization of current directory (option <code>-D</code>)
env	Environment of the job (option <code>-e</code>). Set this to "" or to DEFAULT to force the use of an environment snapshot.
force	Force the job to be rescheduled (option <code>-F</code>)

Field in Array	Description
group	Group the job belongs to (option <code>-g</code>)
group,final	Group the job belongs to including the user subgroup (option <code>-G</code>)
inputs	List of input files (dependencies) (option <code>-i</code>)
interactive,flag	Used for interactive jobs, with values <code>tty_remote</code> (option <code>-Ir</code>) or <code>tty_local</code> (option <code>-Il</code>)
interactive, useXdisplay	Set if the job requires an X display (option <code>-Ix</code>)
logfile	Name of the log file (option <code>-l</code>)
mailuser	Specification of who gets the e-mail notification (option <code>-M</code>)
osgroup	The UNIX group this job. This field is read-only and cannot be changed (see also <i>user</i>)
outputs	List of output files (option <code>-o</code>)
preemptable	A suggestion to determine if the job is preemptable
priority, default	Default priority (NOT USED)
priority, exec	Execution priority
priority, sched	Scheduling priority
proplist	Properties to be added to the job (option <code>-P</code>)
resources	The resources of the job (option <code>-r</code>)
rundir	The running directory for the job (normally)
schedule, date	NOT SUPPORTED YET
setName	The set to which the job belongs (option <code>-set</code>)
user	The user for this job. This field is read-only and cannot be changed (see also <i>osgroup</i>)
wait	Boolean: set it to 1 to wait for the job to complete (option <code>-w</code>)
wait,options	When waiting, these options are passed to the <code>nc wait</code> command. For example, set it to <code>-l</code> to view the log file
wrapper	Wrapper used for the job (option <code>-wrapper</code>)
xdur	Expected duration of the job (option <code>-X</code>)


The following is an example of the populated VOV_JOB_DESC array.

```
VOV_JOB_DESC(autoforget)           = 0
VOV_JOB_DESC(autokill)             = 1
VOV_JOB_DESC(check,directory)      = 1
VOV_JOB_DESC(env)                  = BASE+RTSIM
VOV_JOB_DESC(force)                = 0
VOV_JOB_DESC(group)               = users
VOV_JOB_DESC(inputs)               =
VOV_JOB_DESC(interactive,flag)     = none
VOV_JOB_DESC(interactive,useXdisplay) = 0
VOV_JOB_DESC(logfile)              = vnc_logs/20050920/131409.25563
VOV_JOB_DESC(mailuser)             =
VOV_JOB_DESC(osgroup)              = guests
VOV_JOB_DESC(outputs)              =
VOV_JOB_DESC(priority,default)     = 4
VOV_JOB_DESC(priority,exec)        = 4
VOV_JOB_DESC(priority,sched)       = 8
VOV_JOB_DESC(proplist)             =
VOV_JOB_DESC(resources)            = linux
VOV_JOB_DESC(rundir)               = .
VOV_JOB_DESC(schedule,date)        = 0
VOV_JOB_DESC(setName)              =
VOV_JOB_DESC(user)                 = mary
VOV_JOB_DESC(wait)                 = 0
VOV_JOB_DESC(wait,options)         =
VOV_JOB_DESC(wrapper)              = vw
VOV_JOB_DESC(xdur)                 = 30
```

Use Jobclasses

A jobclass allows multiple job parameters to be set in a single object that can be requested at submission time.

For example, there may be a job that requires 3 different licenses, 4GB of RAM, and 4 cores. Instead of requesting all 3 licenses, a jobclass can be created that is called with the `-C` submission option to the `wx run` command. Jobclasses are often used to emulate queues that are found in other batch processing systems.

 **Note:** A jobclass can only be created by an Accelerator Plus administrator.

Find Jobclasses

To list the available classes from the command line, use the `jobclass` subcommand of the `wx` command.

```
wx: Usage Message
WX JOBCLASS:
    List classes defined for job submission
USAGE:
    % wx jobclass [OPTIONS]
OPTIONS:
    -h                -- This help
```

```
-l          -- Long format (with description)
-l1        -- Longer format.
-v         -- Increase verbosity.
EXAMPLES:
% wx jobclass
% wx jobclass -l
% wx jobclass -l1
```

For example:

```
% wx jobclass
1 short
2 interactive
```

The `jobclass` subcommand accepts the repeatable option `-l`. The first option includes the description, and the second option shows the values to which `VOV_JOB_DESC` slots will be set.

In addition, Accelerator Plus provides the Jobclass page. This page shows a table of the jobclass, with links to the definitions of each class, and to the sets containing the jobs in that class. It also shows the pass/fail status as a bar graph.

Submit Jobs Using Jobclasses

To submit a job in a given class, use the option `-C` of `wx run`.

```
% wx run -C short sleep 10
```

Jobs in a class are automatically added to a set named after the class, for example `Class:interactive`.

