# CGA Pipeline: Option Override Processing

## Background

CGA Pipeline tool description XML files contain the tool options and parameters for a run. These determine the pipeline’s default tool parameters (which may or may not be the tool’s default parameters).

For use in a clinical setting, the current view is that these pipeline defaults will not be able to be overridden, however, a very small set of them might be.

However, for research use, it is important that the default parameters be easily overridden with alternate values. It is desirable that these alternate values be captured for documentation and repeatability purposes.

It is also desirable that a job configuration/submission GUI should be able to use this interface to configure the job, and not have to modify the underlying pipeline XML files.

## High level design

To meet these goals, the pipeline will allow option override files.

There are two designs under consideration, comments are sought. Will either design be satisfactory by itself, or should we support both designs?

Both designs feature a means to pass a specific file path or directory path in which to find the override file(s) into the pipeline through a TBD means (command line option?).

I am considering a search for override files in some specific locations if an override location is not passed into the pipeline; comments on this sought.

* Pro: It makes using overrides easier; they don’t have to be specified on every invocation.
* Con: It becomes too easy to run with altered parameters and not realize it. This could be ameliorated by stating the overrides in effect at the top of the log, but... does *anyone* read log files before diving right into the results?

For the description of the contents of these files in either option, see the Detailed design section, which follows.

### Design 1: One override file for the whole pipeline run

In this design all of the tool option overrides are in a single file.

* Pro: this seems to be easier for a GUI to use. It can accumulate the configuration choices and write out a single config file.
* Con: if being used by a command line researcher who is interested in altering options for multiple tools, this leads to a geometric explosion of files that need to be maintained. For instance, if she wants to explore three different configurations for each of three tools in the pipeline, this design leads to 27 configuration files, as opposed to 9 files needed under Design 2.

### Design 2: An override directory for the run, separate file for each tool

In this design the pipeline is pointed to a directory containing multiple option configuration files, one file for each tool in the pipeline for which an override is desired.

* Pro: Easier to manage in a research / command line environment (see Con in Design 1).
* Con: Possibly a little more difficult for a GUI to use, but probably not much more difficult.

### High level design summary

* The pipeline will be made aware of the presence and location of tool config files in a manner TBD.
* I’m currently leaning toward either using both designs together, or Design 2 if we decide to support only one.
* Comments and alternate suggestions/designs are sought!

## Low Level design

The tool configuration file is tab separated text, with one row per option that is to be changed from its default. The file may contain comment lines, where the first character in the row is a pound (#) character.

Each non-comment row contains from two to four tab-separated fields, explained below.

The four columns in the configuration file correspond to the four attributes in the <option> tag in the Tool description file, in this order:

1. name
2. command\_text
3. value
4. file id

If a field is null or non-existent in a particular row, the value from the corresponding attribute in the XML file is used. If a tool option in the XML file does not have a corresponding row in the config file, the XML file’s values are used.

***Note:*** *it is unclear whether overriding the file id would ever be useful. Perhaps it should not be included in the config file.*

### Name field

If only High Level Design 2 is supported, the name column will consist of only the value of the name attribute in the XML file.

If we also (or only) support High Level Design 1, the name column will consist of *<config\_file\_prefix>.<option\_name>*, where *<config\_file\_prefix>* is the value of the <tool> tag’s config\_file\_prefix attribute, and *<option\_name>* is the value of the <option> tag’s name attribute.

### File name and config file searching

Under High Level Design 2, the files will be named *<tool\_name>*.config, where *<tool\_name>* is the value of the <tool> tag’s name attribute, which will be searched for in a directory which is passed in to the pipeline. If High Level Design 1 is supported, the single file’s name will be passed in to the pipeline.

Searching alternative locations for tool configuration information is TBD.