CIPRES REST v1.1 Users Guide

Introduction

The goal of the CIPRES REST API (CRA) is to allow users to access phylogenetic software supported by CIPRES outside the confines of a point and click browser interface. Unlike the CIPRES Science Gateway (CSG) website , which stores jobs and data indefinitely, the CRA is intended to be a convenient way to run phylogenetic programs on large HPC resources, but does not provide long term data storage. The CRA currently stores jobs for only 4 weeks. This time period is long enough to troubleshoot problems and to ensure that job results aren't lost, but organization and preservation of jobs and results from the CRA is the responsibility of the user.

To use the CRA, you must register as a user, and register any application(s) you wish to develop, as well. Instructions for registration are found below.

The base URL for the API is **https://bumper.sdsc.edu/cipresrest/v1**.

The examples in this guide use the unix `curl` command and assume you have registered with the CRA and have set the following shell variables:

- URL - Base URL.
- PASSWORD - Your CRA password.
- KEY - the application ID assigned when you registered the application.

For example, using the bash shell:

```
$ export URL=https://bumper.sdsc.edu/cipresrest/v1/
```

```
$ export PASSWORD=MyPassWord
$ export KEY=insects-095D20923FAE439982B6D5EBD2E339C9
```

`curl` is of course just one of many ways to interact with a REST API. There are numerous java, php, perl, python, etc., libraries that make it easy to use REST services.

Register

To get started, sign in or register for a CIPRES REST account. Once you've signed in, you can visit "My Profile" to change your account information and password. To register an application, use the Application Management console, found under the "Developer" drop down menu.

> TIP: CIPRES REST API accounts are separate from those used by the CSG website , so you'll still need to register to use the CRA even if you are already a CSG user.

When you register an application, you must choose between DIRECT and UMBRELLA authentication models.

DIRECT is the more common choice, and the choice you want if you wish to use the API from your application immediately. DIRECT authentication means that the username and password of the person running the application will be sent in HTTP basic authentication headers, and jobs will be submitted on behalf of the authenticated user only. If people other than you will be running your application, they will need to register for their own CRA accounts and provide their credentials to your application, so that your code can submit jobs for them.

UMBRELLA is a special case used by web applications that submit jobs on behalf of multiple registered users. Web applications that use UMBRELLA authentication also authenticate with a username and password, that of the person who registered the application. The UMBRELLA application provides the identity of the user that submitted a given job using custom request headers. As a result, users registered with an UMBRELLA application need not register with the CRA. Because UMBRELLA authentication involves a trust relationship (i.e. we are trusting you to accurately identify the individual who submits each request), we will need to talk to you before activating your UMBRELLA application to insure all of our requirements are met. If you are interested in registering an UMBRELLA application, please contact us.

The examples shown in this guide are for DIRECT applications, but with minor changes, they will also work for UMBRELLA Applications, as shown inUMBRELLA Application Examples.

Authenticate

The API requires you to send a username and password in HTTP Basic Authentication headers with each request. The use of SSL ensures that the information is transmitted securely.

In addition to sending a username and password, you must send your application ID in a custom request header named `cipres-appkey`.

List Jobs

Let's get started using the API. Suppose your username is `tom`, you've registered a DIRECT application named `insects`, and set URL, PASSWORD and KEY environment variables as shown in the Introduction. Here's how you would get a list of the jobs you've submitted:

```
$ curl -u tom:$PASSWORD \
        -H cipres-appkey:$KEY\
        $URL/job/tom

&lt;?xml version="1.0" encoding="UTF-8" standalone="yes"?&gt;
&lt;joblist&gt;
    &lt;title&gt;Submitted Jobs&lt;/title&gt;
    &lt;jobs&gt;
        &lt;jobstatus&gt;
            &lt;selfUri&gt;
                &lt;url&gt;$URL/v1/job/tom/NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90&lt;/url&gt;
                &lt;rel&gt;jobstatus&lt;/rel&gt;
                &lt;title&gt;NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90&lt;/title&gt;
            &lt;/selfUri&gt;
        &lt;/jobstatus&gt;
        &lt;jobstatus&gt;
            &lt;selfUri&gt;
                &lt;url&gt;$URL/v1/job/tom/NGBW-JOB-CLUSTALW-CC460782E5FF464CB96791B1E6053AA4&lt;/url&gt;
                &lt;rel&gt;jobstatus&lt;/rel&gt;
                &lt;title&gt;NGBW-JOB-CLUSTALW-CC460782E5FF464CB96791B1E6053AA4&lt;/title&gt;
            &lt;/selfUri&gt;
        &lt;/jobstatus&gt;
    &lt;/jobs&gt;
&lt;/joblist&gt;
```

To get more information about a specific job in the list, use its `jobstatus.selfUri.url`. For example, to retrieve the full jobstatus of the first job in the list above:

```
$ curl -u tom:$PASSWORD \
    -H cipres-appkey:$KEY \
    $URL/job/tom/NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90
```

```
&lt;?xml version="1.0" encoding="UTF-8" standalone="yes"?&gt;
&lt;jobstatus&gt;
        . . .
&lt;/jobstatus&gt;
```

Alternatively, when when you ask for the list of jobs, use the `expand=true` query parameter to request full jobstatus objects.

If you have a CIPRES REST account and have registered a DIRECT application, try getting your list of submitted jobs now. Since you haven't submitted any jobs yet, the list will be empty and will look like this:

```
&lt;?xml version="1.0" encoding="UTF-8" standalone="yes"?&gt;
&lt;joblist&gt;
    &lt;title&gt;Submitted Jobs&lt;/title&gt;
    &lt;jobs/&gt;
&lt;/joblist&gt;
```

TIP: Throughout the API, XML elements named `selfUri` link to the full version of the containing object. All Uri elements, including `selfUri`, contain a `url` which gives the actual url, a `rel` which describes the type of data that the url returns and a `title`. It's good practice to navigate through the API by using the Uris the API returns instead of constructing urls to specific objects yourself.

Submit Jobs

Now that we know how to list jobs; let's consider job submission. You can submit a job by issuing a POST request to `$URL/job/username` with multipart/form-data. Remember to replace `username` with your username, or the username of the person running your application. Most tools can be run minimally using only two fields: a tool identifier and a file to be processed.

Below is an example of a minimal job submission:

```
$ curl -u tom:$PASSWORD \
    -H cipres-appkey:$KEY \
    $URL/job/tom \
    -F tool=CLUSTALW \
    -F input.infile_=@./sample1_in.fasta
```

In this example, the fields used are:

`tool=CLUSTALW`

The `tool` field identifies the tool to be used, in this case, CLUSTALW. Job submissions must always include a `tool`. You can find a list of phylogenetic programs and their respective tool IDs by using the Tool API.

`input.infile_=@./sample1_in.fasta`

The `input.infile_` field is also mandatory; it identifies the main data file to be operated on. `input.infile_` is usually a set of sequences to align or a character matrix. In this example, we're sending the contents of the file named sample1_in.fasta. The '@' tells curl to send sample1_in.fasta as an attached file.

A submission like this will succeed for most tools, and will cause the application to run a job with whatever defaults CIPRES has for that particular tool. You can try a CLUSTALW job this way if you like, using a sample input file. Of course, many job submissions will require configuration of command line options to non-default values, and (often) submission of auxiliary files that specify starting trees, constraints, etc. The appendix of this guide has a section that explains how to configure tool specific parameters.

