**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

##### AuthorCandidate.cs

Constructor:

**AuthorCandidate(UploadedAuthorInfo authorFromFile, int ID)**

* Set MatchedPersonStatistics to null
* Set MatchedPersonExistingInvitations to null
* Set \_authorFromFile private field to incoming UploadedAuthorInfo
* Set \_candidateID private field to incoming ID
* Set Selected property based on incoming UploadedAuthorInfo.Invite
* Set NumberOfInvitationLetters to 1 if incoming UploadedAuthorInfo.Invite == true

Fields:

**\_authorFromFile**

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

Properties:

**AcceptedInvitations**

* Number of invitations to which author has agreed
* Exposes \_matchedPersonStatistics.AcceptedInvitations

**AverageDaysToComplete** - MatchedPersonStatistics.AverageDaysToComplete

**CandidateID** - Represents the manufactured ID of this candidate in the listing of matched candidates

**CompletedInvitations** - MatchedPersonStatistics.CompletedInvitations

**DateLastAccepted**

* MatchedPersonStatistics.DateLastAccepted
* Remains a nullable DateTime – needs to be converted appropriately at display

**DateLastCompleted**

* MatchedPersonStatistics.DateLastCompleted
* Remains a nullable DateTime – needs to be converted appropriately at display

**DateLastDeclined**

* MatchedPersonStatistics.DateLastDeclined
* Remains a nullable DateTime – needs to be converted appropriately at display

**DateLastInvited** -

* MatchedPersonStatistics.DateLastInvited
* Remains a nullable DateTime – needs to be converted appropriately at display

**DeclinedInvitations -** MatchedPersonStatistics.DeclinedInvitations

**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

**InvitationsOutstanding** – MatchedPersonStatistics.InvitationsOutstanding

**InvitedSubmissionsInProcess**

* Number of submissions that author has been invited where there are child submissions that have no final disposition.
* Count of MatchedPersonExistingInvitations where FinalDisposition = None

**InvitedSubmissionsInProgress -** MatchedPersonStatistics.InvitedSubmissionsInProgress

**LastName** - \_authorFromFile.LastName

**MatchedPerson** –

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

**MatchedPersonFullNameWithTitleAndDegree** – MatchedPerson.

FullNameWithTitleAndDegree

**MatchedPersonID** – MatchedPerson.IdentityValue

**MatchedPersonIsBoardMember** – MatchedPerson.Board

**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

**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

**NumberOfInvitationLetters**

* Number of invitation letters to send
* Initially set at constructor to 1

**ORCID** - \_authorFromFile.OrcID

**ORCIDIsAuthenticated** – MatchedPerson.HasAuthenticatedOrcid

**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

**TotalInvitations** – MatchedPersonStatistics.TotalInvitations

**UninvitedAfterAccepting** – MatchedPersonStatistics.UninvitedAfterAccepting

**UninvitedPriorToResponding** – MatchedPersonStatistics.UnivitedPriorToResponding

**UnsolicitedSubmissionsInProcess**

Set externally via 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

**CONSTANTS**:

authorCandidatesCacheDataLocation – “AuthorCandidateList”

uploadedAuthorListCacheDataLocation – “UploadedAuthorList”

**PROPERTIES**:

**int NumberOfCandidates** – returns number of candidates created after call to MatchUploadList.

**METHODS**:

**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
* In loop
  + proxy registers all selected/non-matched (Selected == true, HasMatchedPerson == false) persons in em and returns created person entities using RegistrationManager.ProxyRegisterUser.
  + Updates author candidate list with returned 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.
* Creates one ‘record’ for each invitation – if author candidate has more than one invitation, breaks into single invitation records.
  + Each invitation gets ONE of the UploadedNotes in the *note* element of the record.

