# CGA XML Description

This document describes the XML files used to describe the flow of a pipeline.

# Pipeline XML Description

Each pipeline will be controlled by an XML description file. The outer tag is <pipeline>.

## pipeline

<pipeline name="...">  
 <file />  
 <dir />  
 <foreach />  
 <step />  
</pipeline>

A <pipeline> tag contains one or more <input> tags and one or more <step> tags. It contains exactly one <outputdir> tag. While it is unusual for a pipeline not to specify any output file, this spec does not require an <output> tag. For instance, a pipeline could specify a set of steps that populate a database instead of creating files. The <tempfile> tag is also optional.

## file

The <file> tag specifies a file that is to be used in the pipeline. The filespec (complete file path) may be hardcoded in the tag, or refer to a positional command line parameter.

<file id="..." input=”...” temp=”...” in\_dir=”...” filespec="..." />  
<file id="..." input=”...” temp=”...” in\_dir=”...” parameter="..." />  
<file id="..." input=”...” temp=”...” in\_dir=”...” based\_on=”...”  
 pattern=”...” replace=”...” />

The input attribute is optional, and can have the values True or False (case-blind). If omitted, the attribute defaults to False. Input files must exist at the start of the pipeline. A pipeline is free (even encouraged) to check that all input files exist at the start of the pipeline, to fail early.

The temp attribute is optional, and can have the values True or False (case-blind). If omitted, it defaults to False. Temp files will be deleted at the end of the pipeline’s execution. A filespec attribute may be specified on a temp file. If so, the file will have the name provided, otherwise, it will have an arbitrary name generated, as if by Python’s tempfile.NamedTemporaryFile().

The in\_dir attribute is optional. If present, it specifies the directory id of the directory in which the file or directory is located or to be created.

If the filespec form is used, then that specific file is used as the input to the pipeline. This could be used, for instance, for the path to a reference genome.

If the parameter form is used, the value of the attribute is the positional argument's number (1-based) from the command line invoking the pipeline.

It is possible to create files where the filename is created by manipulating the basename of another file, by using the based\_on attribute. If the based\_on form is used, the parameter and filespec attributes must not be used. The based\_on value specifies a previously-declared file id. This is used in conjunction with either the append attribute, or the combination of pattern and replace attributes. If used, the pattern and replace attributes must both be specified. The append attribute is mutually incompatible with the combination of pattern and replace.

If the append attribute used, its value is simply appended to the basename of the file specified in the based\_on attribute.

If the pattern and replace attributes are used, the file spec for this file will be generated as if by using the Python re.subs() operation applying the values of pattern and replace to the basename of the file specified in the based\_on attribute. This capability is needed, for instance, using samtools, where some commands need the trailing “.bam” stripped off of a file name.

## dir

The <dir> tag specifies a directory. As with files, a dir can specify an input or output directory. The difference is that an output directory need not exist on startup.

<dir id="..." input=”...” filespec="..." in\_dir=”...” create=”...”  
 based\_on="..." pattern="..." replace="..." append=”...” />

A directory which is specified as an output directory and exists prior to the start of the pipeline must be writable.

The lexically first <dir input=”False”> or <dir> (defaulting input to False) tag specifies the default output directory for output files specified with a relative path. It is also the location where various run logs are stored. If all dir tags are specified as input, the first one that is writable will be used as the default output directory. If no <dir> tags are specified, or if none are writable, the current working directory will be used.

The in\_dir attribute is optional and used only on output files or directories. If present, it specifies the directory id of the directory in which the new file or directory is to be created.

The create attribute is optional and defaults to True. It controls whether the directory is created by the pipeline prior to use. We have to be able to control this because some tools want to create output files in an existing directory, and others want to create the directory and will fail if the output directory already exists.

The based\_on, pattern, replace, and append attributes are as documented for the <file> tag.

## filelist

Occasionally, some phase of the processing may require a set of files that can’t be specified at pipeline design time, for instance a set of bam files to be merged. The <filelist> tag allows specifying a regular expression pattern to be matched in a certain directory. At execution time, all the files matching this pattern will be provided in a space separated list.

