PROJECT_NAME Documentation

Release 0.1.0

AUTHOR

Mar 13, 2020

Contents

JBlast - A BLAST service for JBConnect and JBrowse

JBlast (jblast-jbconnect-hook) is JBConnect hook module. It contains both server-side integration with JBConnect and JBrowse that enables Blast analysis that is tightly integrated with JBrowse. JBlast can execute stand-alone NCBI blast commands directly, or it can be configured to use Galaxy for workflow processing. Through the JBrowse user interface the user can choose to submit an existing feature as a blast query or highlight a region to blast. The user can monitor blast execution processing through JBConnect's job queue and the blast search results will appear directly as an inserted track in the track selector.

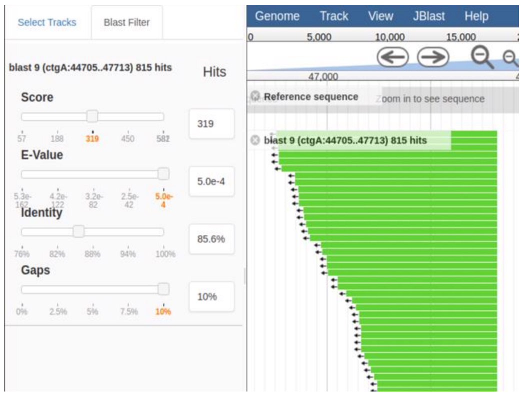


Fig. 1: Filter Panel with results mapped to query space

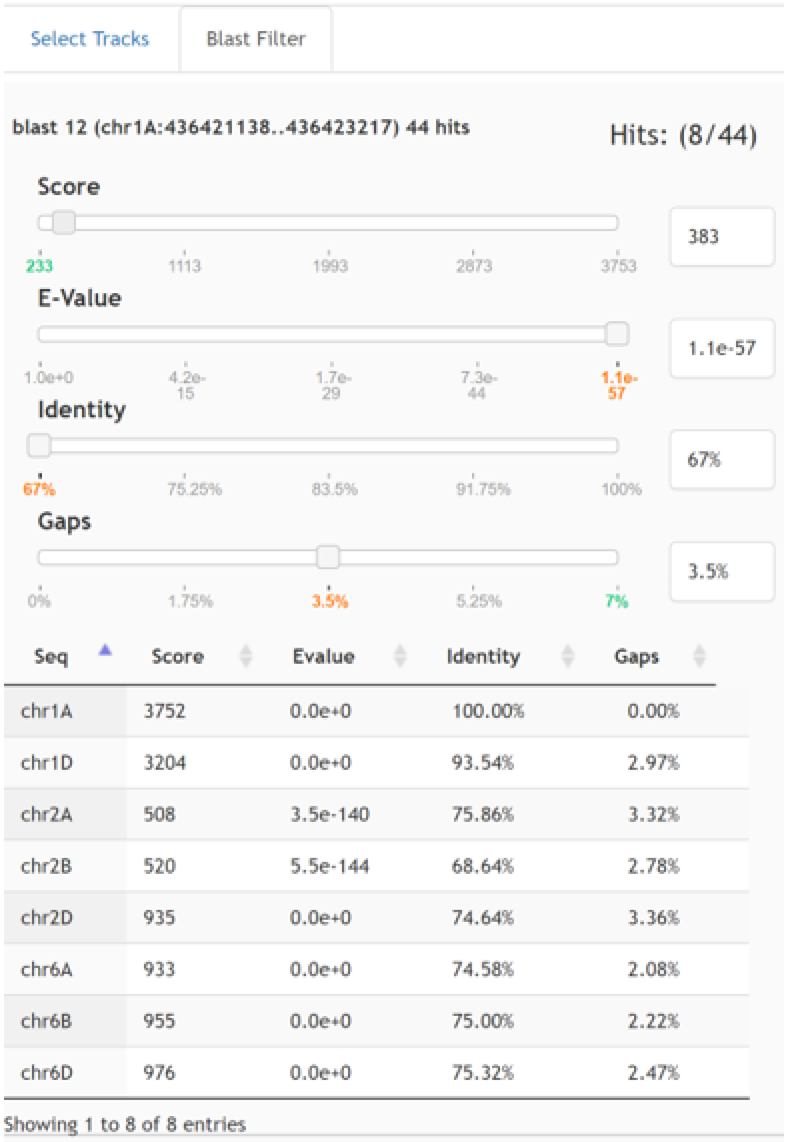
JBlast User Quick Tutorial

JBlast provides the following functionality:

Can leverage Galaxy Server for Blast Analysis or use stand-alone NCBI Blast tools
Basic workflow abstraction and Monitoring

The Client-Side JBlast plugin intgration with JBrowse:

Submit region or existing feature for blast search
Inject result tracks into existing configuration with persistence.
Dynamically filter Blast results and save results.
Extended feature details with blast results



blast 12 (chr1A:436421138..436423217) 44 hits

Hits: (8/44)

Score: 383

E-Value: 1.1e-57

Identity: 67%

Gaps: 3.5%

Seq	Score	Evalue	Identity	Gaps
chr1A	3752	0.0e+0	100.00%	0.00%
chr1D	3204	0.0e+0	93.54%	2.97%
chr2A	508	3.5e-140	75.86%	3.32%
chr2B	520	5.5e-144	68.64%	2.78%
chr2D	935	0.0e+0	74.64%	3.36%
chr6A	933	0.0e+0	74.58%	2.08%
chr6B	955	0.0e+0	75.00%	2.22%
chr6D	976	0.0e+0	75.32%	2.47%

Showing 1 to 8 of 8 entries

Fig. 2: Filter Panel where feature mapping mode is in target space.

CHAPTER 1

Quick Start

Instructions for installing JBlast with stand-alone NCBI Blast tools (non-Galaxy).

To setup Galaxy integration, see **'Configure JBConnect with Galaxy'_**

(Since JBConnect is generally intended to be a companion of JBrowse. JBrowse may also be installed in a separate directory. (See jbs-separate-dir.)

To integrate with Galaxy, see *Setup Galaxy*.

1.1 Pre-Install

JBlast requires redis as a pre-requisite, which is only used by the queue framework (kue). JBConnect depends on Sails.js.

Install *redis* and *sails*

```
yum install redis
redis-server
npm install -g sails@1.0.2
```

1.2 Install

Install the JBConnect and JBrowse. jb_setup.js ensures the sample data is loaded.

```
# install jbconnect
git clone http://github.com/gmod/jbconnect
cd jbconnect
npm install

# install blast tools and sample data
npm install enuggetry/blast-ncbi-tools enuggetry/faux-blastdb
```

(continues on next page)

(continued from previous page)

```
# pull in NCBI blast executables
./utils/blast_getBlastUtils.js 2.8.1

# install jblast
npm install gmod/jblast-jbconnect-hook

# install jbrowse & setup jbrowse demo
npm install @gmod/jbrowse@1.15.1
patch node_modules/@gmod/jbrowse/setup.sh fix_jbrowse_setup.patch
./utils/jb_setup.js
```

The patch operation is needed to make JBrowse 1.15.1 setup.sh run properly. If JBrowse is installed in another location, the patch should be run before setup.sh.

1.3 Run

Launch the server.

```
sails lift
```

From a web browser, access the application (default login: juser/password).

```
http://localhost:1337/jbrowse
```

1.4 Contents

1.4.1 Features

Integrated GUI

JBlast extends JBrowse with a number of GUI elements.

JBlast Brief Tutorial

3 Ways to BLAST

There are three ways to select a query sequence to BLAST:

1. BLAST an arbitrary query sequence (from JBlast Menu)
2. BLAST a highlighted region (from JBlast Menu or click the highlight button)
3. BLAST an existing feature from another track (from the feature's Detail dialog).

BLAST a DNA sequence

This feature provides a way to BLAST an arbitrary sequence.

When logged in, the JBlast menu appears next to the View menu on the menu bar.

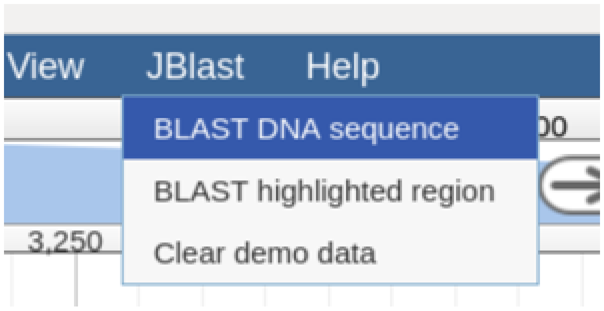
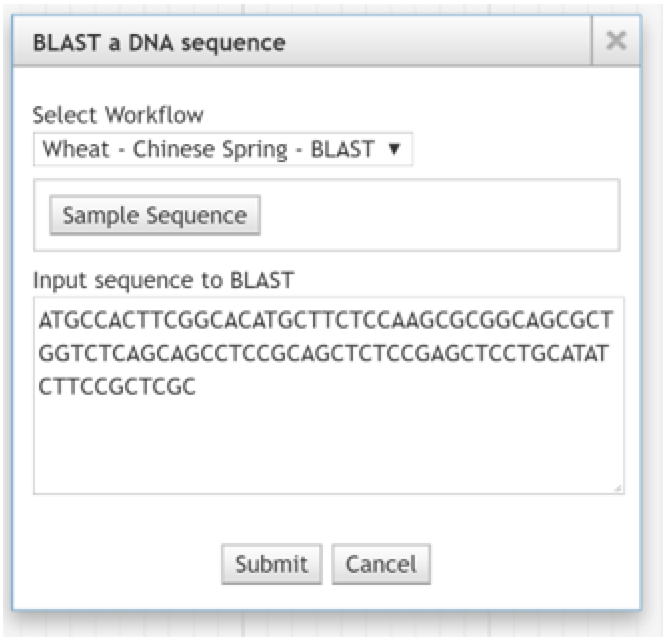


Fig. 1: JBlast Menu



Wheat - Chinese Spring - BLAST

Input sequence to BLAST

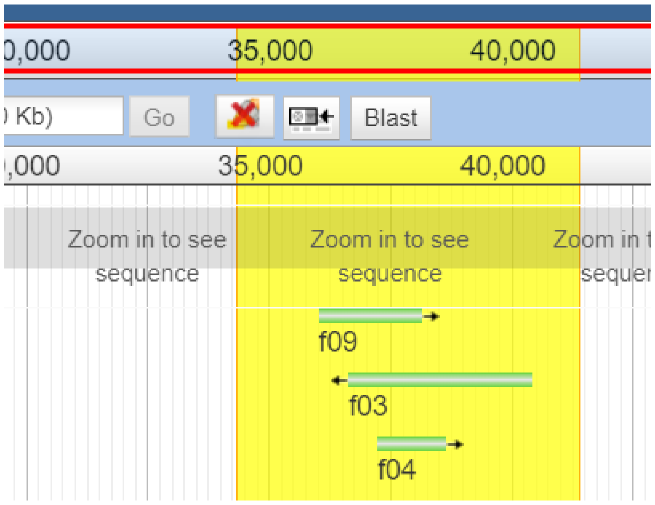
```
ATGCCACTTCGGCACATGCTTCTCCAAGCGCGGCAGCGCT
GGTCTCAGCAGCCTCCGCAGCTCTCCGAGCTCCTGCATAT
CTTCCGCTCGC
```

Fig. 2: BLAST a DNA sequence dialog box

Choose a workflow that will perform the BLAST operation. In the field *Input Sequence to BLAST*, Paste (Ctrl-V) a sequence. The Sample Sequence button may or may not be there depending on the configuration (See *Sample Sequence Button*).

