Appendix 2b: *Humanities Commons* Resource Authorization and Content Syndication Architecture

1. INTRODUCTION
This document provides recommendations as well as a technical proposal and cost estimate addressing the identified use cases for allowing users within the *Humanities Commons* sites to share access to resources and to syndicate content to and from other sites in the network.

1.a. Background and Network Architecture Overview
The MLA Commons site runs on Commons in a Box (hereafter, CBOX), which is a pre-packaged WordPress distribution running BuddyPress along with a collection of third-party WordPress plugins. The *Humanities Commons*, as we understand it, is a network of CBOX installations, each of which allows users to authenticate against a single, central identity provider application. Furthermore, CBOX installations within the network should be able to share and syndicate some CBOX resources to and from other network sites.

It is our understanding that, in addition to the CBOX sites for each network organization, there will be a central CBOX installation and/or blog for the network itself.

This document is informed by the related *Technical Recommendations for Authentication and Authorization within Humanities Commons* document, which outlines a proposal for solving the challenges around authentication and authorization in a distributed, network environment. The final recommendation in that document is that authentication and authorization on the *Humanities Commons* be addressed by a centralized identity management platform, or the “identity provider.” To narrow the scope and focus of this document, we are working under the assumption that the MLA will proceed with that document’s recommendation to deploy a centralized Shibboleth identity provider application—or a comparable solution—which will store identity information for members of the various societies and organizations participating in the network, and that will also function as a repository of user metadata that can be used for determining access to content across the network.

1.b. WordPress and BuddyPress Architectural Challenges
The greatest challenge of this project stems from the fact that neither WordPress nor BuddyPress contains the architectural underpinnings that are necessary for the granular authorization and content syndication that this project suggests. This should not be surprising, as WordPress’ largely procedural, hook-based codebase was developed over a decade ago, at a point in time in which many members of the open source PHP development community were simply not thinking about service-based architecture, API design, object-oriented architecture,
separation of concerns, and the other now mainstream concepts and practices that make for good software design. Moreover, while WordPress does possess an XML-RPC based API, BuddyPress does not currently expose its resources via an API, nor does it provide an abstracted internal API for accessing those resources.

APIs are important because they define layers of abstraction within an application and protect one layer of an application from other layers. A WordPress’ public XML-RPC API, for example, requires that a user authenticate before utilizing the API. If a user sends a request to create a new post via the API, WordPress will first authenticate the user, determine whether she can post by looking at her role, and eventually create the post. When it inserts the post in the database, WordPress will no doubt validate the post and ensure that the database fields are properly updated. The API, then, encapsulates the business logic and validation that is part of creating a post. Without an API, external systems are left inserting data directly into the database, or using lower-level functions within the source code, which can easily lead to data instability.

A similar API may need to be created for BuddyPress to programmatically expose a subset of its core resources to other CBOX installations. In cases where the application follows the MVC pattern (e.g., Rails or Symfony), writing an API layer can be a relatively straight-forward process of exposing create, update, delete, and read methods on the underlying ORM (e.g. ActiveRecord or Doctrine) to the API client via controller methods. Rather than abstracting persistence into a separate layer, the BuddyPress core encapsulates most persistence related methods into classes representing the resource itself. For example, the group class includes a save method that allows it to persist itself to the database. A public API for creating groups, then, would wrap around the group resource class and delegate as much persistence logic as possible to that underlying class, to help ensure forward compatibility.

Another significant challenge is the fact that neither WordPress nor BuddyPress ships with a robust authorization framework. WordPress’ authorization scheme revolves around roles (super admin, administrator, editor, author, contributor, and subscriber). On a WordPress Network installation, except for the super admin role, these roles describe a given user’s relationship to a given blog. BuddyPress also defines a handful of roles in its codex that describe a user’s relationship to a group, which can be that of a member, moderator, or administrator. These roles should not be confused with the detailed access control lists (ACLs) that we see in more complex applications, in which a user’s abilities can be defined on a resource-by-resource basis. Without more sophisticated ACL functionality, we’re unable to easily determine, for example, that a member of a group can access one document, but not another document.
Our strong recommendation here is that the development undertaken as part of this project be treated as a layer on top of CBOX, much like BuddyPress is a layer on top of WordPress. Some small core changes to BuddyPress will likely be necessary, and can probably be contributed back to the BuddyPress core in a non-breaking way that will satisfy the project maintainers. However, introducing large-scale changes to how WordPress and BuddyPress handle resource authorization—for example, introducing detailed ACLs—could effectively be the same as forking these platforms completely, which in our view ought to be avoided.

