**13.0-18**

Select Authors from Uploaded List

**EM Version: 13.0**

# Contributors:

|  |  |
| --- | --- |
| Primary Engineer | Andrew Pokrovski |
| Senior Engineer | Andrew Pokrovski |
| Coding Engineer(s) | Matt Van Voorhies,  Jean Gelinas |

# Document Revision History:

|  |  |  |
| --- | --- | --- |
| 09/09/2015 | Matt Van Voorhies, Andrew Pokrovski | Created Technical Specification |
| 10/08/2015 | Jean Gelinas | V2 - reorg |
| 10/09/2015 | Jean Gelinas | V3 – reorg 4.6 & 4.7 after discussion with Andrew. |

# Summary:

This spec will enhance the existing Author Selection process by adding functionality for uploading and pre-registering a list of authors from an external file.

# General Design:

There are 3 major pieces to the implementation of the Author Upload functionality:

1. The File Import process and related UI enhancements
2. The Parsing of the Author List File
3. The Author List File Results and related UI enhancements

## The File Import Process (authorSelectionSummary.aspx) - Matt

The file import process occurs on the authorSelectionSummary.aspx page. The current page is updated to include a file import button and drag-drop area for importing the Author List File (ALF). The File Import Process is similar to the interface used for adding Submission files during Manuscript creation and hence the existing code will be leveraged as much as possible for implementation of the AuthorSelectionSummary page.

### Pages

#### AuthorSelectionSummary.aspx(.cs)

AuthorSelectionSummary.aspx includes minor changes to add a new option to the existing drop-down list.

At about line 195 in authorSelectionSummary.aspx.cs a “searchModeItem” is created and assigned the constant value: Constants.AUTHOR\_SEARCH\_MODE\_UPLOAD\_AUTHOR\_LIST

Logic is added at about line 779 in the btnGoSearch\_Click event handler to redirect to the new UploadAuthorList.aspx page which implements section 2.3 of the functional spec.

#### + UploadAuthorList.aspx

This new page is described in the functional spec in section 2.3. It utilizes the JQuery File Upload Plugin and passes a list of Authors to the “Search results” style page described in section 2.4 of the functional spec.

The file import will utilize the JQuery File Upload Plug-In, similar to *AttachManuscriptFiles.aspx*. In order to do this the page is updated as follows:

* JQuery libraries are included at about line 23
* JQuery OnReady function is added to script block to initialize the File Upload Plug-in
* A div for the Dynamic Upload (using the plug-in) with class ‘deliveryPanel’ is added. This panel has the fileInput ‘input’ HTML as well as the HTML for the Drag/Drop area.
* A div is added for Static Upload (for browsers that don’t support the drag-drop plug-in) with a standard HTML input of type ‘file’
* A div for the progress Panel is added similar to the HTML in *AttachManuscriptFiles.aspx* at about line 286.

### Code

The file upload process uses the following classes to parse and process the uploaded Author List file.

#### Aries.EditorialManager

##### Constants.cs

This existing class is updated to add a new constant for the Author Search Mode for Upload Author List at about line 1217.

public const int AUTHOR\_SEARCH\_MODE\_UPLOAD\_AUTHOR\_LIST = 5;

#### AEM.Web.Command.Upload

##### + UploadAuthorListFile.cs

This class is similar to the UploadSubmissionFile class used for uploading submission files. This is a new class introduced in this spec.

UploadAuthorListFile

This class is serialized into the JSON return object for the File Upload process. The class definition is contained in the source file for UploadAuthorListFile.

Properties:

Public string FileName

The file name that was uploaded.

Public string ErrorCode

The error code if the upload failed.

Public ValidationResult ValidationResult

The validation results of the file upload process.

Public List<UploadedAuthorInfo> AuthorInfoList

A list of UploadedAuthorInfo objects containing the parsed output generated by parsing the Author File List.

UploadedAuthorInfo

This class represents an individual Author from the uploaded Author List File.

Properties:

Public Guid AuthorID

Public string FirstName

Public string LastName

Public string EmailAddress

Public string AuthorNote

Public string OrcID

Public bool Invite

Public bool AuthorRegistered

#### Aries.EditorialManager.Web

##### UploadScheme.cs

This existing class is updated to provide for the new Upload type - AuthorListFile

At about line 100 the following item is added to the enum UploadScheme:

AuthorListFile = 10

## Author List Overlay Dialog - Matt

The Author List Overlay Dialog appears after the user has uploaded the Author List File. This overlay is modeled after similar dialogs used in EM. The overlay presents the parsed list of Authors using the JSON object list returned by the File Upload Plugin. The overlay itself is presented as a JQuery dialog.