BLAST a region

Selecting an arbitrary region to BLAST. This is done with the highlight feature of JBrowse. the highlight button on the toolbar, when pressed, will allow you to select an arbitrary region to highlight. After highlighting, right click the highlighted region (where there is no track).



This can also be accessed from the JBlast menu.

BLAST an existing feature

Select a feature from an existing track.

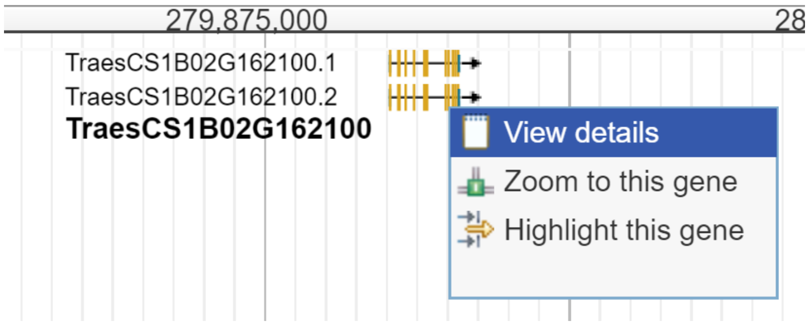
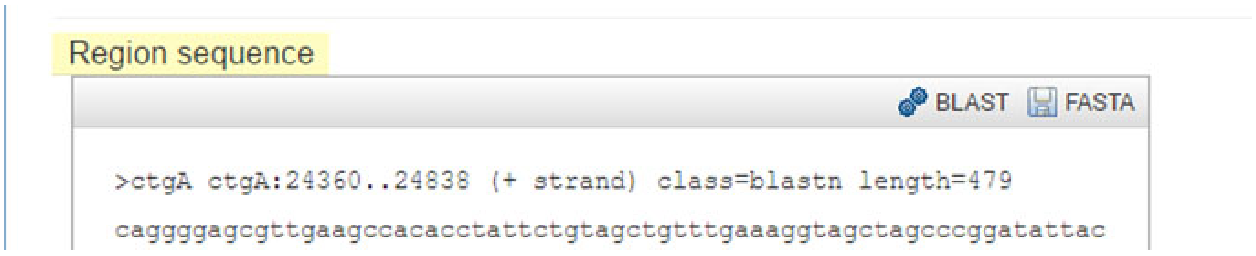


Fig. 3: Open the Details dialog box for the selected feature.

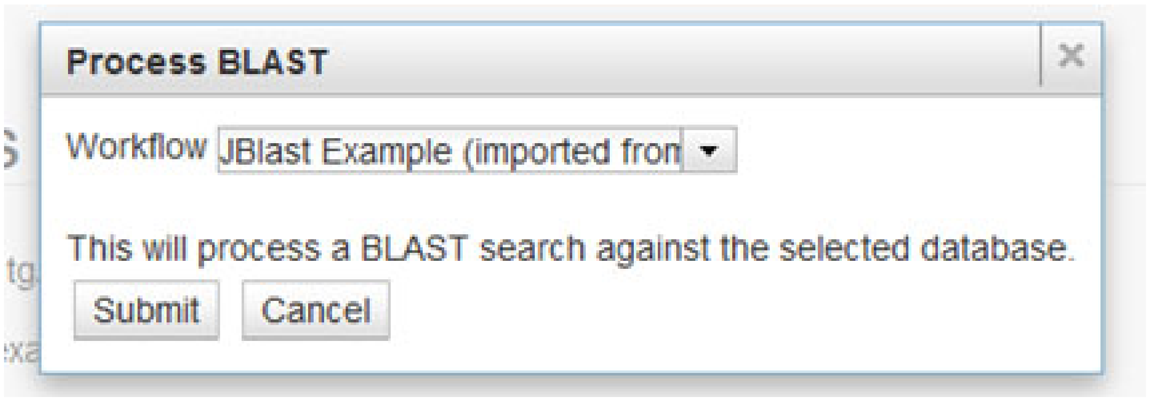


```
>ctgA ctgA:24360..24838 (+ strand) class=blastn length=479
caggggagcgttgaagccacacctattctgtagctgtttgaaaggtagctagcccggatattac
```

Fig. 4: Click the BLAST button in the feature Details dialog box.

Select Workflow

When a region is selected, this dialog box will appear. Choose from the list of workflows to be executed and click Submit. This will submit the selected region for processing using the selected workflow.



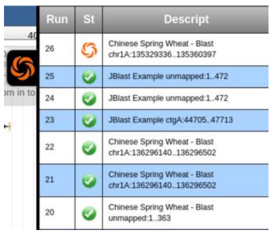
Process BLAST

Workflow: JBlast Example (imported fron

This will process a BLAST search against the selected database.

Job Queue Panel

The job queue side panel is revealed by clicking the Jobs tab on the upper right of JBrowse screen. This contains list of executing jobs, in our case, for processing workflows. It tells the current state of each job and whether completed jobs are completed or errored.

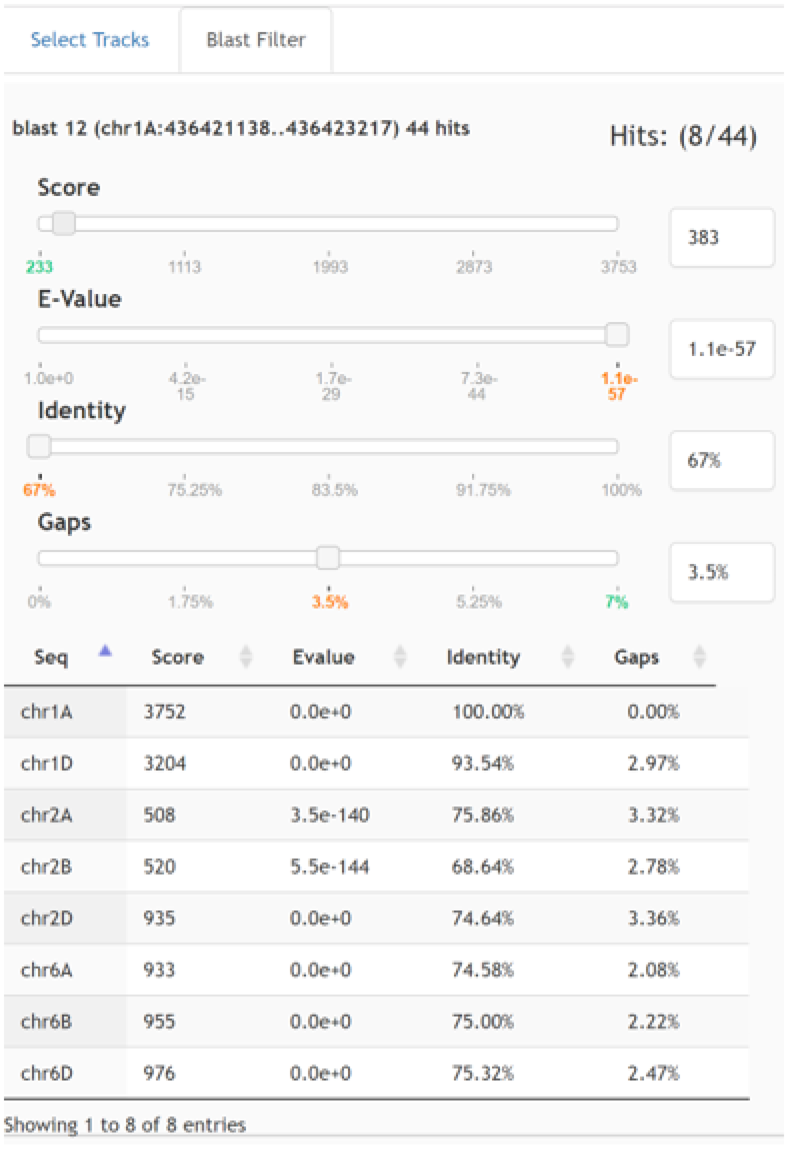


Run	St	Descript
26		Chinese Spring Wheat - Blast chr1A:135329336..135360397
25	✓	JBlast Example unmapped:1..472
24	✓	JBlast Example unmapped:1..472
23	✓	JBlast Example ctgA:44705..47713
22	✓	Chinese Spring Wheat - Blast chr1A:136296140..136296502
21	✓	Chinese Spring Wheat - Blast chr1A:136296140..136296502
20	✓	Chinese Spring Wheat - Blast unmapped:1..363

Filter Panel

The filter panel consists of filter sliders and the result table.

When a blast result track is selected with the track selector, the BLAST Filter panel will appear asa tab next to the Select Tracks tab when the result track is in focus. Filter Sliders (top are available for filtering by score, e-value, identity, and gaps. As the slider positions are moved, the filtered result track will be updated reflecting the filtered hits.



blast 12 (chr1A:436421138..436423217) 44 hits

Hits: (8/44)

