Site Configuration Guide

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Prerequisites **1**

2 Bag of Workstations **1**

2.1 Prerequisites . . . 1
2.2 Create a coaster-service.conf . . . 1
2.3 Modify coaster-service.conf . . . 2
2.4 Starting the Coaster Service . . . 2
2.5 Run Swift . . . 3
2.6 Stopping the Coaster Service . . . 3

3 Beagle **3**

3.1 Introduction . . . 3
3.2 Requesting Access . . . 3
3.3 Connecting to a login node . . . 3
3.4 Larger Runs on Beagle . . . 5
3.5 Resuming Large Runs . . . 6
3.6 Troubleshooting . . . 6
3.7 More Help . . . 7

4 Fusion **8**

4.1 Requesting Access . . . 8
4.2 Projects . . . 8
4.3 SSH Keys . . . 8
4.4 Connecting to a login node . . . 8
4.5 Creating sites.xml . . . 8
4.6 Creating tc.data . . . 9
4.7 Copy a Swift Script . . . 9
4.8 Run Swift . . . 9
4.9 Queues . . . 9
4.10 More Help . . . 10

5 Futuregrid Quickstart Guide **10**

5.1 Requesting Futuregrid Access . . . 10
5.2 Downloading Swift VM Tools . . . 10
5.3 Download your Credentials . . . 10
5.4 Configuring coaster-service.conf . . . 10
5.5 Starting the Coaster Service Script . . . 11
5.6 Running Swift . . . 11
5.7 Stopping the Coaster Service Script . . . 11
5.8 More Help . . . 12

6 Intrepid **12**
6.1 Requesting Access . . . 12
6.2 SSH Keys . . . 12
6.3 Cryptocard . . . 12
6.4 Connecting to a login node . . . 12
6.5 Downloading and building Swift . . . 12
6.6 Adding Swift to your PATH . . . 13
6.7 What You Need To Know Before Running Swift . . . 13
6.7.1 Swift Work Directory . . . 13
6.7.2 Which project(s) are you a member of? . . . 13
6.7.3 Determine your Queue . . . 13
6.8 Generating Configuration Files . . . 14
6.8.1 Manually Editing sites.xml . . . 14
6.9 Manually Editing tc.data . . . 14
6.10 Catsn.swift . . . 15
6.11 Running Swift . . . 15
6.12 More Help . . . 15

7 MCS Workstations **15**
7.1 Create a coaster-service.conf . . . 15
7.2 Starting the Coaster Service . . . 16
7.3 Run Swift . . . 16
7.4 Stopping the Coaster Service . . . 16

8 PADS **17**
8.1 Requesting Access . . . 17
8.2 SSH Keys . . . 17
8.3 Connecting to a login node . . . 17
8.4 Adding Software Packages . . . 17
8.4.1 Which project(s) are you a member of? . . . 17
8.4.2 Creating sites.xml . . . 18
8.4.3 Creating tc.data . . . 18
8.4.4 Copy a Swift Script . . . 18
8.4.5 Run Swift . . . 19
8.4.6 Queues . . . 19
8.5 More Help . . . 19

1 Prerequisites

This guide assumes that you have already downloaded and installed Swift. It assumes that Swift is in your PATH and that you have a working version of Sun Java 1.5+. For more information on downloading and installing Swift, please see the Swift Quickstart Guide.

2 Bag of Workstations

"Bag of workstations" refers to a collection of machines that are not connected together as part of a cluster or supercomputer.

2.1 Prerequisites

In order to run Swift on a bag of workstations, the following prerequisites must be met:

- The machines must be running Linux
- The machines must have Perl available
- A user account must be created on each machine (the username you create must be the same on each machine)
- You must be able to SSH into the accounts without being prompted for a password. This usually involves creating an SSH key and setting up your authorized_keys. More information on how to do this can be found at http://www.openssh.org.

2.2 Create a coaster-service.conf

To begin, copy the text below and paste it into the directory where your swift script is located. Name this file coaster-service.conf.

```
# Keep all interesting settings in one place
# User should modify this to fit environment

# Location of SWIFT. If empty, PATH is searched
export SWIFT=

# Where to copy worker.pl on the remote machine for sites.xml
export WORKER_LOCATION=$HOME/swiftwork

# How to launch workers: local, ssh, cobalt, or futuregrid
export WORKER_MODE=ssh

# SSH hosts to start workers on (ssh mode only)
export WORKER_HOSTS="host1 host2 host3"

# Do all the worker nodes you're using have a shared filesystem? (yes/no)
export SHARED_FILESYSTEM=yes

# Username to use on worker nodes
export WORKER_USERNAME=$USER

# Enable SSH tunneling? (yes/no)
export SSH_TUNNELING=no

# Directory to keep log files, relative to working directory when launching start-coaster- ↩
    service
export LOG_DIR=logs

# Manually define ports. If not specified, an available port will be used
export LOCAL_PORT=
```

```
export SERVICE_PORT=

# This is the IP address to which the workers will connect
# If not given, start-coaster-service tries to automatically detect
#   the IP address of this system via ifconfig
# Specify this if you have multiple network interfaces
export IPADDR=

# Location of the swift-vm-boot scripts
export SWIFTVMBOOT_DIR=$HOME/swift-vm-boot

# Swift information for creating sites.xml
export WORK=$HOME/swiftwork
export QUEUE=prod-devel
export MAXTIME=20
export NODE=64
```

2.3 Modify coaster-service.conf

The coaster-service.conf file contains information about your setup. There are a few settings you must customize.

