"Back End" Working Group Report –2, July 22, 2005

Doug Hiwiller (recorder)

Issues raised by Front-end group for backend consideration:

1. e-mail verification
2. pool id's (user saved sets) addressed in data model
3. term distributed across works
    - not an issue… when the submit button is pressed, the front end updates all the records accordingly
4. object id not acc # (unique object id)
5. save is both search criteria and image id's (save session saves "select object from objects where imageid in (1,2,3,…4)")

Goals for Friday:

1. Data Standard (required set, standard used?) for Ingest
2. Data model (how the data reside in the system)
    a. Map entities, attributes and known values

Data Submitted by Institution (need to determine structure below)

Image data
MIME data (the image)
Image metadata (xml)

Object data
Image Name = the related image
Object ID = local system id
Accession Nbr
Primary_title
Creator
Owner
Materials
Dimensions
Creation Date
Credit Line
Copyright

Additional Text (XML formatted additional fields; this can apply to NON-OBJECT images sent for cataloguing allowing additional image metadata to be displayed.)

- we need to determine what structure the object data will take. Some sort of delimited text similar to the AMICO format, XML, or other.
- Some sort of link to the accompanying image file (e.g. the image name)

Data ingestion routine pseudocode.

a. Dump data file (to spec) and image file (to spec) into a folder (or via ftp to a folder)
b. Run routine to parse data into relational tables
    Subroutine: ProcessData(data file)
    i. Function: ObtainBatchID() returns batch_id
        1. Function: GetNextId(batch) returns next batch_id
            a. Insert batch table next batch id and date/time stamp
            b. RETURN batch_id
    ii. Loop through the objects in the file
        1. Function: ObtainObjectUID(datafile.object_id, datafile.institution_id)
            a. If no object_id then RETURN 0
            b. Does object record exist? Select iobject_id from object where object.object_id = datafile.object_id and object.institution_id = datafile.institution_id
            c. DOES NOT EXIST
                i. Write object record including the batch id to the object table
            d. Write data to object record (ASSUME OVERWRITE IF EXISTS)
            e. RETURN iobject_id
        2. Function: ObtainImageUID(filename, institution_id) returns image_id
            a. Does image record exist? Select images.image_id from images where images.filename_orig = datafile.filename and images.institution_id = datafile.institution_id and images.object_id = datafile.object_id
                i. If EXISTS (assume overwrite)
                    1. Function: ProcessImage(path to source image, destination image info) returns path and filename to system stored image
                    2. write any image metadata back to record
                ii. if DOES NOT EXIST
                    1. If NOT FileExists(datafilepath\filename) and then ERROR("can't find file:" + filename) RETURN
                    2. Function: ProcessImage(path to source image,")

3. Write image record including the batch id and any image metadata to image table
   b. RETURN image_id
3. Write row to associative table to establish the relationship
   a. INSERT IMAGE_OBJECT values(image_id, iobject_id, batch_id)
4. Run exception processing and/or orphan object reports.

Issues raised:

1. we are not requiring descriptive metadata beyond tombstone so how are users going to find objects? How will the system suggest similar objects (this might be mitigated somewhat by the ability to include the additional metadata)

2. Are we showing tombstone data? Is this a user preference setting or Institution preference or both? (we have assumed both for the data model)

3. Terms: do we want Categories for terms?

4. Terms: how to handle the "I can't think of anything" scenario. (Do we have a timer that after X seconds makes a button visible to skip [which then sets a flag on the TERM table indicating the skip.])? **This is confirmed after discussion with Front-end.**

5. How to handle long text feedback? (do we add new table to handle such feedback/user commentary) **The 'blog' type feature for registered users addresses this.**

Questions about new front-end interface:

TRACKING ANONYMOUS USERS:
Front-end wants to track anonymous users independently. How do we handle the requirement to have a user id for various other entities on the back end?

PSEUDOCODE:
1. when an anonymous user starts an Environment (formerly session) trigger FindAnonymousUser("cookie value if exists") [e.g. 'anonymous_00005']
   a. function then looks for a user named 'cookie value'
   b. if cookie.user exists then
      i. RETURN user_id of that user
   c. else
      i. NewAnonymousUser()
         1. gets next user id

2. insert user values ('anonymous_' + string(next number) …) NOTE: anonymous users get a user_status = 0

ii. RETURN user_id of new anonymous user

d. establish new session using user_id from cookie.user record
e. write cookie with new user name

TRACKING INTERFACE SETTINGS FOR RESEARCH:
Front-end may want to have toggling ability on interface features. If so, we may want to track the settings for each session for research.

1. This can be addressed by storing some sort of bit string representing features as being either 'on' or 'off' or some other method such as a key/value pair string.

NOTE: we have created an entity to track MODE history. (i.e. we will track versions of modes and descriptions of changes, as well as capture the mode version within each ENVIRONMENT (aka session information).