Xml data structure:

<PeopleSearchXml>

<InvitedAuthorsXml>

<author>

<numLetters>AuthorCandidate.NumberOfInvitationLetters

</numLetters>

<id>AuthorCandidate.MatchedPersonID</id>

<note>AuthorCandidate.UploadedNotes</notes>

</author>

</InvitedAuthorsXml>

</PeopleSearchXml>

**Support Objects**:

**AuthorCandidateGridData** – object indicating data that can change in author candidate grid.

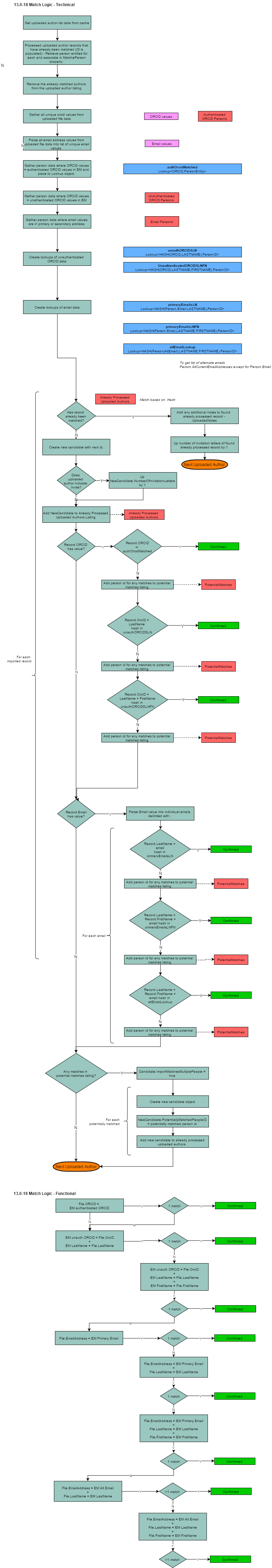
intCandidateID – candidate id given when list was created in *MatchUploadList*

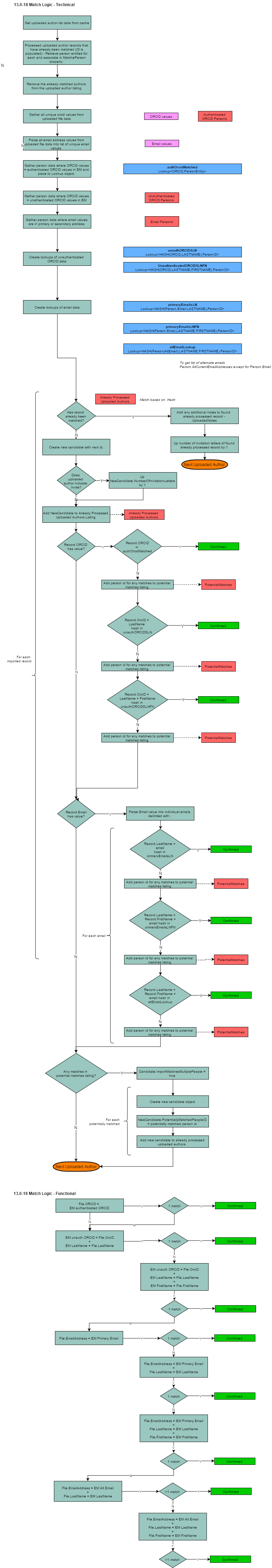
bool Selected – whether candidate is selected for invite in the grid

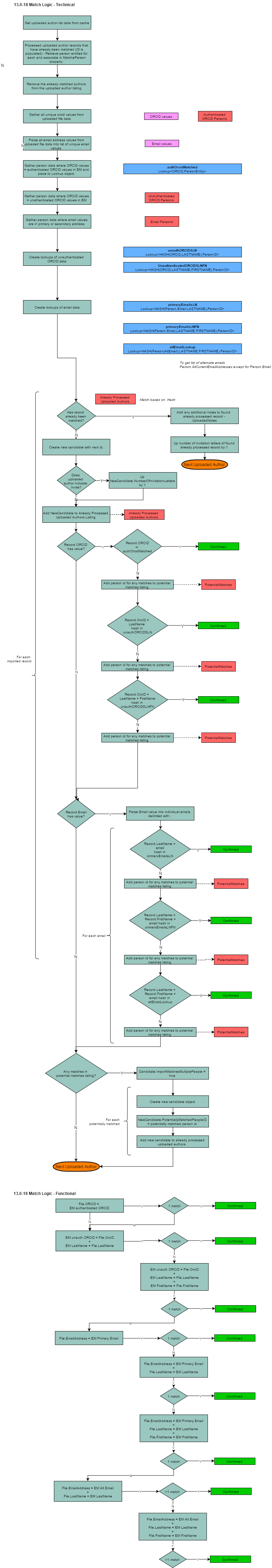
int NumberOfInvites – number of invites indicated in the grid