The first is the name of the machines which will be used as workers. Modify the line below to reflect the names of the machines you want to use.

```
export WORKER_HOSTS="myhost1.mydomain myhost2.mydomain"
```

Update the value for JOBS_PER_NODE to reflect the number of CPUs available per node.

The JOB_THROTTLE setting determines the maximum number of active jobs. Use the formula to determine the ideal value:

jobs per node * number of nodes - 0.1 / 100

Example: Suppose you have 10 machines each with 8 cores

```
8 jobs per node * 10 cores = 80
80 - 0.1 = 79.9
79.9 / 100 = 0.799
```

In this example, you would set the job throttle to 0.799

```
export JOB_THROTTLE=0.799
```

By default, this setup assumes there are no firewall rescrictions. If there is a firewall rescricting SSH access, set tunneling to true with this command

```
export SSH_TUNNELING=yes
```

This setup also assumes that these machines are not using a shared filesystem (NFS/AFS/CIFS, etc). If these systems are all sharing a common filesystem, add the setting below.

```
export SHARED_FILESYSTEM=no
```

2.4 Starting the Coaster Service

Change directories to the location where you would like to run a Swift script and start the coaster service with this command:

```
start-coaster-service
```

This will create a configuration file that Swift needs called sites.xml.

Warning
Any existing sites.xml files in this directory will be overwritten. Be sure to make a copy of any custom configuration files you may have.

2.5 Run Swift

Next, run Swift. If you do not have a particular script in mind, you can test Swift by using a Swift script in the examples/ directory.

Run the following command to run the script:

```
swift -sites.file sites.xml -tc.file tc.data yourscript.swift
```

2.6 Stopping the Coaster Service

The coaster service will run indefinitely. The stop-coaster-service script will terminate the coaster service.

```
$ stop-coaster-service
```

This will kill the coaster service and kill the worker scripts on remote systems.

3 Beagle

3.1 Introduction

Beagle is a Cray XE6 supercomputer at UChicago. It employs a batch-oriented computational model where-in a PBS schedular accepts user's jobs and queues them in the queueing system for execution. The computational model requires a user to prepare the submit files, track job submissions, chackpointing, managing input/output data and handling exceptional conditions manually. Running Swift under Beagle can accomplish the above tasks with least manual user intervention and maximal oppurtunistic computation time on Beagle queues. In the following sections, we discuss more about specifics of running Swift on Beagle. A more detailed information about Swift and its workings can be found on Swift documentation page here: http://www.ci.uchicago.edu/swift/wwwdev/docs/index.php . More information on Beagle can be found on UChicago Beagle website here: http://beagle.ci.uchicago.edu .

3.2 Requesting Access

If you do not already have a Computation Institute account, you can request one at https://www.ci.uchicago.edu/accounts/. This page will give you a list of resources you can request access to. You already have an existing CI account, but do not have access to Beagle, send an email to support@ci.uchicago.edu to request access.

3.3 Connecting to a login node

Once you have account, you should be able to access a Beagle login node with the following command:

```
ssh -l username login.beagle.ci.uchicago.edu -A
```

or to log on to the sandbox:

```
ssh -l username sandbox.beagle.ci.uchicago.edu -A
```

Follow the steps outlined below to get started with Swift on Beagle:

step 1. Load the Swift module on Beagle as follows: `module load swift`

step 2. Create and change to a directory where your Swift related work will stay. (say, `mkdir swift-lab`, followed by, `cd swift-lab`)

step 3. To get started with a simple example running `/bin/cat` to read an input file `data.txt` and write to an output file `f.nnn.out`, start with writing a simple swift source script as follows:

```
type file;

/* App definitio */
app (file o) cat (file i)
{
  cat @i stdout=@o;
}

file out[]<simple_mapper; location="outdir", prefix="f.",suffix=".out">;
file data<"data.txt">;

/* App invocation: n times */
foreach j in [1:@toint(@arg("n","1"))] {
  out[j] = cat(data);
}
```

step 4. The next step is to create a sites file. An example sites file (sites.xml) is shown as follows:

```
<config>
  <pool handle="pbs">
    <execution provider="coaster" jobmanager="local:pbs"/>
    <!-- replace with your project -->
    <profile namespace="globus" key="project">CI-CCR000013</profile>

    <profile namespace="globus" key="providerAttributes">
                    pbs.aprun;pbs.mpp;depth=24</profile>

    <profile namespace="globus" key="jobsPerNode">24</profile>
    <profile namespace="globus" key="maxTime">1000</profile>
    <profile namespace="globus" key="slots">1</profile>
    <profile namespace="globus" key="nodeGranularity">1</profile>
    <profile namespace="globus" key="maxNodes">1</profile>

    <profile namespace="karajan" key="jobThrottle">.63</profile>
    <profile namespace="karajan" key="initialScore">10000</profile>

    <filesystem provider="local"/>
    <!-- replace this with your home on lustre -->
    <workdirectory >/lustre/beagle/ketan/swift.workdir</workdirectory>
  </pool>
</config>
```

step 5. In this step, we will see the config and tc files. The config file (cf) is as follows:

```
wrapperlog.always.transfer=true
sitedir.keep=true
execution.retries=1
lazy.errors=true
use.provider.staging=true
provider.staging.pin.swiftfiles=false
foreach.max.threads=100
provenance.log=false
```

