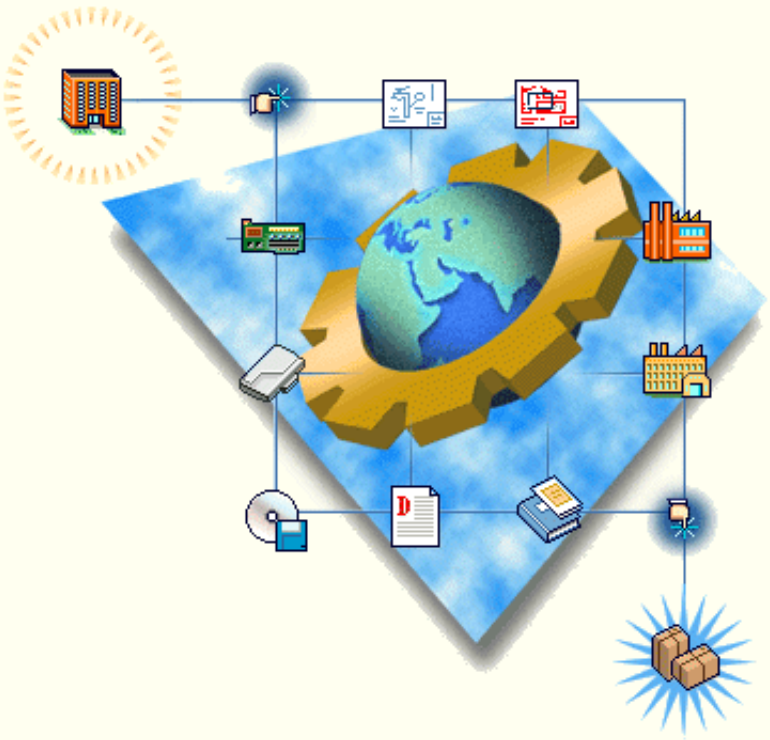
MatrixOne
Common Components

Version 10.5



Administrator Guide

Copyrights and Trademarks

The documentation that accompanies MatrixOne Applications describes the applications as delivered by MatrixOne. This documentation includes readme files, online help, user guides, and administrator guides. If changes are made to an application or to the underlying framework, MatrixOne cannot ensure the accuracy of this documentation. These changes include but are not limited to: changing onscreen text, adding or removing fields on a page, making changes to the administrative objects in the schema, adding new JSPs or changing existing JSPs, changing trigger programs, changing the installation or login process, or changing the values in any properties file. For instructions on customizing the provided documentation, see the Application Exchange Framework Guide.

DM-MX-35-10-50

MatrixOne Inc.
210 Littleton Road
Westford, MA 01886, USA
Telephone: 978-589 4000
Fax: 978-589-5700
Email: info@matrixone.com

Web Address: http://www.matrixone.com

Table of Contents

Introduction to Common Components

Common components includes programs and schema that are common to many MatrixOne applications. When an application installs, it updates common components as needed for its features. Features included in common components dependent on schema, while features in the Application Exchange Framework are less schema dependent.

For version 10.5, common components includes:

- Routes and tasks and route templates
- Vault preference for searching
- Common profile management
- Common document management
- Subscriptions
- Discussions
- Issue Management
- Common file search

An application can use the common components to implement a feature or can use its own programs. Most common components are configurable so each application can tailor it as needed. For example, the checkin pages can be configured to show specific fields and attributes. If an application has requirements that aren't included in the common component, it uses its own programs.

Adding Vaults, Companies, and Users

Overview

To register a company and its employees who need to use the application, perform these steps.

1. Determine the vaulting strategy for the company using the guidelines described in ***Determining Company Vaulting Strategies***.
2. If the company needs one or more vaults that are not already created, create the vault(s) using Matrix System or MQL.
3. Make sure all vaults are registered using the eService Change Administration Property wizard in Matrix Navigator. For instructions on registering objects, see the ***Application Exchange Framework Guide***.
4. Using the Administration pages, create the company. Assign the company's primary vault and secondary vault as determined by the vaulting strategy. For instructions on creating companies and persons, see the ***Common Components User Guide*** or Online help.
5. Using the Administration pages, add at least one employee for each company and designate the person a Company Representative so the person can add additional employees and other profile information for the company. Employees who need to use Program Central must be assigned the Project User or External Project User role.

   If the company has secondary vaults, assign the employee's default vault according to the company's vaulting strategy. By default, a person's default vault is the same as the company's primary vault.

Schema Requirements for Companies and Users

These are the schema requirements needed for a person to successfully access the Common Components application. The system fulfills all these requirements automatically when a company or person is added using the Administration pages.

- To represent the person's company, a business object of type Company must be created and promoted to the Active state. If the company has business units, a Business Unit object should be created for each and promoted to Active. The Business Units are connected to the parent Company with the Division relationship.
- To represent a person who uses the system, a person administrative object must be defined with default settings for privileges. The role assignments that affect a person's ability to log into and use Common Components, and access various menus, tasks, or objects are outlined in the user guide. You can assign the person to as many roles as needed. For example, if the person will need to create projects in Common Components and is an employee of the host company, you should assign the person to the Project User and Project Lead roles.
- Also to represent the person, a business object of type Person must be created with the same name as the administrative object. To log in, the Person business object must be in the Active state.
- The Person business object must be connected to the Company business object with the Employee relationship. If the person is assigned to a business unit, the Person business object must also be connected to the Business Unit with the Business Unit Employee relationship.

- If the person will need to edit profile information for his/her company, including adding locations and persons, the Person business object must also be connected to the Company business object with the Company Representative relationship and assigned the Organization Manager role. The system automatically assigns this role when a person is designated as a Company Representative.

Determining Company Vaulting Strategies

A vault is a container for business objects created within MatrixOne applications. Every company must be assigned to at least one vault and every person must be assigned to one of the vaults assigned to the person's company. Many object types that a person creates—for example, programs, project spaces, project concepts, project templates, and business goals—are stored in the person's default vault. Other objects take on the vault of their parent object. For example, project folders, tasks, documents, version documents, risks, RPNs, assessments, financial items, cost items, benefits items, quality, quality metrics, and routes are created in the same vault as their parent object, which is typically the creator's default vault.

To optimize performance, assign vaults to companies so as to minimize the number of vaults used overall and to distribute objects across vaults as evenly as possible. You shouldn't have some vaults that have very few objects and others with very many objects.

Company Vaults

Every company must have a primary vault and can be assigned one or more secondary vaults. Each person added to the company is then assigned a default vault, which must be either the company's primary vault or one of its secondary vaults.

When host company personnel add a company to Program Central, they must specify a primary vault for the company. They can choose any vault that has been added to the system. A company's primary vault is where the company object is created and, by default, where all the company's objects are created such as business units, locations, and persons. Once set, a company's primary vault cannot be changed.

In addition, the host company can assign one or more secondary vaults for a company. The host company can add additional secondary vaults at any time but cannot remove them from a company.

Person's Default Vault

A person's default vault is the vault the system will store all objects created by the person. The company's primary vault is automatically assigned as the default vault. If the company has secondary vaults assigned, you can choose one of them. Once set, a person's default vault cannot be changed. The host company assigns vaults for companies.

Vaults and Searches

By default, searches are performed across all primary and secondary vaults for the user's company. During each basic search, users can choose to search in a specific company vault or in all company local vaults. (Advanced searches are always executed across all company vaults.) Narrowing a search to specific vaults speeds up the search but can lead to unexpected results if the user isn't sure where objects are stored.

If users typically want to search in one vault or all local vaults, they can set a global preference by clicking **Tools** > **Preferences** on the main toolbar.

Common Document Management

Common Document Management enables all MatrixOne applications to manage documents and files in a similar manner, and promotes the sharing of these objects among the different applications.

Common Document Management includes these features:

- File Versioning
- Document Revising
- Search Capabilities
- Common Checkin
- Common Checkout
- Multiple File Download
- Purge
- Checkin from MQL

File Versioning

A version of a file is created when a user checks out the file from the “Document” object by locking it, then makes changes to the file, and finally checks it back in. Since the root Document object is not locked, other users can work on the remaining files concurrently.

When the new version is checked in, the system revises (using the core revisioning mechanism) the corresponding “Document” (Version) object and updates meta data such as “Description” and “Title”. Files in the Master will be moved to the earlier version of the Document (Version), and the latest file version will be checked into the Master object. The “Latest Version” and “Active Version” relationships are floated on the revision at the “Document” (Version) end. The system handles relationship connects and disconnects.

Users with proper access can delete active file versions. The system will delete the file from the Document object and move the latest version of the file into the Document object (from previous Version object). Also, it deletes the corresponding Version object and connects the previous “Document” (Version) object with the Latest Version and Active Version relationships.

Document Revising

The root Document object can be revised in two ways

- Revise with Files
- Revise without Files

When a user chooses to ***Revise with Files***, the system revises the root Document object, clones the connected “Document” (Version) objects (using the “Active Version” relationship) and connects them to the revised Document.

When a user chooses to ***Revise without Files***, the system revises the root Document object and does not bring forward (clone and connect) the Version objects.

