

PBS Plug-ins:

A Run-time Environment for Agility and Innovation

Scott J. Suchyta

Director, Partner Solutions & Integrations

May 2011



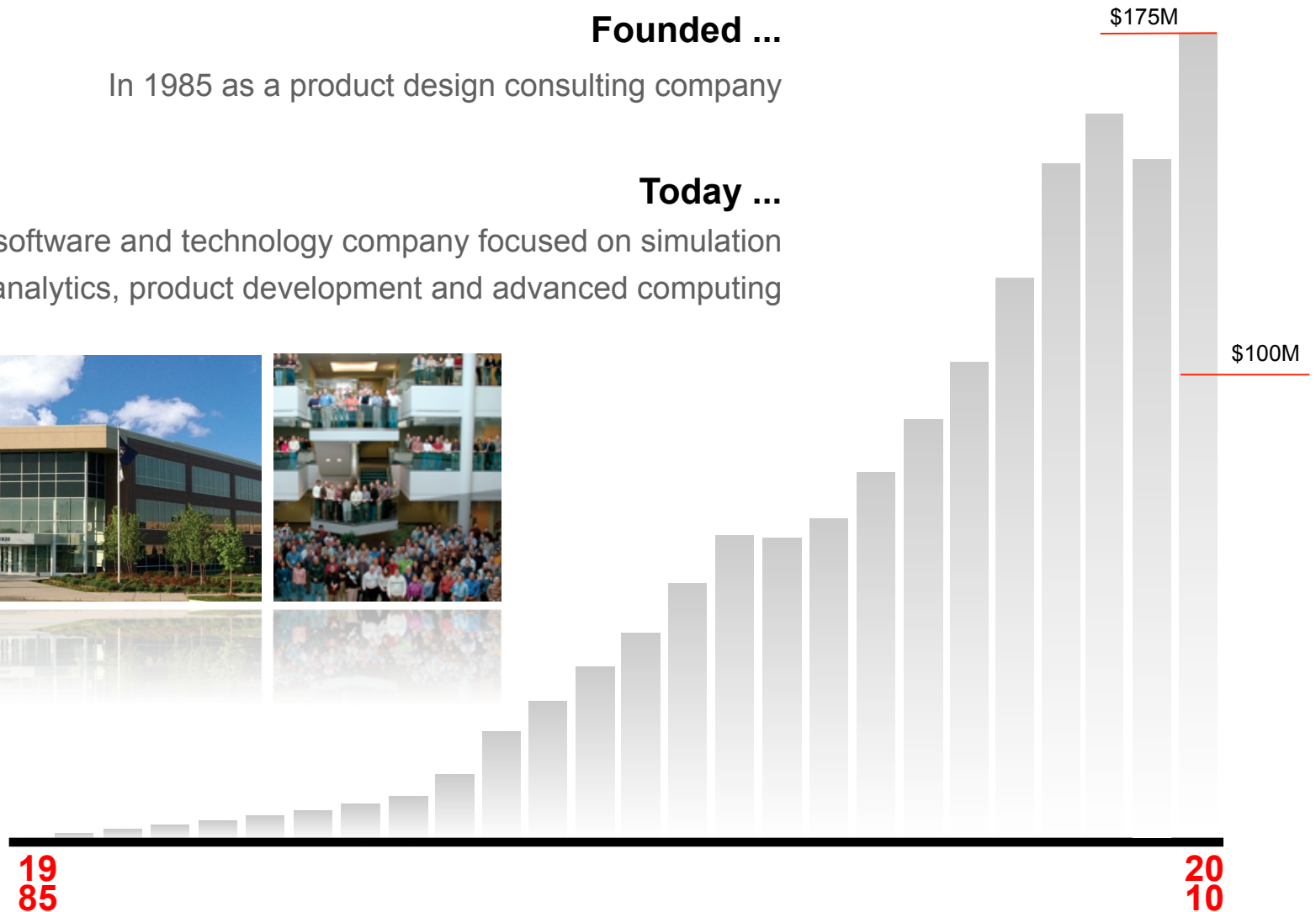
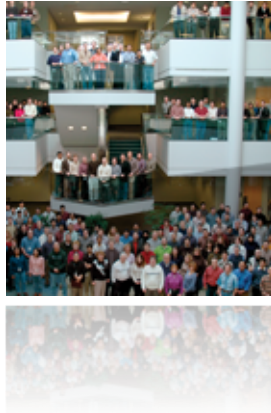
Altair Overview

Founded ...

In 1985 as a product design consulting company

Today ...

A global software and technology company focused on simulation and analytics, product development and advanced computing



Global Presence



A world map with a grid overlay, showing the locations of Altair's global offices. The map is light gray, and the office names are listed in black text over the corresponding geographical areas. The office in Detroit, USA, is highlighted in red text.