The tc file (tc) is as follows:

```
pbs cat /bin/cat null null null
```

More about config and tc file options can be found in the swift userguide here: http://www.ci.uchicago.edu/swift/wwwdev/-guides/release-0.93/userguide/userguide.html#_swift_configuration_properties.

step 6. Run the example using following commandline:

```
swift -config cf -tc.file tc -sites.file sites.xml catsn.swift -n=1
```

You can further change the value of `-n` to any arbitrary number to run that many number of concurrent `cat` tasks.

step 7. Swift will show a status message as "done" after the job has completed its run in the queue. Check the output in the generated `outdir` directory (`ls outdir`)

```
Swift 0.93RC5 swift-r5285 cog-r3322

RunID: 20111218-0246-6ai8g7f0
Progress:  time: Sun, 18 Dec 2011 02:46:33 +0000
Progress:  time: Sun, 18 Dec 2011 02:46:42 +0000  Active:1
Final status:  time: Sun, 18 Dec 2011 02:46:43 +0000  Finished successfully:1
```

Note: Running from sandbox node or requesting 30 minutes walltime for upto 3 nodes will get fast prioritized execution. Suitable for small tests.

3.4 Larger Runs on Beagle

A key factor in scaling up Swift runs on Beagle is to setup the sites.xml parameters. The following sites.xml parameters must be set to scale that is intended for a large run:

- **maxTime** : The expected walltime for completion of your run. This parameter is accepted in seconds.
- **slots** : This parameter specifies the maximum number of pbs jobs/blocks that the coaster scheduler will have running at any given time. On Beagle, this number will determine how many qsubs swift will submit for your run. Typical values range between 40 and 60 for large runs.
- **nodeGranularity** : Determines the number of nodes per job. It restricts the number of nodes in a job to a multiple of this value. The total number of workers will then be a multiple of jobsPerNode * nodeGranularity. For Beagle, jobsPerNode value is 24 corresponding to its 24 cores per node.
- **maxNodes** : Determines the maximum number of nodes a job must pack into its qsub. This parameter determines the largest single job that your run will submit.
- **jobThrottle** : A factor that determines the number of tasks dispatched simultaneously. The intended number of simultaneous tasks must match the number of cores targeted. The number of tasks is calculated from the jobThrottle factor is as follows:

```
Number of parallel Tasks = (JobThrottle x 100) + 1
```

Following is an example sites.xml for a 50 slots run with each slot occupying 4 nodes (thus, a 200 node run):

```
<config>
  <pool handle="pbs">
    <execution provider="coaster" jobmanager="local:pbs"/>
    <profile namespace="globus" key="project">CI-CCR000013</profile>

    <profile namespace="globus" key="ppn">24:cray:pack</profile>

    <!-- For swift 0.93
    <profile namespace="globus" key="ppn">pbs.aprun;pbs.mpp;depth=24</profile>
    -->
```

```
    <profile namespace="globus" key="jobsPerNode">24</profile>
    <profile namespace="globus" key="maxTime">50000</profile>
    <profile namespace="globus" key="slots">50</profile>
    <profile namespace="globus" key="nodeGranularity">4</profile>
    <profile namespace="globus" key="maxNodes">4</profile>

    <profile namespace="karajan" key="jobThrottle">48.00</profile>
    <profile namespace="karajan" key="initialScore">10000</profile>

    <filesystem provider="local"/>
    <workdirectory >/lustre/beagle/ketan/swift.workdir</workdirectory>
  </pool>
</config>
```

3.5 Resuming Large Runs

Oftentimes, the application runs with a large number of tasks needed to be resumed after they have run to a certain point. The reasons for resume could be among others, application error, trainsient errors such as Beagle's availability or accidental shutdowns of the runs. In such cases, the **resume** feature of Swift is very handy. Resume starts the run from the point it left of. One can resume a stopped run using the same swift commandline plus adding the option -resume followed by a resume log (extension .rlog) that is created by Swift in your run directory. An example of such a resume follows:

```
$ swift -resume catsn-ht0adgi315l61.0.rlog <other options> catsn.swift
```

3.6 Troubleshooting

In this section we will discuss some of the common issues and remedies while using Swift on Beagle. The origin of these issues can be Swift or the Beagle's configuration, state and user configuration among other factors. We try to identify maximum known issues and address them here:

- Command not found: Swift is installed on Beagle as a module. If you see the following error message:

```
If 'swift' is not a typo you can run the following command to lookup the
package that contains the binary:
    command-not-found swift
-bash: swift: command not found
```

The most likely cause is the Swift module is not loaded. Do the following to load the Swift module:

```
$ module load swift
Swift version swift-0.93RC5 loaded
```

- Failed to transfer **wrapperlog** for job cat-nmobtbkk and/or Job failed with an exit code of 254. This is a most likely symptom of compute node trying to write to a non-writable filesystem. Check the <workdirectory> element on the sites.xml file.

```
<workdirectory >/home/ketan/swift.workdir</workdirectory>
```

It is likely that it is set to a path where the compute nodes can not write, e.g. your /home directory. The remedy for this error is to set your workdirectory to the /lustre path where swift could write from compute nodes.

```
<workdirectory >/lustre/beagle/ketan/swift.workdir</workdirectory>
```

- Out of heap space error is a typical error that you get when running large number of tasks in parallel from a submit host such as Beagle login nodes.

```
java.lang.OutOfMemoryError: Java heap space
```