### Code

#### + AuthorListFileAuthors.js

This new javascript library file contains the JQuery function called when the file upload and parsing process has completed.

A JQuery dialog is opened with the function defined in this library using *AuthorListFileAuthors.ashx* as the Ajax target and passing the list of Author objects.

The dialog function call passes the parameters:

modal: true,

autoOpen: false,

resizable: false,

closeOnEscape: false,

width: $contentBody.width() \* 0.80,

dialogClass: 'wizard',

open: function () {

$dlgAuthorListFileAuthors.dialog('option', 'position', {

my: 'left top',

at: 'left+10%' + ' top+' + (authorListAuthorsCount > 1 ? '40' : '45'),

of: $contentBody

});

### Pages

#### + AuthorListFileAuthors.ashx

This ashx file is added for this spec and defines the Web Handler for the AuthorListFileAuthors dialog that is displayed as an overlay.

The class AuthorListFileAuthors is defined as follows:

public class AuthorListFileAuthors : BaseHttpHandler, IHttpHandler

The class implements the following methods:

public override void ProcessRequest(HttpContext context)

This method calls the base class method, sets the response context to ‘application/json’ and then loads the WebUserControl AuthorListFileAuthors and generates the HTML and serializes it in the HTTP response.

#### + AuthorListFileAuthors.ascx(.cs)

This is a Web User Control that contains the HTML for displaying the Author list. An Asp Panel and Asp Repeater are used to display the list of Author objects.

The class is defined as follows:

public partial class WebUserControlsAuthorListFileAuthors : System.Web.UI.UserControl

The following constructors are defined:

WebUserControlsAuthorListFileAuthors()

WebUserControlsAuthorListFileAuthors(List<AuthorListFileAuthors> authorList)

This constructor sets the local member variable with the provided author list.

The following methods are defined:

protected override void OnLoad(EventArgs e)

This method populates a repeater control within an ASP panel on the User Control page that contains fields for each of the Author properties defined in the Author List File

### Misc

The following miscellaneous updates are implemented.

#### Web.config

The web.config file requires the following changes to support the new upload type.

A key is added to the *uploadSchemes* section as follows:

<uploadSchemes>

...

<add key="10" value="Aries.EditorialManager.Web.Command.Upload.UploadAuthorListFile"/>

</uploadSchemes>

## Parsing the Author List File - Matt

Parsing the Author File List is done as part of the file upload process. This section details the specific processes for how the file is parsed.

### Code

#### Aries.EditorialManager.Web.Command.Upload

The first step in parsing the file is determining the file type. The acceptable file types are comma separated text file, tab separated text file or Excel spreadsheet file.

Parsing the comma separated and tab separated text is done by opening the file and reading the first few bytes.

##### + UploadAuthorListFile.cs

The parsing occurs in the AEM.Web.Command.Upload.UploadAuthorListFile class in the method:

protected override void OnProcess()

The following method is used to parse the comma separated or tab separated file:

Protected void ParseTextAuthorListFile()

This method uses File.ReadLine() to extract the first line from the source (the uploaded or the converted Excel file) which is used to determine the column order. For both the column row and each Author row, String.Split is used to break apart the provided values. The column order is stored in a hashtable of key = column value=index.

The remainder of the file is read, line by line and each line is stored in a list of *UploadedAuthorInfo* objects.

This list is then returned via the Ajax call in the *UploadAuthorStatusInfo*.

## Author List File Results Page - Matt

### Code – Andrew (completed)

#### Aries.EditorialManager.Framework.Journal.PeopleObjects.Person

##### Mapper.cs

**+ FindByPersonalIdentifierOrcid(List<string> orcIDs, bool authenticated)**

New function (along with supporting CollectionQuery and stored proc) that takes a list of orcid strings and returns all people records with matching orcids. The authenticated parameter controls whether or not we limit ourselves to authenticated orcids only.

**+ FindByEmailMatch(List<string> emails)**

New convenience function that takes a list of strings and invokes FindByEmailMatch(string email) after transforming the string list into a comma separated list (or however else that’s stored)

#### Aries.EditorialManager.Framework.Journal.AuthorObjects.AuthorInvitationStats

##### Mapper.cs

+ FindByPeopleIDs(List<int> peopleIDs)

New function that takes a list of people IDs and finds author invitation statistics records for those people.