Use Optional Metadata

A job submission may include the following optional metadata fields:

`metadata.clientJobId`

Your application's unique ID for the job.**We highly recommended that you use this field**. You may encounter situations where it isn't clear whether or not a submission reached the CRA, in which case, the best thing to do is request a list of your jobs and see whether or not it includes one with the clientJobId you just tried to submit.

`metadata.clientJobName`

A name that the user of your application will recognize the job by.

`metadata.clientToolName`

A name that the user will recognize the tool by.

`metadata.statusEmail`

If "true", email will be sent on job completion. (Delivery, of course, depends upon an valid email address, and functioning delivery infrastructure).

`metadata.emailAddress`

Use this along with `statusEmail` to override the default email destination. By default, job completion emails are sent to the user's registered email address. (Or in the case of UMBRELLA applications, to the address in the cipres-eu-email header of the job submission request). Use this property to direct the email somewhere else.

`metadata.statusUrlPut`

Use this field to specify a URL in your web application where CIPRES will PUT a notification when the job is finished. CIPRES can't guarantee that your application will receive the PUT request so you may still need to poll occasionally. Not implemented yet.

`metadata.statusUrlGet`

Use this field to specify a URL in your web application that CIPRES will GET, with a jh=jobhandle query parameter, when the job is finished. CIPRES can't guarantee that your application will receive the request so you may still need to poll occasionally. Not implemented yet.

All metadata fields are limited to 100 characters, and all are optional. Metadata will be returned with the rest of the information about the job when you later ask for the job's status.

In the following example, Tom, uses some of the metadata fields described above to supply a job ID, generated by his application, and to request email notification of job completion.

```
$ curl -u tom:$PASSWORD \
        -H cipres-appkey:$KEY \
        $URL/job/tom \
        -F tool=CLUSTALW \
        -F input.infile_=@./sample1_in.fasta \
        -F metadata.clientJobId=101 \
        -F metadata.statusEmail=true
```

As noted above, many runs will be more complicated than this because of the need to configure the precise command line. We suggest that you continue through this guide to learn how to check job status, download results, and handle errors, and then read Configure Tool Specific Parameters in the Appendix to learn how to create customized runs.

Understand Job Status

Successful job submission returns a `jobstatus` object that looks like this:

```
&lt;?xml version="1.0" encoding="UTF-8" standalone="yes"?&gt;
&lt;jobstatus&gt;
    &lt;selfUri&gt;
        &lt;url&gt;$URL/v1/job/tom/NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90&lt;/url&gt;
        &lt;rel&gt;jobstatus&lt;/rel&gt;
        &lt;title&gt;NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90&lt;/title&gt;
    &lt;/selfUri&gt;
    &lt;jobHandle&gt;NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90&lt;/jobHandle&gt;
    &lt;jobStage&gt;QUEUE&lt;/jobStage&gt;
    &lt;terminalStage&gt;false&lt;/terminalStage&gt;
    &lt;failed&gt;false&lt;/failed&gt;
    &lt;metadata&gt;
        &lt;entry&gt;
            &lt;key&gt;clientJobId&lt;/key&gt;
            &lt;value&gt;101&lt;/value&gt;
        &lt;/entry&gt;
    &lt;/metadata&gt;
    &lt;dateSubmitted&gt;2014-09-10T15:54:58-07:00&lt;/dateSubmitted&gt;
    &lt;resultsUri&gt;
        &lt;url&gt;$URL/v1/job/tom/NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90/output&lt;/url&gt;
        &lt;rel&gt;results&lt;/rel&gt;
        &lt;title&gt;Job Results&lt;/title&gt;
    &lt;/resultsUri&gt;
    &lt;workingDirUri&gt;
        &lt;url&gt;$URL/v1/job/tom/NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90/workingdir&lt;/url&gt;
        &lt;rel&gt;workingdir&lt;/rel&gt;
        &lt;title&gt;Job Working Directory&lt;/title&gt;
    &lt;/workingDirUri&gt;
    &lt;messages&gt;
        &lt;message&gt;
```

```
            &lt;timestamp&gt;2014-09-10T15:54:59-07:00&lt;/timestamp&gt;
            &lt;stage&gt;QUEUE&lt;/stage&gt;
            &lt;text&gt;Added to cipres run queue.&lt;/text&gt;
        &lt;/message&gt;
    &lt;/messages&gt;
&lt;/jobstatus&gt;
```

Elements of particular interest are:

`jobHandle`
Is a unique, CIPRES assigned, job identifier. It has the format: NGBW-JOB-*toolID-unique_identifier*

`jobStage`
Unfortunately, the current version of CIPRES sets jobstatus.jobStage in a way that's somewhat inconsistent and difficult to explain. You're better off using `jobstatus.messages` to monitor the progress of a job.

`messages`
CIPRES adds a message at each major processing point, as well as when problems are encountered. Each message has a timestamp, processing stage, and textual description. A job progresses through the following stages:

- QUEUE - The job has been validated and placed in CIPRES's queue.
- COMMANDRENDERING - The job has reached the head of the queue and CIPRES has created the command line that will be run.
- INPUTSTAGING - CIPRES has created a temporary working directory for the job on the execution host and copied the input files over.
- SUBMITTED - The job has been submited to the scheduler on the execution host.
- LOAD_RESULTS - The job has finished running on the execution host and CIPRES has begun to transfer the results.
- COMPLETED - Results successfully transferred and available.

`terminalStage`
If true, CIPRES has finished processing the job. If false, there is more to do.

`failed`
This will only be set to true only when the job is finished (i.e. terminalStage=true) and the job has failed. CIPRES has a narrow definition of failure that does *not* take the tool's output or exit code into consideration. A job will only have failed=true if a network or system error prevents the tool from being run or prevents CIPRES from being able to obtain the job's results.

The jobstatus also includes several urls:

`selfUri`
Use this to poll for updated job status.

`workingDirUri`
Use this to monitor the files in the job's working directory, while the job is running.

`resultsUri`
Use this to get the list of result files, once the job has finshed.

Is the Job Finished?

The job is finished when `jobstatus.terminalStage=true`. Use `jobstatus.selfUri.url` to check the status of the job, like this:

```
$ curl -u tom:$PASSWORD \
    -H cipres-appkey:$KEY \
    $URL/job/tom/NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90
```

Alternatively, you can check the status of multiple jobs in a single GET of endpoint `$URL/job` by using multiple instances of the `jh=jobhandle` query parameter. In this case the URL does not include the username (so that UMBRELLA applications can check on jobs for all their end users with a single query).

```
$ curl -u tom:$PASSWORD \
    -H cipres-appkey:$KEY\
    $URL/job/?jh=NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90\&amp;jh=NGBW-JOB-CLUSTALW-CC460782E5FF464CB96791B1E6053AA4
```

We ask users to keep polling frequency as low as possible to avoid overloading CIPRES: in general, please poll once a minute at most, for very short jobs and once every 15 - 30 minutes for longer jobs. As an alternative to frequent polling, consider using `metdata.statusEmail=1` in your job submission so that CIPRES will email you when the job is finished. Showing courtesy here will allow us to avoid having to enforce hard limits.