A simple solution to this problem is to increase the java heap space. This can be solved by increasing the heap space Swift gets by the following environment variable:

```
SWIFT_HEAP_MAX=5000M swift -config cf -tc.file tc -sites.file sites.xml catsn.swift -n ↩
    =10000
```

- Application invocation fails or application returns a non-zero exit status. An application invocation might fail for a variety of reasons. Some of the common reasons include a faulty command line, out-of-memory, non-availability of data, library dependencies unmet, among others. In another set of failures, the application invocation might fail for a partial number of datasets. In these conditions, one might want to to continue for the rest of application invocations. In most cases, these conditions could be handled by catching various exitcodes and logging the erroneous invocations for later inspection. In the rest of this section, we provide some such examples.
  - Handling exitcodes in wrapperscript. The following code snippet from an application, handles the erroneous exitcode so that the erroneous runs could be logged and dealt with later:

```
call_to_app $1 $2
if [ "$exit_status" -ne 0 ]; then
    echo $2 | awk '{ print $1 }' >> /lustre/beagle/ketan/App_FailedList.txt
fi
```

- Advanced Handling of Out of Memory (OOM) Conditions. The following code snippet handles a case of OOM error conditions by monitoring the available memory at each invocation:

```
# if mem is low, wait for it to recover before starting
for i in $(seq 0 $maxtries); do
  freeMB=$(free -m | grep cache: | awk '{print $4}')
  if [ $freeMB -lt $lowmem ]; then
    if [ $i = $maxtries ]; then
      echo "$host $(date) freeMB = $freeMB below yellow mark $lowmem after $maxtries \
       $startsleep sec pauses. Exiting." >>$oomlog
      exit 7
    else
      echo "$host $(date) freeMB = $freeMB below yellow mark $lowmem on try $i. Sleeping \
      $startsleep sec." >>$oomlog
      sleep $startsleep
    fi
  else
    break
  fi
done

app_invocation $args
```

3.7 More Help

If the error messages you get does not give much clue, you can go about one of the following approaches to find more help: - Search for the particular error message on the swift mailing list archive from here: http://www.ci.uchicago.edu/swift/wwwdev/-support/index.php. It is likely someone has encountered the issue before and there is a ready remedy posted by one of the Swift team members. - Subscribe to the swift-user lists and post your questions here: https://lists.ci.uchicago.edu/cgi-bin/mailman/-listinfo/swift-user. Please attach the Swift-generated log file and the sites file with your question.

4 Fusion

Fusion is a 320-node computing cluster for the Argonne National Laboratory community. The primary goal of the LCRC is to facilitate mid-range computing in all of the scientific programs of Argonne and the University of Chicago.

This section will walk you through running a simple Swift script on Fusion.

4.1 Requesting Access

If you do not already have a Fusion account, you can request one at https://accounts.lcrc.anl.gov/request.php. Email support@lcrc.anl.gov for additional help.

4.2 Projects

In order to run a job on a Fusion compute node, you must first be associated with a project.

Each project has one or more Primary Investigators, or PIs. These PIs are responsible for adding and removing users to a project. Contact the PI of your project to be added.

More information on this process can be found at http://www.lcrc.anl.gov/info/Projects.

4.3 SSH Keys

Before accessing Fusion, be sure to have your SSH keys configured correctly. SSH keys are required to access fusion. You should see information about this when you request your account. Email support@lcrc.anl.gov for additional help.

4.4 Connecting to a login node

Once your keys are configured, you should be able to access a Fusion login node with the following command:

```
ssh yourusername@fusion.lcrc.anl.gov
```

4.5 Creating sites.xml

Swift relies on various configuration files to determine how to run. This section will provide a working configuration file which you can copy and paste to get running quickly. The sites.xml file tells Swift how to submit jobs, where working directories are located, and various other configuration information. More information on sites.xml can be found in the Swift User's Guide.

The first step is to paste the text below into a file named sites.xml.

```
<config>
<pool handle="fusion">
  <execution jobmanager="local:pbs" provider="coaster" url="none"/>
  <filesystem provider="local" url="none" />
  <profile namespace="globus" key="maxtime">750</profile>
  <profile namespace="globus" key="jobsPerNode">1</profile>
  <profile namespace="globus" key="slots">1</profile>
  <profile namespace="globus" key="nodeGranularity">1</profile>
  <profile namespace="globus" key="maxNodes">2</profile>
  <profile namespace="globus" key="queue">shared</profile>
  <profile namespace="karajan" key="jobThrottle">5.99</profile>
  <profile namespace="karajan" key="initialScore">10000</profile>
  <workdirectory>_WORK_</workdirectory>
</pool>
</config>
```

This file will require one customization. Create a directory called swiftwork. Modify _WORK_ in sites.xml to point to this new directory. For example

```
<workdirectory>/home/myhome/swiftwork</workdirectory>
```

4.6 Creating tc.data

The tc.data configuration file gives information about the applications that will be called by Swift. More information about the format of tc.data can be found in the Swift User's guide.

Paste the following example into a file named tc.data

```
fusion  echo            /bin/echo       INSTALLED       INTEL32::LINUX
fusion  cat             /bin/cat        INSTALLED       INTEL32::LINUX
fusion  ls              /bin/ls         INSTALLED       INTEL32::LINUX
fusion  grep            /bin/grep       INSTALLED       INTEL32::LINUX
fusion  sort            /bin/sort       INSTALLED       INTEL32::LINUX
fusion  paste           /bin/paste      INSTALLED       INTEL32::LINUX
fusion  wc              /usr/bin/wc     INSTALLED       INTEL32::LINUX
```

4.7 Copy a Swift Script

Within the Swift directory is an examples directory which contains several introductory Swift scripts. The example we will use in this section is called catsn.swift. Copy this script to the same directory that your sites.xml and tc.data files are located.

```
$ cp ~/swift-0.93/examples/misc/catsn.swift .
$ cp ~/swift-0.93/examples/misc/data.txt .
```

Tip
The location of your swift directory may vary depending on how you installed it. Change this to the examples/misc directory of your installation as needed.