<filelist id=”...” in\_dir=”...” pattern=”...” />

The pattern will be processed as if by Python’s re.match() function, against all files in the specified directory.

The <filelist> tag can be used anywhere the <file> tag can be used.

## step

A step is a logical phase of the pipeline. A <step> tag contains one or more tool tags.

<step name="...">  
 <tool />  
</step>

## foreach

The <foreach> tag allows a step to be run on several files in a directory identified by the dir attribute.

***IMPORTANT NOTE:*** The “iterations” of the <foreach> processing may happen in parallel. All processing of one set of files must be completely independent of the processing of another set of files.

The files to be used within this directory are identified by pattern matching the file names, so it is useful primarily when the files are systematically named, for instance, the output directory of an Illumina sequencing run. When there are groups of files that need to be processed together, for instance, paired fastq files, the tag allows constructing the name of the other input or output files based on the name of first one, again via regexp pattern matching / pattern replacement. This could be used, for instance, in performing paired end alignment.

<foreach dir=”...”>  
 <file id=”...” pattern=”...” />  
 <related id=”...” inout=”...” pattern=”...” replace=”...” />  
 <step />  
</foreach>

The <foreach> tag processes a set of files in the directory whose id is specified in the dir attribute. The order in which the files are processed is unspecified; in a cluster environment, they may be processed in parallel.

The <file> tag’s pattern attribute specifies the python regex pattern that will be used to select files for processing; it will be applied to each filename in the directory as if by using python’s re.match() function. There must be exactly one <file> tag in a <foreach>. Any filenames that match will be processed. The tag’s id attribute specifies the id this file will be referenced by in the step’s tools. Since the files identified by this tag must exist to be pattern matched, they are inherently classed as input files.

The <related> tag specifies another file in the same directory, or to be created. The id attribute specifies the id by which this file will be referenced. The inout attribute’s value shall be either in or out. The pattern and replace attributes specify python regex patterns which are used to modify the controlling filename into the desired filename as if by python’s re.sub() function.

For each set of files identified by the above tags, execute the operations specified by the <step> tag(s).

### Example

Assume that the directory /home/example contains the following files, output from an Illumina sequencing run, that we now need to align with our pipeline:

A2\_S1\_L001\_R1\_001.fastq  
 A2\_S1\_L001\_R1\_002.fastq  
 A2\_S1\_L001\_R2\_001.fastq  
 A2\_S1\_L001\_R2\_002.fastq

<inputdir id=”indir” filespec=”/home/example” />  
<foreach dir=”indir”>  
 <file id=”end1” pattern=”.\*\_R1\_.\*fastq” />  
 <related id=”end2” inout=”in”  
 pattern=”(.\*)\_R1\_(.\*fastq)”  
 replace=”\1\_R2\_\2” />  
 <related id=”sam” inout=”out”  
 pattern=”(.\*)\_R1\_(.\*)fastq”  
 replace=”\1\_\2sam” />  
 <step name=”Alignment”>  
 <tool name=”bwa” description=”run\_bwa.xml”  
 input=”end1,end2” output=”sam” />  
 </step>  
</foreach>

The <foreach> tag would result in two invocations of the Alignment step, the first processing A2\_S1\_L001\_R1\_001.fastq and A2\_S1\_L001\_R2\_001.fastq and producing output file A2\_S1\_L001\_001.sam. The second invocation would process A2\_S1\_L001\_R1\_002.fastq and A2\_S1\_L001\_R2\_002.fastq, producing A2\_S1\_L001\_002.sam.

## tool

A tool takes one or more inputs, and creates one or more outputs. For instance, it could be an aligner, or a simple sam to bam converter.

<tool name="..."

input="id [, id]..."

output="id [, id]..."

description="..." />

The description attribute specifies the basename of another XML file which describes the tool and its parameters; see Tool XML Description section, below. The driver will search for the XML file in a set of TBD directories, allowing per-user customization, followed by a system-wide configuration. I am also considering searching for an environment variable <name>.DESCRIPTION, where <name> is the value of the name attribute. Description searching will be described in a separate spec.