##### Matching Logic

**MatchUploadList(DatabaseMapperManager mapperManager, string journal, string emSessionID)**







## AuthorCandidateGrid - Jean

### Data

TRANSLATION

* Pages.Editorial.AuthorCandidateGrid.AuthorInvited – “Author Invited:”
* Pages.Editorial.AuthorCandidateGrid.ConfirmedMatch – “Confirmed Match:”
* Pages.Editorial.AuthorCandidateGrid.InformationFromFile – “Information from File:”
* Pages.Editorial.AuthorCandidateGrid.MultipleMatchedEmails – “Multiple imported Author records contain the same email address. Please select only the record you wish to invite.”
* Pages.Editorial.AuthorCandidateGrid.NoMatchFound – “No Match Found; Account will be proxy-registered.”
* Pages.Editorial.AuthorCandidateGrid.Note – “Note:”
* Pages.Editorial.AuthorCandidateGrid.Declined – “Declined:”

### 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 at currently so that, upon refresh (paginator will refresh on page change and page size) that value can be reset in the paginator.

#### Aries.EditorialManager.Web

##### WholeNumbersOnly\_KeyDown.js

* Add to Resources
* Contains function WholeNumbersOnly\_KeyDown which will be attached to keydown event of invitation number text boxes.

#### common\WebUserControls

##### AuthorCandidateGrid.ascx

**startup()**

* function to be called at document.ready
* hook invite number text boxes keydown event to WholeNumbersOnly\_KeyDown function that will be placed by Page\_Load – selector(.inviteTextBox)
* hook selected check boxes change event to selectCheckbox\_Changed

**selectCheckbox\_Changed(e)**

* handles change event of select check boxes
* selector (.cbSelected)
* Enable invite text box and place value to 1 if checked
* Disable invite text box and clear value if unchecked

**table UploadedAuthorResultsGrid**

* table containing author candidate rows.
* All rows have class *candidateRow*.
* Row uniquely identified through *data-candidate* attribute value of which is the AuthorCandidate.CandidateID.
* All select checkboxes have class *colSelected*.
* Select checkbox uniquely identified through *data-candidate* attribute value of which is the AuthorCandidate.CandidateID.
* All invitation text boxes have class *invitesTextBox*.

**TopPaginator/BottomPaginator**

* Top and bottom of grid – GridPaginator.ascx

##### AuthorCandidateGrid.ascx.cs

To be similar to InvitedReviewersGrid.ascx.

Uses common/WebUserControls/GridPaginator.ascx

Top div includes GridPaginator

Middle div is HTMLTable – *UploadedAuthorResultsGrid*

Bottom div includes GridPaginator

**Private Fields**:

**\_peopleFlagManager** – Aries.EditorialManager.FlagManager – for use in creating custom flags of successfully matched persons.

**Properties**:

**protected EMPage OwnerPage** – allows convenience to containing EMPage object of containing page.

**public EditorRoleManager** Aries.EditorialManager.Framework.Journal.RoleManager.EditorRolePermissions.Entity – \_peopleFlagManager will need this to determine if flags can be shown.

**public int StartIndex** – holds the paginator’s start index value when fired.

**public int EndIndex** – holds the paginator’s end index value when fired.

**public bool PaginationChanged** – allows containing page to know that pagination has changed and retrieve of a new set of data for the grid is necessary.

**public int RecordsCount** – set only – allows external setting of paginator RecordsCount property.

**Paginator Events**:

**GridPaginator\_PageSelected**

* Fires when the grid paginator control’s page number changes, either by pressing the actual number page link or pressing the next/previous buttons.
* Set paginators.CurrentPage to e.SelectedPage
* Do a DataBind on both paginators – not sure why this would be necessary although InvitedReviewersGrid.ascx is doing the same so, when in Rome…
* Capture the start and ending index indicated by the paginator into StartIndex and EndIndex properties.
* Store the current page value in the cache to be retrieved once the owner page refresh occurs, which will be caused by paginator. Will allow the containing page to indicate to the paginator, what page it is currently to display. Save in location name created in Constants.cs, previously.
* Set PaginationChanged = true – used as an indicator in the containing page that pagination has changed and that it should be reset (essentially, there is now a ‘new’ paginator that needs to be recreated/repopulated.