List Results

Once `jobstatus.terminalStage=true`, you can list and then retrieve the final results. Issue a GET request to the URL specified by `jobstatus.selfUrl.url`, like this:

```
$ curl -u tom:$PASSWORD \
    -H cipres-appkey:$KEY \
    $URL/job/tom/NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90/output


&lt;?xml version="1.0" encoding="UTF-8" standalone="yes"?&gt;
&lt;results&gt;
    &lt;jobfiles&gt;
        &lt;jobfile&gt;
            &lt;downloadUri&gt;
                &lt;url&gt;$URL/v1/job/tom/NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90/output/1544&lt;/url&gt;
                &lt;rel&gt;fileDownload&lt;/rel&gt;
                &lt;title&gt;STDOUT&lt;/title&gt;
            &lt;/downloadUri&gt;
            &lt;jobHandle&gt;NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90&lt;/jobHandle&gt;
            &lt;filename&gt;STDOUT&lt;/filename&gt;
            &lt;length&gt;1243&lt;/length&gt;
            &lt;parameterName&gt;PROCESS_OUTPUT&lt;/parameterName&gt;
            &lt;outputDocumentId&gt;1544&lt;/outputDocumentId&gt;
        &lt;/jobfile&gt;
        &lt;jobfile&gt;
            &lt;downloadUri&gt;
                &lt;url&gt;$URL/v1/job/tom/NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90/output/1545&lt;/url&gt;
                &lt;rel&gt;fileDownload&lt;/rel&gt;
                &lt;title&gt;STDERR&lt;/title&gt;
            &lt;/downloadUri&gt;
            &lt;jobHandle&gt;NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90&lt;/jobHandle&gt;
            &lt;filename&gt;STDERR&lt;/filename&gt;
            &lt;length&gt;0&lt;/length&gt;
            &lt;parameterName&gt;PROCESS_OUTPUT&lt;/parameterName&gt;
            &lt;outputDocumentId&gt;1545&lt;/outputDocumentId&gt;
        &lt;/jobfile&gt;
        &lt;jobfile&gt;
            &lt;downloadUri&gt;
                &lt;url&gt;$URL/v1/job/tom/NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90/output/1550&lt;/url&gt;
                &lt;rel&gt;fileDownload&lt;/rel&gt;
                &lt;title&gt;infile.aln&lt;/title&gt;
            &lt;/downloadUri&gt;
            &lt;jobHandle&gt;NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90&lt;/jobHandle&gt;
            &lt;filename&gt;infile.aln&lt;/filename&gt;
            &lt;length&gt;1449&lt;/length&gt;
            &lt;parameterName&gt;aligfile&lt;/parameterName&gt;
            &lt;outputDocumentId&gt;1550&lt;/outputDocumentId&gt;
        &lt;/jobfile&gt;
        &lt;jobfile&gt;
            &lt;downloadUri&gt;
                &lt;url&gt;$URL/v1/job/tom/NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90/output/1551&lt;/url&gt;
                &lt;rel&gt;fileDownload&lt;/rel&gt;
                &lt;title&gt;term.txt&lt;/title&gt;
            &lt;/downloadUri&gt;
            &lt;jobHandle&gt;NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90&lt;/jobHandle&gt;
            &lt;filename&gt;term.txt&lt;/filename&gt;
            &lt;length&gt;338&lt;/length&gt;
            &lt;parameterName&gt;all_outputfiles&lt;/parameterName&gt;
            &lt;outputDocumentId&gt;1551&lt;/outputDocumentId&gt;
        &lt;/jobfile&gt;
        &lt;jobfile&gt;
            &lt;downloadUri&gt;
                &lt;url&gt;$URL/v1/job/tom/NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90/output/1552&lt;/url&gt;
                &lt;rel&gt;fileDownload&lt;/rel&gt;
                &lt;title&gt;batch_command.cmdline&lt;/title&gt;
            &lt;/downloadUri&gt;
            &lt;jobHandle&gt;NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90&lt;/jobHandle&gt;
            &lt;filename&gt;batch_command.cmdline&lt;/filename&gt;
            &lt;length&gt;48&lt;/length&gt;
```

```
            &lt;parameterName&gt;all_outputfiles&lt;/parameterName&gt;
            &lt;outputDocumentId&gt;1552&lt;/outputDocumentId&gt;
        &lt;/jobfile&gt;
        &lt;jobfile&gt;

...
&lt;/jobfiles&gt;
&lt;/results&gt;
```

Download Results

Use the `jobfile.downloadUri.url` links to download individual result files, like this:

```
$ curl -u tom:$PASSWORD \
    -H cipres-appkey:$KEY \
    -O -J \
    $URL/job/tom/NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90/output/1544

% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1243    0  1243    0     0    178      0 --:--:--  0:00:06 --:--:--   313
curl: Saved to filename 'STDOUT'
```

List Working Directory

If you are interested in monitoring the progress of a job while it is running, you can use `jobstatus.workingDirUri.url` to retrieve the list of files in the job's working directory. The job only has a working directory after it has been staged to the execution host and is waiting to run, is running, or is waiting to be cleaned up. If you use this URL at other times, it will return an empty list. Furthermore, if you happen to use this URL while CIPRES is in the process of removing the working directory, you may receive a transient error. Because of this possibility, be prepared to retry the operation.

```
$ curl -u tom:$PASSWORD \
    -H cipres-appkey:$KEY \
    $URL/job/tom/NGBW-JOB-CLUSTALW-3957CC6EBF5E448095A5666B41EDDF90/workingdir

&lt;?xml version="1.0" encoding="UTF-8" standalone="yes"?&gt;
&lt;workingdir&gt;
    &lt;jobfiles/&gt;
&lt;/workingdir&gt;
&lt;?xml version="1.0" encoding="UTF-8" standalone="yes"?&gt;
&lt;workingdir&gt;
    &lt;jobfiles&gt;
        &lt;jobfile&gt;
            &lt;downloadUri&gt;
                &lt;url&gt;$URL/job/tom/NGBW-JOB-CLUSTALW-0171A3F1BFA0477CAF35B79CE075DF9C/workingdir/scheduler.conf&lt;/url&gt;
                &lt;rel&gt;fileDownload&lt;/rel&gt;
                &lt;title&gt;scheduler.conf&lt;/title&gt;
            &lt;/downloadUri&gt;
            &lt;filename&gt;scheduler.conf&lt;/filename&gt;
            &lt;length&gt;11&lt;/length&gt;
            &lt;dateModified&gt;2014-09-20T16:18:05-07:00&lt;/dateModified&gt;
            &lt;parameterName&gt;&lt;/parameterName&gt;
            &lt;outputDocumentId&gt;0&lt;/outputDocumentId&gt;
        &lt;/jobfile&gt;
        &lt;jobfile&gt;
            &lt;downloadUri&gt;
                &lt;url&gt;$URL/job/tom/NGBW-JOB-CLUSTALW-0171A3F1BFA0477CAF35B79CE075DF9C/workingdir/infile.dnd&lt;/url&gt;
                &lt;rel&gt;fileDownload&lt;/rel&gt;
                &lt;title&gt;infile.dnd&lt;/title&gt;
            &lt;/downloadUri&gt;
            &lt;filename&gt;infile.dnd&lt;/filename&gt;
            &lt;length&gt;137&lt;/length&gt;
            &lt;dateModified&gt;2014-09-20T16:18:13-07:00&lt;/dateModified&gt;
            &lt;parameterName&gt;&lt;/parameterName&gt;
            &lt;outputDocumentId&gt;0&lt;/outputDocumentId&gt;
        &lt;/jobfile&gt;
                . . .
    &lt;/jobfiles&gt;
```

```
&lt;/workingdir&gt;
```