Score: 383

E-Value: 1.1e-57

Identity: 67%

Gaps: 3.5%

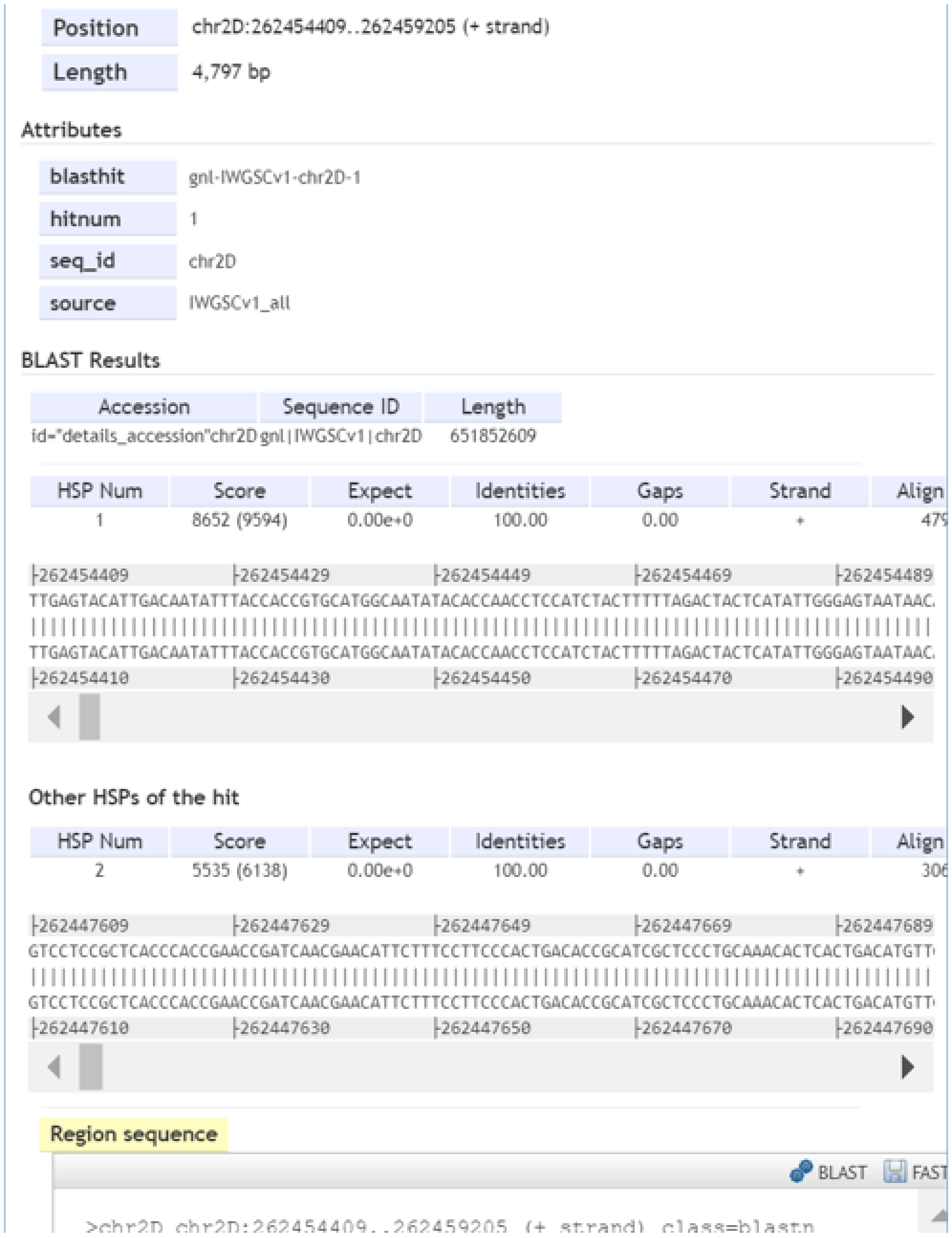
Seq	Score	Evalue	Identity	Gaps
chr1A	3752	0.0e+0	100.00%	0.00%
chr1D	3204	0.0e+0	93.54%	2.97%
chr2A	508	3.5e-140	75.86%	3.32%
chr2B	520	5.5e-144	68.64%	2.78%
chr2D	935	0.0e+0	74.64%	3.36%
chr6A	933	0.0e+0	74.58%	2.08%
chr6B	955	0.0e+0	75.00%	2.22%
chr6D	976	0.0e+0	75.32%	2.47%

Showing 1 to 8 of 8 entries

The Result Table (bottom) which shows the filtered results and allows the user to click the row to jump to the selected location. Note: the Result Table only appears if the *featureMapping='hit'* (see *featureMapping flag*)

View Feature Details

When a blast hit feature is selected, it's feature details will contain information about the blast hit and organism information, accession link, etc.



Position	chr2D:262454409..262459205 (+ strand)
Length	4,797 bp

Attributes

blasthit	gnl-IWGSCv1-chr2D-1
hitnum	1
seq_id	chr2D
source	IWGSCv1_all

BLAST Results

Accession	Sequence ID	Length
id="details_accession"chr2D	gnl|IWGSCv1|chr2D	651852609

HSP Num	Score	Expect	Identities	Gaps	Strand	Align
1	8652 (9594)	0.00e+0	100.00	0.00	+	479

```
├262454409          ├262454429          ├262454449          ├262454469          ├262454489
TTGAGTACATTGACAATATTTACCACCGTGCATGGCAATATACACCAACCTCCATCTACTTTTTAGACTACTCATATTGGGAGTAATAAC
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
TTGAGTACATTGACAATATTTACCACCGTGCATGGCAATATACACCAACCTCCATCTACTTTTTAGACTACTCATATTGGGAGTAATAAC
├262454410          ├262454430          ├262454450          ├262454470          ├262454490
```

Other HSPs of the hit

HSP Num	Score	Expect	Identities	Gaps	Strand	Align
2	5535 (6138)	0.00e+0	100.00	0.00	+	306

```
├262447609          ├262447629          ├262447649          ├262447669          ├262447689
GTCCTCCGCTCACCCACCGAACCGATCAACGAACATTCTTTCCTTCCCACTGACACCGCATCGCTCCCTGCAAACACTCACTGACATGTT
||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
GTCCTCCGCTCACCCACCGAACCGATCAACGAACATTCTTTCCTTCCCACTGACACCGCATCGCTCCCTGCAAACACTCACTGACATGTT
├262447610          ├262447630          ├262447650          ├262447670          ├262447690
```

Region sequence

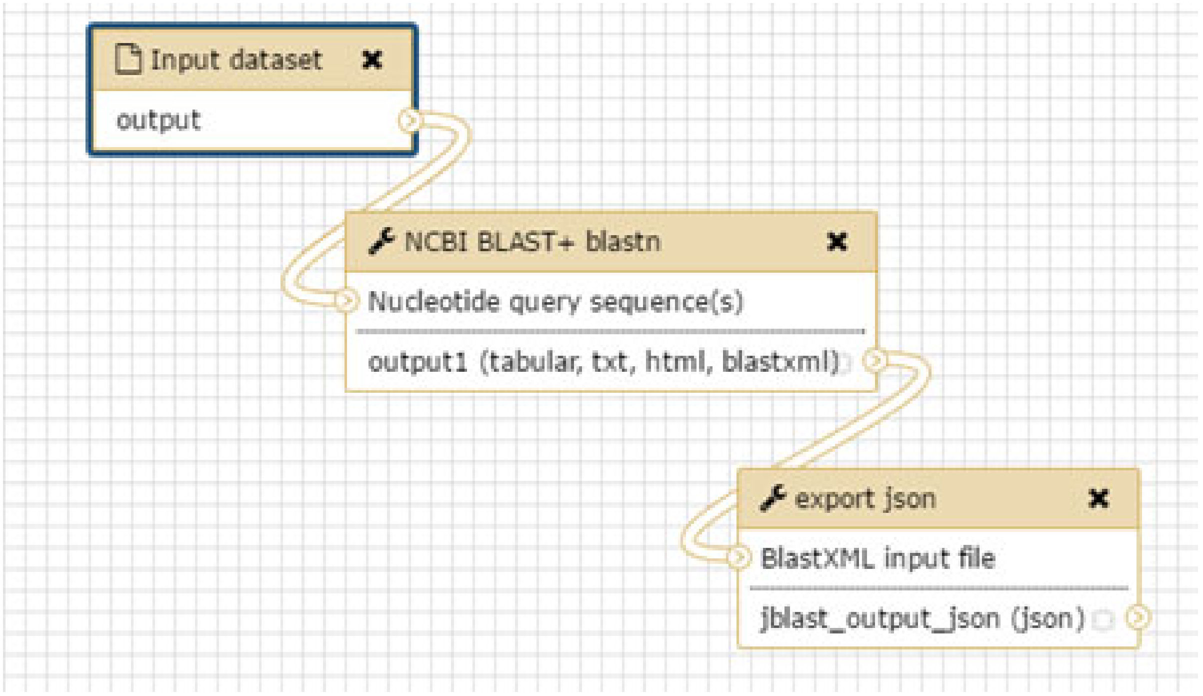
```
>chr2D chr2D:262454409..262459205 (+ strand) class=blastn
```

Galaxy

The following show how JBlast affects the Galaxy interface.

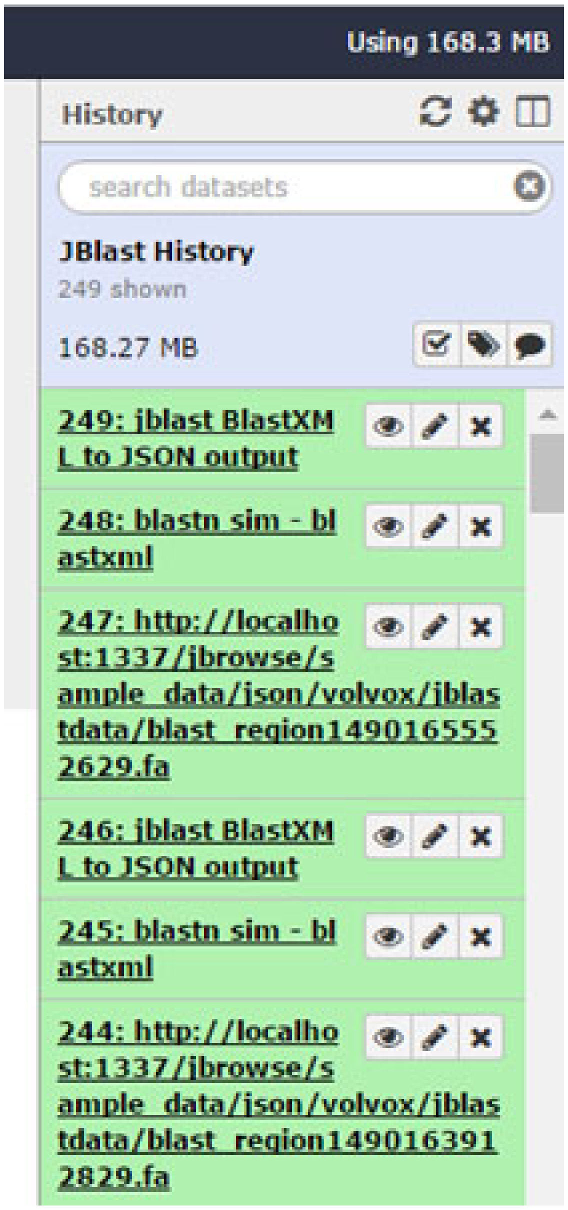
Workflow Graph

This screen shows a sample JBlast workflow's graph.


Input dataset
output
NCBI BLAST+ blastn
Nucleotide query sequence(s)
output1 (tabular, txt, html, blastxml)
export json
BlastXML input file
jblast_output_json (json)

History

JBlast operations are processed in a specific Galaxy history. This history name is defined in the config file (see jbl-globals-js).