North America	Europe	Asia	Africa	Australia
Seattle, USA	Lund, Sweden	Moscow, Russia		
Mountain View, USA	Göteborg, Sweden	Beijing, China		
Los Angeles, USA	Coventry, UK	Shanghai, China		
Austin, USA	Manchester, UK	Delhi, India		
Denver, USA	Stuttgart, Germany	Pune, India		
Mexico City, Mexico	Cologne, Germany	Chennai, India		
	Hamburg, Germany	Hyderabad, India		
	Hanover, Germany	Bangalore, India		
	Munich, Germany			
	Lyon, France			
	Paris, France			
	Sophia Antipolis, France			
	Toulouse, France			
	Torino, Italy			
	Milan, Italy			
Sao Paulo, Brazil				Melbourne, Australia

Over 40 offices across 16 countries

Altair Customers

Automotive	Aerospace	Heavy Equipment	Government	Life/Earth Sciences	Consumer Goods	Energy
                 	              	            	          	               	                  	              


4,000+ customers worldwide

Altair: 25 Years of Innovation



PBS History

PBS turns 20 years old on Jun 17, 2011!

1991-1995	1996-1999	2000-2002	2003-2006	2007-2009	2010-2011
PBS developed for NASA	Early production grids built using PBS 	PBS Pro 5.0: enhanced, hardened, commercial <i>Running on All 7 Continents</i>	Altair acquires PBS Pro <i>Topology-aware scheduling</i>	PBS Analytics PBS Catalyst <i>Green Provisioning</i> <i>GPU Scheduling</i>	PBS Professional 11.0 <i>EAL3+ Security</i> <i>Pleiades Tsubame 2 DoD HPCMP</i> <i>.</i> <i>.</i> <i>.</i>

PBS Works: 5 Strategic Pillars

Easy to Use
Portals & UI

- **More capable, targeted user experience**

Hard to Break
Infrastructure

- **Bullet-proof reliability & unlimited scalability**

Do More (with less)
Scheduling

- **Optimization, optimization, optimization**

Keep Track and Plan
Analytics

- **Integrated whole IT analytics & optimization**

Open Architecture
Extensibility

- **Plugs-in and extends enterprise infrastructure**

PBS Works: 5 Strategic Pillars

Easy to Use
Portals & UI

- More capable, targeted user experience

Hard to Break
Infrastructure

- Bullet-proof reliability & unlimited scalability

Do More (with less)
Scheduling

- Optimization, optimization, optimization

Keep Track and Plan
Analytics

- Integrated whole IT analytics & optimization

Open Architecture
Extensibility

- **Plugs-in and extends enterprise infrastructure**

Plug-in!



home

products

Downloads / Downloads Categ



Add-ins

developerWorks > Information Management > Technical library >

Implement user exit routines

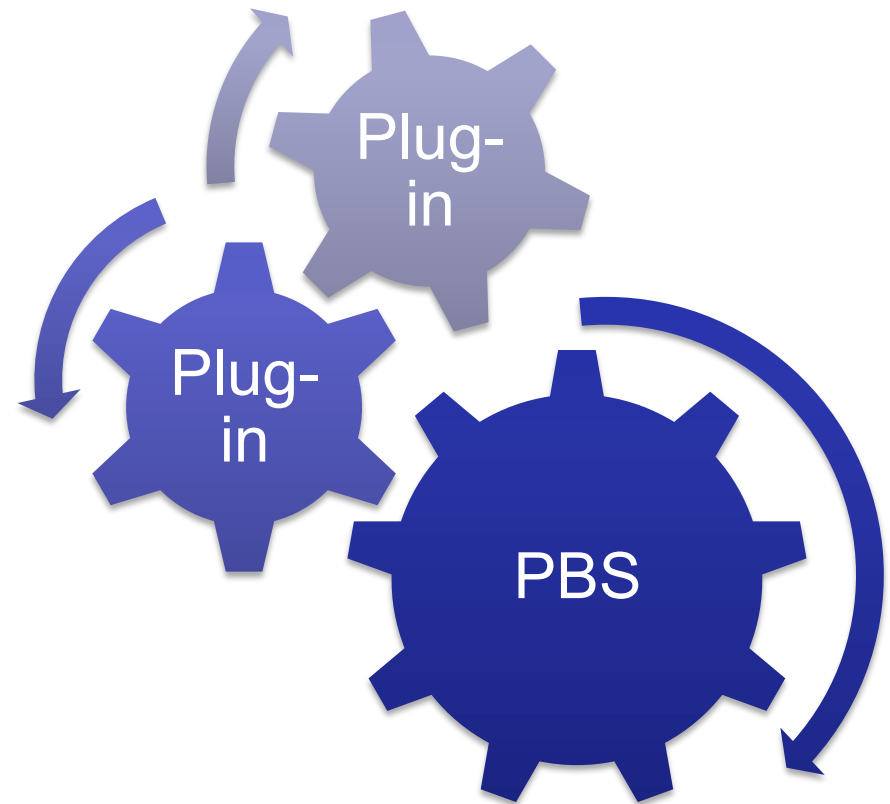
Why?