Download Working Dir Files

To retrieve a file from the working directory list, use its `jobfile.downloadUri.url`. Be prepared to handle transient errors, as well as a permanent 404 NOT FOUND error, once the working directory has been removed.

```
$ curl -k -u tom:tom \
    -H cipres-appkey:$KEY \
    -O -J \
    $URL/job/tom/NGBW-JOB-CLUSTALW-0171A3F1BFA0477CAF35B79CE075DF9C/workingdir/infile.dnd

curl: Saved to filename 'infile.dnd'
```

Delete and Cancel a Job

Once a job has finished and you've downloaded the results, it's a good idea to delete the job. You may also want to delete a job that hasn't finished yet if you, or the user of your application, realize you made a mistake and don't want to waste the compute time.

```
$ curl -u tom:$PASSWORD \
    -H cipres-appkey:$KEY \
    -X DELETE \
    $URL/job/tom/NGBW-JOB-CLUSTALW-CC460782E5FF464CB96791B1E6053AA4
```

There is no data returned from a successful DELETE.

If the job is scheduled to run or is running at the time you delete it, it will be cancelled. Either way, all info associated with the job will be removed. You can verify that the job has been deleted by doing a GET of its jobstatus url. Http status 404 (NOT FOUND) will be returned along with an error object. We demonstrate this below by using curl's -i option, which tells curl to include the http header in its output.

```
$ curl -i -u tom:$PASSWORD \
    -H cipres-appkey:$KEY \
    $URL/job/tom/NGBW-JOB-CLUSTALW-CC460782E5FF464CB96791B1E6053AA4

HTTP/1.1 404 Not Found
Server: Apache-Coyote/1.1
Content-Type: application/xml
Transfer-Encoding: chunked
Date: Thu, 11 Sep 2014 21:43:54 GM

&lt;?xml version="1.0" encoding="UTF-8" standalone="yes"?&gt;
&lt;error&gt;
    &lt;displayMessage&gt;Job not found.&lt;/displayMessage&gt;
    &lt;message&gt;Job Not Found Error: org.ngbw.sdk.jobs.JobNotFoundException: NGBW-JOB-CLUSTALW-CC460782E5FF464CB96791B1E6053AA4&
lt;/message&gt;
    &lt;code&gt;4&lt;/code&gt;
&lt;/error&gt;
```

Handle Errors

Http status codes are used to indicate whether an API request succeeded or failed. When the http status indicates failure (with a status other than 200) an `error` object is returned. A basic `error` object looks like this:

```
&lt;error&gt;
    &lt;displayMessage&gt;Job Not Found&lt;/displayMessage&gt;
    &lt;message&gt;Job Not Found Error: org.ngbw.sdk.jobs.JobNotFoundException: NGBW-JOB-CLUSTALW-261679BE83E245AD8EEECB4592A52B81
    &lt;/message&gt;
    &lt;code&gt;4&lt;/code&gt;
&lt;/error&gt;
```

The `displayMessage` is a user friendly description of the error. The contents of the `message` are not meant for end users, but may be helpful in debugging. The `code` indicates the type of error, for example code = 4 is "not found", as shown in the source code for ErrorData.java

A job validation error may contain a list of field errors. For example:

```
$ curl -u tom:$PASSWORD \
    -H cipres-appkey:$KEY \
    $URL/job/tom \
    -F tool=CLUSTALW \
    -F metadata.clientJobId=110 \
    -F input.infile_=@./sample1_in.fasta \
    -F vparam.runtime_="one hour" \
    -F vparam.foo_=bar

&lt;?xml version="1.0" encoding="UTF-8" standalone="yes"?&gt;
&lt;error&gt;
    &lt;displayMessage&gt;Form validation error.&lt;/displayMessage&gt;
    &lt;message&gt;Validation Error: &lt;/message&gt;
    &lt;code&gt;5&lt;/code&gt;
    &lt;paramError&gt;
        &lt;param&gt;runtime_&lt;/param&gt;
        &lt;error&gt;Must be a Double.&lt;/error&gt;
    &lt;/paramError&gt;
    &lt;paramError&gt;
        &lt;param&gt;foo_&lt;/param&gt;
        &lt;error&gt;Does not exist.&lt;/error&gt;
    &lt;/paramError&gt;
&lt;/error&gt;
```

Data Types

The XML documents or data structures returned by the API, such as `jobstatus`, `results`, `jobfile`, `error`, etc., are not fully documented yet, however the iava source code is available. CIPRES maps the datatypes classes to XML using JAXB. If you happen to be implementing in java you may want to use the datatypes classes to unmarshall the XML.

Tool API

The tool API provides information about the phylogenetic tools that can be run on CIPRES. It's public: no credentials and no special headers are required, so it's easy to use a browser or `curl` to explore it. You can use the Tool API to learn the IDs of the tools you're interested in running and to download their PISE XML descriptions.

Definition: Strictly speaking, a CIPRES *tool* is an interface for configuring command line job submissions. It is defined by a PISE XML document found in the Tool API. Each *tool* deploys jobs for a single phylogenetic program (e.g. CLUSTALW, MrBayes, RaXML, etc.). However, more than one tool may invoke the same program. For example, the RAxML program, is run by two tools, one that provides a simple "blackbox" interface (RAXMLHPC2BB), and one that exposes nearly all RAxML options (RAXMLHPC2_TGB).

Go to `$URL/tool` in the browser, or use curl, as shown below, to see a list of the available tools:

```
$ curl $URL/tool

        &lt;?xml version="1.0" encoding="UTF-8" standalone="yes"?&gt;
        &lt;tools&gt;
        &lt;tool&gt;
                &lt;toolId&gt;MRBAYES_321RESTARTBETA&lt;/toolId&gt;
                &lt;toolName&gt;Tree Inference Using Bayesian Analysis - run on XSEDE&lt;/toolName&gt;
                &lt;selfUri&gt;
                        &lt;url&gt;$URL/tool/MRBAYES_321RESTARTBETA&lt;/url&gt;
                        &lt;rel&gt;tool&lt;/rel&gt;
                        &lt;title&gt;MRBAYES_321RESTARTBETA&lt;/title&gt;
                &lt;/selfUri&gt;
                &lt;piseUri&gt;
                        &lt;url&gt;$URL/tool/MRBAYES_321RESTARTBETA/doc/pise&lt;/url&gt;
                        &lt;rel&gt;Pise XML&lt;/rel&gt;
                        &lt;title&gt;MRBAYES_321RESTARTBETA pise&lt;/title&gt;
                &lt;/piseUri&gt;
                &lt;portal2Uri&gt;
                        &lt;url&gt;$URL1/tool/MRBAYES_321RESTARTBETA/doc/portal2&lt;/url&gt;
                        &lt;rel&gt;Html Web Page&lt;/rel&gt;
                        &lt;title&gt;MRBAYES_321RESTARTBETA type&lt;/title&gt;
                &lt;/portal2Uri&gt;
                &lt;exampleUri&gt;
                        &lt;url&gt;$URL/tool/MRBAYES_321RESTARTBETA/doc/example&lt;/url&gt;
                        &lt;rel&gt;Html Web Page&lt;/rel&gt;
                        &lt;title&gt;MRBAYES_321RESTARTBETA type&lt;/title&gt;
```

```
                &lt;/exampleUri&gt;
                &lt;parameterUri&gt;
                        &lt;url&gt;$URL/tool/MRBAYES_321RESTARTBETA/doc/param&lt;/url&gt;
                        &lt;rel&gt;Html Web Page&lt;/rel&gt;
                        &lt;title&gt;MRBAYES_321RESTARTBETA type&lt;/title&gt;
                &lt;/parameterUri&gt;
        &lt;/tool&gt;
        &lt;tool&gt;
                &lt;toolId&gt;PROBCONS&lt;/toolId&gt;
                &lt;toolName&gt;Probabilistic Consistency-based Multiple Alignment of Amino/Nucleic Acid Sequences&lt;/toolName&gt;
                &lt;selfUri&gt;
                        &lt;url&gt;$URL/tool/PROBCONS&lt;/url&gt;
                        &lt;rel&gt;tool&lt;/rel&gt;
                        &lt;title&gt;PROBCONS&lt;/title&gt;
                &lt;/selfUri&gt;
                . . .
        &lt;/tools&gt;
```