##### Entity.cs

Must add DateLastInvited property, driven by DATE\_LAST\_INVITED from the underlying table. Also, of course, supporting code in mapper and stored procs.

#### Aries.EditorialManager.Framework.Journal.AuthorObjects.InvitedAuthor

##### Mapper.cs

+ FindByPeopleIDs(List<int> peopleIDs)

New function that takes a list of people IDs and finds all author invitation records for those people.

#### Aries.EditorialManager.Framework.Journal.SubmissionObjects.Submission

##### Mapper.cs

**+ Dictionary<int, int> GetUnsolicitedSubmissionCounts(List<int> peopleIDs)**

New function that takes a list of people IDs and returns the count of “unsolicited” submissions in the pipeline for each given person, as per logic in UDF\_getAuthorInvitationStatistics:

SELECT COUNT(1)

FROM dbo.DOCUMENT D WITH (NOLOCK)

INNER JOIN dbo.ROLEAUTH RA WITH (NOLOCK)

ON RA.DOCUMENTID = D.DOCUMENTID

WHERE D.RELATED\_DOCUMENTID IS NULL

AND RA.PEOPLEID = PEOPLE.PEOPLEID

AND D.DDISPOSIID IS NULL

The return value needs to map people IDs to the ‘unsolicited submission count’.

## AuthorCandidate data structure - Jean

### Code

#### Aries.EditorialManager

Private Fields:

These store entity objects parts of which will be exposed through flattened public properties.

##### AuthorCandidate.cs

Constructor:

AuthorCandidate(DatabaseMapperManager mapper, UploadedAuthorInfo authorFromFile)

Fields:

**\_authorFromFile**

* UploadedAuthorInfo of record in file
* Passed/populated through constructor

**\_mapper**

* Passed/populated through constructor

Properties:

**MatchedPersonStatistics**

* A.EM.F.Journal.AuthorObjects.AuthorInvitationStats.Entity
* Based on matched person – null if no match
* Set only

**MatchedPersonExistingInvitations**

* List of A.EM.F.Journal.AuthorObjects.AuthorInvitation.Entity
* Invitation data for matched person
* Set only

**AverageDaysToComplete** - MatchedPersonStatistics

**CompletedInvitation** - MatchedPersonStatistics

**DateLastAccepted** - MatchedPersonStatistics

**DateLastCompleted** - MatchedPersonStatistics

**DateLastDeclined** - MatchedPersonStatistics

**DateLastInvited** - MatchedPersonStatistics

**DeclinedInvitations** - MatchedPersonStatistics

**EmailAddress** – \_authorFromFile.EmailAddress

**FirstName** - \_authorFromFile.FirstName

**Hash**

* \_authorFromFile – firstname,lastname,emailaddress,orcid
* Makes for easier searching of dictionaries/lookups

**HasMatchedPerson** – indicates that a person match has been made

**ImportMatchesMultiplePeople** – indicates this person could match multiple em people

**MultipleImportsWithSameEmail**

* Import record has no EM matches – one or more
* One of the email addresses is duplicated in another non matcheable import record
* Not otherwise a duplicate record in the file

**InvitationsOutstanding** – MatchedPersonStatistics

**InvitedSubmissionsInProgress** – MatchedPersonStatistics

**LastName** - \_authorFromFile.LastName

**MatchedPerson** –

* A.EM.F.Journal.PeopleObjects.Person.Entity
* Set only
* Unique EM match to author from file

**MatchedPersonID** – MatchedPerson.IdentityValue

**NumberOfInvitationLetters**

* Number of invitation letters to send
* Initially set by number of times the author appears in the upload file
* Can be reset thereafter

**ORCID** - \_authorFromFile.OrcID

**PotentiallyMatchedPeopleID**

* Each potential match will have a separate author candidate object created
* For this candidate object, the potential person match

**Selected**

* is this author selected to have an invitation letter sent
* initially determined by \_authorFromFile.Invite
* Can be changed thereafter

**TotalInvitations** – total number of invite letters to send

**UninvitedAfterAccepting** – MatchedPersonStatistics

**UninvitedPriorToResponding** – MatchedPersonStatistics

**UnsolicitedSubmissionsInProcess**

SubmissionMapper.GetUnsolicitedSubmissionCounts(MatchedPerson.