Agility

- Every enterprise is unique
- HPC == “leading edge”

Innovation

- Great ideas grow from lots of not-so-great ideas



Runtime Extensibility

Enterprise integrations

- Integrate with “everything”
- I.e., all 3rd party tools

Site-specific extensions

- 80/20 rule – focus our core engineering on the 80, but also support the 20 too!

Platform-specific features

- Support on day one

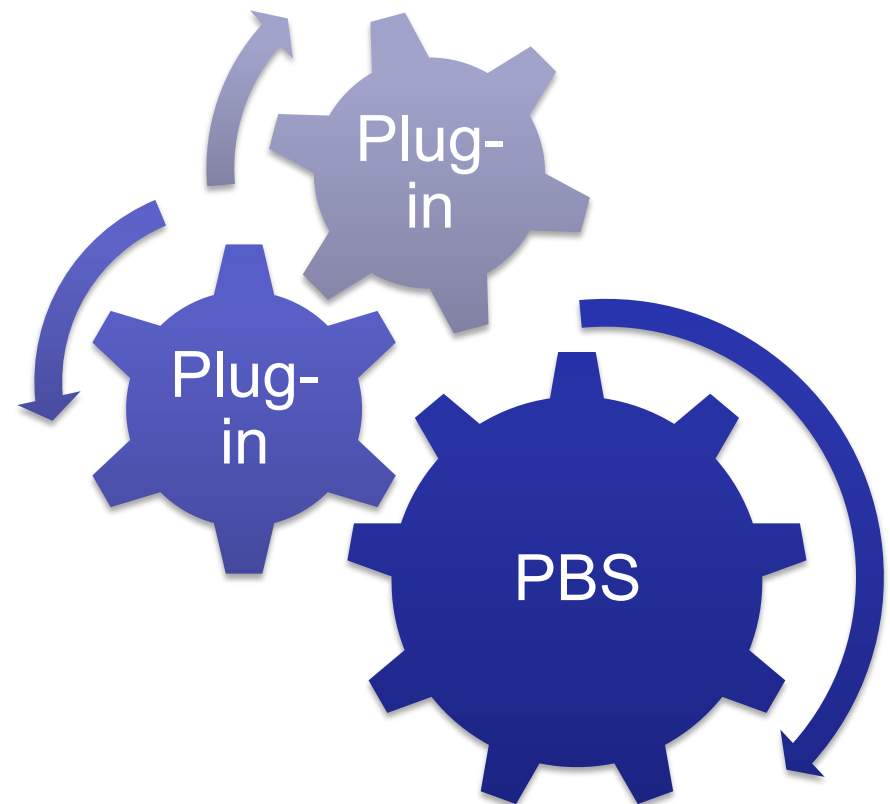
Prototype new capabilities

- Change the behavior of PBS itself

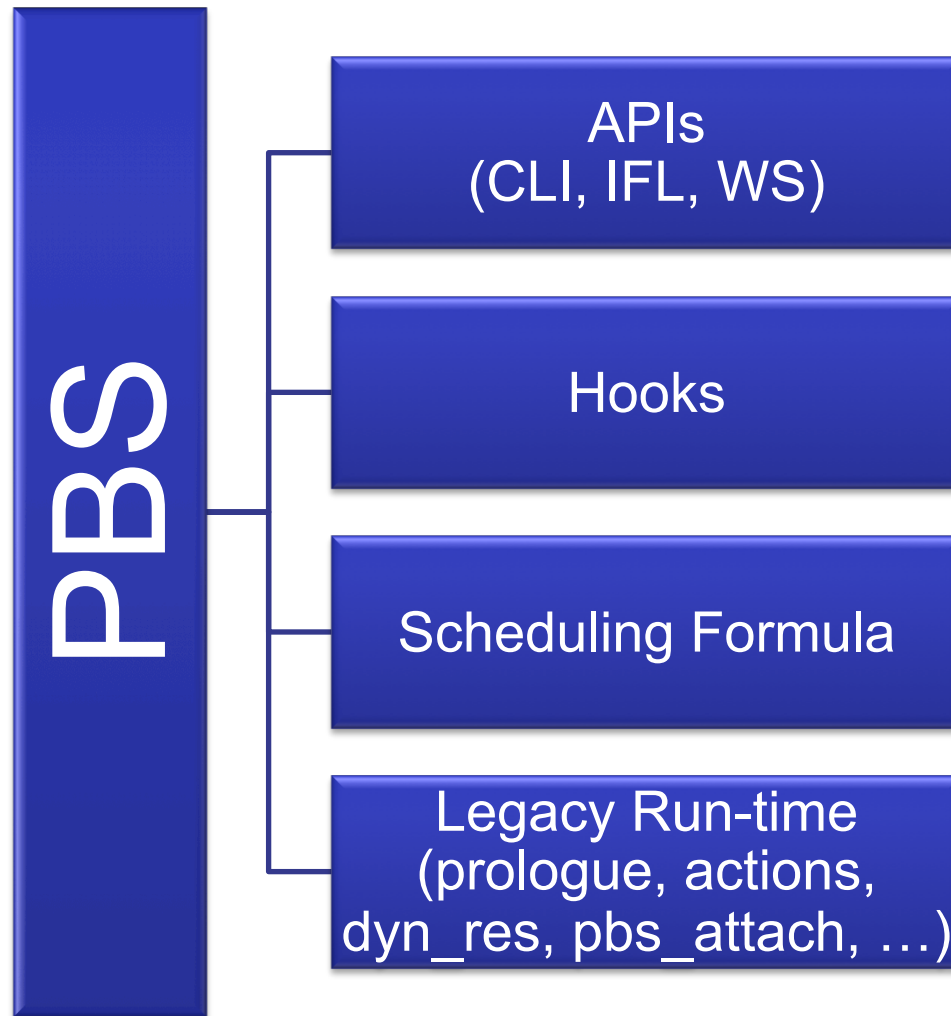
Never say “no”