The options to `wx run` are parsed sequentially, so it is possible to do a command line override of the parameters set in the jobclass. For example, the following commands behave differently:

```
% wx run -C verilog -e DEFAULT -- run_sim chip
% wx run -e DEFAULT -C verilog -- run_sim chip
```

In the first invocation, the option `-e` overrides the specifications for the environment to be used for the job. In the second invocation, the environment is determined by the definition of the `verilog` jobclass.

Resource Management

Altair Accelerator includes a subsystem for managing computing resources. This allows the design team to factor in various constraints regarding hardware and software resources, as well as site policy constraints.


This mechanism is based on the following:

- Resources required by jobs
- Resources offered by taskers
- Resource maps, as described in the file `resources.tcl`

There are several types of resources, which are listed below:

Resource type	Representation	Explanation
Job Resources	<code>name</code>	A resource required by a job. If the quantity is not shown, the default is 1; "unix" is the same as "unix#1".
Uncountable Resources (also called Attributes)	<code>name</code>	These resources represent attributes of a tasker that are not countable. For example, a tasker may have attributes such as "unix" or "linux". The quantity is not shown for these resources and it defaults to MAXINT; "unix" is equivalent to "unix#MAXINT".
Quantitative Resources	<code>name#quantity</code>	Example: The resource <code>RAMTOTAL#2014</code> on a tasker indicates the total amount of RAM on that machine. On a job, it says that the job requires at least the shown amount of <code>RAMTOTAL</code> .
Consumable Resources	<code>name/quantity</code>	Example: <code>RAM/500</code> assigned to a job indicates that the job consumes 500 MB of the consumable resources <code>RAM</code> .
Negated Resources	<code>!name</code>	Example: "unix !linux" on a job indicates that the job requires a UNIX machine but not a Linux one.

The definition of the quantity is related to the context of the resource. If the context is a tasker, quantity represents how much of that resource is available from the tasker. If the context is a job, quality represents how much of that resource is required by the job.

 **Note:** Negated resources are allowed only for the context of a job.

The *unit of measure* is determined by convention for each resource. For example, the resource `RAMTOTAL` is measured in MB. By default, quantity is assumed to be 1; the notation `f oo` is equivalent to `f oo#1`.

A *resources list* is a space-separated list of resources, which are typical resources offered by the taskers. The following example indicates that a job requires at least 128 MB of RAM and a UNIX host, but not a Linux host.

```
RAMTOTAL#128 unix !linux
```

A *resources expression* is a space separated list of resources and operators: typical resources requested by the jobs or mapped in the resource map set. Operators can be one of the following: <blank space>, &, |, OR, AND, !, and NOT. The operators are defined in the table below.

 **Note:** Logical AND has precedence over logical OR operations.

Operator	Description
<blank space>	implicit logical AND
&	explicit logical AND
AND	explicit logical AND
	explicit logical OR
OR	explicit logical OR
!	explicit logical negation
NOT	explicit logical negation

For example, a job may have the following resource requirements:

```
RAMTOTAL#128 unix !linux | RAMTOTAL#512 & linux
```

This job requires either a UNIX host with at least 128 MB of RAM, but not a linux host or a Linux host with at least 512MB of RAM.

Also in this Section

Hardware Resources

All taskers offer a predefined set of hardware resources that can be requested by jobs.

All taskers offer a predefined set of hardware resources that can be requested by jobs. These resources are listed in the following table.

Hardware Resource	Type	Description
ARCH	STRING	The VOV architecture of the machine, for example "linux64", "win64", "armv8"


Hardware Resource	Type	Description
CORES	INTEGER	Consumable resource: the number of logical CPUs/processors used by a job.
CORESUSED	INTEGER	The total number of cores used by the running jobs. It is assumed that each job uses at least one core.
CLOCK	INTEGER	The CPU-clock of at least one of the CPUs on the machine in MHz. If the machine allows frequency stepping, this number can be smaller than expected.
GROUP	STRING	The tasker group for this tasker. Each tasker can belong to only one tasker group.
HOST	STRING	The name of the host on which the tasker is running. Typically this is the value you get with <code>uname -n</code> , except only the first component is taken and converted to lowercase, so that if <code>uname -n</code> returns <code>Lnx0123.my.company.com</code> the value of this field will be <code>lnx0123</code> .
LOADEFF	REAL	The effective load on the machine, including the self-induced load caused by jobs that just started or finished.
L1	REAL	On UNIX, the load average in the last one minute.
L5	REAL	On UNIX, the load average in the last five minutes.
L15	REAL	On UNIX, the load average in the last fifteen minutes.
MACHINE	STRING	Typically the output of <code>uname -m</code> .
MAXNUMACORES	INTEGER	Highest total number of NUMA cores in a single NUMA node.
MAXNUMACORESFREE	INTEGER	Highest number of free cores in a single NUMA node. Note that free NUMA cores are correctly accounted for only if the user specified -jpp pack or -jpp spread for all jobs on the tasker.

