**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 |

# 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)

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

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

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

### Code

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

##### Mapper.cs

**+ FindByOrcIDs(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

### Code

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

Private Fields:

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

##### Entity.cs

the person (if any) associated with the current “unique” uploaded author.

List<UploadedAuthorInfo> \_uploadedAuthors – list of uploaded authors that have been deemed “identical” according to the hash.

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

##### Collection.cs

\_authorInvitations – list of any author invitations.

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

##### Entity.cs

\_authorStatistics – author statistics associated with the registered person (if any).

Properties:

From the UploadedAuthors collection:

**int Hash** - this is a hash from a combination of first name, last name, email address and orcid, used for easy comparison with uploaded author records

**string FirstName**

**string LastName**

**stringEmail**

**string ORCID** - these four properties can be driven off the first element in the collection (if one exists), because the only reason for there to be more than one item in this collection is that the four non-notes fields are identical.

**bool Selected** - initially derived from majority value of Invite property of List<UploadedAuthorInfo>. Can be set otherwise.

**int? RegisteredPersonID** - the previously found match for this author candidate, if any.

**guid ID** - unique identifier, useful for the case where there is no registered person

Derived from AuthorInvitations

**int InvitedSubmissionsInProcess** – Count of all AuthorInvitations with a non-null ChildSubmission that has no final disposition.

Derived from AuthorStatistics

**int AgreedAndAwaitingSubmission** – stats.InvitedSubmissionsInProgress

**int InvitedSubmissionsReceived** - stats.CompletedInvitations

**int UninvitedBeforeAgree** - stats.UninvitedPriorToResponding. Also populates "Un-invited" part of "invitation statistics".

**int UninvitedAfterAgree** - stats.UninvitedAfterAccepting

**string LastInvitedSubmissionAgreed** - Backed by nullable DateTime. stats.DateLastAccepted, formatted according to current journal code, - if none

**string LastInvitedSubmissionReceived** - Backed by nullable DateTime. stats.DateLastCompleted, formatted according to current journal code, - if none

**string LastInvitedSubmissionDeclined** - Backed by nullable DateTime. stats.DateLastDeclined, formatted according to current journal code, - if none

**int AvgDaysToSubmit** - stats.AverageDaysToComplete

**string DateLastInvited** - Backed by nullable DateTime. stats.DateLastInvited, formatted according to current journal code, - if none

**int OutstandingInvitations** – stats.InvitationsOutstanding

**int Agreed** – stats.AcceptedInvitations

**int Declined** – stats.DeclinedInvitations

**int TotalInvitations** – stats.TotalInvitations

stand-alone

**bool ImportMatchesMultiplePeople** - indicates that an imported author has more than one potential match

**bool MultipleImportsWithSameEmail** - indicates that multiple imported authors have the same email

**int UnsolicitedSubmissionsInProcess** – data derived from a call to Aries.EditorialManager.Framework.Journal.SubmissionObjects.Submission.Mapper.GetUnsolicitedSubmissionCounts.

## InvitedAuthorManager class

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

Constructors:

**InvitedAuthorManager**(MapperManager, List<UploadedAuthorInfo>, int operatorID)  
Initializes \_uploadedAuthors, \_mapperManager and \_operatorID with passed in params.

Properties:

**List<string> UploadedOrcids**  
 - a list of the distinct orcids contained in the current \_uploadedAuthors list.

**List<string> UploadedEmails** - a list of distinct email addresses uploaded in the current \_uploadedAuthors list

**List<AuthorCandidate> GetAuthorCandidates**(List<UploadedAuthorInfo> uploadedAuthors, startIndex, endIndex)

This function prunes the list of uploaded authors to just the ones between start and endIndex inclusive, then invokes GetAuthorCandidates(pruned list) and returns that list.

**List<AuthorCandidate> GetAuthorCandidates**(List<UploadedAuthorInfo> uploadedAuthors)

This function generates a list of AuthorCandidate objects given a list of uploaded authors. It's intended to be called up from the author candidate grid.

First, we want to initialize some helper dictionaries:

Dictionary<string, FWJPerson Entity> authenticatedORCIDMatches - populated via call to FindByORCIDs(true). Key is the orcid.

Dictionary<string, List<FWJPerson.Entity>> unauthenticatedORCIDMatches - populated via call to FindByORCIDs(false). Key is the orcid.

Dictionary<string, List<FWJPerson.Entity>> emailMatches - populated via call to Aries.EditorialManager.Framework.Journal.PeopleObjects.Person.Mapper.FindByEmailMatch

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.

Then, we loop through all the AuthorCandidate records again, and assign the FWJP.Entity and Aries.EditorialManager.Framework.Journal.AuthorObjects.AuthorInvitationStats.Entity with the matching peopleID to the current AuthorCandidate record. Additionally, assign the UnsolicitedSubmissionsInProcess property from the GetUnsolicitedSubmissionCounts result. While we are looping through the AuthorCandidate records the second time, keep track of email addresses. If an email address is encountered more than once across multiple AuthorCandidate records, set the MultipleImportsWithSameEmail flag for all affected AuthorCandidate records.

Now, we, we have a list of AuthorCandidate records suitable for use in displaying on the author candidate grid.

**void ProxyRegSelectedUnregisteredUsers**(List<AuthorCandidate> authorCandidates)

This function will proxy register all author candidate records in the given list that have Selected = true but have no Person object.

We will need to generate a Aries.EditorialManager.Framework.Journal.PeopleObjects.Person.Person.Entity using the first name, last name, email address. Additionally, we will need to set its PersonalIdentifiers property to a new entity and set the Orcid there to the author candidate's orcid and unauthenticated.

At this point, we invoke Aries.EditorialManager.RegistrationManager.RegisterUser(person, \_operatorID)

**string SerializeCandidateListForConfirmAndCustomizePage**(List<AuthorCandidate> authorCandidates)

This function produces XML suitable for parsing by the AuthorConfirmAndCustomizeEMPage.ParseSelectedAuthorsXml method.

Note that it assumes all author candidates in the given list have a Person (i.e. have been registered)

The XML needs to be in the format of

<PeopleSearchXml>  
<InvitedAuthorsXml>  
<author> <numLetters>AuthorCandidate.NumInvitationLetters</numLetters> <id>AuthorCandidate.PeopleID</id> <note>AuthorCandidate.NotesString</note>  
</author> ... multiple authors possible  
</InvitedAuthorsXml>  
</PeopleSearchXml>

## AuthorCandidateGrid

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

## UploadedAuthorSearchResults.aspx/cs

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 (?)

## 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:

# 