Foster the ecosystem

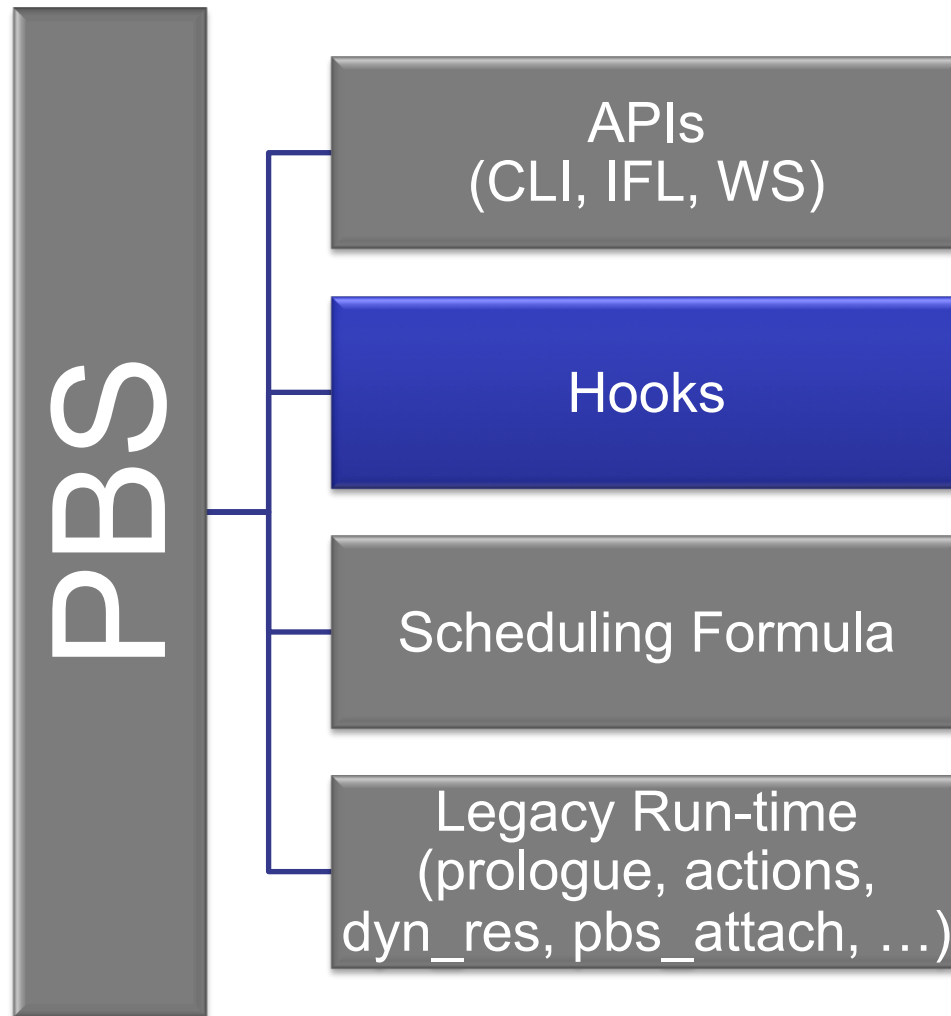
- Single, uniform interface
- “Modern” Python interface
- Shareable!