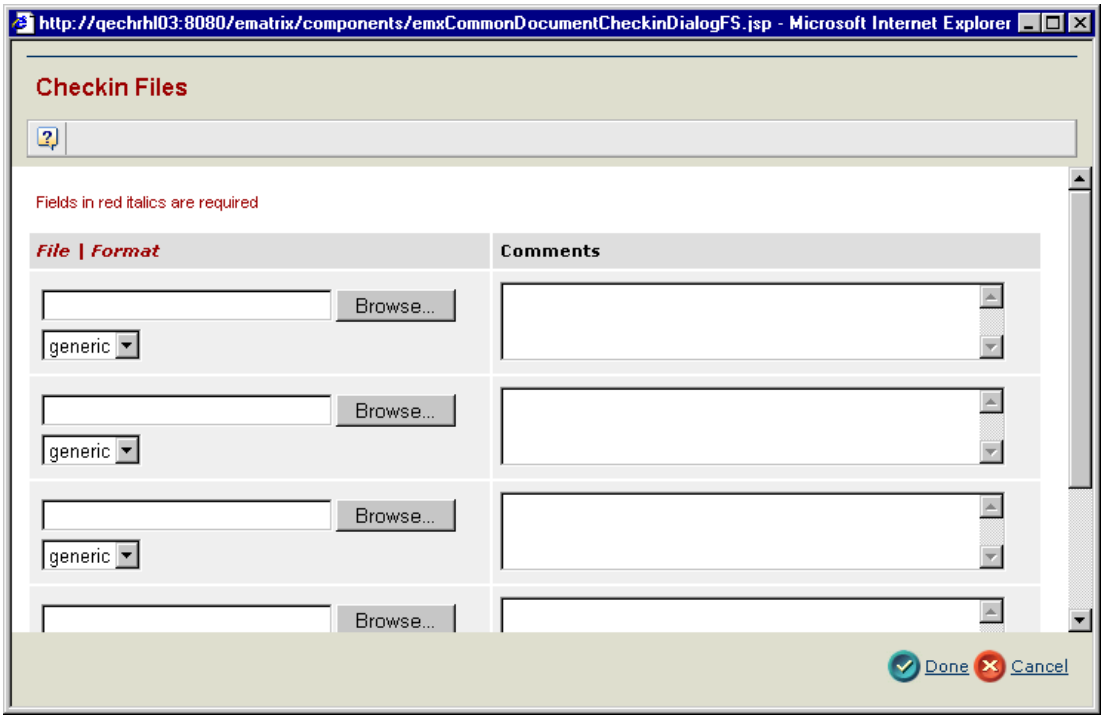
Search Capabilities

Document search pages have a radio button option to search for “Revisions Only” or for “Revisions and Versions”. For full text searches in Documents, searches may also be performed for “Revisions Only” or for “Revisions and Versions”.

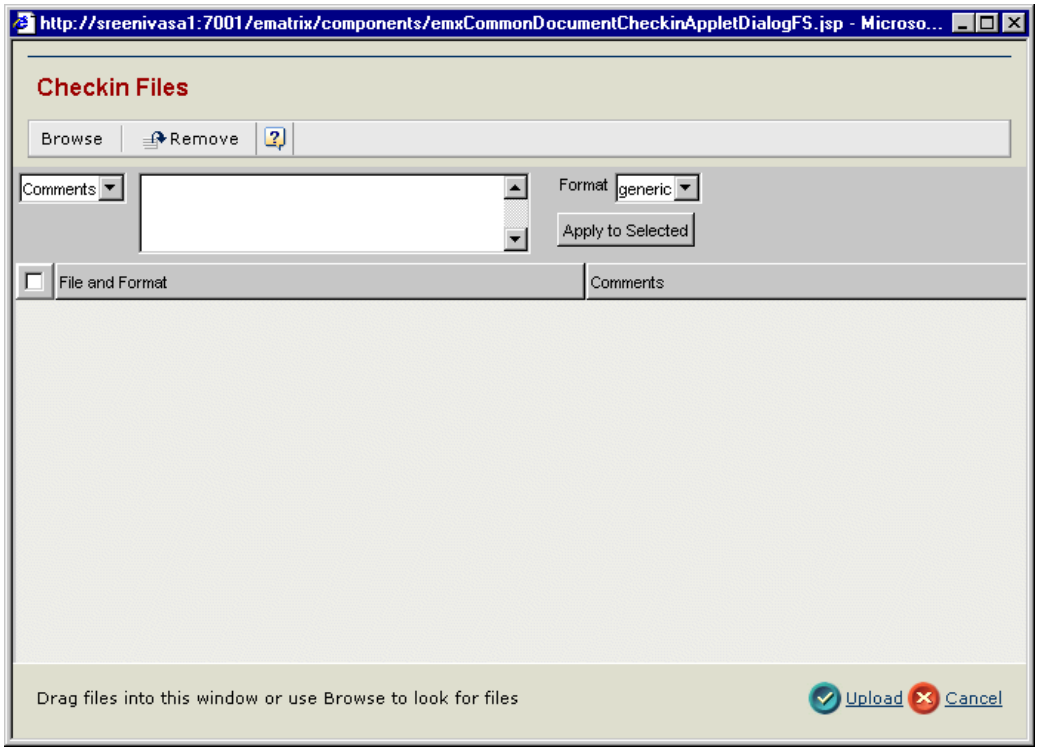
Common Checkin

This section describes the URL parameters and JPO methods to use for common components checkin. Some features are controlled using configurable properties. See ***Checkin Properties***.

Common Components includes the programs (JSPs and JPOs) needed to check in new files and updated versions of existing files. The pages can be configured to work with the Enhanced File Collaboration Servers (EFCS) or without. You can call a page that allows multiple files to be checked in at once:



WIth EFCS enabled, you can use a drag-and-drop Applet to checkin multiple files:



Enabling the Drag-and-Drop Applet with Checkin

You can enable the following applet to allow users to drag-and-drop and checkin multiple files:

`emxCommonDocumentCheckinAppletDialogFS.jsp`

The applet requires JVM 1.4, which is automatically downloaded and installed on the user's machine if the `emxframework.useapplet` setting is set to "TRUE" in the `emxsystem.properties` file. You must also enable FCS, as described in the ***System Manager Guide***. You can also restrict use of the checkin applet by company by specifying the company name as part of the setting. For example, to allow users in the company named acme to use the checkin applet, you would create a setting called `emxframework.useapplet.acme = TRUE`.

When the applet is first installed, the user will see dialogs asking to download the applet. Once installed on the user's client machine, the applet will display automatically.

The drag-and-drop applet is not supported on Macintosh clients nor on non-UTF-8 servers.

Enabling Downloads of Files Packages

Users with read access to an EBOM's component parts can generate and download a package of files connected to a part and its associated objects. To enable downloading of file packages you must enable FCS as described in the ***System Manager Guide***.

Checkin JSPs and URL Parameters

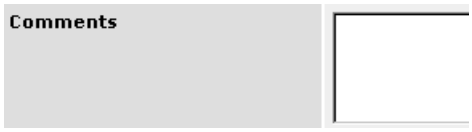
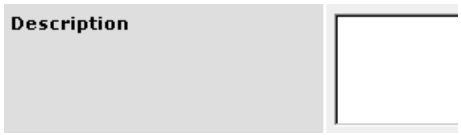
The following table lists the URL parameters accepted by the checkin JSPs. If a parameter is not passed in with the URL, the default value is assumed.

URL Parameter	Description	Accepted Values and Examples
allowFileNameChange	Determines whether the user can change the name of the file when checking in a new version.	true (default)—File name field is not editable for updating files. true—File name field is editable for updating files.
appDir	The name of the directory the application uses.	appDir=programcentral appDir=sourcingcentral
appName	The name of the application that is calling the page.	appName=program appName=sourcing
appProcessPage	Specifies the name of a JSP to call after checkin is complete. For example, in Sourcing Central, a new page is called after checkin for RFQ template standard content. When checkin occurs, the RFQ template is revised so the tree and properties page must be refreshed for the new revision. If this parameter is not sent, the current page is refreshed after checkin.	The name of any valid JSP.
defaultType	Specifies the type of object to create when creating a new object to check the file into. The default type is Document and is used if this parameter is not passed.	Symbolic name of the type to create: defaultType=type_DrawingPrint
folderURL	Used in conjunction with showFolder to specify the page for the range helper URL to call for the folder chooser.	Valid JSP name plus path for folder chooser. `folderURL=../teamcentral/emxTeamUploadFolderDialogFS.jsp`

JPOName	Specifies the name of the JPO that extends emxCommonFile. Most MatrixOne applications install a JPO that extends emxCommonFile to implement application-specific pre-checkin and post-checkin logic.	Name of JPO that extends emxCommonDocument.
		`JPOName=emxTeamDocument`
		`JPOName=emxProgramDocument`

objectAction	Determines the actions that can be taken on a document object.	Create—A two-step wizard at the end of second step. It creates a master object with details given in step 1 and checks in files selected in step 2. This takes defaultType createMasterPerFile—A one-step create of 'Document' Object. By default can be overriden by passing defaultType=tye_xyz. update—Used to update all files locked in the current object. updateHolder—Used to update all files locked under a Holder object, such as a folder. checkin—Used to add new files to the document object.

URL Parameter	Description	Accepted Values and Examples
objectId	The objectId of the object to revise. Use for single file checkin processes that let users check in an updated revision of a file to an existing business object. Use in conjunction with the objectAction=revise parameter to have the system revise the specified object and then check in the file to the new revision. This parameter is not used for multi-file revisions. Use objectAction=revise and the parentId instead.	Valid object ID for the object to be revised. `objectId=33600.6795.43493.8032`
parentId	The object ID for the object the file is associated with. Use when uploading a new file that should be connected to an existing business object, for example, when uploading a new reference document for a part family. Use in conjunction with the objectAction=create, defaultAliasType, and parentRelName parameters to have the system: • create a new object of the type specified in defaultAliasType • connect the new object to the parentID object with the relationship specified in parentRelName • check in the file to the new object Also use in conjunction with objectAction=updateHolder to version multiple files. The system gets all objects connected to the parentId object that have been locked by the current user and displays them in the checkin dialog. Either a parentId or objectId must be specified.	Valid object ID the file is associated with. For example, in Sourcing Central, the parentID for a package attachment would be the object ID for the package. `parentId=33600.6795.26916.54398`
parentRelName	Specifies the symbolic name of the relationship that should connect the parentId object and the new object the file is checked into. Used in conjunction with parentId and objectAction=create. Defaults to reference document if nothing is sent.	The symbolic name of a relationship. parentRelName=relationship_VaultedDocuments
showAccessType	Shows/hides the Access Type list, which lets users select Inherited or Specific. Inherited means the file inherits accesses from the folder. Specific means no accesses are inherited. 	false (default)—Access Type field is not displayed. true—Access Type field is displayed. Inherited is selected by default.