Hardware Resource	Type	Description
NAME	STRING	The name of the tasker.
OS	STRING	The name of the operating system: "Linux" or "Windows".
OSCLASS	STRING	This can be unix or windows.
OSVERSION	STRING	The version of the OS. On Linux, this can usually be found in <code>/etc/system-release</code> .
OSRELEASE	STRING	Typically the output of <code>uname -r</code> .
PERCENT	INTEGER	Consumable resource: The percentage of the machine that is still available.
POWER	INTEGER	The effective power of the tasker, after accounting for both raw power and the effective load.
RAM	INTEGER	A consumable resource expressing the remaining RAM available to run job: <code>RAMTOTAL-RAMUSED</code> , in MB.
RAMFREE	INTEGER	The amount of RAM available to run other jobs. This metric comes from the OS, and on linux it includes both free memory and buffers. In MB.
RAMTOTAL	INTEGER	The total amount of RAM available on the machine, in MB.
RAMUSED	INTEGER	The aggregate quantity of RAM used by all jobs currently running on the tasker, in MB. For each job, the amount of RAM is calculated as the maximum of the requested RAM resource (<code>REQRAM</code>) and the actual RAM usage of the job (<code>CURRAM</code>).
RELEASE	STRING	On Linux machines, this is the output of <code>lsb_release -isr</code> , with spaces replaced by dashes. For example, <code>CentOS-6.2</code>
SLOT	INTEGER	A consumable resource indicating how many more jobs can be run on the tasker.

Hardware Resource	Type	Description
SLOTS	INTEGER	Same as SLOT
SLOTSUSED	INTEGER	Corresponding to the number of jobs running on the tasker.
STATUS	ENUMERATED TYPE	Possible values are BLACKHOLE, BUSY, DEAD, DONE, FULL, OVRLD, NOLIC, NOSLOT OK, PAUSED, READY, REQUESTED, SICK, SUSP, WARN, WRKNG
SWAP	INTEGER	A consumable resource. The swap space in MB.
SWAPFREE	INTEGER	The amount of free swap.
SWAPTOTAL	INTEGER	Total amount of swap configured on the machine.
TASKERNAME	STRING	Same as <i>NAME</i>
TASKERHOST	STRING	Same as <i>HOST</i>
TIMELEFT	INTEGER	The number of seconds before the tasker is expected to exit or to suspend. This value is always checked against the expected duration of a job.
TMP	INTEGER	On UNIX, free disk space in /tmp, in MB.
USER	STRING	The user who started the vovtasker server, which is usually the same user account associated with the vovserver process.
VOVVERSION	STRING	The version of the vovtasker binary (such as '2015.03').

Request Hardware Resources

Each job can request hardware resources.

 **Note:** The consumable resources are CORES, CPUS, PERCENT, RAM, SLOT, SLOTS, and SWAP.

- To request a machine with the name `bison`, request `NAME=bison`. To request any `linux64` machine, request `ARCH=linux64`.

- Consumable resources are added together. For example `-r CORES/2 CORES/4 CORES/6` is a request for a total of 12 cores.
- If redundant resources are specified, the largest value will be taken. For example, if `-r RAMTOTAL#2000 RAMTOTAL#4000` is specified then `RAM#TOTAL4000` will be the resource that is used.

Request examples are listed in the following table:

Request Objective	Syntax for the Request
A specific tasker	<code>NAME=bison</code>
Not on bison	<code>NAME!=bison</code>
One of two taskers	<code>NAME=bison,cheetah</code>
A preference: bison, if it is available; otherwise, cheetah	<code>(NAME=bison OR NAME=cheetah)</code>
A specific architecture, such as Linux	<code>ARCH=linux</code>
A specific tasker group, such as prodLnx	<code>GROUP=prodLnx</code>
2 GB of RAM	<code>RAM/2000</code>
Two cores	<code>CORES/2</code>
Two slots	<code>SLOTS/2</code>
Exclusive access to a machine	<code>PERCENT/100</code>
1 minute load less than 3.0	<code>L1<3.0</code>

Wildcard Tasker Resources

A tasker can also offer resources that contain a wildcard. The wildcard is '*' and can be used instead of the name or the type. Legal values for wildcard resources are:

Wildcard	Description
*	Matches all resources that have no type
.	Matches all resources
*:hsim	Matches all resources with name "hsim"
License:*	Matches all resources of type "License"

Wildcard	Description
JobType:*	Matches all resources of type "JobType"

These resources are particularly useful for indirect taskers, which are used to transfer jobs from FlowTracer to Accelerator.

Resource Mapping

As the vovservers determines which tasker is most suited to execute a particular job, it performs a *mapping* of the job resources, followed by a *matching* of the mapped resources.