4.8 Run Swift

Finally, run the script

```
$ swift -sites.file sites.xml -tc.file tc.data catsn.swift
```

You should see 10 new text files get created, named catsn*.out. If you see these files, then you have succesfully run Swift on Fusion!

4.9 Queues

Fusion has two queues: shared and batch. The shared queue has a maximum 1 hour walltime and limited to 4 nodes. The batch queue is for all other jobs.

Edit your sites.xml file and edit the queue option to modify Swift's behavior. For example:

```
<profile namespace="globus" key="queue">batch</profile>
```

More information on Fusion queues can be found at http://www.lcrc.anl.gov/info/BatchJobs.

4.10 More Help

The best place for additional help is the Swift user mailing list. You can subscribe to this list at https://lists.ci.uchicago.edu/cgi-bin/mailman/listinfo/swift-user. When submitting information, please send your sites.xml file, your tc.data, and any Swift log files that were created during your attempt.

5 Futuregrid Quickstart Guide

FutureGrid is a distributed, high-performance test-bed that allows scientists to collaboratively develop and test innovative approaches to parallel, grid, and cloud computing.

More information on futuregrid can be found at https://portal.futuregrid.org/.

5.1 Requesting Futuregrid Access

If you do not already have a futuregrid account, you can follow the instructions at https://portal.futuregrid.org/gettingstarted to get started. This page provides information on how to create an account, how to join a project, how to set up your SSH keys, and how to create a new project.

5.2 Downloading Swift VM Tools

A set of scripts based around cloudinitd are used to easily start virtual machines. To download, change to your home directory and run the following command:

```
$ svn co https://svn.ci.uchicago.edu/svn/vdl2/usertools/swift-vm-boot swift-vm-boot
```

5.3 Download your Credentials

Run the following commands to retrieve your credentials:

```
$ cd swift-vm-boot
$ scp yourusername@hotel.futuregrid.org:nimbus_creds.tar.gz .
$ tar xvfz nimbus_creds.tar.gz
```

When you extract your credential file, look at the file called hotel.conf. Near the bottom of this file will be two settings called vws.repository.s3id and vws.repository.s3key. Copy these values for the next step.

5.4 Configuring coaster-service.conf

To run on futuregrid, you will need a file called coaster-service.conf. This file contains many options to control how things run. Here is an example of a working coaster-service.conf on futuregrid.

```
# Where to copy worker.pl on the remote machine for sites.xml
export WORKER_LOCATION=/tmp

# How to launch workers: local, ssh, cobalt, or futuregrid
export WORKER_MODE=futuregrid

# Do all the worker nodes you're using have a shared filesystem? (yes/no)
export SHARED_FILESYSTEM=no

# Username to use on worker nodes
export WORKER_USERNAME=root
```

```
# Enable SSH tunneling? (yes/no)
export SSH_TUNNELING=yes

# Directory to keep log files, relative to working directory when launching start-coaster- ↩
    service
export LOG_DIR=logs

# Location of the swift-vm-boot scripts
export SWIFTVMBOOT_DIR=$HOME/swift-vm-boot

# Futuregrid settings
export FUTUREGRID_IAAS_ACCESS_KEY=XXXXXXXXXXXXXXXXXXXX
export FUTUREGRID_IAAS_SECRET_KEY=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
export FUTUREGRID_HOTEL_NODES=0
export FUTUREGRID_SIERRA_NODES=2
export FUTUREGRID_CPUS_PER_NODE=1

# Swift information for creating sites.xml
export WORK=/tmp
export JOBS_PER_NODE=$FUTUREGRID_CPUS_PER_NODE
export JOB_THROTTLE=$( echo "scale=5; ($JOBS_PER_NODE * $(($FUTUREGRID_HOTEL_NODES +  ↩
    $FUTUREGRID_SIERRA_NODES)))/100 - 0.00001"|bc )

# Application locations for tc.data
#app convert=/usr/bin/convert
```

Paste your credentials from the hotel.conf file into the FUTUREGRID_IAAS_ACCESS_KEY and FUTUREGRID_IAAS_SECRET_KE fields. Adjust the number of nodes you would like to allocate here by changing the values of FUTUREGRID_HOTEL_NODES and FUTUREGRID_SIERRA_NODES. Add a list of any applications you want to run in the format "#app myapp=/path/to/app".

5.5 Starting the Coaster Service Script

Now that everything is configured, change to the location of the coaster-service.conf file and run this command to start the coaster service:

```
$ start-coaster-service
```

This command will start the VMs, start the required processes on the worker nodes, and generate Swift configuration files for you to use. The configuration files will be generated in your current directory. These files are sites.xml, tc.data, and cf.

5.6 Running Swift

Now that you have all of your configuration files generated, run the following command:

```
$ swift -sites.file sites.xml -tc.file tc.data -config cf <yourscript.swift>
```

If you would like to create a custom tc file for repeated use, rename it to something other than tc.data to prevent it from being overwritten. The sites.xml however will need to be regenerated every time you start the coaster service. If you need to repeatedly modify some sites.xml options, you may edit the template in Swift's etc/sites/persistent-coasters. You may also create your own custom tc files with the hostname of persistent-coasters. More information about this can be found in the Swift userguide at http://www.ci.uchicago.edu/swift/guides/trunk/userguide/userguide.html.