The PBS Run-time Environment



The PBS Run-time Environment



Python “Everywhere”

Portable Modern Scripting Language Available Everywhere

Can be used for scripts,
prologues, epilogues, actions,
dynamic resources, etc.

Same Python used for Hooks

Allows one script to be used
across all architectures
(Linux & Windows)

Python v2.5

```
import pbs

R = pbs.event().job.Resource_List
sel = repr(R["select"])
tot_ncpus = 0
for chunk in sel.split("+"):
    nchunks = 1
    for c in chunk.split(":"):
        kv = c.split("=")
        if len(kv) == 1:
            nchunks = kv[0]
        elif len(kv) == 2:
            if kv[0] == "ncpus":
                tot_ncpus += (int(nchunks) * int(kv[1]))
    [1])
```

Submission Filtering Hooks

Change / augment capabilities in the field, on-the-fly, without source

Admission control – validate requests

Allocation management

On-the-fly tuning

Custom logging, reporting, debugging, and even patches!

Hooks for:

qsub / qalter / pbs_rsub / qmove

Run Job Hook

Ensure allocation management limits are strictly enforced

Generic run job hook complements submission hook and enables pre-dispatch checks

Jobs can be held, released, and delayed

Enables almost any type of user / group / project /... limits, including limits set by allocation management systems

- E.g., Fred cannot start OptiStruct jobs on Sunday

Dynamic Provisioning Hooks

Automatically Change OS to Match Workload Demands

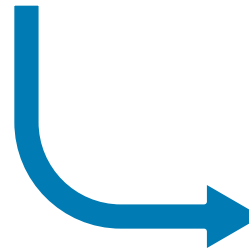
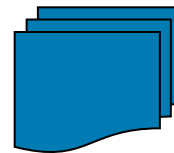
Jobs can individually request operating environments

Support legacy environments with no additional hardware

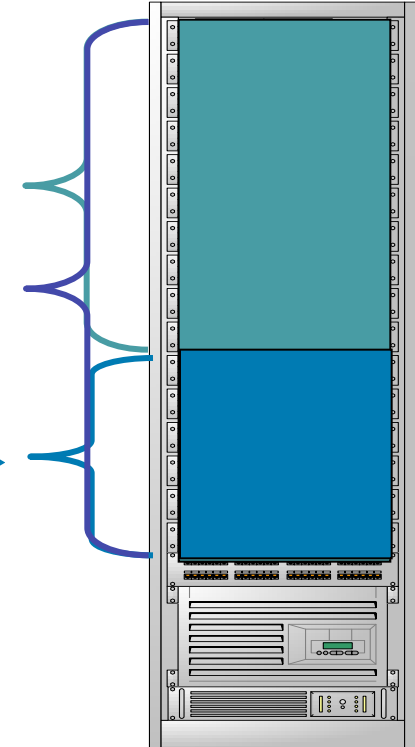
Test new OSES and configurations with minimal disruption

Plug-in architecture supports any cluster manager: SGI Tempo, CMU, ROCKS, Bright, ...

Job Requests
RHEL 4



Operating
Environment
is Dynamically
Provisioned



Example: RequireWalltime.py

```
if j.Resource_List["walltime"] == None :  
    je.reject("Job has no walltime requested")
```


Example: RequireWalltime.py

```
import pbs

je = pbs.event()
j = je.job
if j.Resource_List["walltime"] == None :
    je.reject("Job has no walltime requested")
```

Example: RequireWalltime.py

```
import pbs

try:
    je = pbs.event()
    j = je.job
    if j.Resource_List["walltime"] == None :
        je.reject("Job has no walltime requested")
except SystemExit:
    pass
except pbs.UnsetResourceNameError:
    je.reject("Job has no walltime requested")
```

Example: RequireWalltime.py

```
import pbs

try:
    je = pbs.event()
    j = je.job
    if j.Resource_List["walltime"] == None :
        je.reject("Job has no walltime requested")
except SystemExit:
    pass
except pbs.UnsetResourceNameError:
    je.reject("Job has no walltime requested")
```

And add it to the server via qmgr (as root):

```
# qmgr -c 'create hook RequireWalltime event="queuejob" '
# qmgr -c 'import hook RequireWalltime \
    application/x-python default RequireWalltime.py'
```

Example: RequireWalltime.py

```
import pbs

try:
    je = pbs.event()
    j = je.job
    if j.Resource_List["walltime"] == None :
        je.reject("Job has no walltime requested")
except SystemExit:
    pass
except pbs.UnsetResourceNameError:
    je.reject("Job has no walltime requested")
```

And add it to the server via qmgr (as root):

```
# qmgr -c 'create hook RequireWalltime event="queuejob" '
# qmgr -c 'import hook RequireWalltime \
    application/x-python default RequireWalltime.py'
```