When dispatching a job, the vovservers does the following:

- Gets the list of resources required by a job.
- Appends the resource associated with the priority level, such as `Priority:normal`.
- If it exists, it appends the resource associated with the name of the tool used in the job (reminder: the tool of a command is the tail of the first command argument after the wrappers). The tool resource has type `Tool` and looks like this:
`Tool:toolname`.
- Appends the resource associated with the owner of the job, such as `User:john`.
- Appends the resource associated with the group of the job, such as `Group:time_regression`.
- Expands any special resource, i.e. any resource that starts with a "\$".
- For each resource in the list, the vovservers looks for it in the resource maps. If the resource map is found and there is enough of it, that is, the resource is available, the vovservers maps the resource. This step is repeated until one of the following conditions is met:
 - # The resource is not available. In this case, the job cannot be dispatched and is left in the job queue.
 - # A cycle in the mapping is detected; in this case the job cannot be dispatched at all and is removed from the job queue.
 - # The resource is not in the resource map.
- VOV appends the resource associated with the expected job duration to the final resource list. For example, if the job is expected to take 32 seconds, the resource `TIMELEFT#32` will be appended.
- Finally, the vovservers compares the resulting resource list with the resource list of each tasker. If there is a match - all resources in the list are offered by the tasker - the tasker is labeled as eligible. If there is no eligible tasker, the job cannot be dispatched at this time and remains in the queue; otherwise, the server selects the eligible tasker with the greatest effective power.

Local Resource Maps


Resource maps can be designated as *local*, using the `local` flag.

 **Important:** This flag is only available and supported for a FlowTracer installation, utilizing `vovwxd` and an LSF interface.

Resource maps designated as local will be managed on the "local" (FT) side of the `vovwxd` connection instead of the normal case where resource specifications are expected to be managed on the base queue side.

For example, to limit jobs to running 5 at a time from a specific FlowTracer project, do the following:


1. Enable local resources with run: `vovservermgr configure vovwxd.localresources 1`

 **Note:** Alternatively, you can add the following to the `policy.tcl` file:

```
set config(vovwxd.localresources) 1
```

2. Create the local resource.

- a. Run `vovresourcemgr set mylocallimit -max 5 -local`

 **Note:** Alternatively, you can add the following to the `resource.tcl` file:

```
vtk_resourcemap_set mylocallimit -total 5 -local
```

This results in limiting running jobs with the local resource `mylocallimit` to a maximum of 5 jobs at a time.

For example, FDL to use a local resource named "mylocallimit":

```
R mylocallimit  
J vov /bin/sleep 0
```

Limitations on the number of Resource Expressions

While the number of OR expressions allowed in a job resource request is limited and controlled by a policy setting, the number of AND expressions including plain expressions without an explicit AND, also has limits.

The scheduler evaluates the resource expressions counting them as it goes. For example a list of 4 separate resource requests (without maps) will result in a max count of 7; 4 from the explicit resource requests and 3 from the automatically added resources (Group, Priority and User). Traversal into a resource map increments the count and a return from a resource map restores the count to value prior to the traversal into the map.

During this traversal, OR operators are recorded and used to influence scheduler operation. The count represents the cumulative depth of the resources. Whenever the count exceeds 30, any traversal that increases the depth is curtailed. This is done silently.

Any OR operators that occur at a depth deeper than 30 are ignored and those scheduling solution are effectively ignored by the scheduler resulting in unexpected behavior.

Large numbers of resource expressions do impact scheduler performance. The general guidance is to keep the explicit resource expressions to fewer than 8 including any mapped ones.

For applications that need a much larger number and where the depth may exceed the 30 limit, it is recommended to place the OR operators early in the resource requests (for example, the left hand side) and to place large numbers of ANDed resource into a resource map.

Resources Representing the Sum of Others

The procedure `vtk_resourcemap_sum` is used to define a resource map as the sum of other resource maps. It takes two arguments:

- The name of the resource map
- The list of the resource components

For example, suppose you have a resource map called `License:a` and another called `License:b`. You can create a sum resource called `License:sum` using:

```
# This code fragment typically goes into resources.tcl
vtk_resourcemap_sum License:sum [list License:a License:b]
```

This results in a resource `License:sum` defined as follows:

```
# This is the result of using vtk_resourcemap_sum...
vtk_resourcemap_set License:sum -max [expr $qa+$qb] -map "License:a OR License:b"

## ... or this map if you activate "commas" (see Commas vs. ORs in Resources).
vtk_resourcemap_set License:sum -max [expr $qa+$qb] -map "License:a,License:b"
```

Where `qa` and `qb` are the current max values for `License:a` and `License:b` respectively. The sums are recomputed by `vovresourced` about once a minute, or by `Allocator` every 30 seconds.

Commas vs. ORs in Resources

Motivation for Commas

For various reasons, the current Accelerator scheduler suffers from an growth of complexity depending on the number of OR's in a resources expression. For example, this expression has 2 ORs:

```
"( License:A OR License:B OR License:C ) RAM/200 "
```

Practical considerations limit the allowed number of ORs to about 20. However, in many cases, you do not need to use the expensive OR when instead we are trying to request any resource in a list of resources. For this purpose, there is the "comma operator", and the equivalent expression above can be rewritten as:

```
"License:A,License:B,License:C RAM/200 "
```

The scheduler will pick any one of `License:A`, `License:B` or `License:C`, in that order. Such expression is much cheaper to compute.