URL Parameter	Description	Accepted Values and Examples
showComments	Show/hides the Comments field, which lets users enter comments about checking in an updated version of a file. 	false—Comments field is not shown. true (default)—Comments field is displayed but not required. required—Comments field is required.
showDescription	Shows/hides the Description field, which lets users enter a description for the file. 	false—Description field is not shown. true (default)—Description field is displayed but not required. required—Description field is required.
showFolder	Displays/hides the Folder field, which lets users specify a folder to upload the file to when adding a file to a route. Used in conjunction with the folderURL parameter.	false (default)—Folder field is not displayed. true—Folder field is displayed.
showFormat	Shows/hides the Format list so users can choose the file's format. The available formats are from the object's policy. 	false—Format field is not shown. true (default)—Format field is displayed but not required. required—Format field is required.
showName	Show/hides the Name field, which lets users enter a Name for the object the file will be checked into. 	false—Name field is not shown. true—Name field is displayed but not required. required (default)—Name field is required.
showPolicy	Shows/hides the Policy field, which lets users choose the policy that should govern the file.	false—Policy field is not shown. true—Policy field is displayed but not required. required (default)—Policy field is required.
showTitle	Show/hides the Title attribute field, which lets users enter a title for the object the file will be checked into. Applications often allow the user to enter a title and then use an autoname for the object's real name to ensure uniqueness. 	false—Title field is not shown. true (default)—Title field is displayed but not required. required—Title field is required.
store	Specifies the symbolic name of the store to put the file into. If the parameter is not specified, the system uses the company's store. If the company doesn't have a store defined, the system uses the default store for the policy.	Symbolic name of store to place the file in: `store=store_STORE`

Checkin URL Examples

Create Document URLs

`emxCommondocumentPreCheckin.jsp?objectAction=create` is used for two step create for any type derived from DOCUMENTS.

`emxCommondocumentPreCheckin.jsp?objectAction=createMasterPerFile` is used for a one-step quick create of a Document object.

Checkin Document URLs

`emxCommondocumentPreCheckin.jsp?objectAction=checkin` is used to add new files to the current object derived from a DOCUEMNTS object.

`emxCommondocumentPreCheckin.jsp?objectAction=update` is used to check in locked files in the current object derived from a DOCUMENTS object.

`emxCommondocumentPreCheckin.jsp?objectAction=updateHolder` is used to check in all locked files of the current Holder object.

Checkin Non-Document URL

`emxCommondocumentPreCheckin.jsp?objectAction=objectCheckin&override=false` is used to check in a new file(s) to the current object which is not derived from DOCUMENTS.

Checkout Document URLs

`emxCommonDocumentPreCheckout.jsp?objectId=oid&action=download` is used to checkout all files from the Master object. If there is more than one file it will be downloaded as a Zip file when EFCS is enabled; otherwise, it will display in the File Summary of that Document.

`emxCommonDocumentPreCheckout.jsp?objectId=oid&action=download&fileName=xyz.doc&format=generic` is used to check out a specific file from an object.

Checkin JPOs and Methods

The Checkin JPO used in 10.5 is `emxCommonDocument.java`.

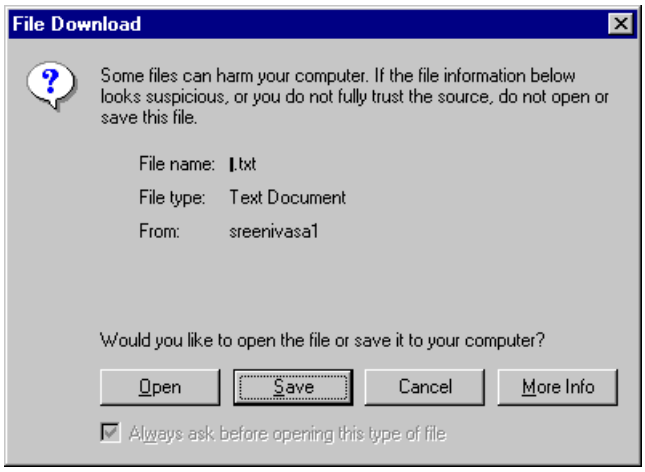
No MatrixOne application out-of-the-box can override any method other than the postCheckin and preCheckin methods.

The method that is called by default is `commonDocumentCheckin` from `emxCommonDocumentCheckinProcess.jsp`. Depending on the objectAction parameter, it then calls appropriate methods and returns objectMap with details.

Common Checkout

Common Components includes the programs (JSPs and JPOs) needed to check out a file that have been checked into an object. Based on an action parameter passed to the checkout JSP, checkout can simply open the file in the application registered for that type of file or checkout can allow the user to save the file to disk. If configured to let the user save the file, the checkout process can be configured to lock the object for edit.

If checkout is configured to let the user save the file, a blank browser window also opens behind the save as dialog box (the File Download dialog in IE and the Save As dialog in Netscape). After saving the file, the user must click Done to close the browser window.



Checkout JSP and URL Parameters

The checkout JSP is called emxComponentsCheckout.jsp and is located in ematrix/components. The following table lists the URL parameters accepted by the checkout JSP. If a parameter is not passed in with the URL, the default value is assumed.

URL Parameter	Description	Accepted Values and Examples
action	Determines whether the file is just opened for view or available for saving to disk and whether the object is locked for edit. This parameter is required.	view—file is opened in registered viewer or other application. In the MatrixOne application user interface, this action is usually implemented when the user clicks a Viewer icon or in some applications, the filename. download—file is downloaded and user specifies the path and filename. A small blank browser window opens behind the save as dialog and user must click Done to close it after saving the file. In the MatrixOne application user interface, this action is implemented when the user clicks a Download icon. checkout—same as download except the object the file is checked into is also locked for edit. In the MatrixOne application user interface, this action is implemented when the user clicks a Checkout or Lock for Edit icon.
appDir	The name of the directory the application uses.	appDir=programcentral appDir=sourcingcentral
appName	The name of the application that is calling the page.	appName=program appName=sourcing
fileName	The name of the file to check out. This parameter is optional. If neither the fileName nor format is sent, the system checks out the first file checked into the object. If only the format is sent, the system checks out the first file for that format.	fileName=designspec1.doc

URL Parameter	Description	Accepted Values and Examples
format	The format to check out the file from. This parameter is optional.	format=text
objectId	Specifies the name of the object to check out the file from. This parameter is required.	Valid object ID for the object. `objectId=33600.6795.43493.8032`

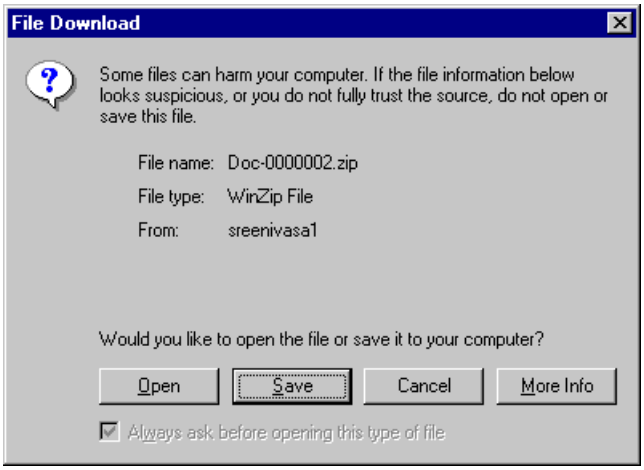
Checkout URL Examples

`emxCommonDocumentPreCheckout.jsp?objectId=oid&action=download` is used to checkout all files from the Master object. If there is more than one file it will be downloaded as a Zip file when EFCS is enabled; otherwise, it will display in the File Summary of that Document.

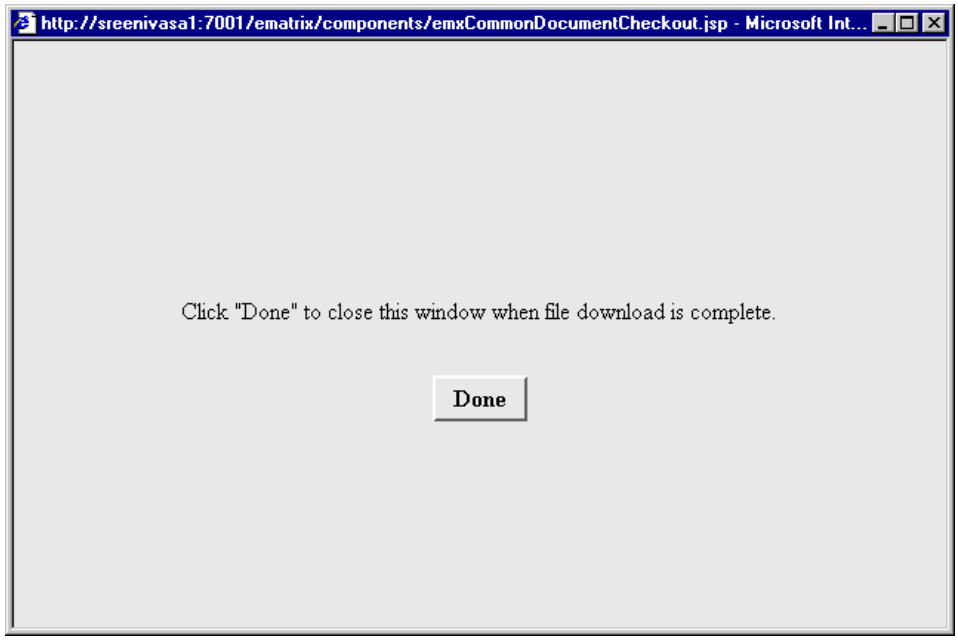
`emxCommonDocumentPreCheckout.jsp?objectId=oid&action=download&fileName=xyz.doc&format=generic` is used to check out a specific file from an object.

Multiple File Download

When you select to download files from a document, multiple files will be downloaded in a single zip file. The zip file name includes the name of the document, as shown below:



When the download completes, you will see the following dialog:



When the download is completed, click **Done**.

Purge

This is a feature for customers who do not want to keep previous versions of a DOCUMENT, which could take up a large amount of disk space. To enable this feature, you need to register a Checkin Trigger on the type used for any DOCUMENT derived type. Then create an eService Trigger Program Parameters object having a revision as Purge. This is especially useful for CAD models. For example, the eService Trigger Program Parameters object might have the name `TypeCADModelCheckinAction` with a revision as Purge.