5.7 Stopping the Coaster Service Script

To stop the coaster service, run the following command:

```
$ stop-coaster-service
```

This will kill the coaster service, kill the worker scripts on remote systems and terminate the virtual machines that were created during start-coaster-service.

5.8 More Help

The best place for additional help is the Swift user mailing list. You can subscribe to this list at http://mail.ci.uchicago.edu/-mailman/listinfo/swift-user. When submitting information, please send your sites.xml file, your tc.data, and any error messages you run into.

6 Intrepid

Intrepid is an IBM Blue Gene/P supercomputer located at the Argonne Leadership Computing Facility. More information on Intrepid can be found at http://www.alcf.anl.gov/.

6.1 Requesting Access

If you do not already have an account on Intrepid, you can request one at https://accounts.alcf.anl.gov/accounts/request.php. More information about this process and requesting allocations for your project can be found at http://www.alcf.anl.gov/support/-gettingstarted/index.php.

6.2 SSH Keys

Accessing the Intrepid via SSH can be done with any SSH software package. Before logging in, you will need to generate an SSH public key and send it to support@alcf.anl.gov for verification and installation.

6.3 Cryptocard

This security token uses one-time passwords for controlled access to the BG/P login systems.

6.4 Connecting to a login node

When you gain access to Intrepid, you should receive a cryptocard and a temporary PIN. You must have a working cryptocard, know your PIN, and have your SSH key in place before you may login.

You can connect to Intrepid with the following command:

```
ssh yourusername@intrepid.alcf.anl.gov
```

You will be presented with a password prompt. The first part of your password is your PIN. Enter you PIN, press the Cryptocard button, and then enter the password your crypocard generates. If this is the first time you are logging in, you will be prompted to change your PIN.

6.5 Downloading and building Swift

The most recent versions of Swift can be found at http://www.ci.uchicago.edu/swift/downloads/index.php. Follow the instructions provided on that site to download and build Swift.

6.6 Adding Swift to your PATH

Once you have installed Swift, add the Swift binary to your PATH so you can easily run it from any directory.

In your home directory, edit the file ".bashrc".

If you have installed Swift via a source repository, add the following line at the bottom of .bashrc.

```
export PATH=$PATH:$HOME/cog/modules/swift/dist/swift-svn/bin
```

If you have installed Swift via a binary package, add this line:

```
export PATH=$PATH:$HOME/swift-<version>/bin
```

Replace <version> with the actual name of the swift directory in the example above.

6.7 What You Need To Know Before Running Swift

Before you can create a Swift configuration file, there are some things you will need to know.

6.7.1 Swift Work Directory

The Swift work directory is a directory which Swift uses for processing work. This directory needs to be writable. Common options for this are:

```
/home/username/swiftwork
/home/username/work
/tmp
```

6.7.2 Which project(s) are you a member of?

Intrepid requires that you are a member of a project. You can determine this by running the following command:

```
$ projects
HTCScienceApps
```

If you are not a member of a project, you must first request access to a project. More information on this process can be found at https://wiki.alcf.anl.gov/index.php/Discretionary_Allocations

6.7.3 Determine your Queue

Intrepid has several different queues you can submit jobs to depending on the type of work you will be doing. The command "qstat -q" will print the most up to date list of this information.

Table 1: Intrepid Queues

User Queue	Queue	Nodes	Time (hours)	User Maxrun	Project maxrun
prod-devel	prod-devel	64-512	0-1	5	20
prod	prod-short	512-4096	0-6	5	20
prod	prod-long	512-4096	6-12	5	20
prod	prod-capability	4097-32768	0-12	2	20
prod	prod-24k	16385-24576	0-12	2	20
prod	prod-bigrun	32769-40960	0-12	2	20
prod	backfill	512-8192	0-6	5	10

6.8 Generating Configuration Files

Now that you know what queue to use, your project, and your work directory, it is time to set up Swift. Swift uses a configuration file called sites.xml to determine how it should run. There are two methods you can use for creating this file. You can manually edit the configuration file, or generate it with a utility called gensites.

6.8.1 Manually Editing sites.xml

Below is the template that is used by Swift's test suite for running on Intrepid. TODO: Update the rest below here

```
<config>

  <pool handle="localhost" sysinfo="INTEL32::LINUX">
    <gridftp url="local://localhost" />
    <execution provider="local" url="none" />
    <workdirectory>/scratch/wozniak/work</workdirectory>
    <!-- <profile namespace="karajan" key="maxSubmitRate">1</profile> -->
    <profile namespace="karajan" key="jobThrottle">0.04</profile>
    <profile namespace="swift"   key="stagingMethod">file</profile>
  </pool>

  <pool handle="coasters_alcfbgp">
    <filesystem provider="local" />
    <execution provider="coaster" jobmanager="local:cobalt"/>
    <!-- <profile namespace="swift"   key="stagingMethod">local</profile> -->
    <profile namespace="globus"  key="internalHostname">_HOST_</profile>
    <profile namespace="globus"  key="project">_PROJECT_</profile>
    <profile namespace="globus"  key="queue">_QUEUE_</profile>
    <profile namespace="globus"  key="kernelprofile">zeptoos</profile>
    <profile namespace="globus"  key="alcfbgpnat">true</profile>
    <profile namespace="karajan" key="jobthrottle">21</profile>
    <profile namespace="karajan" key="initialScore">10000</profile>
    <profile namespace="globus"  key="jobsPerNode">1</profile>
    <profile namespace="globus"  key="workerLoggingLevel">DEBUG</profile>
    <profile namespace="globus"  key="slots">1</profile>
    <profile namespace="globus"  key="maxTime">900</profile> <!-- seconds -->
    <profile namespace="globus"  key="nodeGranularity">64</profile>
    <profile namespace="globus"  key="maxNodes">64</profile>
    <workdirectory>_WORK_</workdirectory>
  </pool>

</config>
```