IdentityValue)

**UploadedNotes** – list of notes from the file

## Imported Author Manager class – Jean

This class encapsulates functionality needed to:

* build lists of AuthorCandidates
* mass-proxy register unregistered authors and generate InvitedAuthorsXml (including notes!) so that it can be harmlessly parsed by AuthorConfirmAndCustomizeEMPage.ParseSelectedAuthorsXml

### Code

#### Aries.EditorialManager

##### ImportedAuthorManager.cs

All static class

**private SessionCacheManager GetSessionCacheManager(string journal, string emSessionID)**

* Convenience function to return a session cache manager for retrieving or updating data in the session cache.

**private static bool UploadedAuthorAlreadyMatched(UploadedAuthorInfo uploadedAuthor, List<AuthorCandidate> processedAuthors)**

* convenience function used in upload list matching
* determines if the current uploaded author data has been processed by searching the processed authors listing.
* If already processed (means that this record is a duplicate of one or more in the author file) adds to the UploadedNotes and NumberOfInvitationLetters of the found processed author candidate.

**public static void MatchUploadList (DatabaseMapperManager, List<UploadedAuthorInfo> uploadedAuthors)**

* performs matching algorithm and returns listing of author candidates to cache

**public static List<AuthorCandidate> GetAuthorCandidatesFromCache(string journal, string emSessionID)**

* gets a cache manager and retrieves author candidates from cache

**public static List<AuthorCandidate> GetMatchedCandidates(string journal, string emSessionID, int startIndex, int endIndex)**

* meant to be called by results page to return the current page’s author candidates
* gets a cache manager and retrieves author candidates from cache and filters based on start and end index passed in.

**public static List<UploadedAuthorInfo> GetUploadedAuthorListFromCache(string journal, string emSessionID)**

* gets a cache manager and retrieves uploaded author info from cache

**public static void PlaceAuthorCandidatesInCache(List<AuthorCandidate> authorCandidates, string journal, string emSessionID)**

* gets a cache manager and places passed in list to cache

**public static void PlaceUploadedAuthorsInCache(List<UploadedAuthorInfo> uploadedAuthors, string journal, string emSessionID)**

* gets a cache manager and places passed in list to cache

**public static void SaveGridChanges(List<AuthorCandidateGridData> gridData, string journal, string emSessionID)**

* retrieves current author candidates list from cache and updates data based on passed in author candidate grid data from the results grid.
* Returns the updates data to the cache
* Allows saving from either the server (the results page) or from the client.

**public static void ProxyRegSelectedUnregisteredUsers(string journal, string emSessionID, DatabaseMapperManager mapperManager, int operatorID)**

* gets a cache manager and retrieves current list of author candidates from cache
* proxy registers all selected/non-matched (Selected == true, HasMatchedPerson == false) persons in em and returns created person entities using RegistrationManager.ProxyRegisterUsers.
* Updates all registered users in author candidates list with their respective person entity.
* Places the author candidates back into cache.

**public static string SerializeCandidatesForConfirmAndCustomize(string journal, string emSessionID)**

* gets a cache manager and retrievs current list of author candidates from cache
* creates and populates an xml structure to be passed to the ConfirmAndCustomizeInvitedAuthors page with data from author candidates list.

Xml data structure:

<PeopleSearchXml>

<InvitedAuthorsXml>

<author>

<numLetters>AuthorCandidate.NumberOfInvitationLetters

</numLetters>

<id>AuthorCandidate.MatchedPersonID</id>

<note>AuthorCandidate.UploadedNotes</notes>

</author>

</InvitedAuthorsXml>

</PeopleSearchXml>

Match algorithm:

* For each uploaded author:
  + Search already processed AuthorCandidates for one with the hash as described in that data structure. Create one if one does not exist.
  + Add current uploadedAuthor record to AuthorCandidate.UploadedAuthors collection
  + Attempt to find a person match, looking greedily in the following order:
    - authenticatedORCIDMatches
    - unauthenticatedORCIDMatches (attempt to resolve “ties” based on matches with the last name, then first name)
    - emailMatches (attempt to resolve “ties” based on matches with the last name, then first name)
  + If no matches, leave AuthorCandidate.Person property blank
  + If just one match, set AuthorCandidate.Person property to that match
  + If more than one match, create that number (less one, due to the existing AuthorCandidate record) of cloned AuthorCandidate records.
  + Set ImportMatchesMultiplePeople flag and RegisteredPersonID for the current and each resulting AuthorCandidate record.
* Having accumulated the initial author candidate collection (we should be seeing some combination of 1:1, many:1 uploaded/person matches, with 1:many being handled as multiple author candidate records), we want to invoke the following data retrieval:
  + Aries.EditorialManager.Framework.Journal.AuthorObjects. AuthorInvitationStats.Mapper.FindByPeopleIDs for all the people IDs in the uploaded author collection to grab statistics for the involved authors. For ease of access, transform the returned list into a dictionary.
  + Aries.EditorialManager.Framework.Journal.AuthorObjects.InvitedAuthor.Mapper.FindByPeopleIDs for all the people IDs in the uploaded author collection to get current and past author invitations for the involved authors. For ease of access, transform the returned list into a dictionary.
  + Aries.EditorialManager.Framework.Journal.SubmissionObjects.Submission.Mapper.FindGetUnsolicitedSubmissionCounts for all the people IDs in the uploaded author collection to get ‘unsolicited submission’ counts for all involved authors.

## AuthorCandidateGrid

### Code

#### Aries.EditorialManager

##### Constants.cs

* CURRENT\_UPLOAD\_AUTHORS\_RESULTS\_PAGE\_CACHE\_KEY (UploadAuthorsResults\_CurrentPage)– to be used to track, in the cache, on which page the candidate grid is currently so that, upon refresh (paginator will refresh on page change and page size) that value can be reset in the paginator.

#### common\WebUserControls

##### AuthorCandidateGrid.ascx/cs

**GridPaginator\_PageSelected**

* Set paginators.CurrentPage to e.SelectedPage
* Do a DataBind on both paginators
* Capture the start and ending index indicated by the paginator
* Store the current page value in the cache to be retrieved once page refresh caused by paginator
* Set PaginationChanged = true

**GridPaginator\_PageSizeSelected**

* Capture the selected page size from e.SelectedPageSize into \_pageSize
* Set PageSizeForGrid in StateManager
* Set PageSize of both paginators
* Do a DataBind on both paginators
* Capture the start and ending index indicated by the paginator
* Store the current page value in the cache to be retrieved once page refresh caused by paginator
* Set PaginationChanged = true

**Page\_Load**

* Set \_currentPage retrieved from cache (CURRENT\_UPLOAD\_AUTHORS\_RESULTS\_PAGE\_CACHE\_KEY).
* Set \_pageSize from StateManager.PageSizeForGrid
* Register any client scripts

**Page\_PreRender**

* Set paginators.CurrentPage = \_currentPage
* Set paginators.PageSize = \_pageSize
* Do a DataBind on paginators

**ShowData(List<AuthorCandidate> candidatesToShow)**

This is a user control that displays a grid containing author candidates. It is driven by a list of AuthorCandidate records.

It will need to have a separate "**RecordCount**" property, which will be passed on to the paginators for proper pagination code.

It will also need to have a readonly **CurrentPage** property, which will be retrieved from the GridPaginator.

It will also need to have a **PageChangeEventHandler** property to assign to the GridPaginators' identically named delegate.

Top: Aries:GridPaginator

Table – contains the following columns:

ID for checkbox and number of invitation letters textbox needs to contain AuthorCandidate.

Select: a checkbox - AuthorCandidate.Selected

Number of Invitation Letters: a textbox, with a client-side numeric validator working off the logic from verifyForm in searchAuthorsByNameAddrResults.asp. Data source: AuthorCandidate.NumInvitationLetters

Author Name

Contains "Confirmed Match: FullNameWithDegree" if AuthorCandidate has a Person  
 Contains "Information from File: First Name, Last Name, Email, Notes" if AuthorCandidate has one or more uploaded authors  
 Contains invitation information for each record in AuthorCandidate.AuthorInvitations. Display logic imitates searchAuthorsByNameAddrResults.asp, block around line 2760  
 Contains warning if ImportMatchesMultiplePeople / MultipleImportsWithSameEmail flags are true.

Board Member

Classification Matches – Driven by ClassificationManager.NumClassificationMatches. If there is time, we need to set ClassificationManager to be able to pre-load classifications for a number of people.

Submissions Being Processed (Invited/Unsolicited) - AgreedAndAwaitingSubmission / UnsolicitedSubmissionsReceived

Author Statistics – Driven directly from appropriate fields in the AuthorCandidate record

Invitation Statistics - Driven directly from appropriate fields in the AuthorCandidate record

Bottom: Aries:GridPaginator

Populated in the same order of records found in the file.