Each tool description includes the `toolId`, `toolName`, and a number of "Uri" elements, which are links to various documents for the specific tool.

As we mentioned earlier, it's good practice to navigate through the API using these returned links rather than hardcoding the urls. For example, all the urls in the table below can all be extracted from the data returned by the top level resource at `$URL/tool`.

Tool API Summary

GET	$URL/tool	Use this to get a list of the available tools.
GET	$URL/tool/*toolId*	Use this to get the URLs that link to the tool's documents (ie. the documents listed below).
GET	$URL/tool/*toolId*/doc/pise	Uset this URL to download the tool's PISE XML file.
GET	$URL/tool/*toolId*/doc/portal2	Use this URL, in a browser, to read a detailed description of the tool. This URL returns http status 303 and a Location header that redirects to the html page on the CIPRES Science Gateway that gives a detailed, human readable, description of the tool.
GET	$URL/tool/*toolId*/doc/example	Not implemented yet. Will give examples showing how to submit jobs to use this tool.
GET	$URL/tool/*toolId*/doc/param	Not implemented yet. Will give a human readable description of each of the tool's parameters.

More Examples

We are in the process of developing samples that you can download and install, but we're not quite finished. In the meantime, it may be helpful to browse through the following source code directories:

Python Example

The code in this directory is an example of a DIRECT application that uses the Requests library to submit jobs.

Java Example

This directory has the source for a very bare bones struts based web application that lets a user login, submit jobs, monitor their progress, download results, etc. It's an example of an UMBRELLA application that uses the Jersey REST Framework.

The demo doesn't actually keep track of users, so you can enter anything on the login screen. Whatever user information you enter will be sent to REST API in the cipres-eu headers. CIPRES creates an account, on the fly, for the "end user". The maximum runtime you can set is 1 min. See the restclient demo in action.

Usage Limits

CIPRES has the following per user limits:

CONCURRENT_LIMIT
The number of concurrent REST API requests.

XSEDE_SU_LIMIT
The number of XSEDE SUs, where 1 SU = 1 hour of computing time on one XSEDE CPU, per year, reset on July 1st.

OTHER_SU_LIMIT

The number of SUs on non-xsede resources such as UCSD's TSCC cluster, per year, reset on July 1st.

SUBMITTED_TODAY_LIMIT

The number of jobs a user may submit in a single day.

ACTIVE_LIMIT

The number of active jobs allowed, where an active job is any job that isn't fully completed (i.e. jobstatus.terminalStage is still false). This includes jobs that are queued, running, or awaiting cleanup.

When a request is rejected due to a usage limit, the http status will be 429 (Too Many Requests). The `error.code` will be 103, which is the CIPRES generic "UsageLimit" error code. The `error` will contain a nested `limitStatus` element which has `type` and `ceiling` fields.

```
&lt;error&gt;
    &lt;displayMessage&gt;Too many active jobs.  Limit is 1&lt;/displayMessage&gt;
    &lt;message&gt;org.ngbw.sdk.UsageLimitException: Too many active jobs.  Limit is 1&lt;/message&gt;
    &lt;code&gt;103&lt;/code&gt;
    &lt;limitStatus&gt;
        &lt;type&gt;active_limit&lt;/type&gt;
        &lt;ceiling&gt;1&lt;/ceiling&gt;
    &lt;/limitStatus&gt;
&lt;/error&gt;
```

Currently, the limits are: concurrent_limit=10, active_limit=50, other_su_limit=30,000 and xsede_su_limit=30,000. These limits can be modified for specific applications and users. If you have a problem with the default limits, please contact us to discuss your needs.

A future release of the REST API will

- Provide a way to programmatically determine the limits that are in effect.
- Provide a way to programmatically determine the number of SUs a user has consumed
- Increase the xsede_su_limit to 50,000 for users in the United States.

Appendix

Configure Tool Specific Parameters

It is impossible to explain job configuration in CIPRES without first explaining the basic method for command line generation. The code for creating command lines and configuring jobs evolved from the Pasteur Institute Software Environment (PISE). PISE is an XML-based standard designed to permit scalable generation of web forms that can be used to create Unix command lines. Each tool offered by CIPRES has a PISE XML document that describes the options supported by that tool. (Please see the Tool API section, for the definition of a CIPRES "tool").

In the CSG website, the PISE XML documents are used to create the browser-based forms that let a user configure a job. In the CRA, they define the fields that may be POST'd in a job submission. In both systems the PISE files are also used to validate job submissions and to create the command line, based on the user supplied fields.

We have already seen that a CRA job submission must include a `tool` and a primary `input.infile_`, and may include optional metadata fields. In this section of the guide we explain how to modify the default values in job submissions. In doing so, we explore the relationship between the command line options offered by a given program, the parameters in a tool's PISE XML files and the `input.` and `vparam.` fields you can use to configure a job submission. The two types of fields that are derived from the PISE XML files and are:

- **Input Files**: these field names have the form `input.`*`parameter_name_`*. Each such field corresponds to a <parameter> in the tool's PISE XML file, where the name of the parameter is *parameter_name* and the parameter's type is `InFile`. Every PISE file defines one special `InFile` parameter that is, by convention, named `infile`. This parameter has the attribute `isinput=1`, which means that it is the primary input, and must always be included in any run of this tool. Other InFile parameters allow you to submit optional files containing constraints, guide trees, etc. (as appropriate for the tool, and the particular analysis).
- **Visible Parameters**: these field names have the form `vparam.`*`parameter_name_`*. Each such field corresponds to a <parameter> in the tool's PISE XML file where the name of the parameter is *parameter_name* and the parameter's type is `Switch`, `String`, `Integer`, etc. These parameters are used to configure the command line and certain other aspects of a run, such as how long the job is allowed to run. They are called visible parameters, because in the CSG website, they correspond to textareas, radio buttons and other visible form controls.

Syntax Recap: Except for the `tool`, all field names are of the form *prefix.name* where allowed values for *prefix* are `metadata.` `input.` or `vparam.` All input and vparam field names have a trailing underscore.