Then activate the trigger.

Following are the eService Trigger Program Parameters that the object needs:

- eService Program Name as `emxCommonDocument`
- eService Method Name as `purgePreviousVersions`
- eService Program Argument 1 `as ${OBJECTID}`
- eService Program Argument 2 as `${FILENAME_ORIGINAL}`

If you do this on every version of a file, the system will delete the previous version of the file, which will replicate the functionality of replace on checkin.

Checkin with MQL

There is a way to check in files through MQL using the Common Document Model. The checked in files will be visible for application to use.

```
execute program emxCommonDocument -method checkinBus objectId c:\temp\
tempFile.txt generic STORE false server 'New File Added';
```

args[0] is DOCUMENTS or its derived object - objectId to checkin

args[1] is the File Path where the file is located, for example, c:\temp\.

args[2] is the File Name that needs to be checked in using the path, for example, xyz.doc.

args[3] is the File Format in which this file needs to be checked in, for example, generic.

args[4] is the store Name where this file should be checked in, for example, STORE.

args[5] is unlock. This should be “true” to unlock the version object or “false”.

args[6] is a server key word to checkin from the server.

This should be `server` when calling from the thick client.

This should be `client` when calling from PowerWeb.

args[7] is Comments about the file getting checked in.

Common Document Data Migration

In MatrixOne applications prior to version 10.5, document management was handled independently for each application. As of version 10.5, all MatrixOne applications (Team, Sourcing, Engineering, Program, Document, Supplier, Specification, and Product) follow a common document model. Thus, all applications upgrading to version 10.5 require data migration.

The following is the document data model for version 10.5:

- Master Document objects are the same.
- Highest version file is in Master Document and all other versions are in Document object with “Is Version Object” attribute set to true.
- These versions of files are managed as Core Revision sequence of Documents. The latest Document of these versions doesn’t contain a file in it, but it contains meta data like Reason For Change and File Name.
- All (version) Documents are not connected to Master. Only the Active (version) Document is connected to Master.

The following outlines the process for migrating all applications to a common document model. For all eight applications, there are 4 migration scripts because some of the applications can use the same migration code for converting their data.

The basic process for migrating the data is to run an initial query that finds all objects that need to be migrated and to store these object IDs in a set of files so they can be processed in groups. Each group is of a size that optimizes the database performance and memory requirements. The separation into these groups also allows for portions of the migration to be done at separate times in order to work around system availability restrictions as needed. The scripts are written in such a way that they can be run multiple times without causing harm to the data.

The following assumptions have been made by the migration scripts.

- Objects cannot meet more than one application’s criteria. If so, migration for those specific objects will be skipped, and Oids written to UnconvertedObjectIdsFile.
- For Team/Sourcing originated Documents, existing document revisions are migrated to become document versions.
- Documents with file formats such as “JT” are skipped. Separate migration scripts should be written to convert these objects.

The above assumptions should be verified as valid for a system before the scripts are run against that system.

You should perform document migration after installing all applications and integrations.

Converting the Data

Data conversion is accomplished by running two programs from MQL.

To convert the data

1. From MQL, type:
   ```
   set context user creator;
   ```

2. To optionally exclude types and subtypes from being migrated:

   In the `emxComponent.properties` file, set the `emxCommonDocumentMigration.Exclude.Types` property to the list of types and subtypes that need exclusion. The list should be comma separated and list the symbolic name of the type (For example, `type_CADModel,type_CADDrawing`). Note that if a type is specified, its subtypes will still be migrated, so you will need to list the specific subtypes you want excluded.

3. Type:

   ```
   execute program emxCommonDocumentFindObjects “PARAMETER1” “PARAMETER2”;
   ```

   PARAMETER1 is the number of Oids per file.

   PARAMETER2 is the directory name to put the files into.

   Note that parameters are separated with spaces, and quotation marks are optional for each parameter. However, quotation marks should be used for values having embedded spaces.

   This program does the initial query to fetch all the objects in the database to migrate and writes Oids into files using the passed-in parameters. The query finds all the objects in the database whose type is a sub-type of DOCUMENTS (by default, these are Document, Document Sheet, Generic Document, CAD Model, CAD Drawing, Drawing Print, Sketch, MarkUp). If any customized sub-types exist, those objects will also be picked up by the query.

4. Type:

   ```
   execute program emxCommonDocumentMigration “PARAMETER1” “PARAMETER2” “PARAMETER3”;
   ```

   PARAMETER1 is the directory name to read the files.

   PARAMETER2 is the minimum range.

   PARAMETER3 is the maximum range

Example

Step 1 : `execute program emxCommonDocumentFindObjects "1000" "c:/migrate105/";`

This will sweep the database for all the objects whose type is a sub-type of "DOCUMENTS" and will write their ids into files (1000 per file). For example, if the query finds 5600 objects in the database, it will create the following files:

C:/migrate105/documentobjectids_1 (first 1000 ids)

C:/migrate105/documentobjectids_2 (second 1000 ids)

C:/migrate105/documentobjectids_3 (third 1000 ids)

C:/migrate105/documentobjectids_4 (fourth 1000 ids)

C:/migrate105/documentobjectids_5 (fifth 1000 ids)

C:/migrate105/documentobjectids_6 (remaining 600 ids)

Step 2: `execute program emxCommonDocumentMigration “C:/migrate105/” “1” “n”;`

This JPO will look for files in “C:/migrate105” i.e. the first parameter starting with 1 to n.

For example, documentobjectids_1 until there are no more files in the folder incrementing the last digit by 1.

For example, documentobjectids_1, documentobjectids_2, documentobjectids_3, documentobjectids_4, documentobjectids_5, documentobjectids_6.

Or alternatively, to convert the objects in just the first file, use the following command:

```
execute program emxCommonDocumentMigration “C:/migrate105/” “1” “1”;
```

Or to convert from documentobjectids_1 to documentobjectids_3, use the following command:

```
execute program emxCommonDocumentMigration 'C:/migrate105/' 1 3;
```

If the migration fails for any reason, all the changes will be rolled back for all the objects in one file (migration proceeds further for the objects in the remaining files). The user will receive an error message with the object ID where it failed. By looking at the error message, the error condition can be fixed, and Step 2 can be performed again. If the same file is run after successful migration, the migration code knows that these are already converted, and skips those objects.

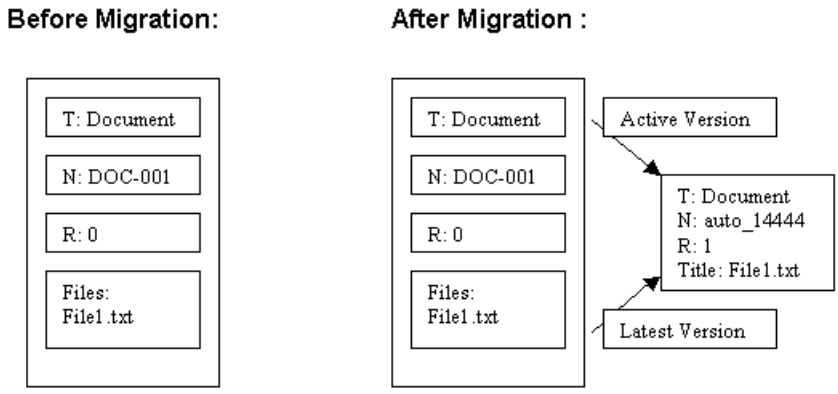
Step 2 of the migration takes an optional fourth argument as "true". When the fourth parameter is set as "true", the migration JPO converts all the "No Model" objects (if they do not have any revisions) to EC model. If they do have revisions, they will not be converted and written to the unConvertedObjectIds.csv file.

Example:

```
execute program emxCommonDocumentMigration “C:/migrate105/” “1” “2” “true”;
```

The above command will migrate all the document objects listed in the files documnetobjectids_1 and documnetobjectids_2 in the directory C:/migrate105/. If during the migration process any "No Model " objects (with out any revisions) are found, they will be automatically converted to ECModel.

The following diagram provides a visual explanation:



Custom Migration

The migration process migrates all objects of type DOCUMENTS. But in some cases, such as when an objects belongs in multiple models or does not belong in any known

model or fails to lock a version object, the migration will skip those specific objects and write the ids to the unConvertedObjectIds.csv file.

The following shows a typical unConvertedObjectIds.csv log after migration:

Master OID	Type	Classification	Version OID	Locker
40903.44534.25829.56752	Document	No Model		
40903.44534.65151.4052	Document	Multiple Model		
40903.44534.38269.41660	Document	Unable to Lock	40903.44534.34199.52107	userabc

The user can sort the above file easily and choose to migrate some of the objects (based on classification, etc.) by running custom migration code. A migration JPO provides a mechanism to execute a custom migration code, as shown below.

```
execute program emxCommonDocumentMigration -method customMigration "parameter1"
"parameter2";
```

where:

"parameter1" is "directory name"

"parameter2" is "file name" that has objectids to be migrated

For example:

```
execute program emxCommonDocumentMigration -method customMigration "c:/migrate/"
"file1.txt";
```

The emxCommonDocumentMigration JPO ships OOTB with empty code in the "migrateCustomObject" method. The user can fill this method based on customization/ implementation specific rules and run it on specific objects.

By passing a third argument to the above command as detailed below, a user can force a specific migration method to run.

```
execute program emxCommonDocumentMigration -method customMigration "c:/migrate/"
"file1.txt" "TeamSourcingModel";
```

This will force the JPO to run method migrateTeamSourcingModel on all objects listed file1.txt.

OR

```
execute program emxCommonDocumentMigration -method customMigration "c:/migrate/"
"file1.txt" "PMCModel";
```

This will force the JPO to run method migratePMCModel on all objects listed file1.txt

OR

```
execute program emxCommonDocumentMigration -method customMigration "c:/migrate/"
"file1.txt" "DocumentProductSpecModel";
```

This will force the JPO to run method migrateDocumentProductSpecModel on all objects listed file1.txt

OR

```
execute program emxCommonDocumentMigration -method customMigration "c:/migrate/"
"file1.txt" "EcModel";
```

This will force the JPO to run method migrateEcModel on all objects listed file1.txt

OR

```
execute program emxCommonDocumentMigration -method customMigration "c:/migrate/"
"file1.txt" "default";
```

This will force the JPO to run method migrateCustomObject on all objects listed file1.txt. If the third argument "default" is not passed, the "migrateCustomObject" method will run by default.