**GridPaginator\_PageSizeSelected**

* Capture the selected page size, if it exists from e.SelectedPageSize or, if it doesn’t, default to a page size of 10 records - place into pageSize local variable.
* Set PageSizeForGrid in StateManager – not quite sure why, but, again, InvitedReviewersGrid is doing it so…
* Set PageSize of both paginators
* Do a DataBind on both paginators – again, not sure why but if you saw someone jump off of a bridge, you’d do it too, right?
* Capture the start and ending index indicated by the paginator into StartIndex and EndIndex properties for later usage.
* Store the current page value in the cache to be retrieved once page refresh caused by paginator
* Set PaginationChanged = true

**Page Events**:

**Page\_Load**

* Retrieve translation values from resource loader into local fields.
* Create html string for multiple matched emails to be used in grid creation.
* Create html string for no match found to be used in grid creation.
* Set \_currentPage retrieved from cache (CURRENT\_UPLOAD\_AUTHORS\_RESULTS\_PAGE\_CACHE\_KEY). – set it to 1 if there is no current page – will be used in Page\_PreRender to reset the CurrentPage of the paginators.
* Set \_pageSize from StateManager.PageSizeForGrid (gues this is why we saved it previously in the paginator events, yes?) – this will be used in Page\_PreRender to reset the PageSize property of the paginators.
* Register any client scripts
  + CUSTOM\_FLAGS\_SCRIPT – for grid creation and adding people flags where appropriate
  + WHOLE\_NUMBERS\_ONLY\_TEXTBOX\_KEYDOWN – created a new function to handle invite text box KeyDown event to prevent all but numeric input.
  + ORCID – includes functions to popup a page to show ORCID data, if authenticated.

**Page\_PreRender**

* Only if PaginationChanged:
  + Set paginators.CurrentPage = \_currentPage
  + Set paginators.PageSize = \_pageSize
  + Do a DataBind on paginators

**Page Methods**:

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

Externally available function to allow containing page to display the correct records requested by the paginator. Containing page will determine, based on StartIndex and EndIndex, the candidate records to request in the grid.

Builds html to append to UploadedAuthorResultsGrid (HTMLTable).

Each table row will contain attribute *data-candidate* with value being AuthorCandidate.CandidateID.

Various private helper functions:

*BuildAuthorStats* – builds the author statistics cell for a particular candidate.

*BuildConfirmedPerson* – builds the html to display a candidate that has been successfully matched against an EM entity, in the AuthorName cell. Uses \_flagManager to get custom people flags/creator, if possible

*BuildInfoFromFile* – builds the file info portion of the AuthorName cell.

*BuildInviteStats* – builds the invite statistics cell for a particular candidate.

## UploadedAuthorSearchResults - Jean

### 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 “
* Pages.Editorial.UploadedAuthors.ProxyWaitText = “Please wait while proxy-registration is performed”
* Pages.Editorial.UploadedAuthors.RemoveFileWarning = “This will remove the uploaded file and discard any existing selections. Are you sure?”
* Pages.Editorial.UploadedAuthors.ProxyRegError = “The following error occurred while attempting to proxy register authors:”
* Pages.Editorial.UploadedAuthors.ProxyRegErrorTitle = “Proxy registration error”
* Pages.Editorial.UploadedAuthors.RemoveFileError = “The following error occurred while attempting to remove uploaded author data:”
* Pages.Editorial.UploadedAuthors.RemoveFileErrorTitle=”Remove file error”
* Pages.Editorial.UploadedAuthors.FileWarningTooltip = “This file contains incomplete or invalid records. Click for more information.”

### Page

#### UploadedAuthorSearchResults.aspx

Include AuthorCandidateGrid.ascx

**startup()**

* function to be called at document.ready

**cancel\_click**:

* shows jquery dialog with OK/Cancel.
* OK returns true to allow server click event to fire.
* Cancel returns false to prevent server click event from firing.