Continuing with the job submission example used earlier in this guide, here's how Tom could submit a CLUSTALW job that uses a guidetree, produces phylip output and has a limited maximum run time:

```
$ curl -u tom:$PASSWORD \
    -H cipres-appkey:$KEY \
    $URL/job/tom \
    -F tool=CLUSTALW \
    -F metadata.clientJobId=102   \
    -F metadata.statusEmail=true \
    -F input.infile_=@./sample1_in.fasta \
    -F input.usetree_@./guidetree.dnd \
    -F vparam.runtime_=1 \
    -F vparam.phylip_alig_=1
```

The `tool`, `metadata` and `input.infile_` fields were explained earlier. The new fields, that Tom just added are:

`-F input.usetree_=@/.guidetree.dnd`

`input.usetree` causes CIPRES to add a `-usetree` option to the CLUSTALW command line. This tells CLUSTALW to use the specified file as a guide tree for the alignment. `usetree` is the name of a parameter of type `InFile`, in CLUSTALW 's PISE XML document, clustalw.xml. At present, the translation from a program's command line option to a field in the REST API (e.g. from clustalw's `-usetree` option to the API's `input.usetree_`) can only be made by inspecting the tool's PISE document. PISE Example 2, An Additional Input File, below, takes a closer look at the `usetree` parameter in clustalw.xml.

`-F vparam.runtime_=1`

Configures a maximum run time of 1 hour, using the `runtime` parameter, found in clustalw.xml. Typically, vparams are specific to a particular tool, but, by convention, `runtime` is found in every tool's PISE XML file. If left unspecified, maximum run time would have been set to the default value specified in the PISE XML file, typically 0.5 h.

When you look at the PISE XML for some tools, you may not see a <parameter> named `runtime`, but will see an entity that includes a shared definition of the parameter, like this:

```
<ENTITY runtime SYSTEM "http://www.phylo.org/dev/rami/XMLDIR/triton_run_time.xml">
```

The exact contents of `triton_run_time.xml` are shown in Example 4, Shared Definition of Runtime Parameter.

`-F vparam.phylip_alig_=1`

The `phylip_alig` parameter, set to `1` means that CLUSTALW should be run with the `-output=PHYLIP` command line option. This is defined in clustalw.xml, as explained in PISE Example 1 below.

Note: In general, only parameters that differ from the defaults specified in the tool's PISE file, need to be specified in the job submission.

More about PISE XML

Since the PISE XML files determine the set of inputs and vparams that can be used to configure a run of a given tool, and form the basis for generating command lines, knowing a bit about PISE XML is essential to using the CRA. We provide a quick introduction to the PISE XML format here.

For complex phylogenetic programs, there is often significant interdependence of parameters. That is, some options are relevant only if others are selected, some combinations may give non-sensical results, and so forth. The PISE XML documents are rich, and embody all of the information required to create successful, meaningful command lines. Where possible, they also prevent creation of incorrect commands that would cause an immediate error. Thus, the PISE XML documents are the definitive reference for configuring CRA job submissions.

Each tool's PISE XML document is esentially a collection of `parameter` elements. Most parameter elements correspond to fields you can use to configure a job submission. It is easiest to explain the XML format through a set of examples:

Example 1. A basic PISE Parameter

A parameter usually defines a single command line flag or input file. Here is an example from clustalw.xml.

```
&lt;parameter issimple="1" type="Switch"&gt;
    &lt;name&gt;phylip_alig&lt;/name&gt;
    &lt;attributes&gt;
        &lt;prompt&gt;Phylip alignment output format (-output)&lt;/prompt&gt;
        &lt;format&gt;
            &lt;language&gt;perl&lt;/language&gt;
            &lt;code&gt; ($value)?" -output=PHYLIP":""&lt;/code&gt;
        &lt;/format&gt;
        &lt;vdef&gt;
            &lt;value&gt;0&lt;/value&gt;
        &lt;/vdef&gt;
        &lt;group&gt;2&lt;/group&gt;
    &lt;/attributes&gt;
&lt;/parameter&gt;
```

Each `parameter` has a name (in this case, "phylip_alig") and a type. In this case, the type is "Switch", which means the allowed values are "0" and "1". To use a parameter from the PISE XML in a CRA job submission, prefix the parameter name with "vparam" (or with "input", if the type=InFile) and add a trailing underscore. So to use this parameter you'd send `vparam.phylip_alig_="1"` or `vparam.phylip_alig_="0"`. If you send this parameter, regardless of whether you set it to 0 or 1, the perl expression in the `format` element will be run. Thus if you set it to 1, "-output=PHYLIP" will be added to the command line, and if you set it to 0, nothing is added to the command line here because `($value)` will evaluate to false. The effect of including `vparam.phylip_alig_=0` and not including any setting for phylip_alig, is the same.

CIPRES PISE XML documents supply default values using the `vdef` element, so it is typically only necessary to send fields where the default value is not correct for the run, even in cases where the parameter has an `ismandatory` attribute.

Note that the PISE XML documents also contain all the information needed to generate a web form. Many elements, such as `prompt`, `label`, `comment`, `issimple` etc. provide information necessary for web form generation, but are not relevant to the CRA.

Note in particular: PISE parameters with the `ishidden=1` attribute, and those of type `Results` may not be sent via the REST API.

PISE Parameter Types

PISE <parameter> elements will have one of the following types:

- `InFile` - an input file. Every tool has one input file that is mandatory. This is indicated with the attribute `isinput=1` and by convention is named "infile". Other input files are optional, or are required only when certain other parameters are set as specified in `precond` or `ctrl` elements.
- `Excl` is a single choice list, the selected value must be in the set of `value` elements given in the `vlist` or `flist` element.
- `List` is a multiple choice list. Allowed values are given in `value` elements. To send multiple values, use multiple form fields with the same name, e.g `-F vparam.hgapresidues_=G -F vparam.hgapresidues_=A`.
- `Switch` must be either "0" or "1".
- Self explanatory types like `Integer, Float, String`
- `Results` specify which files will be returned when the job completes. Users have no direct control over the naming of output files.

PISE Parameter Elements

- `name` is the name of the parameter. Prefix it with "vparam." or "input." and suffix it with an underscore to use in the CRA.
- `vdef` gives the parameter's default value, if any
- `ctrl` elements set constraints on values. `ctrls`, like preconds, have perl expressions. If any of a parameter's `ctrl` elements evaluates to true, the job submission will not pass validation.
- `precond` elements determine whether the parameter is enabled or disabled. When a parameter is disabled, you may not include a value for it in the job submission. A parameter is enabled when all of its precond elements evaluate to true.
- `format` - a perl expression that creates part of the command line.
- `paramfile` - when a `paramfile` is present, CIPRES creates the file in the job's working directory and sends the output from the corresponding `format` to the named paramfile instead of the command line. The user has no control over the creation or naming of these files, they are created and submitted on the back end.

Note: If you don't include a particular `parameter` in your job submission, and that parameter has a default value (i.e. a `vdef` element), and the default value doesn't conflict wth the preconds of any parameters you sent, then the CRA automatically adds that parameter, set to its default value, to your submission. On the other hand, if the preconds do conflict, the parameter is not added. When a parameter isn't present in a job submission, it will be skipped when the PISE XML file is processed. This means that its `ctrl` and `format` code snippets won't be evaluated.

