IngestTool Control Data

The CollectionConfiguration file controls the activities of the ingest tool when working with objects in a specific collection.

```
<cc:collectionConfiguration
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:cc="http://schemas.dlib.indiana.edu/collectionConfiguration/"
    xsi:schemaLocation="http://schemas.dlib.indiana.edu/collectionConfiguration/
        http://www.dlib.indiana.edu/lib/xml/collectionConfiguration/collectionConfiguration.xsd">
</cc:collectionConfiguration>
```


Main section

Details:

- /cc:collectionConfiguration – This is the root element of the CollectionConfiguration. It should always be namespaced as cc: with the shown uri and schemaLocation.

Ingestion

Sample:

```
<cc:ingestionID>3</cc:ingestionID>
<cc:accessCode>C64CF8D17F612A9C26654A2B6C21FF8E5B4830F46756DEF988AE8F57590376FC</cc:accessCode>
```

Details:

- /cc:collectionConfiguration/cc:ingestionID – This is the ingestion ID generated by a request for a new ingestion from the Ingest Tool. This should be used for any unique ingestion. It can be reused on the same ingestion in the case of failure, but if you are providing new data (either content or metadata) then you should get a new ingest id and integrate that into the current CollectionConfiguration.
- /cc:collectionConfiguration/cc:accessCode – This is the access code generated by a request for a new ingestion from the Ingest Tool. See /cc: collectionConfiguration/cc:ingestionID for more details.

The ingest tool is intended to run a single ingestion for a collection; however, a collection may have numerous ingestions over its lifetime. For example, items may be added, modified, or removed as time goes on, and errors may be fixed. This could be handled by altering the collection configuration and re-running the ingestion, but Eric opted to require a separate collection configuration for each new ingestion. This was primarily to handle re-starts of large ingests. We should re-think this, as it is almost always better to do many small ingests rather than a few very large ones.

This causes a complexity in the creation of the collection configuration when a new ingest is required. Each new ingest need to create a new collection configuration xml with a distinct ingestion id, and each ingestion id will have a distinct access code. The ingestion id and access code are generated by the ingest tool servlet, which is documented on the wiki.

A side effect of having this tracking database is that errors and ingests could be more easily tracked to show more historical information, and most importantly, the various ids (short, internal, purl, pid) can be trivially mapped to one another.

Logging

```
<cc:logging>
    <cc:logger type="email" level="summary" trigger="debug">erpeters@indiana.edu</cc:logger>
    <cc:logger level="all" type="std" />
</cc:logging>
```

Details:
The only meaningful logging for the end user is summary/email. All logging also goes through log4j, which is more appropriate for development logging. The file and std logging types were originally intended as development logging, and are now unnecessary.

- /cc:collectionConfiguration/cc:logging – this defines the logging mechanisms for this ingest. It is required and must contain 1 or more /cc:collectionConfiguration/cc:logging/cc:logger elements.
    - /cc:collectionConfiguration/cc:logging/cc:logger – This defines a single logging mechanism
    - /cc:collectionConfiguration/cc:logging/cc:logger/@level – Defines the level of debugging. Must be one of: "none", "summary", "fatal", "error", "warn", "info", "debug", or "all".
    - /cc:collectionConfiguration/cc:logging/cc:logger/@type – Defines where to send the logs. Must be one of: "email", "file", "std".
    - /cc:collectionConfiguration/cc:logging/cc:logger/@trigger – Defines the level of logging when in summary mode. Must be one of: "none", "fatal", "error", "warn", "info", "debug", or "all".

Collection

```
<cc:collectionName>Hohenberger</cc:collectionName>
<cc:contentModel type="image" />
<cc:collectionID>lilly/hohenberger</cc:collectionID>
<cc:collection pid="iudl:10" />
```

Details:

- /cc:collectionConfiguration/cc:collectionName – This is the name of the collectionn being ingested. Just used as a label.
- /cc:collectionConfiguration/cc:contentModel – This defines the data model to use for the ingestion. It controls which portions of the Java code will execute. This must be one of: image, paged.
- /cc:collectionConfiguration/cc:collectionID – This is the prefix used to convert a shortItemID into a fullItemID. It must match results in the input metadata IDs, and must result in a valid purl when added to the system purl base, which is configured by the IngestTool Server. See the Identifiers page for more information on IDs. Note the lack of leading and trailing slashes.
- /cc:collectionConfiguration/cc:collection – defines the collection in Fedora.
    - /cc:collectionConfiguration/cc:collection/@pid – this is the Fedora pid for the collection object. This object must already exist (created manually).

