5 Supplementary Information

The XML is a hierarchical specification. The structure of the protocol specification is specified using a DTD (Document Type Definition). The DTD is as follows.

Pipeline DTD description

```
<pipeline_setup>
  <datasource_setup>
  <iohandler_setup>
  <pipeline_flow_setup>
  <initial_inputs_setup>
</pipeline_setup>
```

5.1 Datasource setup

This specifies the data sources (databases or files) that the pipeline connects to and the adaptor modules that interfaces with them. These adaptors are connected to iohandlers which are a series of method calls on these adaptor modules. These are grouped together as each of them could be connected to many iohandlers.

An example is as follows:

```
my $db = Bio::EnsEMBL::DBSQL::DBAdaptor->new(-dbname=>”homo_sapiens”,
                                             -user  =>”ensembl”);
```

This one would translate into the dbadaptor XML specification as follows.

```
<dbadaptor id="1">
  <dbname>homo_sapiens</dbname>
  <driver>mysql</driver>
  <host>localhost</host>
  <user>ensembl</user>
  <password></password>
  <module>Bio::EnsEMBL::DBSQL::DBAdaptor</module>
</dbadaptor>
```

If the datasource is not a database but a file or some encapsulating module (API), then stream adaptor should be used. It is specified as follows.

```
<stream_adaptor id="2">
   <module>Bio::DB::Fasta</module>
</stream_adaptor>
```

5.2 IOhandler_setup

This groups together all the iohandlers that are required for input/output handling of runnables (analyses).
An IOHandler is a series of method calls on a datasource adaptor. Each IOHandler could be used by to more than one analysis if they use the same data source and object type for inputs or outputs.

For the above database adaptor, if a gene is to be obtained, the method call would be as follows:

```
my $gene = $db->get_GeneAdaptor->fetch_by_stable_id('XXXXX');
```

This would translate into the IOHandler XML specification as follows:

```
<iohandler_setup>
  <iohandler id="1">
    <adaptor_id>1</adaptor_id>
    <adaptor_type>DB</adaptor_type>
    <iohandler_type>INPUT</iohandler_type>
    <method>
      <name>get_GeneAdaptor</name>
      <rank>1</rank>
    </method>
    <method>
      <name>fetch_by_stable_id</name>
      <rank>2</rank>
    </method>
  </iohandler>
```

Note that if the runnable inputs are simple values(like filenames) which do not require fetching methods, no IOHandler need to be specified.

5.3 Pipeline_flow_setup

This specifies the analysis and rules of the pipeline. Analysis refer to the runnables that will be used in this pipeline while the rules specify the order in which these analysis are to be run, including any specific pre-processing actions that are to be carried out.
Each analysis specified here requires a Runnable that should follow the `Bio::Pipeline::RunnableI interface`. A runnable may encapsulate more than one wrapper depending on how the wrapper has been designed. Biopipe does not dictate how wrappers are written.

A blast program with parameters like '-p blastx -e 0.001 -F SEG+XNU' and on a db file 'db.fa' will be specified in XML as follows.

```
<analysis id="1">
   <logic_name>blast</logic_name>
   <runnable>Bio::Pipeline::Runnable::Blast</runnable>
   <db>swissprot</db>
   <db_file>/data/db.fa</db_file>
   <program>blastall</program>
   <parameters>-p blastx -e 0.001 -F SEG+XNU</parameters>
   <output_iohandler id="5"> </output_iohandler>
 </analysis>
```

If there are two analyses specified as above, to indicate that analysis 2 would be run after analysis for same input name that analysis 1 was run but with different method of fetching inputs (different iohandler), a rule would be specified as follows.

```
<rule>
  <current_analysis_id>1</current_analysis_id>
  <next_analysis_id>2</next_analysis_id>
  <input_iohandler_mapping>
  <current_analysis_iohandler_id>1</current_analysis_iohandler_id>
  <next_analysis_iohandler_id>2</next_analysis_iohandler_id>
  </input_iohandler_mapping>
  <action>COPY_ID</action>
</rule>
```

One of the actions of the rule is WAITFORALL which would only start the next analysis after it had finished running the current analysis for all its inputs. An example where you would need this is that the Blast may be needed to be run on all the contigs before the Tribe clustering analysis is started.

The input names for the analysis could be set using special module called Bio::Pipeline::Runnable::Datamonger. This special module allows all the flexibility that is needed to preprocess the inputs for an analysis. A Datamonger is specified by InputCreates and Filters. An InputCreate modules has to inherit from Bio::Pipeline::InputCreate.pm and a Filter from Bio::Pipeline::Filter.pm. A DataMonger runs the Filter and InputCreates on its inputs (mostly a list of ids) and creates the inputs for the next analysis. The Plugging in of Filter and InputCreate allows the flexibility that the pipeline specification and design cannot cater. A Datamonger could be specified as follows.

```
<analysis id="1">
  <data_monger>
    <input>
<!-- Inputs that the datamonger itself needs, on which the Filter and
InputCreate would be run -->
      <name>Ensembl_ids</name>
      <iohandler>1</iohandler>
    </input>
    <input>
      <name>BioperlDB_ids</name>
      <iohandler>2</iohandler>
    </input>

    <filter>
       <module>feature_coverage</module>
```

```
            <rank>1</rank>
            <argument>
                 <tag>feature_type</tag>
                 <value>xxx</value>
             </argument>
            <argument>
                 <tag>xxxxx</tag>
                 <value>xxx</value>
             </argument>
          </filter>

         <input_create>
            <module>setup_genewise</module>
            <rank>1</rank>
            <argument>
                 <tag>contig_ioh</tag>
                 <value>3</value>
             </argument>
            <argument>
                 <tag>protein_ioh</tag>
                 <value>3</value>
             </argument>
            <argument>
                 <tag>data_handler_id</tag>
                 <value>3</value>
             </argument>
            <argument>
                 <tag>dbadaptor</tag>
                 <value>Bio::Pipeline::SQL:DBAdatptor</value>
             </argument>
         </input_create>
       </data_monger>
    </analysis>
```

5.4 Input_setup

Inputs for the first analysis could be setup using a datamonger. This is the preferable way of setting up the initial inputs when the analyses are to be run for a large set of inputs. But when only few inputs are to be used, then they can be specified as follows.

```
<initial_input_setup>
   <analysis_id> 1 </analysis_id>
   <input>
     <name>xyz1</name>
     <iohandler_id>3</iohandler_id>
   </input>
   <input>
     <name>xyz2</name>
     <iohandler_id>4</iohandler_id>
   </input>
</initial_input_setup>
```

But for the downstream analyses, the inputs are either copied over from the inputs of the previous analysis(only names could be copied over and the iohandlers could be different) or setup cleanly using a Datamonger.

For example, in bioperl-db, a sequence maybe fetched with the following snippet of code:

```
use Bio::DB::BioSQL::DBAdaptor;

my $db = Bio::DB::BioSQL::DBAdaptor->new(-user=> 'root',
                                         -dbname=> 'seqdatabase',
                                         -host => 'host1',
                                         -driver => 'mysql');

my $seq_adaptor = $db->get_BioDatabaseAdaptor;

my $seq = $seq_adaptor->fetch_Seq_by_display_id('104K_THEPA');
```