Confirmed Match –

Author name is link to reviewer\_info.asp?id=*peopleid*&docid=*Session.DocumentID*

If Editor has rights, show people flags editor link/icons

ORCID ID is link to [https://orcid.org/*ORCIDID*](https://orcid.org/ORCIDID)

Display the ORCID icon on only authenticated ORCID

Any notes values are truncated at 25 chars, followed by ellipsis

SELECTED

INVITE = TRUE

UNLESS

AuthorCandidate.MutlipleImportsWithSameEmail

OR

AuthorCandidate.ImportMatchesMutliplePeople

AuthorCandidate.NumberOfInvitationLetters > 1 (means that the author candidate is a duplicate in the file)

## UploadedAuthorSearchResults

### Data

TRANSLATION

* Pages.Editorial.UploadedAuthors.UploadedAuthorSearchResults – “Uploaded Author Search Results”
* Pages.Editorial.UploadedAuthors.AuthorListFile – “Author List File”
* Pages.Editorial.UploadedAuthors.CandidateHeading – “Author Candidates”
* Pages.Editorial.UploadedAuthors.Instructions – “Check the box for each person you want to invite to submit and then proceed.”
* Pages.Editorial.UploadedAuthors.HeaderText = “’Author Candidates’ Results – Manuscript “

### Page

#### UploadedAuthorSearchResults.aspx/cs

Heading – ‘Authur Candidates’ Results – Manuscript SUBNUMBER <br> DTITLE

Page\_Load

Page\_PreRender

* SaveChanges() – save current grid page changes to cached dataset.
* Gather necessary data from cache, depending on current GridControl.StartIndex and GridControl.EndIndex
* Pass gathered data to GridControl.SetData

SaveChanges()

* Save the changes in the current grid page to the author candidates data being held in cache.

btnCancel\_Click

btnProceed\_Click

* proxy registration should have happened by now, so all selected items should have a person object by now.
* Call SaveChanges() to save all changes made to the current grid page data.

File Action Remove

New page. Layout looks like this:

Page Header (results - ms#<br/>"title")

Author List File "grid" which always contains only the one file, including its name, "error" indicator, size and a 'remove' link, which needs to (after displaying its validation text) invoke the same server side code as the cancel button

Clicking the yellow icon shows an instance of the Author List Overlay Dialog (1.3) with the 'Ignore' checkboxes disabled.

Author Candidates

Instructions

Instance of AuthorCandidateGrid. In a non-postback case, DataSource is InvitedAuthorManager.GetAuthorCandidates(uploadedAuthors, 1, default/stored page size)

uploadedAuthors is a List<UploadedAuthorInfo> retrieved from session cache

In any case, AuthorCandidateGrid.RecordCount needs to be uploadedAuthors.Count

Methods:

PageChangeEventHandler - passed to AuthorCandidateGrid.PageChangeEventHandler

Sets the AuthorCandidateGrid data source to InvitedAuthorManager.GetAuthorCandidates(uploadedAuthors, startIndex, endIndex).

Updates the in-cache collection of UploadedAuthorInfo objects with selections from the previous page

Submit button handler -

Client-side

Sets up spinner

Invokes server side web method to ProxyRegSelectedUsers

Sets "selectedAuthors" hidden field to value generated by InvitedAuthorManager.SerializeCandidateListForConfirmAndCustomizePage

Post to ConfirmAndCustomizeInvitedAuthors.aspx once we have completed the web method invocation.

Remove/Cancel button handler -

Client-side

Displays appropriate heads up message

Goes back to bare bones upload page (?)

WebMethod to do proxy registration.

## ConfirmAndCustomizeInvitedAuthors

### Page

#### ConfirmAndCustomizeInvitedAuthors.aspx/cs

Modify AuthorConfirmAndCustomizeEMPage.ParseSelectedAuthorsXml to emit (via output parameter perhaps) a dictionary of <peopleID, list<note>> for storage and use later in the page cycle.

Modify GetAuthorInvitationNotes – if the invite comes back as null, get the first element of the item with the given people ID from the dictionary emitted by ParseSelectAuthorsXml, then remove that element

Tokens

Pages.Editorial.UploadAuthorList.HeaderUploadAuthorList

Upload Author List

Pages.Editorial.UploadAuthorList.LegendUploadAuthorList

Author List File - Manuscript {0} <br/>

{1}

# DB Content

# FIELD

# Short Description:

# 

# Long Description:

# 

# Engineering Notes:

# 