Application Qualification Matrix

The following Qualification Matrix indicates the criteria used to determine which types of objects belong to which applications. A description of the criteria follows.

Application	Criteria													
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Doc					X									
Engineering									X	X	X			
Team	X	X	X											
Sourcing				X										
Specification								X						
Program												X	X	X
Product						X	X							

Criteria A

Object Type: Document or its Sub-Type

From object type: “Workspace Vault”

Relationship Type: “Vaulted Objects”

Number of files == 1

File Format != “JT”

Application: Team

Model: Model 1

Criteria B

Object Type: Document or its Sub-Type

From object type: “Meeting”

Relationship Type: “Meeting Attachments”

Number of files == 1

File Format != “JT”

Application: Team

Model: Model 1

Criteria C

Object Type: Document or its Sub-Type

From object type: “Message”

Relationship Type: “Message Attachments”

Number of files == 1

File Format != “JT”

Application: Team

Model: Model 1

Criteria D

Object Type: Document or its Sub-Type

From object type: Package, RFQ, RTS-Template, RTS-Quotation, Line Item, Supplier Line Item, Part, ECR

NOT from object type: Products, Builds, Features, Incident, Requirement, Test Case, Issue, Use Case, Technical Specification, Financials, Assessment, Risk, Business Goal, Quality

Relationship Type: Reference Document

Number of files == 1

File Format != “JT”

Application: Sourcing

Model: Model 1

Example: Suppose a document is connected to a package and also to a product with the Reference Document relationship, then it will NOT fall into this criteria.

A document object will fall into this category if it is connected from any object whose type falls into “From object type”, and at the same time, it should NOT be connected from “NOT from object type” list.

Criteria E

Object Type: Generic Document or its Sub-Type, Document Sheet or its Sub-Type

Application: Document

Model: Model 3

Criteria F

Object Type: Specification or its Sub-Type

Application: Product

Model: Model 3

Criteria G

From object type: Products, Builds, Features, Incident, Requirement, Test Case, Issue, Use Case

NOT from object type: Package, RFQ, RTS-Template, RTS-Quotation, Line Item, Supplier Line Item, Part, ECR, Financials, Assessment, Risk, Business Goal, Quality

Relationship Type: Reference Document

Application: Product

Model: Model 3

Criteria H

From object type: Technical Specification

NOT from object type: Package, RFQ, RTS-Template, RTS-Quotation, Line Item, Supplier Line Item, Part, ECR, Financials, Assessment, Risk, Business Goal, Quality

Relationship Type: Reference Document

Application: Spec

Model: Model 3

Criteria I

Object Type: CAD Model, CAD Drawing, Drawing Print, Sketch, MarkUp or their Sub-Types

Application: Engineering

Model: Model 4

Criteria J

Object Type: Document or its Sub-Type

From object type: Part

NOT from object type: ProductsPackage, RFQ, RTS-Template, RTS-Quotation, Line Item, Supplier Line Item, ECR, Products, Builds, Features, Incident, Requirement, Test Case, Issue, Use Case, Technical Specification, Financials, Assessment, Risk, Business Goal, Quality

Relationship Type: Reference Document

Number of files > 1

Application: Engineering

Model: Model 4

Criteria K

Object Type: Document or its Sub-Type

From object type: Part Family

Relationship Type: Part Family Reference Document

Application: Engineering

Model: Model 4

Criteria L

Object Type: Document or its Sub-Type

From object type: Financials, Assessment, Risk, Business Goal, Quality

NOT from object type: Package, RFQ, RTS-Template, RTS-Quotation, Line Item, Supplier Line Item, Part, ECR, Products, Builds, Features, Incident, Requirement, Test Case, Issue, Use Case, Technical Specification

Relationship Type: Reference Document

Number of files == 1

Application: Program

Model: Model 2

Criteria M

Object Type: Document or its Sub-Type

From object type: Workspace Vault

Relationship Type: Vaulted Documents Rev2

Number of files == 1

Application: Program

Model: Model 2

Criteria N

Object Type: Document or its Sub-Type

From object type: Task

Relationship Type: Task Deliverable

Number of files == 1

Application: Program

Model: Model 2

Application to Conversion Model Mapping

The following table maps the conversion model used for each application:

Application	Model 1	Model 2	Model 3	Model 4
Document			X	
Engineering				X
Team	X			
Sourcing	X			
Specification			X	
PMC		X		
Product			X	

Migration Logic

Migration for Team/Sourcing Documents (Model 1)

Migrate all existing Document (that is, master document) revisions to Document versions.

If the “Title” of the master object is null, set it to the master object name.

If there are Document revisions (that is, revisions > 1):

1. Clone (without files) the highest Document revision (the master object). The newly created object corresponds to the highest file version in the new Common Document data model.
2. Disconnect the highest Document revision from the core revision chain.
3. Insert the cloned object created in Step 1 into the core revision chain.
4. Connect the highest Document revision to the cloned object with Latest Version and Active Version relationships.
5. Loop through core revision chain, and set the “Is Version Object” attribute to True and modify the policy to Version of each object in the revision chain.
6. Since the highest file version object is created during the migration process, the owner and originated date of this object are incorrect. To rectify this, the owner and originated date are set equal to that of highest Document revision.
7. In the new Common Document data model, the corresponding highest version object for the file is locked. So if the highest Document revision is locked, then the corresponding highest version object is locked, and the highest Document revision is unlocked.

If there are NO Document revisions (that is, revisions == 1)

1. Clone the Document without a file. (This object corresponds to the highest file version in the new Common Document data model.)
2. Connect the Document to the cloned object with the Latest Version and Active Version relationships.
3. Set “Is Version Object” attribute to True and modify the policy to Version for the cloned object.

4. Since the highest file version object is created during the migration process, the owner and originated date of this object are incorrect. To rectify this, the owner and originated date are set equal to that of the highest Document revision.

5. In the new Common Document data model, the corresponding highest version object for the file is locked when the user locks a file. So if the highest Document revision is locked, then the corresponding highest version object is locked, and the highest Document revision is unlocked.

Migration for Engineering Central Documents (Model 2)

For each file in the object (master object):

1. If the "Title" of the master object is null, set it to the master object name.
2. Get the list of files in the master object.
3. For each file found in Step 2:
   - **a)** Create a new Version object with type as that of the master object type with Version policy.
   - **b)** Set the Is Version Object attribute to True. Modify the policy to Version for the cloned object.
   - **c)** Set the Originator attribute to that of the master object owner.
   - **d)** Since the version object is created during the migration process, the owner of this object (User Agent) is incorrect. Set the Owner to that of the master object owner.
   - **e)** Since the version object is created during the migration process, the originated date of this object (migration run date) is incorrect. Set the originated date to that of the master object originated date.
   - **f)** Connect this version object to the master object with Latest Version and Active Version relationships.
   - **g)** In the new Common Document data model, the corresponding highest version object for the file is locked when the user locks a file. If the master object is locked, then the corresponding highest version object is locked.
4. If the master object is locked, unlock the master object.

Migration for Program Central Documents (Model 3)

In this model, a Document (that is, master object) can have only one file, and each file can have multiple versions. Also, a file can be renamed on versioning. For each file version, there is a corresponding "Version Object" connected to the master with the Version relationship (except for the latest version). That is, the latest version of the file is in the master object. There is no corresponding "Version Object." Previous versions of files are in the corresponding "Version Object."

If the "Title" of the master object is null, set it to the master object name.

For each file in the object:

1. Expand the document object with the Version relationship for all connected "Version document" objects.
2. Create a stack (in ascending order of "File Version" attribute) of "Version Document" objects found in Step 1.
3. Create a core revision for the objects in the stack.

4. Create a new Version Object to correspond to the highest version of the file, and add this object to the top of the revision chain created in Step 3.

   Since the highest file version object is created during the migration process, the owner and originated date of this object are incorrect. To rectify this, the owner and originated date are set equal to that of the Document object.

5. Connect the highest file version object created in Step 4 with Latest Version and Active Version relationships.

6. For each object in the version object revision chain created in Step 3:

   a) Set the “Is Version Object” attribute to True.

   b) Modify the type to the master object type.

   c) Modify the policy to Version.

   d) Disconnect the Version relationship.

7. Since the highest file version object is created during the migration process (Step 4), the owner and originated date of this object are incorrect. To rectify this, the owner and originated date are set equal to that of the Document object.

8. In the new Common Document data model, the corresponding highest version object for the file is locked when the user locks a file. So if the Document is locked, then the corresponding highest version object for the file is locked, and the Document is unlocked.

9. Update the “Document Store” attribute on the DOCUMENTS subtype with the STORE the original file is checked into.

Migration for Document Central/Specification Central /Product Central (Model 4)

In this model, a Document (that is, master object) can have multiple files, and each file can have multiple versions. For each file version, there is a corresponding “Version Object” connected to the master with the Version relationship. The latest version of each file is in the master object. The corresponding “Version Object” does not contain the file. It is just a dummy place holder. Previous versions of files are in the corresponding “Version Object.”

1. If the “Title” of the master object is null, set it to the master object name.

2. Expand the document object with the Version relationship for all connected “Version Document” objects.

For each file in the object:

3. Filter out the Version Document objects pertaining to this file by comparing the “Title” attribute of the objects found in Step2.

4. Create a stack (in ascending order of “File Version” attribute) of Version Document objects found in Step 3.

5. Create a core revision for the objects in the stack.

6. Connect the highest file version object created in Step 5 (the object at the top of the revision chain) with the Latest Version and Active Version relationships.

7. For each object in the version object revision chain created in Step 5:

   a) Set the “Is Version Object” attribute to True.

   b) Modify the type to the master object type.

c) Modify the policy to Version.

d) Disconnect the Version relationship.