As of version 2016.09u15, full support of this comma operator is offered.

Activation of Commas in vovresourced

To activate the use of commas in resource expressions in `vovresourced`, add the following setting in the config file (either `resources.tcl` or `vovresourced/config.tcl`)

```
# Add this to vovresourced/config.tcl
set RESD(useCommas) 1
```

Activation of Commas in Allocator

For the time being, the use of commas to represent groups of resources is activated by setting the environment variable `VOV_USE_COMMAS_IN_MAPS` to 1 when calling `vtklanc`. This is most easily accomplished by setting the variable in the `setup.tcl` file and restarting the taskers.

```
# Set this in la.swd/setup.tcl
setenv VOV_USE_COMMAS_IN_MAPS 1
```

It is expected that finer control for this will be provided in future versions of Allocator.

Frequently Asked Questions and Troubleshooting Tips

This section provides recommendations to obtain the maximum performance from Accelerator Plus.

Use the Latest Release

The performance of the Accelerator Plus scheduler is frequently updated. Using the most current version is recommended.

Use the vwn Wrapper

The wrapper vwn (alias for vw -N) avoids communication with vovserver. An example is shown below:

```
% wx run -wrapper vwn sleep 0
```

The benefit is of using vwn is speed.

The disadvantage is that jobs that require the -wl option cannot be run. However, this disadvantage may be not be significant, as -wl adds a relatively high load for what it does: -wl requires an extra *notify client* to handle the event generated when the job terminates.

Disable Wait Reasons

If analyzing what causes *wait time* in the workload, the wait reason analysis can be disabled as shown below:

```
# In policy.tcl  
set config(enableWaitReasons) 0
```

Wait time analysis can then be re-enable as needed as shown below:

```
% wx cmd vovsh -x 'vtk_server_config enableWaitReasons 1'  
### collect some data for a few minutes, then  
% wx cmd vovsh -x 'vtk_server_config enableWaitReasons 0'
```


Disable File Access

Disabling file access is mostly a high-reliability option. By disabling file access, the vovserver never looks at any of the files in the user workspaces, which avoids the risk of disk slowness or disk unavailability. An example is shown below:

```
% wx cmd vovsh -x 'vtk_server_config disablefileaccess 2'
```

Reduce Update Rate of Notify Clients

Notify clients, clients that are tapping the event stream from vovserver (such as wx gui, voveventmon or wx run -wl), are updated immediately in the inner loop of the scheduler. If the environment includes hundreds of such clients, it may be beneficial to slow down the update rate by setting the parameter *notifySkip*. The default value is 0: no skip. Typically, the more events that take place, the more events that can be skipped without notice. For example, if several events are taking place, setting *notifySkip* to 100, fewer updates may not be noticed. If the number of events is small, a one-second delay may be noticed in some updates of the GUI. skipped without notice.

 **Note:** Regardless of the setting, the maximum time between updates is one second.

```
# In policy.tcl
set config(notifySkip) 100
```

Accelerator Plus Modulation

When using Accelerator Plus, jobs launched in Accelerator Plus are essentially bundled into groups that are run by vovtaskers on hosts allocated by the base scheduler. This means that it is harder to depend on job retirement to free up slots in the base scheduler, because the bundle of jobs is of course many times longer than the individual jobs.

This section describes a means of freeing up slots more quickly by preempting the vovtaskers that have been assigned to Accelerator Plus, based on FairShare statistics. A preempted vovtasker will stop accepting jobs (tasker status "DONE") and will still finish any running job.

The preemption rule drives the system and this is the main place to influence the systems behavior. A sample rule is found in `$THISGIT/vovpreempt/config.tcl` and this should be appended to any existing preemption rules in `XXXX.swd/vovpreemptd/config.tcl`.

While the rule can be tuned there are some key elements that must be retained. Preemptable jobs should be have the predicate `JOBNAME~${WXQueueName}` and the method should send SIGUSR2 but only to the vovtasker process:
`0:*:EXT,SIGUSR2,vovtasker`.

The preemptable job sort predicate is "FS_EXCESS_RUNNING DESC, PRIORITY, AGE DESC" which chooses vovtaskers ordered on greatest excess FairShare, lowest priority and oldest age.

Here is an example of a preemption rule for job modulation in Accelerator Plus:

```
# Taken from $VOVDIR/etc/config/vovpreemptd/config_wx_modulation.tcl
set WXQueueName wx
VovPreemptRule \
  -pool      "WXJobModulation" \
  -rulename  "fastFairshare_${WXQueueName}" \
  -ruletype  "FAST_FAIRSHARE" \
  -method    "0:*:EXT,SIGUSR2,vovtasker" \
  -killage   0 \
  -numjobs   10 \
  -maxattempts 1 \
  -waitingfor "HW" \
  -preempting "JOBNAME~${WXQueueName} FS_EXCESS_RUNNING<0" \
  -preemptable "JOBNAME~${WXQueueName} FS_EXCESS_RUNNING>0 FSRANK9>>@FSRANK9@" \
  -resumeres "" \
  -enable    1 \
  -sortjobs  "FS_EXCESS_RUNNING DESC,PRIORITY,AGE DESC"
```

Monitoring

This is a dynamic system with quite a few moving parts and this makes monitoring a bit challenging. Some suggestions follow.