Example 2. An Additional Input File

This is an example of a parameter that specifies an additional input file. To include it in your job submission you would use `vparam.input.usetree_`, as we did in the example job submission shown earlier. When CIPRES receives the input file contents, it disregards your original filename for the data (i.e guidetree.dnd, in the job submission example), and stores the data in a file named "usetree.dnd" in the job's working directory. The filename CIPRES uses is specified by the parameter's `filenames` element. When the `format` code is executed, it adds "-usetree=usetree.dnd" to clustalw's command line.

```
&lt;parameter type="InFile"&gt;
        &lt;name&gt;usetree&lt;/name&gt;
        &lt;attributes&gt;
                &lt;prompt&gt;File for old guide tree (-usetree)&lt;/prompt&gt;
                &lt;format&gt;
                        &lt;language&gt;perl&lt;/language&gt;
                        &lt;code&gt; " -usetree=usetree.dnd"&lt;/code&gt;
                &lt;/format&gt;
                &lt;group&gt;2&lt;/group&gt;
                &lt;comment&gt;&lt;value&gt;You can give a previously computed tree (.dnd file) - on the same data&lt;/value&gt;&lt
;/comment&gt;
                &lt;precond&gt;
                        &lt;language&gt;perl&lt;/language&gt;
                        &lt;code&gt;($actions =~ /align/ )&lt;/code&gt;
                &lt;/precond&gt;
                &lt;filenames&gt;usetree.dnd&lt;/filenames&gt;
        &lt;/attributes&gt;
&lt;/parameter&gt;
```

Example 3. A Parameter That Builds A Configuration File.

The following example is for GARLI v.2.0, taken from garli2_tgb.xml. It adds a setting to a configuration file, garli.conf, that garli will read. The `paramfile` element is what tells CIPRES to direct the output of the `format` element to a file named garli.conf instead of to the command line. Each time CIPRES processes a parameter with a `paramfile` element, it either creates the specified file in the job's working directory (if it doesn't already exist) or adds text to it.

This parameter defines a choice list named `d_statefrequencies`. The allowable values are given by the `vlist.value` elements and are `equal`, `empirical`, `estimate`, and `fixed`. The default value is `estimate`. The `precond` for this parameter dictates that it is only allowed when a second parameter, `datatype_value`, has the value `nucleotide`. The output of the perl code in the `format` element will be directed to garli.conf, thereby adding a "statefrequencies" setting to the file.

```
&lt;parameter type="Excl"  ismandatory="1"&gt;
    &lt;name&gt;d_statefrequencies&lt;/name&gt;
    &lt;attributes&gt;
        &lt;paramfile&gt;garli.conf&lt;/paramfile&gt;
        &lt;prompt&gt;Base Frequencies (statefrequencies)&lt;/prompt&gt;
        &lt;precond&gt;
            &lt;language&gt;perl&lt;/language&gt;
            &lt;code&gt;$datatype_value eq "nucleotide"&lt;/code&gt;
```

```
        &lt;/precond&gt;
        &lt;format&gt;
            &lt;language&gt;perl&lt;/language&gt;
            &lt;code&gt;"statefrequencies = $value\\n" &lt;/code&gt;
        &lt;/format&gt;
        &lt;vlist&gt;
            &lt;value&gt;equal&lt;/value&gt;
            &lt;label&gt;equal&lt;/label&gt;
            &lt;value&gt;empirical&lt;/value&gt;
            &lt;label&gt;empirical&lt;/label&gt;
            &lt;value&gt;estimate&lt;/value&gt;
            &lt;label&gt;estimate&lt;/label&gt;
            &lt;value&gt;fixed&lt;/value&gt;
            &lt;label&gt;fixed&lt;/label&gt;
        &lt;/vlist&gt;
        &lt;vdef&gt;
            &lt;value&gt;estimate&lt;/value&gt;
        &lt;/vdef&gt;
        &lt;paramfile&gt;garli.conf&lt;/paramfile&gt;
        &lt;group&gt;2&lt;/group&gt;
    &lt;/attributes&gt;
&lt;/parameter&gt;
```

Example 4. Shared Definition of Runtime Parameter.

Many tools include a file named `triton_run_time.xml` that contains a definition of the `runtime` parameter. It looks like this:

```
&lt;parameter type="Float" issimple="1" ismandatory="1"&gt;
        &lt;name&gt;runtime&lt;/name&gt;
        &lt;attributes&gt;
                &lt;group&gt;1&lt;/group&gt;
                &lt;paramfile&gt;scheduler.conf&lt;/paramfile&gt;
                &lt;prompt&gt;Maximum Hours to Run (click here for help setting this correctly)&lt;/prompt&gt;
                &lt;vdef&gt;&lt;value&gt;1.0&lt;/value&gt;&lt;/vdef&gt;
                &lt;comment&gt;
                        &lt;value&gt;
                                Estimate the maximum time your job will need to run (up to 72 hrs).
                                Your job will be killed if it doesn't finish within the time you specify, however jobs with shorter
                                maximum run times are often scheduled sooner than longer jobs.
                        &lt;/value&gt;
                &lt;/comment&gt;
                &lt;ctrls&gt;
                        &lt;ctrl&gt;
                                &lt;message&gt;Maximum Hours to Run must be between 0.1 - 72.0.&lt;/message&gt;
                                &lt;language&gt;perl&lt;/language&gt;
                                &lt;code&gt;$runtime &amp;lt; 0.1 || $runtime &amp;gt; 72.0&lt;/code&gt;
                        &lt;/ctrl&gt;
                &lt;/ctrls&gt;
                &lt;format&gt;
                        &lt;language&gt;perl&lt;/language&gt;
                        &lt;code&gt;"runhours=$value\\n"&lt;/code&gt;
                &lt;/format&gt;
        &lt;/attributes&gt;
&lt;/parameter&gt;
```

This defines a field named `vparam.runtime_`. The `ctrl` element says that you are allowed to set values between .1 hrs and 72.0 hrs. The default value is 1 hr. This definition is used by many tools that run on the `TSCC` supercomputer. Tools that run on XSEDE resources define `runtime` differently, usually allowing up to 168 hours, with a default of .5 hrs.

This parameter works by writing a line that looks like "runhours=1" (for example) to a file named scheduler.conf. CIPRES uses the information in scheduler.conf to limit the runtime as specified.

Strategies for using PISE xml files

Some tools, notably BEAST and MrBayes allow or require users to configure most options in the main input file. This can greatly simplify the development of REST submissions since there will be little to configure via the API. Others, such as GARLI and RAxML have PISE XML files that contain a significant numbers of parameters, and require familiarity with the defaults and potential interaction between parameters. At present, there are basically two strategies that can be useful in learning to configure a job.

1. Download and examine the PISE XML file of interest, and identify the elements that control the command line flags you are interested in using.

The utility of this strategy will depend on the complexity of the interface (in terms of preconds and ctrls) and how many non-default values you need to use. This strategy should work fine for simpler PISE XML documents.
2. Display the functioning parameter web form in the CSG website, and note names of the fields, the default values, the interdependency of the fields, and the logical organization of the form. This information can help identify the parameters of interest within the PISE XML document. To find the PISE `parameter` names, you'll have to create, configure and save a task in the CSG, then use the links on the Task Details or Tasks (list) pages to view the inputs and parameters.

In our experience, these two techniques are optimal when used in combination, according to the complexity of the interface, and the familiarity of the user with the code in question and the PISE XML document.

It may be comforting to know that many commonly used job configurations require only one or two parameter fields to be set manually.