8. In the new Common Document data model, the corresponding highest version object for the file is locked when the user locks a file. So if the Document is locked, then the corresponding highest version object for the file is locked, and the Document is unlocked.

Configurable Properties

The file called emxComponents.properties contains properties that let you configure features installed with common components. This file is located in ematrix/properties. ***As with all properties files, after making changes, save the file and restart the Web server.***

Properties You Should Not Change

The emxComponents.properties file contains several properties that should not be changed unless you are directed to do so by MatrixOne personnel. These properties are:

- The properties that control the date format for meetings.

```
# Default Date Format
emxComponents.CreateMeetingDateFormat= M/d/yy hh:mm:ss a zzz
emxComponents.WebexDateFormat= M/d/yy hh:mm:ss a
```

Miscellaneous Properties

Property Use	Property Comment, Key, and Value

Specify the option that should be selected by default in the Default Vault (Search) preference. This preference is installed with common components and is used by some applications, such as Engineering Central, to let users choose the vaults they want to search in by default. When executing a specific search, users can choose a different vault.	`# This is the property for default search vaults`
	
	`# It can have values 'ALL_VAULTS' (All Vaults of the Comapny),`
	`# 'LOCAL_VAULTS' (All vaults of the company Local to the connected server),`
	`# 'DEFAULT_VAULT' (logged in person default vault)`
	`#`
	`emxComponents.DefaultSearchPreference = ALL_VAULTS`

Define alternate category list (tree) menu objects for common types. These properties define category list menu objects to be used for types that are used in more than one MatrixOne application.	Here are a few examples of Common Component's alternate category list properties. See the properties file for a complete list.
	`eServiceSuiteComponents.emxTreeAlternateMenuName.type_Company=type_Company`
	`eServiceSuiteComponents.emxTreeAlternateMenuName.type_BusinessUnit=type_BusinessUnit`
	`eServiceSuiteComponents.emxTreeAlternateMenuName.type_Location=type_Location`
	`eServiceSuiteComponents.emxTreeAlternateMenuName.type_Person=type_Person`
	`eServiceSuiteComponents.emxTreeAlternateMenuName.type_Department=type_Department`
	`eServiceSuiteComponents.emxTreeAlternateMenuName.type_Message=type_Message`

Checkin Properties

Property Use	Property Comment, Key, and Value
Define the policies users can choose from when checking in a file. The ability to specify a policy for a file is configurable.	`emxComponents.PoliciesList = policy_Document,policy_DataDocument,policy_DocumentSheet`
Define the number of files users can specify when checking in files.	`emxComponents.MultiFileUpload.NoOfFiles = 5`
Defines the delimiter used in the document name	`emxComponents.DocumentNameDelimiter`
Defines the list of excluded attributes	`emxComponents.CreateDocument.ExcludeAttributeList`

Route Properties

Property Use	Property Comment, Key, and Value
Determines whether routes are listed on the Routes summary pages for route members. If true, routes are listed for the route owner and all members. If false, routes are listed only for the route owner.	`emxFramework.Routes.RouteVisibility = true`

Property Use	Property Comment, Key, and Value
Determines the default action for route tasks. Available choices are Approve, Comment, Notify Only, Investigate, and Information Only.	`# Flag to set the RouteDefaultAction parameter of Route Page.` `emxComponents.RouteDefaultAction=Approve`
Controls whether the Comments field is shown for tasks with Approve actions. Allowing assignees to enter a comment and choose an approval option opens the possibility of a user entering conflicting information. For example, the assignee could choose Approve but enter a negative comment.	`emxComponents.Routes.ShowCommentsForTaskApproval=true`
Controls whether Abstain is displayed as an option for route tasks with Approve actions.	`emxComponents.Routes.ShowAbstainForTaskApproval=true`

Requires signature verification when indicating an approval option (approve, reject, or abstain) for route tasks when using Single Signon (SSO).
These properties are only used when external authentication or SSO is being used and verification is required for route task approvals. To use the properties, emxFramework.Routes.EnableFDA, in emxSystem.properties, must be set to true.
When the InboxTask.ExternalAuthentication property is set to TRUE, users are redirected to the URL defined in the InboxTask.ExternalAuthenticationURL property for authentication. These properties should be set only in SSO environments. For more information, see “Configuring Approval Verification with External Authentication” in the AEF guide.
	`emxComponentsRoutes.InboxTask.ExternalAuthentication=FALSE`
	For SiteMinder:
	`emxComponentsRoutes.InboxTask.ExternalAuthenticationURL=http://siteMinderServer:siteMinderPort/siteminderagent/forms/emxTaskSSOLogout.fcc`
	For ClearTrust:
	`emxComponentsRoutes.InboxTask.ExternalAuthenticationURL=/cleartrust/ct_logon.jsp?ct_orig_uri=/ematrix/components/emxTaskSSOAuthentication.jsp?callbackFunctionName=passwordCallback()`

Person Profile Properties

Property Use	Property Comment, Key, and Value
TRUE - Will use person’s company vault. FALSE - Allows the user to select a vault.	`eServiceSuiteComponents.VaultAwareness=false`
Define the roles that should be available when a person is added to the applications (when common profile is used). This is a list of symbolic names of roles separated by semicolons.	`emxComponents.DefaultRoles=role_OrganizationManager;role_Employee;role_CompanyRepresentative;role_ExchangeUser;`

Page Definition Properties

Common Component pages built using the common frameset, exmCommonFS.jsp, can be configured using properties in emxComponents.properties. For each of these pages, the properties file contains a property that defines the page’s heading, buttons, action bars, content URL, page toolbar and pagination options, and help marker.

Property Use	Property Comment, Key, and Value

Define the heading text for a page. The value for the property can be the actual text string or a string resource ID defined in the emxComponentsStringResource.properties files. For internationalization, the value must be a string resource ID.	`eServiceSuiteComponents.PAGE.heading=STRING_RESOURCE_ID`
	For example, this property defines the heading text for the Discussions Subscription page:
	`eServiceSuiteComponents.DiscussionsSubscribe.heading = emxComponents.DiscussionsSubscribe.heading`

Define the buttons that are to be shown in the lower right action bar. Only the standard buttons of Done, Cancel, Previous, and Next are supported.	`eServiceSuiteComponents.PAGE.buttons =Done,Cancel,Previous,Next`
	For example, this property defines the buttons for the Discussions Subscription page:
	`eServiceSuiteComponents.DiscussionsSubscribe.buttons =Done,Cancel`

Define the action links that are to be shown at the top and bottom.	`eServiceSuiteComponents.PAGE.topLinks=`
	`eServiceSuiteComponents.PAGE.bottomLinks=`
	For example, these properties define the links for the top and bottom of the Create Distribution List Step 2 page:
	`eServiceSuiteComponents.CreateDistributionListStep2.topLinks=AddMembers`
	`eServiceSuiteComponents.CreateDistributionListStep2.bottomLinks=RemoveSelected`

Define whether to display the Printer Friendly tool, Pagination controls, and Show Conversion tool respectively. A true will show the respective icon/control.The Printer Friendly and Show Conversion tools are in the page toolbar. The pagination controls are in the lower right side of table pages.	`eServiceSuiteComponents.PAGE.options=false/true|false/true|false/true`
	For example, this property defines the page tools for the Discussions Subscription page. By default, only the Help tool displays.
	`eServiceSuiteComponents.DiscussionsSubscribe.options=false|false|false`

Define the JSP to call for the page. This JSP should be located in ematrix/common.	`eServiceSuiteComponents.PAGE.contentURL=JSP_NAME`
	For example, this property defines the content URL for the Discussions Subscription page:
	`eServiceSuiteComponents.DiscussionsSubscribe.contentURL=emxComponentsCreateSubscriptionsDialog.jsp`

Define the help marker for the page. This marker is passed to the help system so the help opens to the topic that explains the page.	`eServiceSuiteComponents.PAGE.help=MARKER`
	For example, this property defines the help marker for the Discussions Subscription page:
	`eServiceSuiteComponents.DiscussionsSubscribe.help=emxhelpdiscussionsubscription`

Document Properties

The property `emxComponents.AllowChangePolicy` defaults to True to allow policy change for Documents.

Old UI Properties

These are properties used by pre-95x (UI2) applications. They are no longer used by 95x and higher applications.

```
# Command Menu Properties
# Icon: The icon an application menu should be associated with
# MouseOverText: The Text which will be displayed when the mouse is moved over the icon
# TargetJSP: The JSP page to be loaded when this command button is clicked
# TargetFrame: The frame to be targeted when the JSP page is loaded

eServiceCommandComponentsInboxTask.Icon = inboxtask.gif
eServiceCommandComponentsInboxTask.MouseOverText = emxComponents.ApplyMarkup.InboxTask
eServiceCommandComponentsInboxTask.TargetJSP = emxengchgInboxFrameset.jsp
eServiceCommandComponentsInboxTask.TargetFrame = folderFrame
eServiceCommandComponentsInboxTask.HelpMarker = emxhelpinboxtask

eServiceCommandComponentsIconMail.Icon = iconmail.gif
eServiceCommandComponentsIconMail.MouseOverText = emxComponents.Common.IconMail
eServiceCommandComponentsIconMail.TargetJSP = emxCompInboxDialog.jsp
eServiceCommandComponentsIconMail.TargetFrame = PopUp
eServiceCommandComponentsIconMail.Javascript.WindowHeight = 700
eServiceCommandComponentsIconMail.Javascript.WindowWidth = 900
eServiceCommandComponentsIconMail.Javascript.Scrollbar = yes
eServiceCommandComponentsIconMail.Javascript.Resize = no

eServiceCommandComponentsChangePassword.Icon = changepasswd.gif
eServiceCommandComponentsChangePassword.MouseOverText = emxComponents.DesignTOP.ChangePassword
eServiceCommandComponentsChangePassword.TargetJSP = emxengchgChangePasswordDialog.jsp
eServiceCommandComponentsChangePassword.TargetFrame = folderFrame

eServiceCommandComponentsHome.Icon = home.gif
eServiceCommandComponentsHome.MouseOverText = emxComponents.DesignTOP.Home
eServiceCommandComponentsHome.TargetJSP = ../emxHome.jsp
eServiceCommandComponentsHome.TargetFrame = _parent

eServiceCommandComponentsInformation.Icon = about.gif
eServiceCommandComponentsInformation.MouseOverText = emxComponents.DesignTOP.Information
eServiceCommandComponentsInformation.TargetJSP = emxDesignAbout.jsp
eServiceCommandComponentsInformation.TargetFrame = folderFrame
```

```
eServiceCommandComponentsHelp.Icon = help.gif
eServiceCommandComponentsHelp.MouseOverText =
emxComponents.DesignTOP.Help
eServiceCommandComponentsHelp.TargetJSP = Javascript:openHelp()
eServiceCommandComponentsHelp.TargetFrame =

eServiceCommandComponentsLogout.Icon = close.gif
eServiceCommandComponentsLogout.MouseOverText =
emxComponents.DesignTOP.Logout
eServiceCommandComponentsLogout.TargetJSP = emxDesignLogout.jsp
eServiceCommandComponentsLogout.TargetFrame = _parent
```