Turn on the debug option in the preemption rule - the preemption activity will be logged in a property attached to the preemption rule object. Use the global preemption debug flag to get the info also in the main vovserver log.

```
% vovsh -x 'vtk_server_config set_debug_flag PreemptRules'
```

```
% vovsh -x 'vtk_server_config reset_debug_flag PreemptRules'
```

Altair Accelerator Plus Agent Preemption

Preempting Accelerator Plus Agents from an Accelerator queue using Accelerator Preemption

vovtasker understands Signal USR1 as killing all the jobs running on it and exiting as soon as possible. This signal can be used to preempt Accelerator Plus agents running on an Accelerator queue. When an Accelerator Plus agent receives a USR1, it kills all jobs, and sets the states of jobs as WITHDRAWN. These jobs get rescheduled in Accelerator Plus and will be dispatched on a new tasker. The Accelerator Plus agent on Accelerator exits and becomes a valid job. The following is an example to preempt an Accelerator Plus agent job on Accelerator.

```
nc preempt -method "0:*:SIGUSR1" 159500
```

Legal Notices

Intellectual Property Rights Notice

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

Copyright ©1986-2024 Altair Engineering Inc. All Rights Reserved.

This Intellectual Property Rights Notice is exemplary, and therefore not exhaustive, of the intellectual property rights held by Altair Engineering Inc. or its affiliates. Software, other products, and materials of Altair Engineering Inc. or its affiliates are protected under laws of the United States and laws of other jurisdictions.

In addition to intellectual property rights indicated herein, such software, other products, and materials of Altair Engineering Inc. or its affiliates may be further protected by patents, additional copyrights, additional trademarks, trade secrets, and additional other intellectual property rights. For avoidance of doubt, copyright notice does not imply publication. Copyrights in the below are held by Altair Engineering Inc. or its affiliates. Additionally, all non-Altair marks are the property of their respective owners. If you have any questions regarding trademarks or registrations, please contact marketing and legal.

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

Altair HyperWorks®, a Design & Simulation Platform

Altair® AcuSolve® ©1997-2024

Altair® Activate® ©1989-2024

Altair® Automated Reporting Director™ ©2008-2022

Altair® Battery Damage Identifier™ ©2019-2024

Altair® Battery Designer™ ©2019-2024

Altair® CFD™ ©1990-2024

Altair Compose® ©2007-2024

Altair® ConnectMe™ ©2014-2024

Altair® DesignAI™ ©2022-2024

Altair® EDEM™ ©2005-2024

Altair® EEvision™ ©2018-2024

Altair® ElectroFlo™ ©1992-2024

Altair Embed® ©1989-2024

Altair Embed® SE ©1989-2024

Altair Embed®/Digital Power Designer ©2012-2024

Altair Embed®/eDrives ©2012-2024

Altair Embed® Viewer ©1996-2024

Altair® e-Motor Director™ ©2019-2024

Altair® ESAComp® ©1992-2024
Altair® expertAI™ ©2020-2024
Altair® Feko® ©1999-2024
Altair® Flow Simulator™ ©2016-2024
Altair® Flux® ©1983-2024
Altair® FluxMotor® ©2017-2024
Altair® GateVision PRO™ ©2002-2024
Altair® Geomechanics Director™ ©2011-2022
Altair® HyperCrash® ©2001-2023
Altair® HyperGraph® ©1995-2024
Altair® HyperLife® ©1990-2024
Altair® HyperMesh® ©1990-2024
Altair® HyperMesh® CFD ©1990-2024
Altair® HyperMesh® NVH ©1990-2024
Altair® HyperSpice™ ©2017-2024
Altair® HyperStudy® ©1999-2024
Altair® HyperView® ©1999-2024
Altair® HyperView Player® ©2022-2024
Altair® HyperWorks® ©1990-2024
Altair® HyperWorks® Design Explorer ©1990-2024
Altair® HyperXtrude® ©1999-2024
Altair® Impact Simulation Director™ ©2010-2022
Altair® Inspire™ ©2009-2024
Altair® Inspire™ Cast ©2011-2024
Altair® Inspire™ Extrude Metal ©1996-2024
Altair® Inspire™ Extrude Polymer ©1996-2024
Altair® Inspire™ Form ©1998-2024
Altair® Inspire™ Mold ©2009-2024
Altair® Inspire™ PolyFoam ©2009-2024
Altair® Inspire™ Print3D ©2021-2024
Altair® Inspire™ Render©1993-2024
Altair® Inspire™ Studio ©1993-2024
Altair® Material Data Center™ ©2019-2024

