Passing parameters to Extractors at runtime

As I have worked with Clowder for over a year now, I have some to appreciation the extractor pattern in its simplicity. I start this post mentioning that, because I feel that what I am about to suggest MUST strive to continue that spirit of simplicity, even though it might sound complex.

Background

Having been involved in extractor creation, it didn't take long before I was faced with a situation where I wanted an extractor to behave in two different ways depending on context outside of the file(s) being processed. The two different behaviors didn't really seem like two different extractors, but more like the same extractor with branching logic based on a parameter. There are two situations that I encountered, but you have probably even more examples from your own experience.

1) Within a space an extractor needs to behave slightly differently on different invocations. The extractor wrapped another application that took additional parameters and we needed to pass different parameters in different situations.

2) In one space we wanted an extractor to behave one way, in a separate space we wanted the extractor to behave slightly differently. The differences were very minor, so the code was 99% the same.

In both of the above cases our only option was to create separate extractors. While not terrible, it really seems like it shouldn't be necessary to do that and if we could pass parameters to the extractors at runtime, then we could significantly extend the utility of an extractor to be even more flexible and powerful.

Proposal

Add the capability of passing parameters to extractors through a couple of different mechanisms that allow for various situations to be handled differently, giving the admins and users considerable flexibility. The following is the approach that comes to my mind of how this might be best accomplished. This is by no means prescriptive but only intended to help illustrate the concept.

1. Add the ability to have extractor registration include "Parameters". Parameters should not be considered strongly typed. Parameter registration should declare that name of the parameter and whether it is required (or optional).
2. The Instance level extractor description page should include these parameters in a list (optional whether it in the default view or requires you to access a link to get to) with the ability to provide and save a default value at the instance level.
3. The Space level extractor description page should default from the instance level and allow for the Space administrators to change and save values at the Space level.
4. (I hope this makes sense) At the dataset level, I propose that we use the existing metadata capability to allow for the creation on a type of metadata (type being controlled by URI not datatype) that designate the label/value pair as being for parameters. If a value is stored against the correct metadata label then that value is intended to be used as input to the extractor that has a parameter with that name.
5. Finally, to provide the lowest level of granularity on affecting extractor execution...if a metadata label exists (same metadata label referenced in #4) on a file being processed that value is intended to override all previous values and all parent levels.
6. When an extractor execution is being processed and queued, have Clowder gather the available parameters starting at the Instance level and then walking down from there to Space level, Dataset level and finally File level. Each level overrides the previous level if the parameter is provided (not blank). This final set of parameters is package into a name/value pair object (probably JSON) and included in the payload to queue for the extractor submission.
7. (Continuing #6) If a "required" (not optional) parameter is still missing after evaluating the available levels...AND... the extractor is being submitted by the user via the UI...prompt the user for the missing value. If the extractor is being auto triggered or invoked via an API, then log an error and don't finish the extractor queueing.
8. When an extractor is submitted via the API, the developer is allowed to provide a set of parameters via a JSON string (same as referenced in #7). This does not rationalize the auto generated parameters with the developer provided parameters, it is an either/or.
9. (Optional) Also, when the submission takes place the actual parameters that were used (rationalized and passed to the extractor) should be stored in the parameter metadata labels (see #4) to track for later reference which values where used when the extractor ran. This would allow users to discover that one file was run with one set of parameters long after the execution occurred and defaults may have been changed, facilitating the ability to conditionally re-process some files with new parameters while skipping those that don't require it.

Please feel free to build/comment on any and all points/suggestions.

Thanks!

Dan