1.c. Use Cases
It is our view that the primary use cases can be broken down into two separate conceptual categories. The first category addresses use cases that are related to authorization—as defined in the Technical Recommendations for Authentication and Authorization within Humanities Commons document—which involves building a subsystem that is capable of determining whether or not a user can access a given BuddyPress group on a given CBOX site. The second category addresses use cases that are related to resource syndication, which involves pushing content from one CBOX instance to another and, in some cases, synchronizing content between instances.

2. PROPOSAL FOR ADDRESSING AUTHORIZATION USE CASES

2.a. Authorization Use Cases
The main resources on a CBOX installation are groups, users, announcements, private messages, blogs, blog posts, forums, and forum posts. For the most part, those resources that generally require authorization are resources that exist within a BuddyPress group. We recommend that the authorization use cases be reduced to determining whether or not a user belongs to a given group, and then allowing BuddyPress’ role-based permission logic to determine access to specific resources within that group. Not modifying BuddyPress’ permission logic simplifies the task at hand considerably, because we are limiting the changes that need to be made to BuddyPress to the relatively narrow domain of determining group membership. This approach would address, for example, the following use cases:

1. Local users on CBOX sites should be able to be created and added to groups, even if that user doesn’t exist in the central identity provider database.
2. A specific user on a CBOX instance that exists in the central identity provider database should be able to be given member, moderator, or administrator access to a group on another CBOX instance within the Humanities Commons.
3. A collection of users belonging to one or more organizations in the central identity provider database can be given access to one or more groups on any CBOX instance within the Humanities Commons.
4. For the purposes of group membership, a collection of users can be defined based on their organization affiliation or on their affiliations within a given organization.

2.b. Authorization Proposal Overview
In brief, our proposal is that the Humanities Commons incorporate a mechanism into CBOX by which users can be automatically enrolled in groups on any networked CBOX instance as an administrator, moderator, or member. Once a user has been given a role in a group, how the user interacts with the resources contained within that group is determined by the role-based abilities that exist in the BuddyPress core (see http://bit.ly/R9VCPg for a detailed description of what actions each role can take on public and private groups).¹

Under this approach, the primary development task for addressing the authorization use cases will be to create a mechanism by which a user can be automatically added to a group on a CBOX installation.

2.c. Prerequisites for Authorization Use Cases
Before users can be authorized as a member of a given group, the Humanities Common will need to store metadata about these users in its centralized identity management system. We propose that the identity provider store (at least) two key pieces of meta-data about every user. One will be a list of network organizations to which a user belongs. The second will be a list of organization-specific affiliations that will be provided by the network organization when it submits its member data to the identity provider instance via a data transfer mechanism. Both identifiers will be simple alpha-numeric string identifiers, akin to tags. Because we can’t possibly predict the organizational structure of the various network CBOX instances, they should be allowed to tag their users using whatever identifiers make the most sense for their organization.

2.d. Data Transfer Application
One challenging part of this project will be the creation of a data transfer application. This application would be used by administrative users at each organization to import their membership data into the central identity provider. Access would be restricted to logged in users (who would be authenticated against the identity provider). These select users would be able to upload member data into the identity provider on a regular basis from a variety of formats (e.g. CSV, JSON, XML).

¹ If it is ultimately determined that the three core BuddyPress roles are too limiting, then we would recommend that additional roles be added to all CBOX instances (e.g: read-only member) either via a stand-alone plugin or a modification to the BuddyPress core. The key point is that the question of what a user in a given role is able to do in a given group should be treated as a separate concern from the task of assigning a user to a given role in a given group.
We anticipate a need for a separate, stand-alone import application so that there is a UI in place to simplify the process of adding data to the identity provider. The data transfer application would also be responsible for defining the format of the input data and for validating that data before it was sent to the identity provider. Our recommendation would be to use a development framework—Rails and Laravel would both be good choices—to create this application.²

When the data is imported into the identity provider, the identity provider is responsible for creating or updating each user’s record. Business logic will need to be developed to determine how the identity provider knows that a given user is the same user across all three membership databases. Most likely we will need to use each user’s email address as a globally unique identifier, but this likely warrants further discussion. The import mechanism will also need to make decisions about when and how to update a user’s information, and which data source—if any—should be considered authoritative.

To illustrate how this import process will work, we’ll look at a hypothetical user named Jane Doe. Jane belongs to the Modern Languages Association (MLA), the American Historical Association (AHA), and the Renaissance Society of America (RSA). At the MLA, Jane is a member of the Committee on Scholarly Editions. At the AHA, Jane is on the Book Prizes and Awards Committee. At the RSA, Jane is a member of the History Working Group.