PBS Module

Natural mappings to PBS objects

```
q = s.queue("workq")  
q.total_jobs      ← returns the # of jobs on "workq"  
q.job("22.fest")  ← returns a job in the queue
```

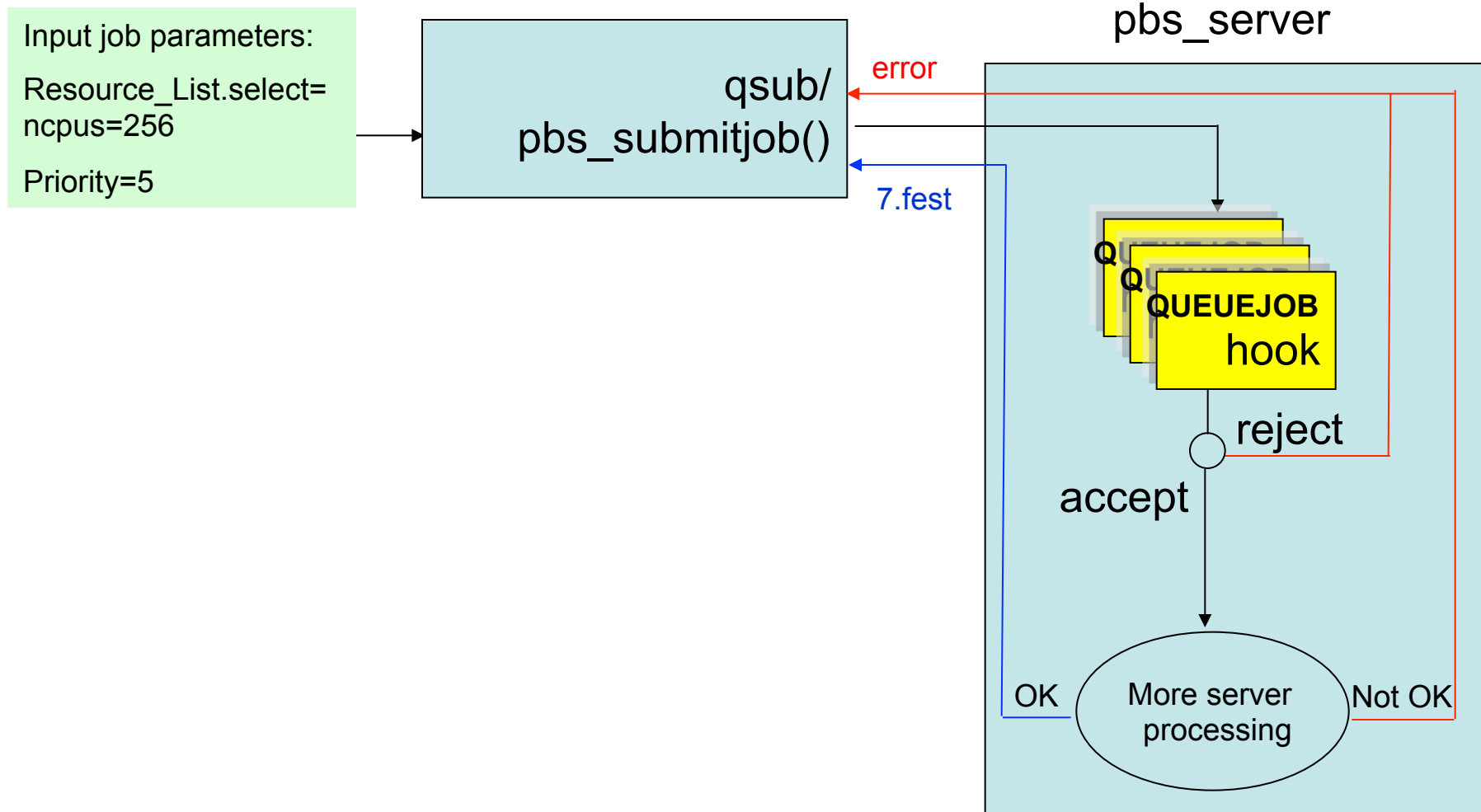
Write log/debug info directly to PBS logs

```
pbs.logmsg(pbs.LOG_DEBUG, "Hooks are great!")
```

Standard Python exception handling

Multiple hooks per event (including ordering execution)

Internally, hooks are run inside PBS



Ex: Put Interactive Jobs into Q 'interq'

```
if j.interactive:  
    q = pbs.server().queue("interq")  
    j.queue = q
```


Ex: Restrict qalter to admins only...

```
who = e.requestor
pbs.logmsg(pbs.LOG_DEBUG, "requestor=%s" % (who,))

admins = ["PBS_Server", "Scheduler", "root"]

if who not in admins:
    e.reject("Normal users are not allowed to qalter jobs")
```

And add it as a 'modifyjob' hook:

```
# qmgr -c 'create hook No_qalter event="modifyjob" '
# ...
```