Object Lifecycle and Process Automation

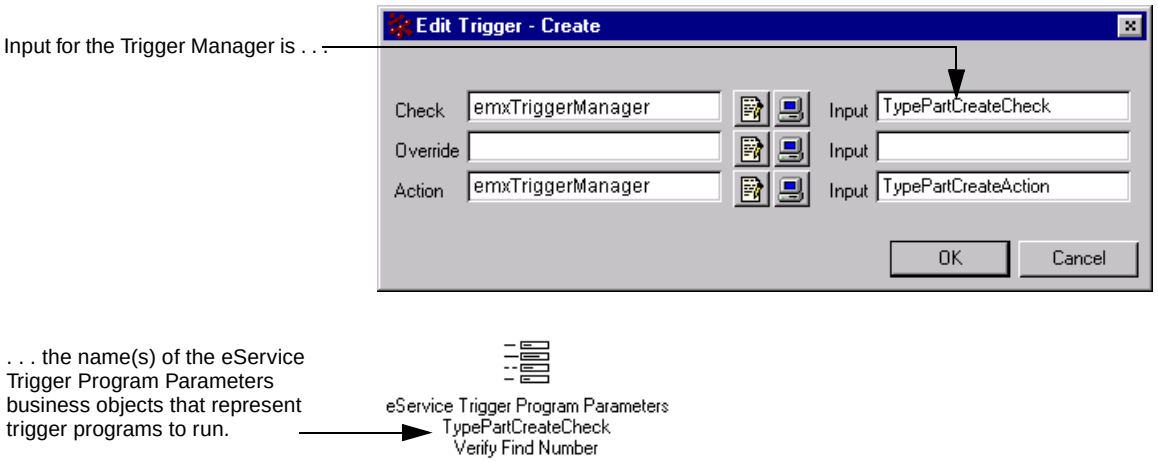
This section lists the triggers that have been added to policies and other administrative objects.

Trigger programs are run using a Trigger Manager program called emxTriggerManager. This Java program is specified as the Action for all triggers. The name of the eService Trigger Program Parameters business object that represents the specific trigger program to run is specified as the Input to pass to the Trigger Manager. The naming convention for the eService Trigger Program Parameters object indicates the schema object the trigger is associated with, trigger event, and the type of trigger.

[ADMIN OBJECT TYPE][ADMIN OBJECT NAME][TRIGGER EVENT][TYPE OF TRIGGER]

The revision describes the purpose of the trigger.

For example, an eService Trigger Program Parameters object named "TypePartCreateCheck" indicates that the trigger is a Create Check trigger on the Part type. The revision indicates that the purpose of the trigger is to verify the find number for the part.



Trigger programs can be TCL programs or Java programs. Attributes on the eService Trigger Program Parameters object contain the name of the trigger program to run and parameters to pass to it. For Java programs, the object's attributes also contain the specific method to call and constructor arguments. Commented sections within each trigger program describe the parameters accepted by the program.

For a trigger program to run, its eService Trigger Program Parameters object must be in the Active state. To turn off a trigger, demote its eService Trigger Program Parameters object to Inactive.

For more information about the Trigger Manager and eService Trigger Program Parameters objects, see the ***Application Exchange Framework Guide***.

Triggers

This table lists triggers that are installed with common components. For lists of triggers installed with the framework, see Appendix A of the ***Application Exchange Framework Guide***. For version 10 and higher framework and applications, triggers are installed with

the applications and common components, not with the framework. For lists of triggers installed with an application, see that application's Administrator's Guide.

Type of Object Being Automated	eService Trigger Program Parameters Object, Name and Revision	Trigger Program and JPO Method	Description of Action
Route and Task	N: RelationshipRouteNodeCreateAction R: populateRouteNodeId	emxRoute populateRouteNodeId method	Populates Route Node Id attribute on relationship Route Node.
	N: PolicyInboxTaskStateReviewPromoteAction R: Review Completion	emxInboxTask triggerActionPromoteOnReviewState method	Sends a notification to the task assignee informing her/him of the review completion.
	N: PolicyInboxTaskStateReviewPromoteAction R: Complete Task	eServicecommonTrigcCompleteTask_if.tcl	When a person completes a task, the Inbox Task object is promoted from Review to Complete and this trigger is fired. The trigger: – Enters the actual completion date. – If the approval status is Reject, stops the route and notifies the route owner. – If there are no other tasks to be completed for the route, notifies the route owner that route is complete and changes the Route status to Finished.
	N: PolicyInboxTaskStateReviewPromoteCheck R: Check Route Owner	emxInboxTask triggerCheckPromoteOnReviewState method	Checks if the user is the route owner, else sends an error message to the user and prevents the promote.
	N: PolicyInboxTaskStateReviewDemoteAction R: Reject Task	emxInboxTask triggerActionDemoteOnReviewState method	Sends a notification to the task assignee informing him of the rejection of the task.
Organization, Company, Business Unit, Location	N: TypeCompanyCreateAction R: Set Company Key	emxKeyUtil generateAndSetKey method	Create unique company key for company and set as Primary Key.
	N: RelationshipOrganizationLocationCreateAction R: Set Company Key	emxKeyUtil setToKey method	Set company key as Primary Key.
	N: RelationshipDivisionCreateAction R: Set Company Key	emxKeyUtil setToKey method	Set company key as Primary Key.

Type of Object Being Automated	eService Trigger Program Parameters Object, Name and Revision	Trigger Program and JPO Method	Description of Action
Person	N: TypePersonChangeNameAction R: syncPersonAdminObject	emxPerson syncPersonAdminObject method	When the name of a Person business object changes, the name of the corresponding person admin object is updated to match. If the Full Name format, as defined in emxFramework.FullName.Format, contains <User Name>, update the Full Name field. *Input Arguments: Arg[0] = ${EVENT} Arg[1] = ${NAME} Arg[2] = ${NEWNAME}
	N: AttributeFirstNameModifyOverride R: syncPersonAdminObject	emxPerson syncPersonAdminObject method	If a person's First Name is changed and the Full Name format, as defined in emxFramework.FullName.Format, contains <First Name>, update the Full Name field on the person admin object. *Input Arguments: Arg[0] = ${EVENT} Arg[1] = ${NAME} Arg[2] = " Arg[3] = ${ATTRNAME} Arg[4] = ${ATTRVALUE} Arg[5] = ${NEWATTRVALUE}
	N: AttributeLastNameModifyOverride R: syncPersonAdminObject	emxPerson syncPersonAdminObject method	If a person's Last Name is changed and the Full Name format, as defined in emxFramework.FullName.Format, contains <Last Name>, update the Full Name field on the person admin object. Input Arguments are same as above trigger.
	N: RelationshipEmployeeCreateAction R: addMemberRelationship	emxCompany addMemberRelationship method	When a person is connected as an employee to a company, this trigger makes sure a member relationship is also created and its roles parameter are filled in with all of the person's roles.
	N: AttributeEmailAddressModifyOverride R: Notify Email Change	eServicecommonTrigaNotifyEmailChange_if.tcl	Sends email to person whenever a change is made to their email address. The email is sent to both the old and new email address and gives the old and new address, the person who made the change, and the date and time of the change.
	N: RelationshipEmployeeCreateAction R: Set Company Key	emxKeyUtil setCompanyKey method	Set company key as Primary Key.

* The input arguments for the person name change triggers are as follows:

- Event is changeName or modifyAttribute
- Name is the person business object name
- AttrName is Last Name or First Name
- AttrValue is value on the Last Name or First Name.

Activating Route State-Based Blocking

State-based blocking can be set up for non-document content items in a route. If you specify a block promotion state for an item, the system prevents the item from being promoted from that state until the route is completed. For example, suppose a buyer is routing an RFQ to be reviewed internally before sending it to suppliers. The buyer can make sure the RFQ is not sent to suppliers until the route (and associated review) is complete by specifying Initial Review as the block promotion state. Similarly, suppliers can make sure a quotation is not returned to the buyer until its route (which lets others in their company review it) is complete by choosing Review as the block promotion state.

State-based blocking must be enabled in Matrix Business Modeler before it can be used in a route.

To enable state-based blocking

1. For each type that you want use state-based blocking, open its corresponding policy in Matrix Business Modeler. For example, if you want to enable state-based blocking for Engineering Central parts, open the EC Part policy.
2. On the States tab, double-click and open the state on which you want state-based blocking enabled.
3. On the Triggers tab, locate the promote trigger and double-click it. (If it does not exist, add the promote trigger.)
   a) In the **Check** field, type emxTriggerManager.
   b) In the **Input** field, give a name that is unique. (If already present, do not modify it.) For example, on the Preliminary state in the policy EC Part policy, use `PolicyECPartStatePreliminaryPromoteCheck`.
4. In Matrix Business Modeler, search for the object specified in the promote trigger check input field. For example:

   Type: eService Trigger Program Parameters

   Name: PolicyECPartStatePreliminaryPromoteCheck

   Revision: *

   If found, you will notice in its attributes that the eService Sequence Number is entered. If multiple entries are found, note the last sequence number. This will be used in creating a new object.
5. Click **Object > New > Original** to create a new object, using the name entered in Step 3. Use a unique revision. For example:

   Type: eService Trigger Program Parameters

   Name: PolicyECPartStatePreliminaryPromoteCheck

   Revision: CheckRoutesForPreliminaryState

   Vault: eService Administration

   Policy: eService Trigger Program Policy
6. Enter the following for its attributes:

   eService Program Name: eServiceCheckRoutes_if.tcl

   eService Sequence Number: last sequence number (from Step 4)+ 1

7. Promote the object to the Active state.
8. Restart the RMI and web/application server.

Selecting Objects for Routing

Any type of object can be included for routing. To set the types of objects that can be included in a route, use Business Modeler to open the "Object Route" relationship and add the type names, separated by commas, on the "From Types" tab. In previous versions, some applications used a properties setting, such as `emxComponentsRoutes.RoutableTypes.APP_NAME` to define which types of objects could be included in a route. The property setting is no longer used.