The input and output attributes are used to map from the pipeline file id space to a tool file id space. In this way, a tool description file can be utilized in multiple pipelines or steps without having to have a global id assignment registry.

The ids listed in the input or output attributes are in the pipeline / step id space. The order in the list determines the id they will have in the tool’s description file. The first id in the list in this tag will receive the id “in\_1” or “out\_1”, as appropriate, in the tool description XML file. Input and output lists use separate indexes, so the first file in each list will be \_1, the second in each list will be \_2, etc.

# Tool XML Description

A tool XML description file is separate from a pipeline description file. This allows utilizing a tool description in multiple pipelines, and also conveniently tweaking a tool’s configuration without editing a whole pipeline file.

The outer tag is <tool>.

## tool

<tool name=”...” tool\_config\_prefix=”...” threads=”...”  
 walltime=”...” error\_strings="...">  
 <description />   
 <option />  
 <command />  
 <file />  
 <validate />  
 <module />  
</tool>

The tool\_config\_prefix, threads and walltime attributes are optional. The <description> tag is optional. If not present, threads defaults to one, and walltime defaults to one hour. There may be zero or more <option> tags. There must be one or more <command> tags. If multiple <command> tags are specified, they are executed serially, in lexical order.

See the <option> tag for a description of the tool\_config\_prefix attribute.

The threads attribute specifies the maximum number of threads any command in this tool will use. If not specified, the tool will be run in one thread. If executing in a cluster environment, it specifies the number of processors that will be allocated. If the commands in a tool run for a significant length of time and use widely different numbers of threads, consider splitting it into multiple tools within one step, so that each tool can specify the number of threads appropriate to its command(s).

The error\_strings attribute allows specifying a comma-separated list of strings. If specified and the tool's error output contains any of the strings, the tool is deemed to have failed, even if its last command returns a zero exit status.

When executing in a cluster environment, a tool will be scheduled as a job.

## description

The <description> tag contains free form information about the tool.

<description> ... </description>

## option

The information in the <option> tag is used in the <command> tag.

<option name="..." from\_file=”...” command\_text=”...” value="..." />

The <option> tag must contain all the defined attributes. See the description of the <command> tag for how these are used.

Option names are in the same name space as the tool file ids, but separate from the name space of the invoking pipeline. All names in the tool’s option name / file id name space must be unique.

Occasionally, the value associated with an option is complex and is derived from the processing being done in the pipeline. An example is specifying read group information during bwa alignment. In this case, the from\_file attribute specifies the id of a file containing a single line which is used as the option’s value instead of what would have been specified in the value attribute.

For each option specified, if the tool\_config\_prefix attribute is specified in the <tool> tag, option processing will search for an configuration file in a manner TBD. If the configuration file exists and the option’s name is listed, the value(s) from the configuration file will be used instead of the values specified in this tag. A full description of configuration processing is in a separate document TBD.

## command

The <command> tag specifies how to construct the command line that will be executed is constructed.

<command  
 program=”...”  
 delimiters=”...”  
 stdout\_id=”...”> ... </command>

The command line to be executed will be constructed from the value of the program attribute followed by the text contained within the tag.

The text within this tag consists of literal text that will be inserted in the command line after the program filename, interspersed with option names or file ids enclosed within braces ({}). Option names in this context means the values of the name attributes in the <option> tags.

If a command to be executed uses braces in its syntax (e.g., the find command, it is necessary to specify an alternate set of delimiters or an escaping mechanism. This specification allows for alternate delimiters. If the optional delimiters attribute is specified, its value is a two character string. The first character is used in place of open brace ({) to indicate the start of an option name, and the second character is used in place of close brace (}). If a literal brace is required in the command line, use the delimiters attribute to specify an alternate pair of characters.

The optional stdout\_id attribute allows IO redirection. The process’ stderr output channel is managed by the infrastructure and is not redirectable at this time.

When a brace-enclosed option name is encountered, the value of the command\_text attribute will be inserted, if present. Then the value of the value attribute if specified, and finally the filename represented by the id attribute, if specified. But see the specification of tool configuration files for processing that overrides the values specified in the XML file.