Item

```
    <cc:existingItem>
        <cc:fedoraItemExists action="purge" />
        <cc:databaseItemExists action="skip" />
    </cc:existingItem>
<cc:items>
```

- Item ID Configuration
- Item Title Configuration

```
</cc:items>
```

Details:

- /cc:collectionConfiguration/cc:existingItem – This defines what the ingest process should do when it encounters an item that already exists in Fedora or in the tracking database. The presence will be detected by the PURL in the default DC datastream.
    - The action attribute must be one of the following:
        - alter – this triggers an alteration of the existing item, this is the default behavior if the element is not specified in the config.
        - purge – this triggers a delete of the existing item, followed by a clean ingest of the new item.
        - skip – this triggers the item to be skipped altogether.

There is no implementation to process the databaseItemExists at this time, and there is no checking for erroneous cases (like an item existing in the database, but not in Fedora). This is complex to implement, because there must be rules for dealing with multi-object items. What happens if half the pages in a book are ingested, and an error occurs? Should the entire book be re-ingested?

BookID

```
<cc:bookID>
    <cc:dataReference source="mods" />
    <cc:defaultDataReference action="skip" />
</cc:bookID>
```

The cc:bookID block defines the source data for a paged collection. This example shows that the master data source is the mods records, and any item encountered that is not in the mods should be skipped (ignored). For details on where the mods is located, see the descriptive metadata specification.

Title

```
<cc:title>
    <cc:dataReference source="mods" />
    <cc:defaultDataReference action="text">[unknown]</cc:defaultDataReference>
</cc:title>
```

The title block data for each item is stored in the [title] placeholder. Please see cc:idformat, below, for more information.

- /cc:collectionConfiguration/cc:items/cc:title – This defines where to find the title for any particular item.
    - /cc:collectionConfiguration/cc:items/cc:title/cc:dataReference – This is the same as /cc:collectionConfiguration/cc:items/cc:itemID/cc:dataReference, but deals with the title instead of the ID.
    - /cc:collectionConfiguration/cc:items/cc:title/cc:defaultDataReference – This is the same as /cc:collectionConfiguration/cc:items/cc:itemID /cc:defaultDataReference, but deals with title instead of ID.
    - /cc:collectionConfiguration/cc:items/cc:title/cc:defaultDataReference/@action – Must be one of the following:
        - text – the specified content will be used as the title when the title is not found by the dataReference specifications.
        - skip – the item will be skipped when the title is not found by the dataReference specifications.

Item ID

```
<cc:itemID>
    <cc:dataReference source="masterContent" />
    <cc:dataReference source="ead" />
    <cc:defaultDataReference action="skip" />
</cc:itemID>
```

The cc:itemID block defines the source data for an image collection, or for the pages in a paged collection. This example shows that the master data source is the masterContent, and any items not found in the master content should be skipped (ignored). Details on where the master content is located, is specified below in the master content specification.

Details:

- /cc:collectionConfiguration/cc:items/cc:itemID – This defines how to build the list of Items to ingest. The processor will search for ID's in each of the collections listed, in the order shown, and will build a master ID list from the superset of all items. If an item is found in a collection that is not listed, the default action will be performed.
    - /cc:collectionConfiguration/cc:items/cc:itemID/cc:dataReference – This defines a location to search for ID's.
        - /cc:collectionConfiguration/cc:items/cc:itemID/cc:dataReference/@source – this is the actual location to search, may be one of:
            - ead – the ead will have a shortID.
            - masterContent – id will be pulled from the masterContent filename: everything between the last / character and the last . character will be interpreted as a shortID.
            - mods – not yet supported
    - /cc:collectionConfiguration/cc:items/cc:itemID/cc:defaultDataReference – This is what to do with the items don't have an ID available. For instance, if the EAD file is missing the appropriate field.
    - /cc:collectionConfiguration/cc:items/cc:itemID/cc:defaultDataReference/@action – This is what to do when the items don't have an ID available. Must be one of the following:
        - skip – item will be skipped

Overview Content

```
    <cc:overviewContent derivativeType="pdf">
        <cc:source location="url">http://inharmony.dlib.indiana.edu/inharm/[institution]/[institution]-[score]-
[copy]/</cc:source>
        <cc:filenameFormat>[institution]-[score]-[copy].pdf</cc:filenameFormat>
    </cc:overviewContent>
```

The filename format specifier defines the filename for the item. The data_label fields are placeholders for data that comes from the cc:idformat mapping.

Master Content

```
    <cc:masterContent type="image" subtype="tiff">
        <cc:source location="localfs">C:\Documents and Settings\erpeters\My
Documents\workspace\infrastructure\hohenberger\master_images</cc:source>
        <cc:extension>.tif</cc:extension>
    </cc:masterContent>
```

Derived Content

```
    <cc:derivedContent derivativeType="images">
        <cc:source location="url">http://inharmony.dlib.indiana.edu/inharm/[institution]/[institution]-[score]-
[copy]/</cc:source>
        <cc:extension item="thumb">-thumb.jpg</cc:extension>
        <cc:extension item="screen">-screen.jpg</cc:extension>
        <cc:extension item="large">-full.jpg</cc:extension>
    </cc:derivedContent>
```

This represents that 3 derivatives are expected to have been created for each master, 1 each: thumb, screen, and large. The data_label fields are placeholders for data that comes from the cc:idformat mapping (see below).