**proceed\_click**

* Creates an JSON object to mimic ImportedAuthorManager.AuthorCandidateGridData for each author candidate grid row (AuthorCandidateGrid control) – into array.
* Opens jquery proxy wait dialog ($dialogProxyWait) – modal
* Makes ajax call to UploadedAuthorSearchResults.aspx/ProxyRegSelectedAuthorCandidates – passing array created previously to proxy register any necessary selected candidates.
* Proxy register fails – return false to prevent proceed button server event from firing.
* Proxy register success – remove proxy wait dialog and return true to allow proceed button server event to fire.

**removeFile**

* Show warning message with OK/Cancel.
* ajax call to RemoveUploadedAuthorsFileData that calls to remove all uploaded info from cache.
* Return to file upload page.

#### UploadedAuthorSearchResults.aspx.cs

Took paginator/grid control handling direction from SearchReviewers.aspx which includes the InvitedReviewersGrid.ascx – on which the AuthorCandidateGrid.ascx is based.

**Page Events**:

**Page\_Load**

* Get submission data using EMPage.CurrentDocumentID
* Set header text to Pages.Editorial.UploadedAuthors.HeaderText + Submission.ManuscriptNumber + line break + Submission.Title.
* Create file data row in tblFileData.
* Do a DataBind – not sure if this is necessary but other search results pages with grid controls on them do it so…

**Page\_PreRender**

* If AuthorCandidateGrid.PaginationChanged, call SaveChanges() – save current grid page changes to cached dataset.
* Gather necessary author candidates from cache, depending on current AuthorCandidateGrid.StartIndex and AuthorCandidateGrid.EndIndex
* Pass gathered data to AuthorCandidateGrid.SetData to populate the current grid page.

**Button Events**:

**btnCancel\_Click**

User will have been given a jquery dialog indicating that all cache data will be removed if they cancel. If we have reached this point, they have agreed to that.

* Remove author candidates from cache (ImportedAuthorManager.authorCandidatesCacheDataLocation)
* Remove uploaded authors from cache (ImportedAuthorManager.uploadedAuthorListCacheDataLocation)
* Remove uploaded file from database ?
* Redirect to upload author list page.

**btnProceed\_Click**

* proxy registration should have happened by now, so all selected author candidate items should have a person object by now. As well, all grid changes will have taken place prior to proxy registration.
* CurrentSession.CurrentPeopleSearchXML = ImportedAuthorManager.SerializeCandidatesForConfirmAndCustomize – creates xml and places it in session object location for use by ConfirmAndCustomizeInvitedAuthors.aspx.
* Call ConfirmAndCustomizeInvitedAuthors.aspx?authorSearchMode = Constants.AUTHOR\_SEARCH\_MODE\_UPLOAD\_AUTHOR\_LIST.

**Page Functionality**:

**private void SaveChanges()**

Wrapper function to prepare data and call save function via *ImportedAuthorManager.SaveGridChanges*.

**private string BuildFileGrid()**

* return html string for file data grid row
* show warning graphic if import file errors
* get data from ImportedAuthorManager.GetAuthorListFileFromCache()

**Web Methods**:

**public static void ProxyRegSelectedAuthorCandidates (List<ImportedAuthorManager.AuthorCandidateGridData gridData)**

* Create a new EMData object
* Save any grid changes passed in via gridData parameter via *ImportedAuthorManager.SaveGridChanges*
* Gather author candidates from cache (authorCandidatesCacheDataLocation) and pass them for proxy registration via *ImportedAuthorManager.ProxyRegSelectedUnregisteredUsers*.

**public static void RemoveUploadedAuthorsFileData()**

* Create a new EMData object
* Call ImportedAuthorManager.ClearAuthorListFileCache

## ConfirmAndCustomizeInvitedAuthors - Jean

### Page

#### AuthorConfirmAndCustomizeEMPage.cs

This is basis of ConfirmAndCustomizeInvitedAuthors.aspx page.

**New**:

**private string \_authorSearchMode**

holds passed in authorSearchMode from query string.

**private static List<KeyValuePair<int,string>> \_peopleNotes**

will hold a listing of peopleIDs (KEY) and related note passed in (VALUE) from the UploadedAuthorSearchResults page through note element in the author element of <InvitedAuthorsXml> portion of <PeopleSearchXml>.