After all three organizations have imported their data into the identity provider via the data transfer application web UI, the identity provider will contain a record for Jane that includes the following meta data:

```json
memberships: ['MLA', 'AHA', 'RSA']
org_affiliations: {
  'MLA': ['committee_on_scholarlyEditions'],
  'AHA': ['book_prizes_and_awards_committee'],
  'RSA': ['history']
}
```

The identity provider is storing—likely as JSON—a machine readable representation of the organizations that she belongs to (each of which is given a unique identifier when their membership data is imported into Shibboleth), as well as the arbitrary organization specific affiliations.

² This approach assumes that the identity provider possesses APIs for creating and updating member data, or that an alternative identity provider is used that accomplishes the same basic function—a central data store for user identity and metadata—while also providing an API. Indeed, when the MLA gets closer to selecting an identity provider, we would argue that the final solution must have include a SOAP or REST API.
2.6. Humanities Commons Authentication Plugin

After addressing the identity storage and import mechanisms, we will need to develop a CBOX component that can authenticate users and authorize resources against the central identity providers. To do this, we would create a Humanities Commons Authentication plugin to be installed on every CBOX instance within the network. This plugin will be similar to the MLA Commons’ MLA-Auth plugin, only it will rely on the identity provider rather than the MLA membership web service for its data. When a user authenticates against a CBOX site running this plugin, the provided username and password will be sent to the identity provider. If the user is authenticated, the plugin will check to see if a local instance of the user exists in the WordPress database. If it does not, it will be created based on the information returned by the identity provider. If it does, the user may or may not be updated depending on whether the local user or the remote user has the most recent data. At least two custom fields will be added to the CBOX user table, including one that stores the user’s network affiliations, and one that stores the user’s organizational affiliations.

Continuing with the example of Jane Doe, above, all three of the organization to which she belongs maintain their own CBOX installation running the Humanities Commons Authentication plugin. All three of these organizations regularly provide membership information to the identity provider by way of the data import application. When they provide this information, they include a field for local affiliations. This field would contain alphanumeric string representations of Jane’s organization specific affiliations. Along these lines, the MLA would list “committee_on_scholarly_editions” as one of Jane’s her affiliations:

```json
memberships: ['MLA', 'AHA', 'RSA']
org_affiliations: {
    'MLA': ['committee_on_scholarly_editions'],
    'AHA': ['book_prizes_and_awards_committee'],
    'RSA': ['history']
}
```

Returning to the Humanities Commons Authentication Plugin, it’s important to understand what happens when a user logs in to a network CBOX instance. On login, the plugin will update the local representation of the user’s memberships and organizational affiliations in the local user table. This means that on authenticating to any network CBOX site, that site will become aware of all the organization affiliations that Jane possesses. This awareness of every user’s affiliations will be the basis for authorizing access to groups on an CBOX instance-by-instance basis.
2.f. Assigning Users to BuddyPress Groups Based on Memberships and Affiliations

By way of the central identity provider, every networked CBOX instance can be made aware of all network users memberships and organizational affiliations. To utilize this information, group administrators must be able to access it while managing group membership.

To this end, we will write a WordPress plugin that extends the BuddyPress group administration views so that group admins have the ability to set organizations and organization affiliations that should allow a given user to gain automatic membership to the groups. For this to work intuitively for the group administrator, she should be shown an interface that allows her to first select from among the organizations (MLA, AHA, RSA, etc) that are included in the identity providers and then to select from among the affiliations (Committee on Scholarly Editions, History Working Group, etc) for the selected organization. Because a given CBOX instance would only be aware of the organizations and affiliations belonging to member who had already logged in, this interface will likely need to make an asynchronous (AJAX) request to either the user data transfer application (mentioned above), or directly to the identity provider to retrieve a list of all existing organizations and affiliations.

When a user logs in to a CBOX site, the *Humanities Commons Authentication* plugin would look at the user’s memberships and organizational affiliations and compare them to the existing groups in the installation. If they matched the group membership settings, the user would automatically be enrolled in the group after logging in.

These modifications to the group settings and invitation interfaces would also need to allow a user on the CBOX instance to invite a specific user from another site. To accomplish this, we would need to extend the BuddyPress components that search for users in the local database so that they also queried the external identity provider for users and imported those users into the local database on being added or invited to a group. The UI should be carefully designed to permit users to easily find one user among the potentially tens of thousands of users that exist in the central identity provider database.