Note that derived content is expected to be generated by an outside source and already exist at the location specified.

Structural Metadata

```
<xs:element name="structuralMetadata">
<xs:complexType mixed="false">
<xs:choice minOccurs="1" maxOccurs="1">
<xs:element ref="cc:source" minOccurs="1" maxOccurs="1" />
</xs:choice>
<xs:attribute name="type">
<xs:simpleType>
<xs:restriction base="xs:string">
<xs:enumeration value="derived"></xs:enumeration>
<xs:enumeration value="mets"></xs:enumeration>
</xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:complexType>
</xs:element>
```

Descriptive Metadata

```
    <cc:descriptiveMetadata>
        <cc:metadataItem type="mods" authoritative="true" level="item">
            <cc:source location="localfs">C:\Documents and Settings\erpeters\My
              Documents\workspace\infrastructure\inharmony\mods.xml</cc:source>
            <cc:idFormat regexp="^([^\-]*)\-(.*)\-([^\-]*)$">
                <cc:mapping>institution</cc:mapping>
                <cc:mapping>score</cc:mapping>
                <cc:mapping>copy</cc:mapping>
            </cc:idFormat>
            <cc:datalookup key="title">/mods:modsCollection/mods:mods/mods:titleInfo[not(@type)]/mods:title</cc:
datalookup>
        </cc:metadataItem>
    </cc:descriptiveMetadata>
```

The ID number is retrieved using a hardcoded XPath in dlib.metadata.Mods, then parsed by the regexp above and the (matching) group values are saved in a hashtable with the key being [institution] for match 1, [score] for match 2, and [copy] for match 3. Warnings are thrown if a match count doesn't equal the number of mappings. These data mappings are then used for data lookup, derivative name generation, or other (currently undefined) locations. This is currently working but has not been fully developed into the full vision I had for it.

I found that the metadata is not going to be consistent across collections, even the collections that we are going over specifically with the infrastructure project in mind, subtle variations occur. For example, many collections use a single /mods:modsCollection/mods:mods/mods:titleInfo/mods:title entry as the title, so this is the default cc:datalookup for title in the MODS. However, with the addition of IN Harmony, there are multiple ...mods:title references, and the main title is the one that doesn't have a (sub) type attribute. So, while this variant title xpath could be used as a new default value, I saw the very real likeliness that various collections will have different metadata formats as time goes on. Therefore I created a mechanism and template for overriding some of the xpath expressions used for data lookup.

Technical Metadata

```
    <cc:technicalMetadata>
              <!--
        <cc:metadataItem type="mix" authoritative="true" level="masterContent">
            <cc:metadataGeneration>
                <cc:autoGeneration method="jhove">
 -->
                <!-- TODO:
                    <cc:augmentation>
                        <cc:source location="localfs">C:\Documents and Settings\erpeters\My
                         Documents\workspace\infrastructure\inharmony\inharmony_mix.xml</cc:source>
                    </cc:augmentation>
                -->
```

This augmentation block is not supported at this time at any level. The idea is that the mix provided here would somehow override, or fill in blanks, or somehow otherwise alter the pre-generated technical MIX MD for the items. The idea comes from the fact that many of the collections store technical MD in a MS Excel file, which should be stored with the item in some way. However, the process of doing this has not at all been developed. Excel does have a mechanism to export data to an XML format, which could be used as source for this, then transformed with an XSLT into MIX, which could then easily be used as an augmentation data source. However, that's about all the further my thoughts had gone with this mechanism. (Note: the XML output I'm referring to is not the MS Excel 'save-as xml' format, but a custom export to XML so that the structure could be standardized.)

```
<!--
                </cc:autoGeneration>
            </cc:metadataGeneration>
        </cc:metadataItem>
-->
```

This entire set of 3 comment blocks (above) is because the masters aren't available for MIX generation. This is really a little bit of a problem since the MIX is expected (not required, but expected) in the Fedora object, but the masters aren't available for IN Harmony. I have not brought this issue up to anybody at this time.

```
        <!--
        <cc:metadataItem type="mix" authoritative="true" level="overviewContent">
            <cc:metadataGeneration>
                <cc:autoGeneration method="jhove" />
            </cc:metadataGeneration>
        </cc:metadataItem>
            -->
```

This block is commented out because I currently don't support MIX for PDF's. It should, however, be fairly simple to implement, perhaps trivial. I just haven't gotten around to it yet.

```
        <cc:metadataItem type="mix" authoritative="true" level="derivedContent">
            <cc:metadataGeneration>
                <cc:autoGeneration method="jhove" />
            </cc:metadataGeneration>
        </cc:metadataItem>
    </cc:technicalMetadata>
</cc:collectionConfiguration>
```