All of the text within the <command> </command> tags will be reconstructed as a single line, with any line breaks treated as spaces and all spaces collapsed.

### Example 1

The following is a fragment from a tool description file, intended to illustrate the use of an option in a command line, not to be complete. The fragment

<tool>  
<option name=”bowtie\_max\_multi” command\_text=”-m” value=”40” />  
<command program=”bowtie”>{bowtie\_max\_multi} ... </command>

would result in the following being emitted for the command:

bowtie –m 40 ...

### Example 2

The following fragment demonstrates use of a file id being used in a command. This fragment assumes that the file “fred.sam” was passed as id “out\_1” into the tool. The fragment:

<command program=”bowtie”>–s ... {out\_1}</command>

would result in the following being emitted:

bowtie –s ... fred.sam

### Example 3

The following fragment demonstrates use of the delimiters attribute. It assumes that the pipeline’s outputdir is named “myoutput” and its id was passed as id out\_3. The fragment:

<command delimiters=”%%” program=”find”>%out\_3% -name ”\*.tmp” –exec rm {} \+</command>

would result in this command being executed:

find myoutput –name=”\*.tmp” –exec rm {} \+

(Please excuse Word’s insistence on using “smart quotes” in this example...)

## validate

The validate tag allows us to specify a file for validation, that it has not been altered since the pipeline  
was initially validated. This does not need to be specified for the command program names; they are automatically added to the list. However, for java jar files, etc., that aren’t the initial word in a command, we can check them this way. The tag has two forms:

<validate>filepath</validate>

<validate id=”...” />

In the first form, the file is searched for by name on the PATH, or in the current working directory. In the second form, the file is checked for using the “in\_X” and “out\_X” file IDs passed into the tool.

## version\_command

One of the requirements on this pipeline tools is repeatability. Part of that is the ability to log the versions of tools that are used. When specified within a <command> tag, the <version\_command> tag allows us to specify a command that will cause the tool to emit its version string to either stdout or stderr.

<version\_command output="..."> ... </version\_command>

The value of the output attribute shall be either "stdout" or "stderr". This identifies the IO channel on which the result of the command is printed. If not specified, defaults to "stdout".

If this tag is specified, the version information for the tool will be recorded in a version log.

Example:

<version\_command output="stdout">--version</version\_command>

Note that since this tag is within the <command> tag, we don’t specify the program name. That is contained as an attribute of the <command> tag.

## file

See description above. When specified in the context of a tool description file, the parameter form is not allowed. If a temporary file is declared in the context of a tool, that temporary file is deleted at the end of the tool’s execution.

# Changes:

|  |  |  |
| --- | --- | --- |
| Date | Who | What |
| 21-Jan-2013 | Simons | Renamed <process> to <step> to avoid confusion with UNIX processes.  Discussed file and option-name name spaces and uniqueness requirements.  Added <version\_command> tag.  Added examples to <command>.  Responded to DOW comments.  Proposed removing the <input> and <output> tags from the <step> tag.  Clarified that the Tool description file ids for files passed in use separate indexes for input and output file. |
| 24-Jan-2013 | Simons | Added <inputdir> and <foreach>. |
| 18-Feb-2013 | Simons | Many small tweaks for problems found during implementation. |
| 4-Mar-2013 | Simons | Add tool subtags <validate>, <wallclock> and <modules> |
| 11-Mar-2013 | Simons | Merged input and output, and inputdir and outputdir, into file and directory, each with an input attribute. Added option attribute from\_file and file attributes based\_on and replace. |
| 13-Mar-2013 | Simons | More rationalization of files. Removed the <tempfile> tag, added a temp attribute. Temp files may be named, or have generated names. Tools can have named files; they don’t have to be passed in or out of the tool invocation. Use for instance with tool-specific data files. |
| 29-Mar-2013 | Simons | Removed capability to redirect stderr, so that we can add the error\_strings attribute. Added the append attribute to <file>, for easy (easier) filename construction in simple cases. |
| 1-Apr-2013 | Simons | Added the <filelist> tag... No foolin! |