2.g. Maintaining Member Data Integrity

The current *MLA Commons* site’s approach to authorization—in which the user’s credentials are passed to a data source external to WordPress on login, at which point the WordPress account is updated—contains shortcomings that we will want to avoid on the *Humanities Commons*. The most significant problem is that by only synchronizing authorization data (organization affiliations, group affiliations, etc.) at the moment of authentication, it is not possible to ensure that member data is current. If, for example, a user logs in and is automatically added to a local BuddyPress group, only to be removed from that group in the central identity provider, that user
will continue to be a member of the group until she logs in again. While simple to maintain, this model is insufficient for ensuring relatively current data around resource authorization.

There are two fundamental approaches to solving this problem. The first approach is for each of the CBOX instances to pass all authorization requests to the central identity provider whenever an authorization check is required. While this approach would provide the most current information, it will likely be prohibitively resource intensive. There are numerous points in BuddyPress where the software needs to make determinations around whether or not a given user belongs to a given group, and sending all of these requests to an external server would introduce a single point of failure for all CBOX instances while also likely degrading the performance of each instance.

The second approach is to attempt to synchronize data to the CBOX instances on a more regular basis. To accomplish this, we expect that the Humanities Commons will need to provide each CBOX instance with a mechanism for regularly updating local membership data from the remote identity provider. Some discovery will be required to determine what form this mechanism should take. If all of the CBOX instances are running on dedicated or virtualized hardware, one approach would be to write a background daemon (perhaps in Ruby) that was responsible for regularly updating a set of local records with current member data. Another approach would be to create a WordPress plugin that hooked into WordPress’s cron integration to periodically update sets of local records from the central identity provider. When these updates occurred, local CBOX instances would receive up-to-date profile information and group affiliations for users that exist in the CBOX instance. To accomplish this, we would likely want to create a JSON API that returns member data as part of the import web application that will sit in front of the identity provider.

There are some security implications here, of course, as each CBOX instance would have access to a fairly broad swath of member data. The architectural approach to this component would require some additional planning to ensure that proper monitoring was in place, and that the identity provider was able to easily revoke access for a given CBOX instance.

2.h. Advantages and Disadvantages of Authorizing Resources through Group Membership
The approach we’ve described for automatically authorizing users in BuddyPress groups has a number of advantages:

- It likely does not require large-scale changes to the BuddyPress core, which means that it’s conceivable that most of the logic can be added at the plug-in level, which will increase the lifespan and upgrade path of CBOX sites.
The underlying role-based authorization logic of WordPress and BuddyPress is by and large kept as-is. Our custom extensions are responsible for making sure that users are assigned to the right groups with the right roles, not for determining what a user can do in a group once she has attained a given role.

CBOX instances do not need to be able to communicate directly with one another for this approach to work. Following the recommendations of the Authentication and Authorization document, each CBOX instance only needs to be able to send messages to and receive messages from the central identity provider or an application that sits in front of it. As a result, the overall structure of the network is significantly simpler than if it were truly distributed.

There are, however, also disadvantages:

• Because CBOX instances are not communicating directly with other CBOX instances, auto enrollment in groups is only possible for users whose profiles are stored in the central identity provider. That is, if I make a local user in CBOX instance A, that user will not necessarily be able to login on CBOX instance B.

• There is an additional layer of abstraction that may cause some confusion. If CBOX sites were speaking directly to one another, it might be possible for a user in group A on instance A to automatically belong to group B on instance B based on a relationship created directly between those two groups. Under our proposal, however, both group A on CBOX instance A and group B on CBOX instance B would need to include a shared affiliation or organization for a user to automatically belong to both.

• This approach requires the member data that's provided by each organization to include all of the specific member affiliations and for this data to be stored centrally. This is not the same as creating a true, federated network, but it is in keeping with the realistic recommendations made in the authentication and authorization document.

3. PROPOSAL FOR ADDRESSING RESOURCE SYNDICATION USE CASES

3.a. Resource Syndication Use Cases

Our proposal for handling content sharing and syndication assumes that all content can have one authoritative location. A user on one network site may need to be informed of content that was modified or created on another CBOX site, but the content itself does not need to be moved between sites. Accordingly, we envision the following core use cases dealing with content sharing and syndication:
1. Activity that occurs on a group on one CBOX instance is available to a specific group on another CBOX instance. For example, the History Group on the RSA site wants to be notified when announcements are made in the MLA Committee on Scholarly Editions group.

2. A user who regularly uses multiple CBOX instances in the network sees her profile data updated consistently across all sites in the Humanities Commons.

3. All posts belonging to a certain category (e.g., CFP) on all Humanities Commons sites appear on the main Humanities Commons site.