The values to note here are the ones that are listed between underscores. In the example above, they are _QUEUE_, and _WORK_. Queue is the PADS queue to use and WORK is the swift work directory. These are placeholder values you will need to modify to fit your needs. Copy and paste this template, replace the values, and call it sites.xml.

6.9 Manually Editing tc.data

Below is the tc.data file used by Swift's test suite for running on PADS.

```
coasters_alcfbgp        cp              /bin/cp         INSTALLED       INTEL32::LINUX   ↩
    null
```

Copy these commands and save it as tc.data.

6.10 Catsn.swift

The swift script we will run is called catsn.swift. It simply cats a file and saves the result. This is a nice simple test to ensure jobs are running correctly. Create a file called data.txt which contains some simple input - a "hello world" will do the trick.

```
type file;

app (file o) cat (file i)
{
  cat @i stdout=@o;
}

file out[]<simple_mapper; location=".", prefix="catsn.",suffix=".out">;
foreach j in [1:@toint(@arg("n","10"))] {
  file data<"data.txt">;
  out[j] = cat(data);
}
```

6.11 Running Swift

Now that everything is in place, run Swift with the following command:

```
swift -sites.file sites.xml -tc.file tc.data catsn.swift -n=10
```

You should see several new files being created, called catsn.0001.out, catsn.0002.out, etc. Each of these files should contain the contents of what you placed into data.txt. If this happens, your job has run successfully on PADS!

6.12 More Help

The best place for additional help is the Swift user mailing list. You can subscribe to this list at https://lists.ci.uchicago.edu/cgi-bin/mailman/listinfo/swift-user. When submitting information, please send your sites.xml file, your tc.data, and any Swift log files that were created during your attempt.

7 MCS Workstations

This sections describes how to use the general use compute servers for the MCS division of Argonne National Laboratory.

7.1 Create a coaster-service.conf

To begin, copy the text below and paste it into your Swift distribution's etc directory. Name the file coaster-service.conf.

```
# Keep all interesting settings in one place
# User should modify this to fit environment

# Location of SWIFT. If empty, PATH is referenced
export SWIFT=

# Where to place/launch worker.pl on the remote machine for sites.xml
export WORKER_WORK=/home/${USER}/work

# How to launch workers: local, ssh, or cobalt
export WORKER_MODE=ssh

# Worker logging setting passed to worker.pl for sites.xml
export WORKER_LOGGING=INFO
```

```
# Worker host names for ssh
export WORKER_HOSTS="crush.mcs.anl.gov thwomp.mcs.anl.gov stomp.mcs.anl.gov crank.mcs.anl. ↩
    gov
steamroller.mcs.anl.gov grind.mcs.anl.gov churn.mcs.anl.gov trounce.mcs.anl.gov
thrash.mcs.anl.gov vanquish.mcs.anl.gov"

# Directory to keep log files, relative to working directory when launching start-coaster- ↩
    service
export LOG_DIR=logs

# Manually define ports. If not specified, ports will be automatically generated
export LOCAL_PORT=
export SERVICE_PORT=

# start-coaster-service tries to automatically detect IP address.
# Specify here if auto detection is not working correctly
export IPADDR=

# Below are various settings to give information about how to create sites.xml
export work=$HOME/work
export queue=prod-devel
export maxtime=20
export nodes=64
```

7.2 Starting the Coaster Service

Change directories to the location you would like to run a Swift script and start the coaster service with this command:

```
start-coaster-service
```

This will create a configuration file that Swift needs called sites.xml.

Warning
Any existing sites.xml files in this directory will be overwritten. Be sure to make a copy of any custom configuration files you may have.

7.3 Run Swift

Next, run Swift. If you do not have a particular script in mind, you can test Swift by using a Swift script in the examples/ directory.

Run the following command to run the script:

```
swift -sites.file sites.xml -tc.file tc.data yourscript.swift
```

7.4 Stopping the Coaster Service

The coaster service will run indefinitely. The stop-coaster-service script will terminate the coaster service.

```
$ stop-coaster-service
```

This will kill the coaster service and kill the worker scripts on remote systems.

8 PADS

PADS is a petabyte-scale, data intense computing resource located at the joint Argonne National Laboratory/University of Chicago Computation Institute. More information about PADS can be found at http://pads.ci.uchicago.edu.

8.1 Requesting Access

If you do not already have a Computation Institute account, you can request access at https://www.ci.uchicago.edu/accounts. This page will give you a list of resources you can request access to. Be sure that PADS is selected. If you already have an existing CI account, but do not have access to PADS, send an email to support@ci.uchicago.edu to request access.