Our plan is to develop automatically generated documentation from the XML files in the near future. Please contact us if you have questions about how to configure specific tools.

How to Make a Test Run

Once you have a job submission ready, you can validate it by POST'ing it to `$URL/job/validate/username` instead of POST'ing it to `$URL/job/username`. CIPRES will validate the parameters but won't actually submit the job. If your submission is fine CIPRES will return a `jobstatus` object with a `commandline` element that shows the Linux command line that would be run if the job were submitted. On the other hand if there are errors in the submission, CIPRES will return an error object that explains the problems.

Umbrella Application Examples

Umbrella applications can use the commands in this guide, with additional request headers that identify the end user. Behind the scenes, CIPRES creates an account for the user with a username of the form *application_name.cipres_eu_header* and it is this qualified username that goes in the URLs.

Request Headers

All requests to the job API use request headers. The required headers depend on the type of authentication the application uses, as noted below.

Basic authentication credentials	ALL	DIRECT applications send the user's CIPRES REST username and password. UMBRELLA applications send the username and password of the person who registered the application. See Authentication.
cipres-appkey	ALL	Application ID generated by CIPRES when you registered the application. It can be changed later if necessary.
cipres-eu	UMBRELLA	Uniquely identifies the user within your application. Up to 200 characters. Single quotes are not allowed within the name.
cipres-eu-email	UMBRELLA	End user's email address. Up to 200 characters. You can't have 2 users with the same email address.
cipres-eu-institution	UMBRELLA	End user's home institution, if any.
cipres-eu-country	UMBRELLA	Two letter ISO 3166 country code for the end user's institution. Optional, but can get the user higher SU limits.

List Jobs

For example, suppose your username is `mary` and you're integrating an existing web application with the CIPRES REST API. You've registered the application with the name `phylobank` and set the authentication method to UMBRELLA.

Now suppose a user named `harry` logs into your application and your application needs to get a list of jobs that harry has submitted to CIPRES. First, you go to your database or user management component and retrieve harry's email address, institutional affiliation, and optional ISO 3166 2 letter country code. Now you're ready to issue this curl command (or the equivalent statement in the language you're using):

```
$ curl -i -u mary:password \
    -H cipres-appkey:$KEY \
    -H cipres-eu:harry \
    -H cipres-eu-email:harry@ucsddd.edu \
    -H cipres-eu-institution:UCSD \
    -H cipres-eu-country:US \
    $URL/job/phylobank.harry
```

Notice that although the value of the `cipres-eu` header is `harry`, in the URL, you must use `phylobank.harry`.

Submit a Job

Here you submit a basic clustalw job for harry and get back a jobstatus object.

```
$ curl -u mary:password \
    -H cipres-appkey:$KEY \
    -H cipres-eu:harry \
    -H cipres-eu-email:harry@ucsddd.edu \
    -H cipres-eu-institution:UCSD \
    -H cipres-eu-country:US \
    $URL/job/phylobank.harry\
    -F tool=CLUSTALW \
    -F input.infile_=@./sample1_in.fasta \

&lt;?xml version="1.0" encoding="UTF-8" standalone="yes"?&gt;
&lt;jobstatus&gt;
    &lt;selfUri&gt;
        &lt;url&gt;$URL/cipresrest/v1/job/phylobank.harry/NGBW-JOB-CLUSTALW-CB8D053F9033487E9B4F9BAF8A3AA47A&lt;/url&gt;
        &lt;rel&gt;jobstatus&lt;/rel&gt;
        &lt;title&gt;NGBW-JOB-CLUSTALW-CB8D053F9033487E9B4F9BAF8A3AA47A&lt;/title&gt;
    &lt;/selfUri&gt;
    &lt;jobHandle&gt;NGBW-JOB-CLUSTALW-CB8D053F9033487E9B4F9BAF8A3AA47A&lt;/jobHandle&gt;
    &lt;jobStage&gt;QUEUE&lt;/jobStage&gt;
    &lt;terminalStage&gt;false&lt;/terminalStage&gt;
    &lt;failed&gt;false&lt;/failed&gt;
    &lt;metadata&gt;
        &lt;entry&gt;
            &lt;key&gt;clientJobId&lt;/key&gt;
            &lt;value&gt;010007AQ&lt;/value&gt;
        &lt;/entry&gt;
    &lt;/metadata&gt;
    &lt;dateSubmitted&gt;2014-09-12T12:36:31-07:00&lt;/dateSubmitted&gt;
    &lt;resultsUri&gt;
        &lt;url&gt;$URL/cipresrest/v1/job/phylobank.harry/NGBW-JOB-CLUSTALW-CB8D053F9033487E9B4F9BAF8A3AA47A/output&lt;/url&gt;
        &lt;rel&gt;results&lt;/rel&gt;
        &lt;title&gt;Job Results&lt;/title&gt;
    &lt;/resultsUri&gt;
    &lt;workingDirUri&gt;
        &lt;url&gt;$URL/cipresrest/v1/job/phylobank.harry/NGBW-JOB-CLUSTALW-CB8D053F9033487E9B4F9BAF8A3AA47A/workingdir&lt;/url&gt;
        &lt;rel&gt;workingdir&lt;/rel&gt;
        &lt;title&gt;Job Working Directory&lt;/title&gt;
    &lt;/workingDirUri&gt;
    &lt;messages&gt;
        &lt;message&gt;
            &lt;timestamp&gt;2014-09-12T12:36:31-07:00&lt;/timestamp&gt;
            &lt;stage&gt;QUEUE&lt;/stage&gt;
            &lt;text&gt;Added to cipres run queue.&lt;/text&gt;
        &lt;/message&gt;
    &lt;/messages&gt;
&lt;/jobstatus&gt;
```

Check Job Status

You can check the status of a single job, using the `jobstatus.selfUri.url` that was returned when the job was submitted, like this:

```
$ curl -u mary:mary \
    -H cipres-appkey:$KEY \
    -H cipres-eu:harry \
    -H cipres-eu-email:harry@ucsddd.edu \
    -H cipres-eu-institution:UCSD \
    -H cipres-eu-country:US \
    $URL/job/phylobank.harry/NGBW-JOB-CLUSTALW-CB8D053F9033487E9B4F9BAF8A3AA47A
```

or you can get the status of multiple jobs, submitted on behalf of multiple users with a single GET of `$URL/job`. Indicate which jobs you're interested in with a query parameters named `jh` (for "job handle"). Use separate `jh` parameters for each job. With this request, the `cipres-appkey` header is required, but end user headers are not. For example:

```
$ curl -u mary:mary \
```

```
    -H cipres-appkey:$KEY \
    $URL/job/?jh=NGBW-JOB-CLUSTALW-CB8D053F9033487E9B4F9BAF8A3AA47A\&amp;jh=NGBW-JOB-CLUSTALW-553D534D355C4631BBDCF217BB792A01
```

If you're using curl in a typical unix shell, you must place a backslash before the `&` that separates the query parameters to escape it from interpretation by the shell.

Other Operations

The other things you may need to do are 1) retrieve files from a job's working directory while it's running, 2) retrieve final results once a job has finished, 3) cancel and/or delete a job. The DIRECT application examples in this guide are applicable to UMBRELLA applications too. Just remember to add the appropriate CIPRES end user headers and prefix the username in the URL with the the application name and a period.