Chinook User Documentation (CUD)

How-to Add New Services

Stephen Montgomery Oct. 22nd, 2003

Adding new services in Chinook is more often than not trivial. It requires the addition of a new tag in the XML file that is used to start-up your Chinook server *(see CUD: How-to Start a Chinook Server)*.

Chinook is only designed to work with command-line applications where sequence is the input. Current input sequence formats that are supported in Chinook are:

1) Fasta
2) Multi-Fasta

However, static databases or data retrieval mechanisms are allowed in your command-line applications; for a transcription factor binding search algorithm it is perfectly valid to create a service that takes in a matrix name and have your service download this matrix and search an inputted sequence.

The most complicated thing that may need to be done when adding a new Chinook service is to extend the available parsing classes. This would have to be done in the case that your output format cannot be read by Chinook or you want to return special data – data that conforms to a more complex return type. It is often better to return your format as GFF and include attributes that the client can interpret. The currently support parsers are:

1) Multi-fasta (LAGAN, MLAGAN services)
2) GCG (Clustalw services)

Very soon an implementation of GFF will be available.

All new parsers must implement the class `ca.bcgsc.chinook.server.runner.impl.ExecutableImpl` and extend the getReport(ReportFactory rf) function. For an example see the Multi-fasta parsers for LAGAN `ca.bcgsc.chinook.server.runner.alignment.Lagan`.

The default available services that are packaged with Chinook are currently:

1) MLAGAN
2) LAGAN.
3) CLUSTALW

So how do you add a new application? Let's look at the LAGAN implementation as an example.

```
<application>
        <name>MLAGAN</name>
        <type>ALIGNMENT</type>
        <path>/opt/mlagan</path>
        <executable>mlagan</executable>

        <format>exe_path/executable SEQUENCE(2)(*) parameter -out output_path/temp_outfile</format>

        <parsing_class>ca.bcgsc.chinook.server.runner.alignment.Lagan</parsing_class>
        <output_path>/tmp</output_path>
        <description>Lagan is developed at Stanford by Mike Brudno</description>
        <creator>http://lagan.stanford.edu</creator>

        <parameter>
            <descriptor>tree_STRING</descriptor>
            <regex_format>["]([*]+)["]</regex_format>
            <description>This is of the form "(Homo_sapiens Mus_musculus) Danio_rerio)" for your input sequences</description>
            <user_defined>true</user_defined>
            <equals>false</equals>
        </parameter>
        <parameter>
            <descriptor>translate_</descriptor>
            <description>Use translated anchoring</description>
            <user_defined>true</user_defined>
        </parameter>
        <parameter>
            <descriptor>nested_</descriptor>
            <description>Runs iterative improvement in a nested fashion</description>
            <user_defined>true</user_defined>
        </parameter>
        <parameter>
            <descriptor>postir_</descriptor>
            <description>Runs iterative improvement on final alignment</description>
            <user_defined>true</user_defined>
        </parameter>
        <parameter>
            <descriptor>fastreject_</descriptor>
            <description>Abandon alignment if homology looks weak</description>
            <user_defined>true</user_defined>
        </parameter>
</application>
```

All new applications specs are defined between the `<application>` `</application>` tags. The first tag `<name>` defines the application that is being run. This can be anything. The next tag `<type>` is more important. This is an ontological definition that marks the type of service class you belong to. It is planned that the website will carry a dictionary of the terms that are wildly used. For now, the only well-defined term is ALIGNMENT. Work is being done to add PRIMER PREDICTION and MOTIF SCANNING. The `<path>` tag simple points to the directory that the main service is in and `<executable>` tag holds the name of the application that will be run.