Ex: Convert “words” to “bytes”

```
bpw = 8 # bytes per word
```

```
sel=repr(j.Resource_List["select"])  
newsel=sel
```

```
# find any memory request that's using words  
mc=re.findall('mem=(\d+)([pPtTgGmM]?[wW])', sel)
```

```
if len(mc) >= 1:  
    for m in mc:  
        r_mem="mem=%s%s" % (m[0],m[1],)  
        s_mem="mem=%s%s" % (str(long(m[0])*bpw), m  
[1].replace("w", "b").replace("W", "B"),)  
        newsel=newsel.replace(r_mem,s_mem)
```

```
j.Resource_List["select"]=pbs.select(newsel)
```

Ex: Per-Q Primetime (~20 lines!)

```
my_queue = j.queue

if (not my_queue.resources_available["night_end"]) and (not
my_queue.resources_available["night_start"]):
    je.accept()

night_end = int(my_queue.resources_available["night_end"])
night_start = int(my_queue.resources_available["night_start"])

today = datetime.datetime.now()
end_epoch = today.replace(hour=night_end, minute=0, second=0, microsecond=0)
start_epoch = today.replace(hour=night_start, minute=0, second=0, microsecond=0)
start_buffer = today.replace(hour=night_start + 1, minute=0, second=0, microsecond=0)
next_start_epoch = start_epoch + datetime.timedelta(1)

now = datetime.datetime.now()

If j.Resource_List["walltime"]:
    job_length = datetime.timedelta(0,j.Resource_List["walltime"])
else:
    job_length = datetime.timedelta(0,24 * 3600)

if (now > start_epoch) and ((now < start_buffer) or (now + job_length < end_epoch)):
    je.accept()
else:
    j.Execution_Time = time.mktime(next_start_epoch.timetuple())
    je.reject("Delayed until %s" % time.ctime(this_job.Execution_Time))
```

Ex: Limit Subjobs Run per Array (~20 lines!)

```
if (j.array == False) or (j.Resource_List['max_subjobs_running'] is None):
    je.accept()

# get the id part of a job array. i.e. the 123 in 123[1]
job_suffix = j.id[0:len(j.id)-2]

regexp = job_suffix + "\\[[\\d]+\\]"
job_array_re = re.compile(regexp)

num_subjobs_running = 0

for job in pbs.server().jobs():
    m = job_array_re.match(job.id)
    if m:
        if int(str(job.job_state)) == pbs.JOB_STATE_RUNNING:
            num_subjobs_running += 1

if num_subjobs_running >= j.Resource_List['max_subjobs_running']:
    pbs.logmsg(pbs.LOG_DEBUG, "Not running %s: subjob limited" % j.id)
    je.reject()
```

Ex: Name-based Job Dependencies

In production!

Only ~100 lines of
Python (including
debug stmts...)

➔ Agile Prototyping

```
import pbs
import sys
import os

e=pbs.event()
my_name = e.hook_name
debug_me = False

# If resources_available.debug_hooks contains the name of this hook, then we
# turn on the debug flag.
if ( "debug_hooks" in pbs.server().resources_available and
    my_name in str(pbs.server().resources_available['debug_hooks']).split(',') ):
    debug_me=True

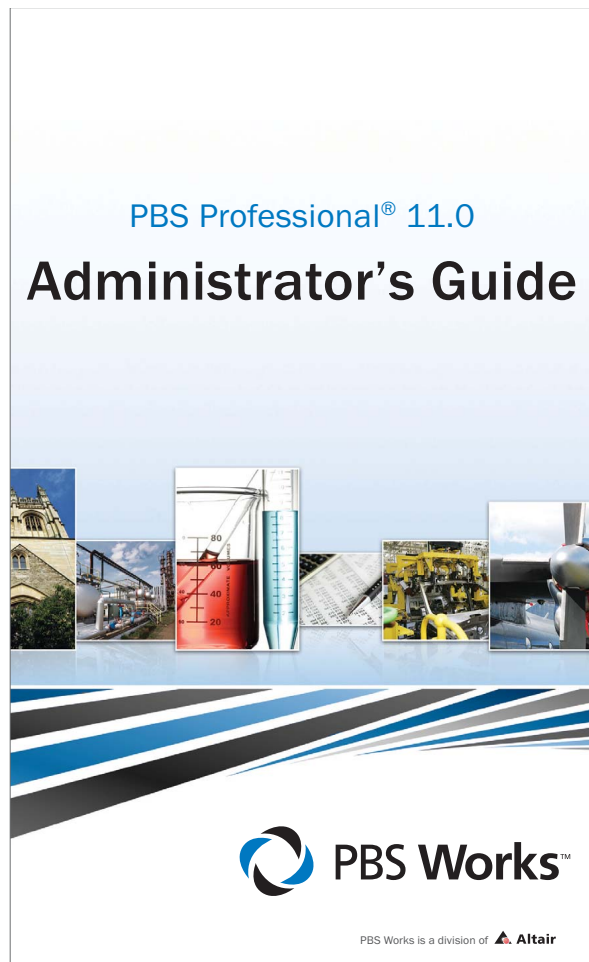
def dbg_svr_log(string):
    '''quick function to wrap debug logging to the server'''
    # Abort if the hook_debug value is not set
    if(debug_me):
        header = "DEBUG"+"".join(["%s" % "*" for s in range(19)])
        footer = "".join(["%s" % "*" for s in range(79)])
        pbs.logmsg(pbs.LOG_ERROR, "%s\n%s\n%s" % ( header, string, footer ))

my_job = e.job

# Exit if we don't have a named based dependency request, no need to go any
# further
if( my_job.Resource_List['ndepend'] == None ):
    dbg_svr_log("Exiting because no one asked for a dep")
    sys.exit()

# Exit if they specified an actual dependency as well as a name based one, they
# can not be combined. This is to avoid people getting their syntax mixed up,
# if requests come in to allow both to be combined I will do that, it's not
# difficult.
if ( my_job.depend != None ):
    dbg_svr_log("Exiting because they tried to use the real depend option")
    e.reject("Can not combine -Wdepend syntax and -l ndepend syntax!")
```