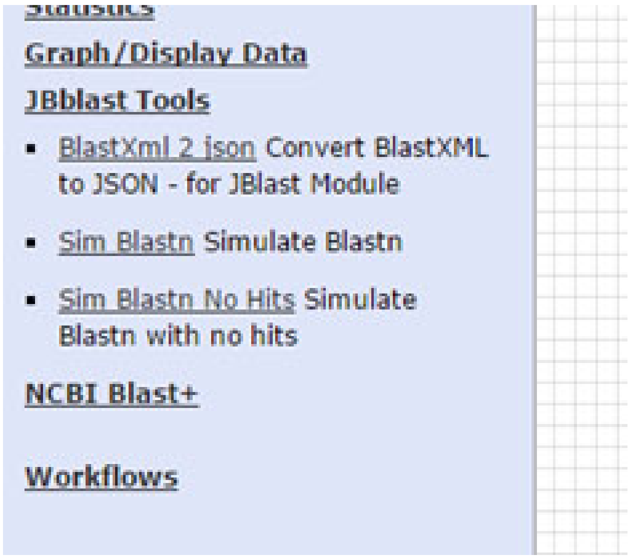


JBlast History
249 shown
168.27 MB

249: jblast BlastXML to JSON output
248: blastn sim - blastxml
247: http://localhost:1337/jbrowse/sample_data/json/volvox/jblastdata/blast_region1490165552629.fa
246: jblast BlastXML to JSON output
245: blastn sim - blastxml
244: http://localhost:1337/jbrowse/sample_data/json/volvox/jblastdata/blast_region1490163912829.fa

JBlast Tools

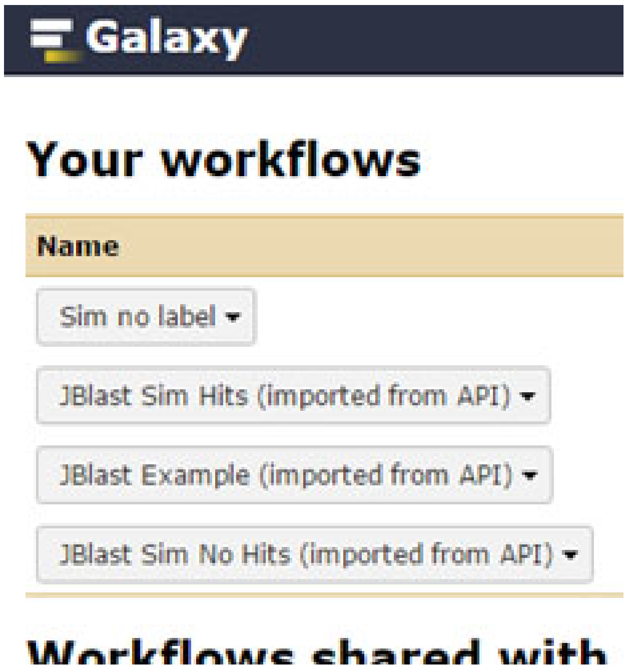
This shows the JBlast and NCBI tools in the tools sidebar.


Graph/Display Data
JBblast Tools
BlastXml 2 json Convert BlastXML to JSON - for JBlast Module
Sim Blastn Simulate Blastn
Sim Blastn No Hits Simulate Blastn with no hits
NCBI Blast+
Workflows

Note: NCBI Blast tools are not installed by the ‘‘jbutils –setuptools‘‘ script. the user must manually install these through the Tool Shed as admin.

JBlast Galaxy Workflows

JBlast has a fully functional workflow and 2 simulation workflows. The simulation workflows will only simulate a fixed blast result for demonstration purposes.


Galaxy
Your workflows
Name
Sim no label
JBlast Sim Hits (imported from API)
JBlast Example (imported from API)
JBlast Sim No Hits (imported from API)

JBlast Process

The typical JBlast process starts with the JBrowse client.

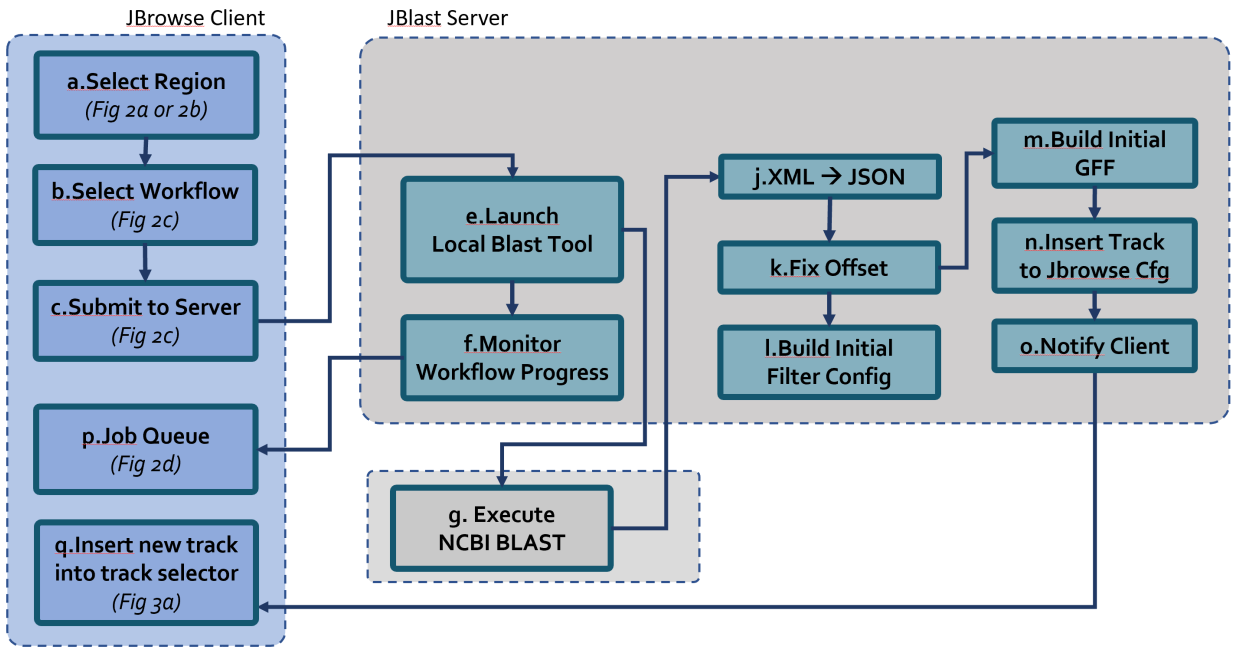

JBrowse Client
JBlast Server
a.Select Region (Fig 2a or 2b)
b.Select Workflow (Fig 2c)
c.Submit to Server (Fig 2c)
p.Job Queue (Fig 2d)
q.Insert new track into track selector (Fig 3a)
e.Launch Local Blast Tool
f.Monitor Workflow Progress
g. Execute NCBI BLAST
j.XML → JSON
k.Fix Offset
l.Build Initial Filter Config
m.Build Initial GFF
n.Insert Track to Jbrowse Cfg
o.Notify Client

Fig. 5: Processing for Stand-Alone mode

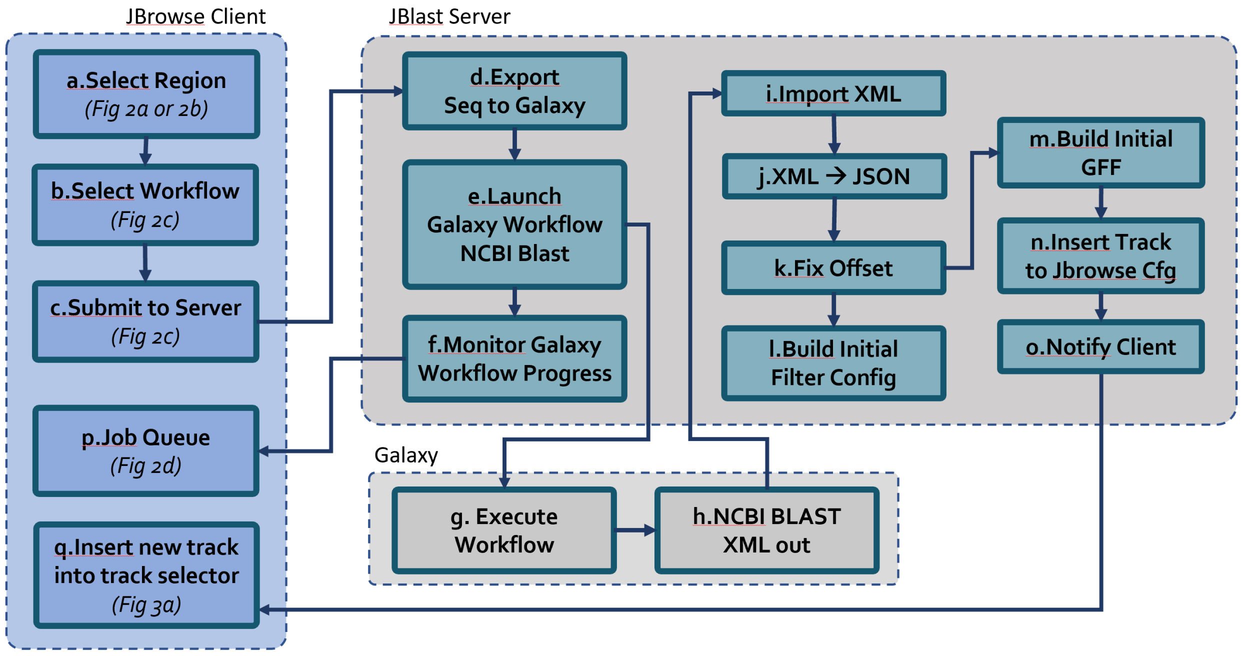
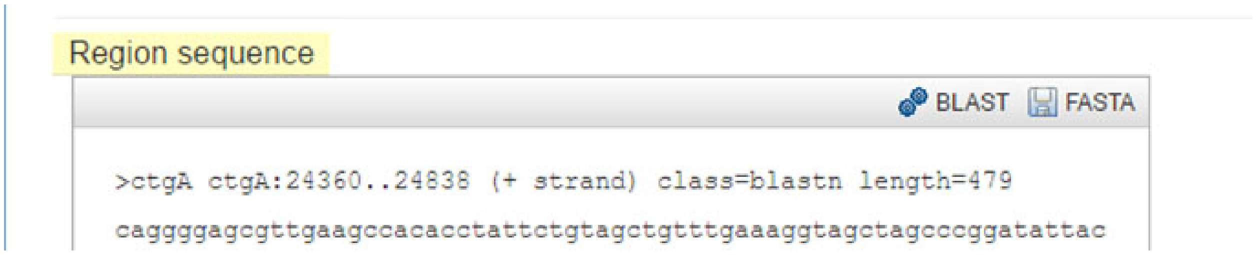

JBrowse Client
JBlast Server
a.Select Region (Fig 2a or 2b)
b.Select Workflow (Fig 2c)
c.Submit to Server (Fig 2c)
p.Job Queue (Fig 2d)
q.Insert new track into track selector (Fig 3a)
d.Export Seq to Galaxy
e.Launch Galaxy Workflow NCBI Blast
f.Monitor Galaxy Workflow Progress
Galaxy
g. Execute Workflow
h.NCBI BLAST XML out
i.Import XML
j.XML → JSON
k.Fix Offset
l.Build Initial Filter Config
m.Build Initial GFF
n.Insert Track to Jbrowse Cfg
o.Notify Client

Fig. 6: Processing for Galaxy integration mode

Select Region

A region is selected using one of two methods, either by highlighting an arbitrary region of a sequence or by selecting an existing feature.