See *ParseSelectedAuthorsXml* below for population.

See *GetAuthorInvitationNotes* below for usage.

**InitializeBaseObjects()**

*fromSearchResultsPage = TRUE*

determined if Request does not have ‘fromLetterCustomizationPage’ in query string which should be the case when called by *UploadedAuthorSearchResults.aspx*

*\_pageXml = CurrentSession.CurrentPeopleSearchXML*

This should already be populated in the call from *UploadedAuthorsSearchResults*.*aspx at the* btnProceed\_Click event by calling *ImportedAuthorManager.SerializeCandidatesForConfirmAndCustomize* – which will populate the <*InvitedAuthorsXml*> portion of the <*PeopleSearchXml*>

**CHANGE:**

* Capture authorSearchMode from queryString into \_authorSearchMode private field to be used later in StoreSessionVariables() and InitializeHiddenFields
* Use \_authorSearchMode in InitializeHiddenFields parameter – fromSearchResultsPage == true && \_authorSearchMode != Constants.AUTHOR\_SEARCH\_MODE\_UPLOAD\_AUTHOR\_LIST
* In the case of coming from the UploadedAuthorSearchResultsPage, fromSearchResults will be TRUE and second condition will be FALSE, so, parameter value passed to InitializeHiddenFields will be false, meaning that the majority of the hidden fields will be populated either from the current session object and/or the \_pageXml object, which is the correct way of populating these fields in this case.

**StoreSessionVariables()**

* When \_authorSearchMode = Constants.AUTHOR\_SEARCH\_MODE\_UPLOAD\_AUTHOR\_LIST (meaning the page has been arrived at from the UploadedAuthorSerachResults.aspx page) –
  + set the eventID node of <PeopleSearchXml> to EventType.InviteAuthorsForProposal.
  + DO NOT populate the \_pageXml<InvitedAuthorsXml> - it will already exist as it was set via the session object when we serialized the selected author candidates at ImportedAuthorManager.SerializeCandidatesForConfirmAndCustomize.
* Set AuthorSearchMode node of <PeopleSearchXml> to \_authorSearchMode

**InitializeHiddenFields**

Just populate hidden field hdnAuthorSearchMode from \_authorSearchMode captured at InitializeBaseObjects, no matter the input parameter.

**ParseSelectedAuthorsXml**

This function will be called within the ConfirmAndCustomizeInvitedAuthors.Page\_Load event.

At this event, it is asking for just an array of people ids that came in through CurrentSession.CurrentPeopleSearchXML, that it can use in the call to \_ELM.AddRecipients.

Take advantage of this call and the loop through all selected author candidates and capture the <note> element value in a KeyValuePair (KEY = peopleID & VALUE = <note>) and place in \_peopleNotes list for later use in creating populating the notes text box with a value in *GetAuthorInvitationNotes*.

**GetAuthorInvitationNotes**

This function is arrived at via

Page\_Load->generateRecipientListHTML->

GenerateLettersTableCell->GetAuthorInvitationNotes

GenerateLettersTableCell creates the author cell which may include a notes text box if it is visible through config settings.

GenerateLettersTableCell calls GetAuthorInvitationNotes for a value for the notes text box if it cannot find a note value in an existing invitation (\_xmlManager.InvitationNotes) to see if one can be gathered.

In the case of arrival to this page via UploadedAuthorSearchResults, there will be nothing in \_xmlManager, since there will not yet be any invitations, so GetAuthorInvitationNotes will be called for each created row.

In this case, after the check of the database for a note through \_authorInvitationManager.GetInvitedAuthorForDocIDAndEventLetterID (which will return null in this case) – go through the \_peopleNotes field KeyValuePair listing created previously (at ParseSelectedAuthorsXml), and see if there is a note that can be found related to the personID (meaning there was a related note in the uploaded authors file for this person). Use this value at the existing note.

#### ConfirmAndCustomizeInvitedAuthors.aspx.cs

**cancelButtonClick**

* Server.Transfer to UploadedAuthorSearchResults.aspx

# DB Content

# FIELD

# Short Description:

# 

# Long Description:

# 

# Engineering Notes:

# 