More Information



Chapter 6

Hooks

Hooks are custom executables that can be run at specific points in the execution of PBS. They accept, reject, or modify the upcoming action. This provides job filtering, patches or workarounds, and extends the capabilities of PBS, without the need to modify source code.

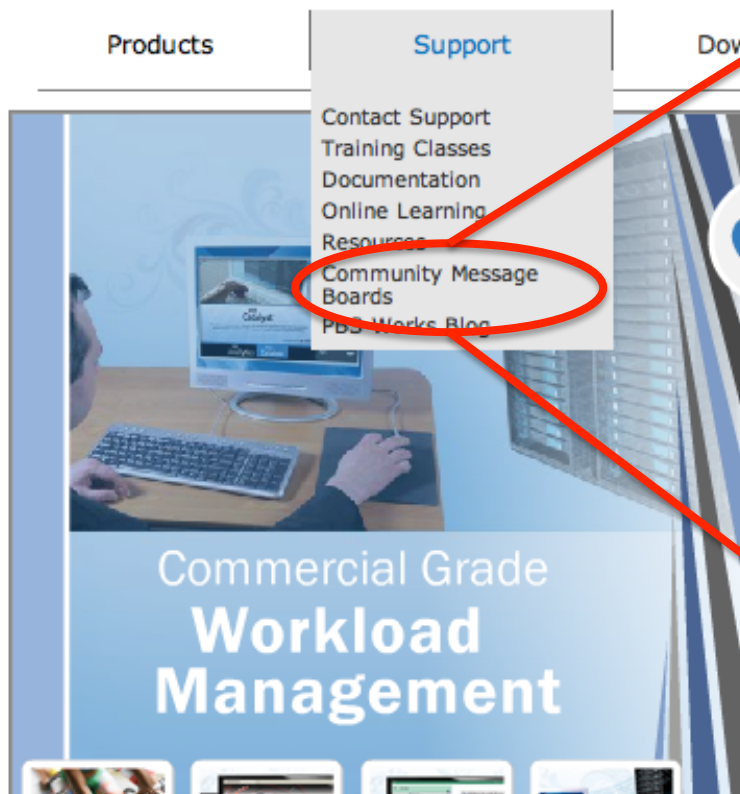
This chapter describes how hooks can be used, how they work, the interface to hooks provided by the `pbs` module, how to create and deploy hooks, and how to get information about hooks.

Please read the entire chapter before writing any hooks.





6.1 Introduction to Hooks

A hook is a block of Python code that is triggered in response to queuing a job, modifying a job, moving a job, running a job, provisioning a vnode, or submitting a PBS reservation. Each hook can *accept* (allow) or *reject* (prevent) the action that triggers it. The hook can modify the input param-

Community



PBS WORKS FORUM » COMMUNITY DISCUSSIONS

	Forum	Topics	P
	<u>COMMUNITY DISCUSSIONS</u>		
	<u>TROUBLESHOOTING</u> (1 Viewing) Have a problem? Post your questions here.	42	
	<u>ADMINS</u> Tips & Tricks for administrators	23	
	<u>USERS</u> Tips & Tricks for users	7	
	<u>BUGS AND FEEDBACK</u>		

Thank You!

Altair is the only company that...

makes HPC Tools



and makes HPC Apps,



and uses HPC Apps!



**500 Altair engineers worldwide
use HPC every day for
real-world modeling
& simulation**