Altair® Material Modeler™©2019-2024
Altair® Model Mesher Director™ ©2010-2024
Altair® MotionSolve® ©2002-2024
Altair® MotionView® ©1993-2024
Altair® Multi-Disciplinary Optimization Director™ ©2012-2024
Altair® Multiscale Designer® ©2011-2024
Altair® newFASANT™©2010-2020
Altair® nanoFluidX® ©2013-2024
Altair® NVH Director™ ©2010-2024
Altair® NVH Full Vehicle™ ©2022-2024
Altair® NVH Standard™ ©2022-2024
Altair® OmniV™ ©2015-2024
Altair® OptiStruct® ©1996-2024
Altair® physicsAI™ ©2021-2024
Altair® PollEx™ ©2003-2024
Altair® PSIM™ ©1994-2024
Altair® Pulse™ ©2020-2024
Altair® Radioss® ©1986-2024
Altair® romAI™ ©2022-2024
Altair® RTLvision PRO™ ©2002-2024
Altair® S-CALC™ ©1995-2024
Altair® S-CONCRETE™ ©1995-2024
Altair® S-FRAME® ©1995-2024
Altair® S-FOUNDATION™ ©1995-2024
Altair® S-LINE™ ©1995-2024
Altair® S-PAD™ © 1995-2024
Altair® S-STEEL™ ©1995-2024
Altair® S-TIMBER™ ©1995-2024
Altair® S-VIEW™ ©1995-2024
Altair® SEAM® ©1985-2024
Altair® shapeAI™ ©2021-2024
Altair® signalAI™ ©2020-2024
Altair® Silicon Debug Tools™ ©2018-2024