The `<format>` tag deserves special mention. It is probably best understood by going through several examples:

```
1)  <format>exe_path/executable SEQUENCE(2)(*) parameter -out output_path/temp_outfile</format>
2)  <format>exe_path/executable 1SEQUENCE(2)(*) parameter -outfile=output_path/temp_outfile</format>
3)  <format>exe_path/executable SEQUENCE(2)(2) parameter > output_path/temp_outfile</format>
```

Here the examples have been numbered for convenience. In the `<format>` tag, several terms are special and are replaced by appropriate values when the script is run.

1) `exe_path` is replaced by the contents of the `<path>` tag
2) `executable` is replaced by the contents of the `<executable>` tag
3) `SEQUENCE` is replaced by the location of the sequence files (auto-generated)
4) `parameter` is replaced by the service specific parameters
5) `output_path` is replaced by the `<output_path>` tag
6) `temp_outfile` is replaced by a temporary outfile name (auto-generated)

You are able to do anything you want in the `<format>` tag that allows your program to run; note the different ways of pointing to the output file. For instance if this was a blast service, you could specify the databases that are required. However, as we shall see below it is better to provide this information within the `<parameter>` tags.

HINT: What is so special about `SEQUENCE`

`SEQUENCE` has a 1 in front of it in one example above and it has two numbers in parenthesis after it. The 1 in front of sequence specifies that all the sequences are going to be put into 1 fasta file, the default is that they are all put into separate fasta files. The numbers trailing indicate how many sequences this service can handle. Some services can only operate on one sequence and should be defined as `(1)(1)`. Some can work on 1, 2 or 3 sequences and would be defined `(1)(3)`. Services that can use 2 or more sequences are defined as `(2)(*)` - which is the case for our multiple alignment services.

The `<parsing_class>` tag defines what class will parse the output placed in the `temp_outfile`. The currently supported parsing classes are:

1) `ca.bcgsc.chinook.server.runner.alignment.Lagan` for Fasta
2) `ca.bcgsc.chinook.server.runner.alignment.Clustalw` for GCG

As mentioned above, if a parsing class isn't defined, you will have to make one in the runner package of the Chinook server code – see notes at the beginning of this document.

The `<output_path>` tag points to your temporary directory for formatting files. The `<description>` tag points to a description of the service. The `<creator>` tag is the website of the original author of the services implementation (not the service providers). So in the case of Lagan, it points to the site at Stanford.

Implementing `<parameter>` tags:

The `<parameter>` tag is a another special tag in the XML description of your service. These tags define what parameters you want your client to input, what parameters you'd prefer they didn't, and what default values you'd like to maintain. The `<parameter>` tag offers extensive control over how your client uses your service.

The first part of the `<parameter>` tag if the `<descriptor>` tag. This tag defines what type of parameter it is. Fundamentally, there are two types `STRING` and empty (boolean). When defining the `<descriptor>` tag, define it as the name than the type, i.e. `tree_STRING` tells Chinook that you have a parameter called tree that needs a string whereas `translate_` tells Chinook that you have a parameter that is either there or it isn't; for instance,

```
mlagan -tree “mytree” -translate
```

The `<regex_format>` tag describes the regular expression that you want your input string data to match. This is a security feature that allows you to guarantee that parameters will be inputted in the way that you expect to get them – as users are prevented from entering parameters that don't match. This tag is not required however.

The `<description>` tag describes to the user what this parameter does. The `<user_defined>` tag tells Chinook whether you want the user to be able to change this parameter; it has two options `true` or `false`. The `<use_equals>` tag tells Chinook whether the parameter is of the form `-tree=value` or `-tree value.` It takes two options `true` or `false`. The `<on>` tag tells Chinook whether this boolean parameter is active or not by default. Finally, the `<default_value>` tag holds the data that you want this parameter to input, i.e the default string data. It is very possible as a service provider to use these tags in way that doesn't make sense. Be very careful about what you want users to do and what the default parameters are. If you find that something is missing to fully describe your input parameters, e-mail me at smontgom@bcgsc.bc.ca

After, you have defined your new service you should be able to share it with the world using Chinook. Visit our online applet to see if our discovery service has picked it up (in development).