This is an example of selecting a feature to blast:

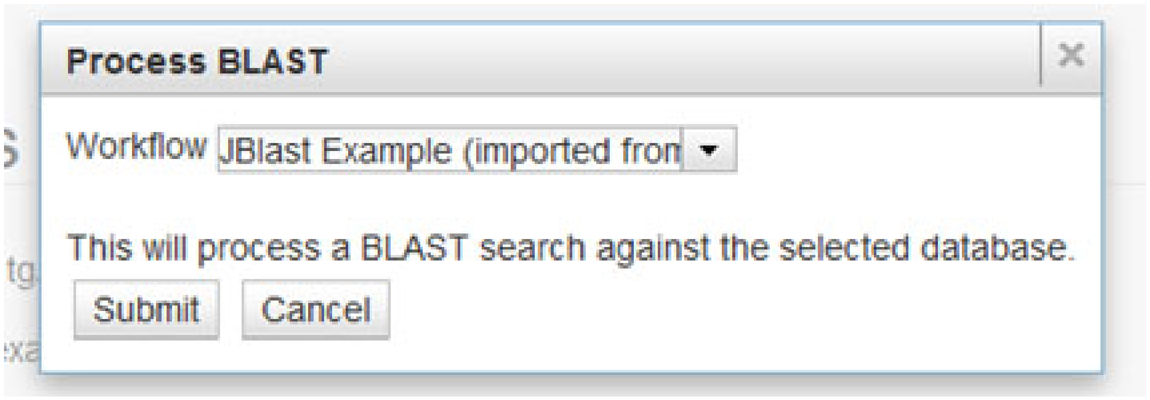


```
>ctgA ctgA:24360..24838 (+ strand) class=blastn length=479
caggggagcgttgaagccacacctattctgtagctgtttgaaggtagctagcccggatattac
```

See: *BLAST a region* for alternate selection method.

Select & Submit

User selects workflow from the list and the region is submitted to the server.



This will process a BLAST search against the selected database.

Details:

The software enumerates the available workflows from the server. The workflows may be Galaxy workflows or JBConnect workflows (stand-alone). The user should choose workflows with blast operations and the result files are blastxml.

See: *Select Workflow*

Upload selected sequence and start workflow

Upon submitting, the selected region is passed to the server (in the submit operation). The submission causes a FASTA file is generated for the region in the jblastdata directory. The selected workflow is started, which uploads the FASTA file to the workflow engine.

The offset of the sequence is saved for later use.

Monitor workflow

A workflow monitor thread is kicked off to monitor the progress of the workflow. The result files for JBlast workflows are generally blastxml files.

The workflow monitor is designed to monitor a workflow that may generate multiple result files, assuming blast searches may occur in serial or parallel, depending on the configureation of Galaxy or the JBConnect workflow engine.

The workflow monitor code is specific to the blast operation monitoring.

XML to JSON

For each resulting blastxml file, XML is converted to a JSON file with hits arranged as a associative array, for easier lookup.

The results are referred to as an *asset* with a generated asset id.

Offset Fix

Since the resulting blastxml hit results are independent of offset of the original sequence location, the offset must be applied to the results. This operation is done to the resulting JSON file so that the data can be represented as a result relative to the original dataset sequence.

Filter Settings

The filter settings file `<asset>.filtersettings` is the persistence mechanism for the current state filter settings for the asset.

The initial state of the blast filter is first built in this file. Later, when the user tunes the graphical sliders in jbrowse, the current vals will change.

The file is used as the basis for generating the filtered GFF file, which is generated each time the filter settings are changed/updated.

The format of the file looks like this:

```
{
   "score":{
      "type":"abs",
      "min":58,
      "max":593,
      "val":440
   },
   "evalue":{
      "type":"exp",
      "min":-164.2246437232114,
      "max":-3.535684861138325,
      "val":-3.535684861138325
   },
   "identity":{
      "type":"pct",
      "min":78,
      "max":100,
      "val":78
   },
```

(continues on next page)

(continued from previous page)

```
   "gaps":{
      "type":"pct",
      "min":0,
      "max":13,
      "val":13
   }
}
```

The format of the files is designed to be extensible to potentially contain other filterable values. Although, the current filter interface is not flexible enough to fully take advantage of it. In other words, the code currently only supports these 4 values.

Generate Initial GFF

The `<asset>.GFF` file contains the visible features that are the result of the dynamic filter operation and the results are driven by the `<asset>.filtersettings`.

The initial state of the `<asset>.GFF` file is unfiltered (i.e. contains all feature hits.)

Build track and add to configuration.

With `<asset>.filtersettings` and `<asset>.GFF` created, now, the a track configuration is built with `inMemTemplate.json` as a baseline. The track configuration is then inserted in the track database and the client is notified.

This is the basic track config that is built:

```
{
    "maxHeight": 1200,
    "storeClass": "JBrowse/Store/SeqFeature/GFF3",
    "blastData": "jblastdata/sampleResult.json",
    "type": "JBrowse/View/Track/HTMLFeatures",
    "metadata": {
        "description": "Sample JBlast result track"
    },
    "category": "JBlast Results",
    "key": "Sample result track",
    "label": "jblast_sample",
    "urlTemplate": "/jbapi/gettrackdata/jblast_sample/sample_data%2Fjson%2Fvolvox%2F",
    "baseUrl": "/",
    "storeCache": false
}
```

Note `storeCache:  false` configuration. This tells JBrowse not to cache the track so that each time the GFF track is redrawn, it will reread the data from the filtered GFF file.

Test Framework

Test framework uses

- Mocha for unit test
- Nightwatch for end-to-end, supporting phantomjs, selenium and online service such as browserstack.
- Istanbul for coverage

To execute

```
npm test
```

by default nightwatch is setup for phantomjs. Selenium requires running an additional selenium server Browserstack has not been tested.

Documentation Framework

For integrated documentation, JSdoc3 is used to generate API docs from code with jsdoc-sphinx, a jsdoc template that generates RestructuredText (RST) and Sphinx. This enables support for readthedocs.

See: RST/Sphinx Cheatsheet

```
npm run gendocs
```

1.4.2 Configuration Options

Setup Galaxy Server for Blast processing.

Configuration Files

globals.js

Modify the configuration file as necessary.

To view aggregate configuration: `./jbutil --config`

The aggregate config file is the merged config of JBConnect and its installed jbconnect-hook-* modules.

Edit config file: `nano config/globals.js`

```
module.exports.globals = {
    jbrowse: {
        galaxy: {
            galaxyUrl: "http://localhost:8080",                 // URL of Galaxy

            galaxyPath: "/var/www/html/galaxy",                 // path of Galaxy
            //galaxyPath: "/var/www/html/galaxy_jblast",        // path of Galaxy, if␣
↪docker

            galaxyAPIKey: "c7be32db9329841598b1a5705655f633",   //

            // jblast will use this Galaxy history
            historyName: "Unnamed history"                      // default history␣
↪name
        },
        jblast: {
            blastResultPath: "jblastdata",
            blastResultCategory: "JBlast Results",
            insertTrackTemplate: "inMemTemplate.json",
            import: ["blastxml"]
        }
    }
};
```

- `blastResultPath` is the sub directory within the dataset directory where the blast results are stored

- `blastResultCategory` is the name of the JBrowse track selectory category.
- `insertTrackTemplate` is the track insertion template.
- `import` is the file extension to process from the Galaxy workflow.

Standalone Blast Job Service

The job service, basicWorkdlowService, is the job runner service that manages stand-alone Blast processing.

Requirements

These requirements are generally installed as part of the JBlast project.

```
npm install enuggetry/blast-ncbi-tools   (NCBI blast)
npm install enuggetry/faux-blastdb       (a small sample blast database)
```

Configuration

Add the following to the jbconnect.config.js file, enabling basicWorkflowService:

```
module.exports  = {
    jbrowse: {
        services: {
            'basicWorkflowService':     {enable: true,  name: 'basicWorkflowService',␣
↪ type: 'workflow', alias: "jblast"},
            'galaxyService':            {enable: false, name: 'galaxyService',       ␣
↪ type: 'workflow', alias: "jblast"}
        },
    }
};
```

featureMapping flag

featureMapping defines how the target features mapped into coordinate space.

'hit' - target features will be mapped into target coordinates. 'query' (default) - target features will be mapped into query space on the in the contig where the query is made.

If featureMapping='hit', then contigHandler() function will be needed to decipher the proper contig of the feature. If featureMapping='query', then the contigHandler() is ignored.

```
dataSet: {
    ChineseSpring: { path: 'IWGSC',
        featureMapping: 'hit',
        contigHandler(hit) {
            console.log("IWGSC contigHandler");
            return hit.Hit_accession;
        }
    },
},
```

The contigHandler() function allows for selecting the value from a hit will be treated as the sequence id of the feature. The hit is passed to the function.

Blast Profiles

Blast profiles are parameter lists that translate to NCBI blast command line parameters sets.

For example: for the following blast command

```
blastn -db nt -query nt.fsa -out results.out
```

the blast profile would look like:

```
'myprofile': {
    'db': 'nt',
    'query': 'nt.fsa',
    'out': 'result.out'
},
```

`config/globals.js` defines the blast profiles that are preconfigured with jblast. (note, only faux blast database is autoatically loaded by JBlast project. So, 'htgs' shouldn't be used unless htgs blast database is first installed.)

```
jblast:
    defaultBlastProfile: 'faux',

    blastProfiles: {
        'htgs': {
            'db': 'htgs'
        },
        'faux': {
            'db': 'faux'
        },
        'remote_htgs': {
            'db': 'htgs',
            'remote': ""
        }
    }
}
```

The blastProfile can be specified the `POST /job/submit`. For example:

```
var postData = {
      service: "jblast",  // this can be the name of the job service or its alias
      dataset: "sample_data/json/volvox",
      region: ">ctgA ctgA:44705..47713 (- strand) class=remark␣
↪length=3009\nacatccaatggcgaacataa...gcgagttt",
      workflow: "NCBI.blast.workflow.js"
      blastProfile: 'faux'    // selects the 'faux' profile that is defined in␣
↪globals.js.
  };
$.post( "/job/submit", postData , function( result ) {
    console.log( result );
}, "json");
```

Alternatively, an previously undefined profiled may be specified in `/job/submit`.

```
var postData = {
      service: "jblast",  // this can be the name of the job service or its alias
      dataset: "sample_data/json/volvox",
      region: ">ctgA ctgA:44705..47713 (- strand) class=remark␣
↪length=3009\nacatccaatggcgaacataa...gcgagttt",
```

(continues on next page)

(continued from previous page)

```
      workflow: "NCBI.blast.workflow.js"
      blastProfile: {
        'db': 'nt',
        'query': 'nt.fsa',
        'out': 'result.out'
      }
  };
$.post( "/job/submit", postData , function( result ) {
    console.log( result );
}, "json");
```

If defaultBlastProfile is defined in globals.js will be used if no blast profile is specified in the `/job/submit` call.

Blast profiles only apply to basicWorkflowService.

JBrowse Peer Configuration

JBrowse can be configured in a peer directory instead of a module. In this case, bypass the Install JBrowse and Setup Demo step.

For example:

```
/home
  /zuser
    /JBConnect
    /jbrowse
```

Create a file called `jbconnect.config.js` in the JBConnect app directory that contains the following:

```
module.exports  = {
    jbrowse: {
        jbrowsePath: "/home/zuser/jbrowse/"
    }
};
```

Galaxy Blast Job Service

The galaxyService requres the presence of Galaxy.