8.2 SSH Keys

Before accessing PADS, be sure to have your SSH keys configured correctly. There is some basic information about SSH and how to generate your key at http://www.ci.uchicago.edu/wiki/bin/view/Resources/SshKeys. Once you have followed those instructions, you can add your key at https://www.ci.uchicago.edu/support/sshkeys/.

8.3 Connecting to a login node

Once your keys are configured, you should be able to access a PADS login node with the following command:

```
ssh yourusername@login.pads.ci.uchicago.edu
```

8.4 Adding Software Packages

Softenv is a system used for managing applications. In order to run Swift, the softenv environment will have to be modified slightly. Softenv is configured by a file in your home directory called .soft. Edit this file to look like this:

```
+java-sun
+maui
+torque
@default
```

Log out of PADS, and log back in for these changes to take effect.

8.4.1 Which project(s) are you a member of?

PADS requires that you are a member of a project. You can determine this by running the following command:

```
$ projects --available

The following projects are available for your use

Project      PI                      Title
CI-CCR000013 Michael Wilde           The Swift Parallel Scripting System
```

If you are not a member of a project, you must first request access to a project at http://www.ci.uchicago.edu/hpc/projects.

You should make sure that you have a project set as default. Run the projects command with no arguments to determine if you have a default.

```
$ projects
You have no default project set.
```

To set your default project, use projects --set

```
$ projects --set CI-CCR000013 --all
Your default project for all CI clusters has been set to CI-CCR000013.
```

8.4.2 Creating sites.xml

Swift relies on various configuration files to determine how to run. This section will provide a working configuration file which you can copy and paste to get running quickly. The sites.xml file tells Swift how to submit jobs, where working directories are located, and various other configuration information. More information on sites.xml can be found in the Swift User's Guide.

The first step is to paste the text below into a file named sites.xml.

```
<config>
<pool handle="PADS-coasters">
  <execution jobmanager="local:pbs" provider="coaster" url="none"/>
  <filesystem provider="local" url="none" />
  <profile namespace="globus" key="maxWallTime">2</profile>
  <profile namespace="globus" key="maxTime">300</profile>
  <profile key="jobsPerNode" namespace="globus">1</profile>
  <profile key="slots" namespace="globus">1</profile>
  <profile key="nodeGranularity" namespace="globus">1</profile>
  <profile key="maxNodes" namespace="globus">1</profile>
  <profile key="queue" namespace="globus">fast</profile>
  <profile key="jobThrottle" namespace="karajan">5.99</profile>
  <profile key="initialScore" namespace="karajan">10000</profile>
  <workdirectory>_WORK_</workdirectory>
</pool>
</config>
```

This file will require just a few customizations. First, create a directory called swiftwork. Modify _WORK_ in sites.xml to point to this new directory. For example

```
<workdirectory>/home/myhome/swiftwork</workdirectory>
```

8.4.3 Creating tc.data

The tc.data configuration file gives information about the applications that will be called by Swift. More information about the format of tc.data can be found in the Swift User's guide.

Paste the following example into a file named tc.data

```
PADS-coasters   echo            /bin/echo       INSTALLED       INTEL32::LINUX  null
PADS-coasters   cat             /bin/cat        INSTALLED       INTEL32::LINUX  null
PADS-coasters   ls              /bin/ls         INSTALLED       INTEL32::LINUX  null
PADS-coasters   grep            /bin/grep       INSTALLED       INTEL32::LINUX  null
PADS-coasters   sort            /bin/sort       INSTALLED       INTEL32::LINUX  null
PADS-coasters   paste           /bin/paste      INSTALLED       INTEL32::LINUX  null
PADS-coasters   wc              /usr/bin/wc     INSTALLED       INTEL32::LINUX  null
```

8.4.4 Copy a Swift Script

Within the Swift directory is an examples directory which contains several introductory Swift scripts. The example we will use in this section is called catsn.swift. Copy this script to the same directory that your sites.xml and tc.data files are located.

```
$ cp ~/swift-0.93/examples/misc/catsn.swift .
$ cp ~/swift-0.93/examples/misc/data.txt .
```

Tip
The location of your swift directory may vary depending on how you installed it. Change this to the examples/misc directory of your installation as needed.

8.4.5 Run Swift

Finally, run the script

```
$ swift -sites.file sites.xml -tc.file tc.data catsn.swift
```

You should see several new files being created, called catsn.0001.out, catsn.0002.out, etc. Each of these files should contain the contents of what you placed into data.txt. If this happens, your job has run successfully on PADS!

Tip
Make sure your default project is defined. Read on for more information.

Read on for more detailed information about running Swift on PADS.

8.4.6 Queues

As you run more application in the future, you will likely need to change queues.

PADS has several different queues you can submit jobs to depending on the type of work you will be doing. The command "qstat -q" will print the most up to date list of this information.

Table 2: PADS Queues

Queue	Memory	CPU Time	Walltime	Node	Run	Que	Lm	State
route	—	—	—	—	0	0	—	E R
short	—	—	04:00:00	—	64	0	—	E R
extended	—	—	—	—	0	0	—	E R
fast	—	—	01:00:00	1	0	152	—	E R
long	—	—	24:00:00	—	232	130	—	E R

When you determine your computing requirements, modify this line in your sites.xml:

```
<profile key="queue" namespace="globus">fast</profile>
```

8.5 More Help

The best place for additional help is the Swift user mailing list. You can subscribe to this list at https://lists.ci.uchicago.edu/cgi-bin/mailman/listinfo/swift-user. When submitting information, please send your sites.xml file, your tc.data, and any Swift log files that were created during your attempt.