4. A user can see a centralized record of her activity on all networked CBOX instances on the Humanities Commons CBOX installation.

3.b. Synchronization vs. Syndication

There are two principal approaches that we can take in order for a group on one CBOX instance to make its content available to a group on another CBOX instance. One approach is to attempt to synchronize that content between the two groups. Synchronizing content entails that the same piece of content exist in the database of both CBOX instances, and there are a number of challenges that make this approach error-prone and more likely to fail. Under a content synchronization scenario, a group on the MLA site might synchronize content—for example, group docs—from a group on the RSA site. Any change to a doc on the RSA site would need to be pushed to the MLA site, and both systems would need to maintain globally unique identities of their documents. Attempting to synchronize data between two CBOX databases will quickly lead to complex problems:

- What happens when one site is not available? Do we need to implement content synchronization queuing?
- How do we maintain global identity for resources across CBOX installations when BuddyPress does include UUIDs for its resources? Do we have to make significant changes to the BuddyPress core?
- How do we authorize synchronized resources across different sites and groups, when a user in a group on site A may have very different permissions from the user in a group on site B?
- What happens when content is edited in both locations? Which version is considered authoritative?
- Do we need to implement locking mechanisms to prevent race conditions when content can be edited in two places?

Because a synchronization approach so quickly becomes complex, our recommendation is that the Humanities Commons instead attempt to solve the content sharing use cases by approaching
them as a problem of syndication—something WordPress excels because of its strong RSS support—rather than synchronization.

By syndicating, rather than synchronizing content from one CBOX instance to another, the task at hand is made simpler. BuddyPress already records most activity that occurs within a given group as part of its activity feed, and it renders this feed as RSS so that it can be consumed by other applications. Our recommendation is that we utilize these feeds to pull content from one group’s activity feed into another group’s feed, much like we might syndicate blog posts from one blog to another using RSS.

3.c. Syndicating Group Content and Blog Posts
To accomplish this, the development team would expand the existing CBOX “External Group Blogs” to provide extra UI component in the administrative view for a BuddyPress view that allow group admins to specify the URLs of one or more activity feeds for another group, which would be merged into that group’s own activity feed. Similarly, if a group wanted to include content from a blog somewhere else in the Humanities Commons, they would be be able to do so by syndicating activity from that blog’s RSS feed.

Depending on project scope and budget, it would also be possible to simplify this interface by allowing users to select any public group from any network CBOX instance. To accomplish this, we would need to add functionality to CBOX that allowed each instance to report its groups to the central data import application, which would maintain a list of all public groups at all network sites. With this approach, a group administrator could potentially select a relevant group by name in an admin interface, and a scheduled task could run in WordPress to periodically pull public activity from the remote group into the local group.

This approach is not without its challenges, of course. With syndication, content that is pulled from a remote source can be modified remotely after it’s been pulled, which can lead to an inaccurate or stale local representation. We’ve worked with community syndication solutions like FeedWordPress (http://wordpress.org/plugins/feedwordpress/) and have seen performance issues on large network sites with thousands of blogs.

3.d. Syndicating CBOX Instance Content to a Central Site
Ideally, it should be possible for content from one network site to be syndicated to the central Humanities Commons site, and for this content to be syndicated back to the individual network sites. To accomplish this, we recommend that each network CBOX installation provide the central site with a single RSS feed to be used for global announcements. The central site will consume an RSS feed for each network site and combine the syndicated content into a single “Network Announcements” RSS feed that includes links back to the member sites. Each CBOX
site would have the option of rendering the Network Announcements feed on their own home pages, thereby making these announcements available to all users. The same approach could be used for CFPs, in which the central network site would aggregate CFP feeds from each CBOX site into a single, combined feed.

Perhaps the addresses of these feeds could be maintained in the central data import application. In addition to serving as a tool for uploading member data, the central data import could provide a configuration UI in which network sites could manage which feeds of theirs should be consumed by the central Humanities Commons site. Some further discovery will be required here to further articulate and define the various use cases.

3.e. Synchronizing Profile Data Across All Network Sites

Another key piece of data that needs to be synchronized across all sites is user profile data. Where possible, the Shibboleth identity provider should be treated as the authoritative data store for member data. Upon logging into a network site, those fields for which the identity provider is authoritative will be synchronized to the network site’s local user database. Likewise, changes to local data may need to be synced back to the identity provider, assuming the local data has a more recent timestamp than the remote data.

Network sites should also be provided with a synchronization mechanism that allows them to pull all remote user records from the identity provider into the local database, and to update local data with remote data on request. This mechanism will likely be executed via the WordPress administrative interface.