See *Setup Galaxy* for instructions on how to configure Galaxy for JBlast.

JBlast jbutil Command

`jbutil` is a setup/configuration utility for JBConnect. JBConnect hooks can extend `jbutil` command options. (see: jbs-hooks-extend)

This example shows that `jbconnect-hook-jblast` adds a number of commands to `jbutil`

```
$ ./jbutil --help
Usage: node jbutil

-c, --config           display merged configuration
    --setupworkflows   (jblast-galaxy) installs demo galaxy workflows (must have API␣
↪key configured
```

(continues on next page)

(continued from previous page)

```
    --setuptools       (jblast-galaxy) setup jblast tools for galaxy
    --setupdata        (jblast) setup jblast demo data and samples
-o, --overwrite        (jblast) used with --setupdata - overwrite samples
-d, --dbreset          reset the database to default and clean kue db
-f, --force            --dbreset without verifying
-a, --setadmin         set admin flag
-r, --removeall        remove JBConnect components from JBrowse
    --pushplugins      link plugins into JBrowse dir
    --coverage=PLUGIN  used with --pushplugins to add coverage instrumentation
    --buildwebpack     build jbrowse webpack
-h, --help             display this help
```

–setupworkflows

This option setus up sample JBlast workflows in galaxy. This requires having configured the Galaxy API key in config.

–setuptools

This option sets up Jblast tools for Galaxy. After this is called, Galaxy will need to be restarted.

Note: NCBI Blast tools are not installed by the ‘‘jbutils –setuptools‘‘ script. the user must manually install these through the Tool Shed as admin.

–setupdata

This options sets up samples and sample data for JBlast.

JBlast Plugin

JBlast has integrated GUI features that must be enabled with by installing the `JBlast` plugin and the `JBClient` on the client side.

In `trackList.json`, within the dataset's path, add `JBlast` and `JBClient` plugin to the configuration.

```
"plugins": [
  "JBClient",                   <-----
  "JBlast",                     <-----

  "NeatHTMLFeatures",
  "NeatCanvasFeatures",
  "HideTrackLabels"
],
```

Note: the JBlast and JBClient plugins are not physically in the JBrowse plugin directory. They are made available as route by the JBConnect framework and are only accessible at runtime.

See *Integrated GUI* for more details.

Limiting the BLAST query size

In dataset's trackList.json, define *bpSizeLimit* to limit the size of the sequence query.

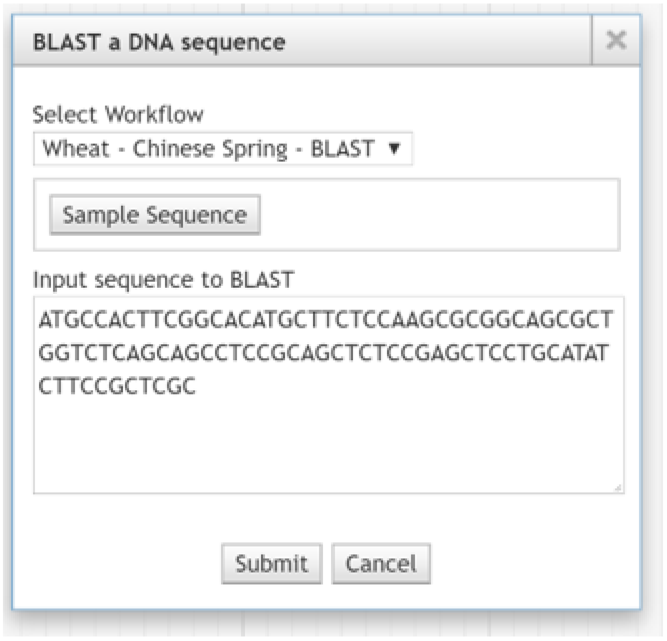
```
"plugins": [

    ...

    "JBClient",
    {
        "name": "JBlast",
        "bpSizeLimit": 15000
    }
],
```

Sample Sequence Button

Configure a Sample Sequence Button in the BLAST a DNA sequence dialog box.



```
ATGCCACTTCGGCACATGCTTCTCCAAGCGCGGCAGCGCT
GGTCTCAGCAGCCTCCGCAGCTCTCCGAGCTCCTGCATAT
CTTCCGCTCGC
```

Fig. 7: BLAST DNA Sequence Dialog box

Add the following section in the JBrowse trackList.json file.

```
"demo": {
    "blastButtons": [
        {
            "button": "Sample Sequence",
            "description": "Insert sample sequence",
            "sequence": [

↪"ATGCCACTTCGGCACATGCTTCTCCAAGCGCGGCAGCGCTGGTCTCAGCAGCCTCCGCAGCTCTCCGAGCTCCTGCATATCTTCCGCTCGCTCTCTA
↪",

↪"CCATCCGGAGCCTCTGCCGCGCCATCCAACCGCTCTCCTAGGCAAATTCAGCTGCCCCAAACCTTGCCTTGCTCCAATGCCAACCCACTTGGCGCCG
↪",

↪"CACATCGACGTTGTTGACGATGACCTCTGGCCCACTTCCTTCGGCTTCTCCTCAGATCCCATGACTGGTGATGAGTGTCTTGATACCTTCCAAGAAC
↪",
```

(continues on next page)

(continued from previous page)

```
↪"GAAGAACAAGTGCACGACTCGGATGATGAGATAGATGACATGAGGCACCGCAAGCAGCTGTTCTACAAGCTGGACAGGGGGTCCAAGGAGTTTGAGGA
↪",
                    "AACTTGCCCTTGCGCCGCAGATGGAAGAGAGATAAACCCAATGCCAAGAATCCATCCGATTGCG"
               ]
          }
     ]
},
```

1.4.3 Setup Galaxy

Instructions for installing and configuring Galaxy for use with JBlast.

Install Galaxy

Instructions for installing galaxy: Get Galaxy

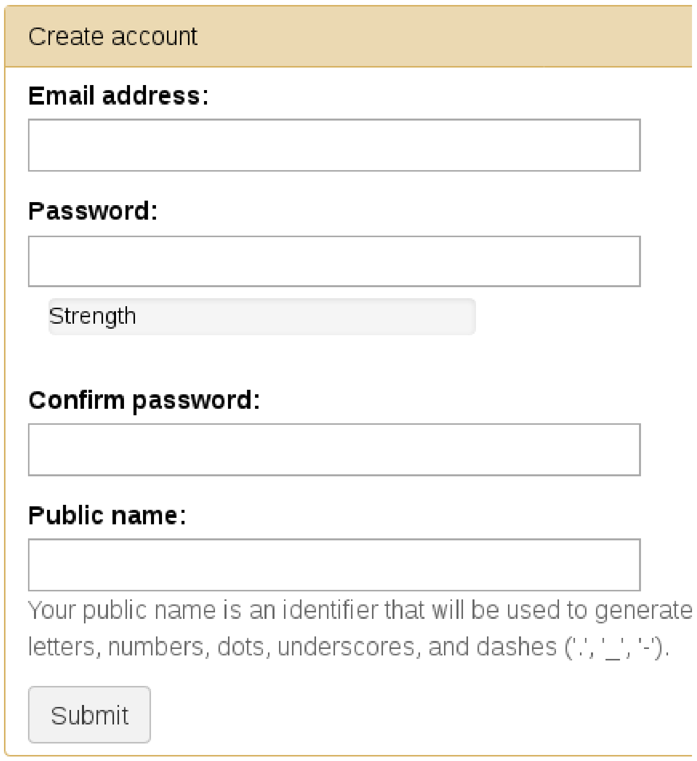
`git clone -b release_17.09 https://github.com/galaxyproject/galaxy.git` (tested)

Run galaxy: `sh run.sh` (From galaxy dir. First time run will take a while)

By default Galaxy is hosted on port 8080: `http://localhost:8080`

Create a user with admin privilage

Register a new user (**User** Menu –> Register).


Create account

Email address:

Password:

Strength

Confirm password:

Public name:

Your public name is an identifier that will be used to generate
letters, numbers, dots, underscores, and dashes ('.', '_', '-').

In your `jbconnect` directory, edit `jbconnect.config.js` and create a `galaxy:` section under `jbrowse:` section. Add the Galaxy installation path.

```
module.exports = {
  jbrowse: {
    galaxy: {
      galaxyPath: '/var/www/galaxy'
    }
  }
}
```

These settings will override any settings in `node_modules/jbconnect-hook-jblast/config/globals.js` and `jbconnect/config/globals.js`.

From the JBrowse directory, type `./jbutil --setuptools`

This will copy some JBlast specific Galaxy tools into the `galaxy` directory as as well as replace `config.galaxy.ini` in the `galaxy` directory.

In `galaxy` directory, add the following line to `config/galaxy.ini` add the user email for the user you created as an admin:

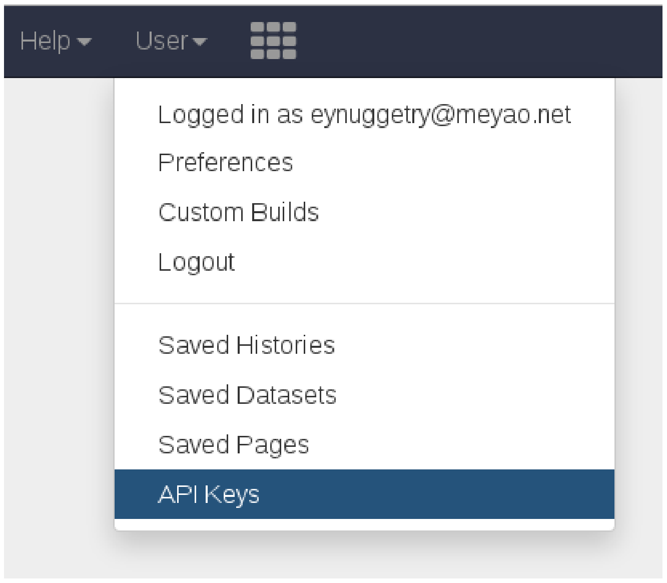
```
admin_users = me-user@gmail.com
```

Start Galaxy again from the galaxy directory (`sh run.sh`)

Now you should see and **Admin** menu appear in Galaxy.

Generating the Galaxy API key

Create an API key (**User** Menu –> Preference), then select **Manage API Key**, click the **Create a new key** button.