Altair® SimLab® ©2004-2024

Altair® SimLab® ST ©2019-2024

Altair® SimSolid® ©2015-2024

Altair® SpiceVision PRO™ ©2002-2024

Altair® Squeak and Rattle Director™ ©2012-2024

Altair® StarVision PRO™ ©2002-2024

Altair® Structural Office™ ©2022-2024

Altair® Sulis™©2018-2024

Altair® Twin Activate®©1989-2024

Altair® ultraFluidX® ©2010-2024

Altair® Virtual Gauge Director™ ©2012-2024

Altair® Virtual Wind Tunnel™ ©2012-2024

Altair® Weight Analytics™ ©2013-2022

Altair® Weld Certification Director™ ©2014-2024

Altair® WinProp™ ©2000-2024

Altair® WRAP™ ©1998-2024

Altair HPCWorks®, a HPC & Cloud Platform

Altair® Allocator™ ©1995-2024

Altair® Access™ ©2008-2024

Altair® Accelerator™ ©1995-2024

Altair® Accelerator™ Plus ©1995-2024

Altair® Breeze™ ©2022-2024

Altair® Cassini™ ©2015-2024

Altair® Control™ ©2008-2024

Altair® Desktop Software Usage Analytics™ (DSUA) ©2022-2024

Altair® FlowTracer™ ©1995-2024

Altair® Grid Engine® ©2001, 2011-2024

Altair® InsightPro™ ©2023-2024

Altair® Hero™ ©1995-2024

Altair® Liquid Scheduling™©2023-2024

Altair® Mistral™ ©2022-2024

Altair® Monitor™ ©1995-2024

Altair® NavOps® ©2022-2024

Altair® PBS Professional® ©1994-2024

Altair® PBS Works™ ©2022-2024

Altair® Software Asset Optimization (SAO)® ©2007-2024

Altair® Unlimited™ ©2022-2024

Altair® Unlimited Data Analytics Appliance™ ©2022-2024

Altair® Unlimited Virtual Appliance™ ©2022-2024

Altair RapidMiner®, a Data Analytics & AI Platform

Altair® AI Hub ©2001-2023

Altair® AI Edge ©2001-2023

Altair® AI Cloud ©2001-2023

Altair® AI Studio ©2001-2023

Altair® Analytics Workbench™ ©2002-2024

Altair® Knowledge Hub™ ©2017-2024

Altair® Knowledge Studio® ©1994-2024

Altair® Knowledge Studio® for Apache Spark ©1994-2024

Altair® Knowledge Seeker™ ©1994-2024

Altair® IoT Studio™ ©2002-2024

Altair® Monarch® ©1996-2024

Altair® Monarch® Classic ©1996-2024

Altair® Monarch® Complete™ ©1996-2024

Altair® Monarch® Data Prep Studio ©2015-2024

Altair® Monarch Server™ ©1996-2024

Altair® Panopticon™ ©2004-2024

Altair® Panopticon™ BI ©2011-2024

Altair® SLC™ ©2002-2024

Altair® SLC Hub™ ©2002-2024

Altair® SmartWorks™ ©2002-2024

Altair® RapidMiner® ©2001-2023

Altair One® ©1994-2024

Altair® License Utility™ ©2010-2024

Altair® TheaRender® ©2010-2024

Altair® OpenMatrix™ ©2007-2024

Altair® OpenPBS® ©1994-2024

Altair® OpenRadioss™ ©1986-2024

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

Telephone and E-mail

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

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

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

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

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

Index

A

Accelerator Plus Administrator Guide [4](#)
Accelerator Plus and Accelerator [6](#)
Accelerator Plus customization [30](#)
Accelerator Plus on Accelerator theory of operation & troubleshooting [15](#)
Accelerator Plus server and system configuration [12](#)
activation of commas in Allocator [57](#)
activation of commas in vovresourced [57](#)
advanced taskers topics [50](#)
Altair Accelerator Plus agent preemption [61](#)
array_VOV_JOB_DESC [44](#)
array, VOV_JOB_DESC [45](#)

C

classDescription [42](#)
classEditable [42](#)
commas vs ORs in resources [57](#)
configure Accelerator Plus to request PBS resources [14](#)
configure list cache expiration [32](#)
configure the tls/ssl protocol [39](#)
consumable resources [49](#)
control access to Accelerator Plus [25](#)
crash recovery [32](#)
create jobclasses [42](#)
customize submission policy [27](#)
customize the wx list command [32](#)
customize the wx run command [31](#)

D

daemons for Accelerator Plus [14](#)
define a default jobclass [44](#)
define jobclasses [45](#)
direct drive [19](#)
directories, jobclass [43](#)
disable file access [59](#)
disable listing by job name [32](#)
disable wait reasons [59](#)

E

enable list cache [32](#)
external webserver [38](#)

F

find and remove "stuck" jobs using a jobclass [45](#)
find jobclasses [47](#)
frequently asked questions and troubleshooting tips [59](#)

G

global setting for reconciliation [45](#)
guest access port [37](#)

H

hardware resources [50](#)
HTTP access models [37](#)

I

integration with PBS [12](#)
internal webserver [37](#)

J

job placement policies [37](#)
job resources [49](#)
job status [26](#)
job submission customizing procedures [27](#)
jobclass definitions examples [44](#)
jobclass directories [43](#)
jobclass_global [43](#)
jobclass_site [43](#)
jobclass, define default [44](#)
jobclass, submit [48](#)
jobclasses [42](#)
jobclasses, define [45](#)
jobs aren't being submitted [32](#)
jobs aren't running [32](#)

L

legacy webserver [37](#)
license violation [32](#)
local resource maps [55](#)

M

many Accelerator Plus queues on many Accelerator queues [41](#)
migrate from a previous version [10](#)
modes of operation [5](#)
modulation [60](#)

monitoring [60](#)
motivation for commas [57](#)
multi-instance vovwxd [10](#)

N

NC_DEFAULT_JOBCLASS [44](#)
negated resources [49](#)

O

open access to many users, example [25](#)

P

policy, customize [27](#)
procedures for job customization [27](#)

Q

quantitative resources [49](#)

R

reconcile unused resources [45](#)
reduce update rate of notify clients [59](#)
request hardware resources [50](#)
resource management [49](#)
resource mapping [55](#)
resources representing the sum of others [57](#)
restrict access to a single user, example [25](#)

S

server doesn't start [32](#)
start Accelerator Plus [7](#)
start and stop Accelerator Plus [12](#)
status of jobs [26](#)
submit jobs using jobclasses [48](#)
system overview [5](#)

T

tasker suspension in base scheduler [32](#)
theory of operation [5](#)
throttle job submission rate [27](#)
troubleshooting [32](#)

U

uncountable resources [49](#)

unix classes do not start [32](#)
use Accelerator as a base scheduler [40](#)
use additional jobclass directories [43](#)
use jobclasses [47](#)
use latest release [59](#)
use the vwn wrapper [59](#)

V

VncJobClassSearchPath [42](#)
VncPolicyDefaultPriority [27](#)
VncPolicyDefaultResources [27](#)
VncPolicyGetJobInfo [27](#)
VncPolicyUserPriority [27](#)
VncPolicyUserPriorityExec [27](#)
VncPolicyValidateCommand [27](#)
VncPolicyValidateEnvironment [27](#)
VncPolicyValidateResources [27](#)
vov security keys [20](#)
VOV_JOB_DESC [27](#), [31](#), [42](#), [45](#)
VOV_JOBCLASS_DIRS [42](#), [43](#)
VOV_USE_COMMAS_IN_MAPS [57](#)
VOVARCH [27](#)
VOVDIR [10](#)
vovelasticd [10](#)
vovlsfd [10](#)
VovPreemptPolicy [27](#)
vovresgrab [27](#)
vovresourced [42](#), [57](#)
vovresreq [27](#)
vovwxconnect [9](#), [10](#), [12](#)
vovwxd [10](#)
vovwxd configuration [17](#)
vovwxd, direct drive [19](#)

W

web interface [40](#)
web server configuration [37](#)
wildcard tasker resources [54](#)
wx list [32](#)
wx run [31](#)
WXLauncher [10](#)