Issue Management

MatrixOne Common Components include issue reporting, which allows users to report and resolve issues against specific object types.

This section contains procedures for the following:

- Adding Object Types for Issue Relationships
- Adding Issue Subtypes
- Adding Issue Categories and Classifications
- Assigning Users as Issue Managers

Adding Object Types for Issue Relationships

You can add object types for issue relationships in order to report and resolve issues against these objects.

The Issue relationship appears as "Reported Against" on Issue pages; the "Resolved To" relationship appears as "Resolved By" on Issue pages.

To add types to the Issue or Resolved To relationship

1. Log in to the Business Administrator application.

   You must have Business Administrator access.
2. Search for the relationship "Issue" or "Resolved To".
3. Open either the "Issue" or the "Resolved To" relationship in edit mode.
4. Click the **To Type** tab in the Edit Relationship dialog.
5. Add any types you want to appear on the "to" end of the relationship.

   When creating an Issue or filling in the Resolved By in the application, the types added will be available for selection.

If you want to be able to access Issues in the context of the newly added types (for the Issue relationship), the tree for the corresponding type must be modified to add the common command `ContextIssueListPage`.

For example:

1. Add the type "Part" to the to end of the "Issue" relationship.
2. Add the command `ContextIssueListPage` to the type_Part menu.

Adding Issue Subtypes

Issue Management supports the ability to create new issue subtypes. Issue subtypes allow you to define different business processes for each issue type. For example, you may need to implement two different business processes for Issues reported by internal users vs. external users, such as for customers and field services (help desk type of issues).

To add a new issue subtype, you must perform these procedures in the following order:

1. Add the new issue subtype using Business Modeler.
2. Register the new type in Matrix Navigator using the eService Administration Property Wizard.

3. Add the number Generation business objects for each subtype using Matrix Navigator.

The rest of this section contains these procedures, in the correct order.

To add a new Issue subtype

1. Log in to Business Modeler.

   You must have Business Administrator access.

2. Search for the type **Issue**.

3. Select the type **Issue** and select from the menu **Object** > **Clone**.

   The Clone dialog will ask you for the new Issue Subtype name.

4. Enter the new issue subtype name.

   The Edit Type dialog is displayed.

5. Change the Description and Derived From fields:

   For Description, type a new description or the purpose of the new type.

   For Derived From, type **Issue**.

For Derived From, you must enter the type Issue.

6. Click the **Triggers** tab and add or remove any of the inherited triggers.

   If you don't want the automatic Issue Manager assignment to take place, remove the inherited trigger “ChangeOwner”.

7. Click the **Edit** button when you have completed the definition of your new Issue subtype.

To register the new issue subtype

To register the new type, refer to “Registering Your Own Administrative Objects” in the ***AEF Administrator Guide***. This will explain how to use the eService Change Administration Property Wizard.

Add a number generator for the new issue subtype

1. Log in to Matrix Navigator.

   You must have Business Administrator access.

2. From the main menu, click **Object** > **New** > **Original**.

3. For the **eService Object Generator**, on the Original Dialog, enter the following:

   Type: eService Object Generator

   Name: <the symbolic name for the new type, e.g., type_CustomerIssue>

   Revision: (leave blank)

   Vault: eService Administration

   Policy: eService Object Generator

4. Enter the following for the **eService Object Generator** attribute:

   eService Next Number = 0000001

5. On the Original Dialog, enter the following:

   Type: eService Number Generator

Name: <the symbolic name for the new type, e.g., type_CustomerIssue>

Revision: (leave blank)

Vault: eService Administration

Policy: eService Object Generator

6. Enter the following **eService Number Generator** attribute:

   eService Name Prefix = ISS-

   eService Name Suffix =

   eService Retry Delay = 60

   eService Retry Count = 5

   eService Safety Vault = vault_eServiceAdministration

   eService Processing Time Limit = 60

The eService Name Prefix and eService Name Suffix allow the business administrator to set a different prefix and suffix for each issue subtype.

7. Once you have created both the eService Number Generator business object and the eService Object Generator you are ready to connect the two business objects together:

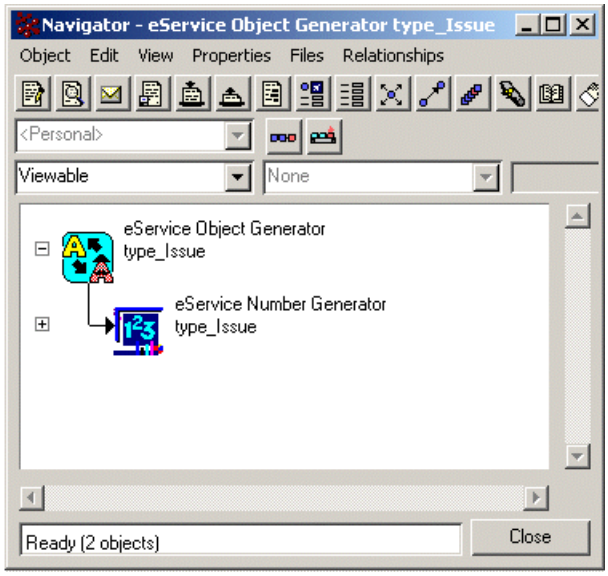
   a) Select the business object **eService Object Generator**.

   b) From the main menu, select **Relationship** > **Connect**.

   c) From the Connect dialog on the left frame, select the relationship **eService Number Generator**, and on the right frame select the business object **eService Number Generator**.

   d) Click the **Connect** button.

Below is an example of what the auto-number generators should look like:



Adding Issue Categories and Classifications

Issue Category and Classification provide the organization the ability to organize/group issues by functional or technical categories. Furthermore, Issue Management provides the ability to automatically assign all new issues to a predefined issue manager. An Issue Manager is a role that is assigned to Product Managers, Project Leads, etc.

Issue Categories and Classifications work in the following ways:

- Issue Management provides a sample set of Categories and Classifications that can be changed based on your business needs.
- The Issue Category and Classification is a required field on the Issue Create dialog. All issues must be classified.
- The list of Categories and Classifications displayed on the Issue Create dialog and the Issue Edit Details are business objects managed by the Matrix Navigator application.
- Categories must have at least one Classification.
- Each Classification is assigned to a registered user that has the Issue Manager role.

This section describes how to add your own Category/Classification assignments.

To add new Issue Categories

1. Log in to Matrix Navigator.

   You must have Business Administrator access.

2. From the main menu select **Object** > **New** > **Original**.

3. On the Original Dialog, enter the following:

   Type: Issue Category

   Name: <the name of the category>

   Revision: -

   The Revision value must be “-”.

   Vault: eService Production (or your own)

   Policy: Issue Categorization

To add new Issue Classifications

1. Log in to Matrix Navigator.

   You must have Business Administrator access.

2. From the main menu select **Object** > **New** > **Original**.

3. On the Original Dialog enter the following:

   Type: Issue Classification

   Name: <the name of the category>

   Revision: -

   The Revision value must be “-”.

   Vault: eService Production

   Policy: Issue Categorization

To Connect the Category to the Classification

Categories must be connected to one or more Classifications via the “Issue Category Classification” relationship. Classifications should not be created without being connected to a Category.

The following steps describe how to connect the two business objects together and then assign an Issue Manager to the Classification.

1. Once the Issue Category and Issue Classification are defined, select the Issue Category business object.
2. From the menu, select **Relationships** > **Connect**, or use the connect icon from the tool bar.
3. From the Connect dialog, select the Classification object on the right frame, and on the left frame select the relationship “Issue Category Classification”.
4. Click the **Connect** button.
5. Once the relationship is created, the relationship attribute “Issue Owned By” will be displayed. In this attribute you must enter a valid user ID having the role Issue Manager. Refer to the next section, ***Assigning Users as Issue Managers***.

The “Issue Owned By” field must contain a valid user ID with an Issue Manager role. If left blank, the context user is set as the owner, who should then reassign issue ownership to an Issue Manager.

Assigning Users as Issue Managers

Issue Managers have a very important role throughout the life of an issue. The Issue Manager is responsible for monitoring and responding to all issues assigned to him. The Issue Manager role is also responsible to notify the originator as soon as the issue is received and assigned.

The sample data that comes with the Framework assigns the following registered persons to specific Issue Categories and Classifications:

- Test IssueManager
- Test1 IssueManager
- Test2 IssueManager

In order to create an Issue, the default setup for Common Issues requires the above registered persons.

If you do not run the Framework sample data, you must register a new user in the application, assign it to the Issue Management Role, and assign it to the default Issue Category/Classification.

1. Open the application and click **Tools** > **Administrator** to define all new persons that are responsible for managing issues and to assign them to the role Issue Manager.
2. Once the Issue Managers have been defined in the application, proceed to the Matrix Navigator to assign them to the proper Issue Category and Classification.

For information about running the Framework sample data, see the ***AEF User Guide***.

User External Authentication

To set up external authentication for users accessing MatrixOne applications, see “Login Behavior When External Authentication is Used” in the ***Installation Guide.***

If you are using Site Minder or Clear Trust for user authentication, custom JPOs and special APIs must be set up in order to authenticate users accessing lifecycles, routes, and FDA approvals. To enable custom JPO authentication, see “Enabling External Authentication” in the ***Installation Guide***.

Index