In the JBConnect directory, add the API key to jbconnect.config.js under the galaxy: section.

```
module.exports = {
  jbrowse: {
    galaxy: {
      galaxyPath: '/var/www/galaxy',
```

(continues on next page)

(continued from previous page)

```
        galaxyAPIKey: "c7be32db9329841598b1a5705655f633"
      }
    }
}
```

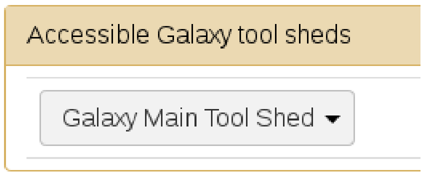
Now, restart galaxy: `sh run.sh`

Install NCBI Blast+ Tools

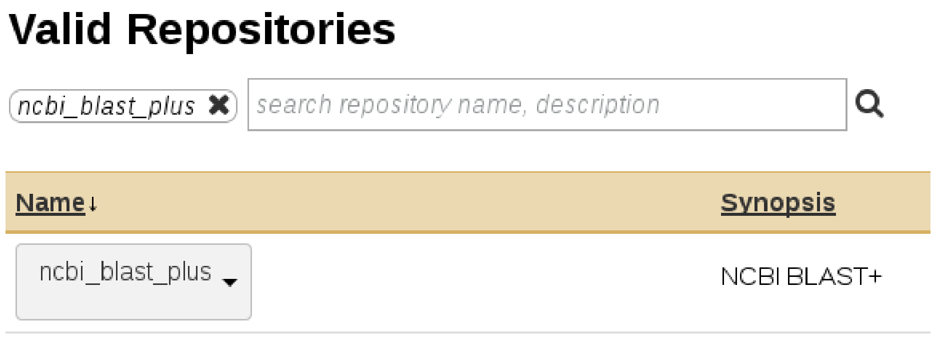
At the same level as `jbconnect` and `galaxy` directories, create a directory called `shed_tools`, making sure it has the same permissions as the `galaxy` and `jbconnect` directories.

Select the **Admin** menu and **Search Tool Shed** from the left side bar.

Select the **Galaxy Main Tool Shed**:

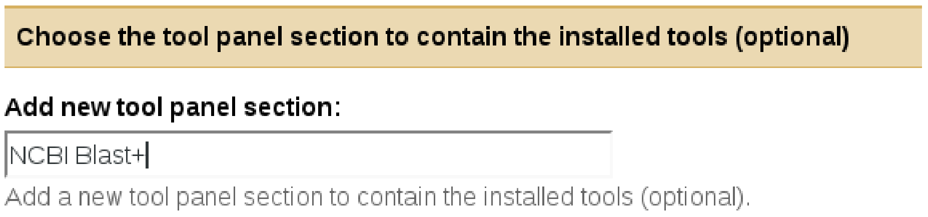


In the search box enter `ncbi_blast_plus`.



When you come to the Install to Galaxy button, click it.

When you arrive at the screen with **Add new tool panel section**, type in “NCBI Blast+”


Choose the tool panel section to contain the installed tools (optional)
Add new tool panel section:
NCBI Blast+
Add a new tool panel section to contain the installed tools (optional).

Then click Install button.

The NCBI blast tools and dependencies will proceed to be installed.

Sometimes you will have to do this procedure a 2nd or 3rd time to ensure all the dependencies are installed.

Install demo workflows

Install sample workflows used in demo. (this step require the API key to be configured and Galaxy should be running.)

```
./jbutil --setupworkflows
```

Registering a Blast Database

A default database called faux is a small sample blast database is loaded by the project.

Additional databases may be loaded if necessary.

Download the blast database if you haven't already done it.

```
./bin/blast_downloadDb.js htgs.05   (setup sample database)
         // you can also download the full "htgs" database, but this will
         // take a while on slower lines. (ie. "./blast_downloadDb.js htgs" )
```

This downloads and installs **"htgs"** BLAST database from `ftp://ftp.ncbi.nlm.nih.gov/blast/db/` into blastdb/htgs directory.

In the `galaxy` directory, edit `tool-data/blastdb.loc`.

Add this line to the end of the file:

```
htgs{tab}High Throughput Genomic Sequences (htgs){tab}/var/www/jbconnect/
blastdb/htgs/htgs
```

It's important to get the name "htgs" correct. The name counts in our demo workflow. The directory should reflect the directory of the installed BLAST database.

Setup Galaxy Job Service

In `jbconnect` directory, edit `node_modules/jbconnect-hook-jblast/config/globals.js` and add the galaxy job service.

```
services: {
    'basicWorkflowService':      {enable: false, name: 'basicWorkflowService',  type:
↪'workflow', alias: "jblast"},
    'galaxyService':             {enable: true,  name: 'galaxyService',         type:
↪'workflow', alias: "jblast"},
    'filterService':             {name: 'filterService',         type: 'service'},
    'entrezService':             {name: 'entrezService',         type: 'service'}
},
```

Restart Galaxy: `sh run.sh`

Lift sails: `sails lift`

1.4.4 API

Module: `hooks/jblast`

Local Navigation

Description

This module is the main subclass of a Sails Hook incorporating *Marlinspike*.

Function: `_interopRequireDefault`

`_interopRequireDefault``()`

Function: `_classCallCheck`

`_classCallCheck``()`

Function: `_possibleConstructorReturn`

`_possibleConstructorReturn``()`

Function: `_inherits`

`_inherits``()`

Function: `value`

`value``()`

Function: `value`

`value``()`

Function: `value`

`value``()`

Function: `value`

`value`()

Module: `services/basicWorkflowService`

Local Navigation

Description

This is a job services that executes local NCBI blast by either excuting NCBI.blast or Sim.blast, defined by the job.

This job service is functionally equivelant to galaxyService, which does blast search through Galaxy API.

Job submission example:

```
var postData = {
      service: "jblast",
      dataset: "sample_data/json/volvox",
      region: ">ctgA ctgA:44705..47713 (- strand) class=remark␣
↪length=3009\nacatccaatggcgaacataa...gcgagttt",
      workflow: "NCBI.blast.workflow.js"
  };
$.post( "/job/submit", postData , function( result ) {
    console.log( result );
}, "json");
```

Configuration:

```
 jblast: {
    // The subdir where blast results will be deposited (i.e. ``sample_data/json/
↪volvox/jblastdata``)
    blastResultPath: "jblastdata",

    // The category for successful blast results in the track selector
```

(continues on next page)

(continued from previous page)

```
    blastResultCategory: "JBlast Results",

    // Track template of the blast result track that will be inserted in trackList.
↪json
    trackTemplate: "jblastTrackTemplate.json",

    // Type of file that will be imported for processing blast.
    import: ["blastxml"],


    // BLAST profiles
    // blast profiles are parameter lists that translate to blastn cli parameters sets
    // (i.e. for "remote_htgs" would translate to "blastn -db htgs -remote")
    // These will override any default parameters defined in ``blastjs``
    //
    // Blast profiles generally apply to basicWorkflowService only
    // and do no apply to galaxyService.
    //
    // Our example uses a subset of htgs, an NCBI curated blast database.
    // So, it is our default profile.
    defaultBlastProfile: 'htgs',
    blastProfiles: {
        'htgs': {
            'db': 'htgs'
        },
        'remote_htgs': {
            'db': 'htgs',
            'remote': ""
        }
    }
},
// list of services that will get registered.
services: {
    'basicWorkflowService':     {name: 'basicWorkflowService',  type: 'workflow',␣
↪alias: "jblast"},
    'filterService':            {name: 'filterService',         type: 'service'},
    'entrezService':            {name: 'entrezService',         type: 'service'}
},
```

Job queue entry example:

```
{
  "id": 145,
  "type": "workflow",
  "progress": "0",
  "priority": 0,
  "data": {
    "service": "jblast",
    "dataset": "sample_data/json/volvox",
    "region": ">ctgA ctgA:44705..47713 (- strand) class=remark␣
↪length=3009\nacatccaatggcgaacataagcgagttttgt...tggccc",
    "workflow": "NCBI.blast.workflow.js",
    "name": "NCBI.blast.workflow.js",
    "sequence": {
      "seq": "ctgA",
      "start": "44705",
      "end": "47713",
```

(continues on next page)

(continued from previous page)

```
      "strand": "-",
      "class": "remark",
      "length": "3009"
    },
    "blastData": {
      "name": "JBlast",
      "blastSeq": "/var/www/html/jbconnect/node_modules/jbrowse//sample_data/json/
↪volvox/jblastdata/blast_region1517044304838.fa",
      "offset": "44705"
    },
    "seqFile": "http://localhost:1337/jbrowse/sample_data/json/volvox/jblastdata/
↪blast_region1517044304838.fa",
    "blastOptions": {
      "db": "htgs"
    },
    "blastOptionFile": "/tmp/blast_option1517044304843.json"
  },
  "state": "failed",
  "promote_at": "1517044302842",
  "created_at": "1517044302842",
  "updated_at": "1517044310134",
  "createdAt": "2018-02-01T05:38:27.406Z",
  "updatedAt": "2018-02-01T05:38:27.406Z"
},
```

Function: `init`

`init`()

Function: `validateParams`

job service validate

`validateParams`(*params*)

Arguments

- **`params`** (`object`) – parameters

Return int 0 if successful

Function: `generateName`

job service generate name.

`generateName`(*params*)

Arguments

- **`params`** (`object`) – parameters

Return string string job name

Function: `get_workflows`

Enumerate available workflow scripts

`get_workflows`(*req*, *res*)

Arguments

- **`req`** (*`object`*) – request
- **`res`** (*`object`*) – response

Function: `get_hit_details`

`get_hit_details`()

Function: `beginProcessing`

Job service - job execution entry point

`beginProcessing`(*kJob*)

Arguments

- **`kJob`** (*`object`*) – reference to the kue job object

Function: `determineBlastProfile`

`determineBlastProfile`()

Function: `beginProcessing2`

`beginProcessing2`()

Function: `_runWorkflow`

`_runWorkflow`()

Function: `_postProcess`

`_postProcess`()

Module: `services/blastxml2json`

Local Navigation

Description

Convert BlastXML to JSON (not a straight conversion) This script not only converts the XML to json, it flattens the hits per hsp where there are multiple hsp.

Creates an indexed list by feature ID. Essentially, it simpifies the hit array into an associative array and makes it indexed by key, where key is <Hit_id>;<Hsp_num>

Function: `convert`

Perform the conversion operation

`convert`()

Arguments

- **`convert()`** – kJob - kue job object
- **`convert()`** – trackJson
- **`convert()`** – cb - callback function

Member: `err`:

Module: `services/entrezService`

Local Navigation

Description

This job service enables accession value lookup utilizeing Entrez API.

Ref: Entrez

Function: `init`

Initialize the module

`init`(*req*, *res*, *cb*)

Arguments

- **`req`** (*`object`*) – request
- **`res`** (*`object`*) – response
- **`cb`** (*`function`*) – callback function

Function: `lookup_accession`

This does an esummary lookup (using Entrez api), adding the link field into the result.

`lookup_accession`(*req*, *res*)

Arguments

- **`req`** (*`object`*) – request
- **`res`** (*`object`*) – response

Module: `services/filter`

Local Navigation

Description

Supporting methods for the filterService jservice.

Function: `filterInit`

create initial filter settings file

`filterInit`(*kJob*)

Arguments

- **kJob** (*object*) – kue job object
- **filterInit (kJob)** – cb - callback

Function: getFilterSettings

get filterData

getFilterSettings (*requestData*, *cb*)

Arguments

- **requestData** (*object*) – eg. { asset: 'jblast_sample', dataset: 'sample_data/json/volvox' }
- **cb** (*object*) – function(filterData)

::

eg. filterData: { contig: "ctgA", score: {type: 'abs', min: 58, max: 593, val: 421 }, evalue: { type: 'exp', min: 5.96151e-165, max: 0.000291283, val: 0.000291283 }, identity: { type: 'pct', min: 78, max: 100, val: 78 }, gaps: { type: 'pct', min: 0, max: 13, val: 13 }

}

Function: writeFilterSettings

write new data to filter settings file, given requestData

writeFilterSettings (*requestData*, *cb*)

Arguments

- **requestData** (*object*) – eg. { asset: 'jblast_sample', dataset: 'sample_data/json/volvox', filterParams: filterData }
- **cb** (*object*) – updated filterData function(filterData)

::

eg. filterData: { contig: "ctgA", score: {type: 'abs', min: 58, max: 593, val: 421 }, evalue: { type: 'exp', min: 5.96151e-165, max: 0.000291283, val: 0.000291283 }, identity: { type: 'pct', min: 78, max: 100, val: 78 }, gaps: { type: 'pct', min: 0, max: 13, val: 13 }

}

Function: applyFilter

Based on the filterData, generate a new gff3 file. Also announces the track to subscribed clients.

applyFilter (*filterData*, *requestData*)

Arguments

- **filterData** (*object*) – the output of writeFilterSettings or getFilterSettings.
- **requestData** (*object*) – eg. { asset: 'jblast_sample', dataset: 'sample_data/json/volvox' }

callback:

```
cb({
   totalFeatures: x,                 // total number of features
   filteredFeatures: x               // filtered features.
})
```

Function: `getHitDataFiltered`

`getHitDataFiltered``()`

Function: `_announceTrack`

`_announceTrack``()`

Function: `getHitDetails`

return hit details given hit key, including all HSPs of the original hit. The hit key looks like this "gi-402239547-gb-JN790190-1–3" Separate the hit id ==> "gi-402239547-gb-JN790190-1–" (basically remove the last number) Returns multiple HSPs for each hit id: data for "gi-402239547-gb-JN790190-1–1", "gi-402239547-gb-JN790190-1–2"…

`getHitDetails``(`*hitkey*, *cb*`)`

Arguments

- **`hitkey`** (*`string`*) – eg. "gi-402239547-gb-JN790190-1–3"
- **`cb)`** (*`getHitDetails(hitkey,`*) – dataSet - eg. "sample_data/json/volvox"
- **`cb`** (*`function`*) – callback

Function: `getHighest`

`getHighest``()`

Function: `getLowest`

`getLowest``()`

Function: `getHighest10`

`getHighest10``()`

Function: `getLowest10`

`getLowest10``()`

Function: `getHighestPct`

`getHighestPct``()`

Function: `getLowestPct`

`getLowestPct`()

Function: `convert2Num`

`convert2Num`()

Function: `getHitId`

`getHitId`()

Constant: _:

Module: `services/filterService`

Local Navigation

Description

This jservice provides restful APIs for processing filter requests.

Function: `init`

`init`()

Function: `set_filter`

Based on new filter settings provided by the caller, updates the associated filtersettings file and the resulting GFF3 file containing filtered features.

REST Request: POST */service/exec/set_filter*

`set_filter`(*req*, *res*)

Arguments

- **`req`** (`object`) – request

```
req.body = {
  filterParams: {score:{val: 50}, evalue:{val:-2}...
  dataSet: (i.e. "sample_data/json/volvox" generally from config.dataRoot)
  asset: asset id
}
```

Arguments

- **`res`** (*`object`*) – response

Function: `get_blastdata`

Determine filter details, like number of hit results. REST

> *GET /service/exec/set_filter* data: eg. *{asset: '151_1517462263883', dataset: 'sample_data/json/volvox'}*

Return data: eg. *{ result: 'success', hits: 792, filteredHits: 501 }*

`get_blastdata`(*req*, *res*)

Arguments

- **`req`** (*`object`*) – request
- **`res`** (*`object`*) – response

Function: `get_trackdata`

Fetch the GFF3 file of the prior filter operation

`GET /service/exec/set_filter`

`get_trackdata`(*req*, *res*)

Arguments

- **`req`** (*`type`*) – request
- **`res`** (*`type`*) – response

Function: `fixNumber`

`fixNumber`()

Module: `services/galaxyService`

Local Navigation

- *Function:* `generateName`
- *Function:* `beginProcessing`
- *Function:* `get_workflows`
- *Function:* `get_hit_details`

Description

This job service is functionally equivelant to basicWorkflowService, however, NCBI operations are sent through galaxy workflow for processing.

Job submission example:

```
var postData = {
      service: "jblast",
      dataset: "sample_data/json/volvox",
      region: ">ctgA ctgA:44705..47713 (- strand) class=remark␣
↪length=3009\nacatccaatggcgaacataa...gcgagttt",
      workflow: "NCBI.blast.workflow.js"
  };
$.post( "/job/submit", postData , function( result ) {
    console.log( result );
}, "json");
```

Configuration:

```
 // Galaxy settings
 galaxy: {
     // Galaxy API path
     galaxyUrl: "http://localhost:8080",

     // Galaxy installation path
     galaxyPath: "/var/www/html/galaxy",

     // Galaxy API key (you must obtain this from your Galaxy installation)
     galaxyAPIKey: "c7be32db9329841598b1a5705655f633",

     // The default Galaxy History where workflows will execute
     historyName: "Unnamed history"
 },

 jblast: {
    // The subdir where blast results will be deposited (i.e. ``sample_data/json/
↪volvox/jblastdata``)
    blastResultPath: "jblastdata",

    // The category for successful blast results in the track selector
    blastResultCategory: "JBlast Results",

    // Track template of the blast result track that will be inserted in trackList.
↪json
    trackTemplate: "jblastTrackTemplate.json",

    // Type of file that will be imported for processing blast.
    import: ["blastxml"],
```

(continues on next page)

(continued from previous page)

```
    // BLAST profiles
    // blast profiles are parameter lists that translate to blastn cli parameters sets
    // (i.e. for "remote_htgs" would translate to "blastn -db htgs -remote")
    // These will override any default parameters defined in ``blastjs``
    //
    // Blast profiles generally apply to basicWorkflowService only
    // and do no apply to galaxyService.
    //
    // Our example uses a subset of htgs, an NCBI curated blast database.
    // So, it is our default profile.
    defaultBlastProfile: 'htgs',
    blastProfiles: {
        'htgs': {
            'db': 'htgs'
        },
        'remote_htgs': {
            'db': 'htgs',
            'remote': ""
        }
    }
},
// list of services that will get registered.
services: {
    'galaxyService':          {name: 'galaxyService',         type: 'workflow',␣
↪alias: "jblast"},
    'filterService':            {name: 'filterService',         type: 'service'},
    'entrezService':            {name: 'entrezService',         type: 'service'}
},
```

Function: `init`

`init`()

Function: `validateParams`

job service validation

`validateParams`(*params*)

Arguments

- **`params`** (*`object`*) – parameters

Return val 0 if successful, otherwise failure

Function: `generateName`

job service generate name

`generateName`(*params*)

Arguments

- **params** (*object*) – parameters

Return string name of job

Function: `beginProcessing`

job service begin

beginProcessing(*kJob*)

Arguments

- **kJob** (*object*) – kue job object

Function: `get_workflows`

get_workflows()

Function: `get_hit_details`

get_hit_details()

Module: `services/galaxyUtils`

Local Navigation

Description

This provides functional support to galaxyService job service.

Function: `init`

Initialize module

`init`(*cb*, *cberr*)

Arguments

- **`cb`** (*type*) – Initialize module
- **`cberr`** (*type*) – Initialize module

Return undefined Initialize module

Function: `galaxyGetPromise`

`galaxyGetPromise`()

Function: `galaxyPostPromise`

`galaxyPostPromise`()

Function: `galaxyGET`

send JSON GET request to galaxy server

`galaxyGET`(*api*, *cb*)

Arguments

- **`api`** (*type*) – i.e. '/api/histories'
- **`cb`** (*type*) – callback i.e. function(retval)

Function: `galaxyPOST`

`galaxyPOST`()

Function: `getHistoryId`

`getHistoryId`()

Return string history id

Function: `getHistoryName`

`getHistoryName`()

Return string history name

Function: `initHistory`

acquire history id from galaxy

`initHistory`(*cb*)

Arguments

- **`cb`** (*`type`*) – acquire history id from galaxy

Function: `getWorkflows`

get workflows

`getWorkflows`(*cb*)

Arguments

- **`cb`** (*`type`*) – get workflows

Return undefined get workflows

Function: `sendFile`

send file to galaxy

`sendFile`(*theFile*, *hId*, *cb*, *cberr*)

Arguments

- **`theFile`** (*`type`*) – send file to galaxy
- **`hId`** (*`type`*) – send file to galaxy
- **`cb`** (*`type`*) – send file to galaxy
- **`cberr`** (*`type`*) – send file to galaxy

Return undefined send file to galaxy

Function: `beginProcessing`

Job service, job entry point.

`beginProcessing`(*kJob*)

Arguments

- **`kJob`** (*`object`*) – reference to kue job object

Function: `beginProcessing2`

`beginProcessing2`()

Function: `monitorWorkflow`

Monitor workflow and exit upon completion of the workflow

`monitorWorkflow`(*kJob*)

Arguments

- **`kJob`** (*`object`*) – Monitor workflow and exit upon completion of the workflow

Function: `doCompleteAction`

Read output of last generated file, copy results to /jblastdata, insert track to trackList.json.

`doCompleteAction`(*kJob*, *hista*)

Arguments

- **`kJob`** (*`object`*) – kue job object
- **`hista`** (*`object`*) – associative array of histories

Module: `services/jblastPostAction`

Local Navigation

Description

This module implements the actions that occur after a galaxy workflow completes. It supports galaxyService job service.

Function: `postMoveResultFiles`

`postMoveResultFiles`()

Function: `getHits`

`getHits`()

Function: `processFilter`

`processFilter`()

Function: `postMoveResultFiles`

this generates track template

`postMoveResultFiles`(*kJob*, *cb*)

Arguments

- **`kJob`** (*`type`*) – kue job object
- **`cb`** (*`type`*) – callback

Function: `processFilter`

Generate the GFF file

`processFilter`(*kJob*, *newTrackJson*, *cb*)

Arguments

- **`kJob`** (*`type`*) – = kue job object
- **`newTrackJson`** (*`type`*) – working track object
- **`cb`** (*`type`*) – callback

Function: `getHits`

return number of hits

`getHits`(*kJob*, *newTrackJson*)

Arguments

- **`kJob`** (*`object`*) – kue job object
- **`newTrackJson`** (*`JSON`*) – working track object

Return Number number of hits

Member: `requestp`:

Member: `path`:

Member: `Promise`:

Member: `fs`:

Member: `deferred`:

Member: `filter`:

Member: `offsetfix`:

Member: `blast2json`:

Member: `galaxy`:

Member: _:

Member: `newTrackJson`:

Module: `services/offsetfix`

Local Navigation

Description

This module fixes the offsets of blast search results.

Function: `process`

`process`()

Index