



EMu Documentation

Archive View

Document Version 1

EMu Version 4.1



Contents

SECTION 1	Archive View	1
	Multi-part objects	3
SECTION 2	Enable / disable Archive View	4
SECTION 3	Using Archive View	5
	Index	9

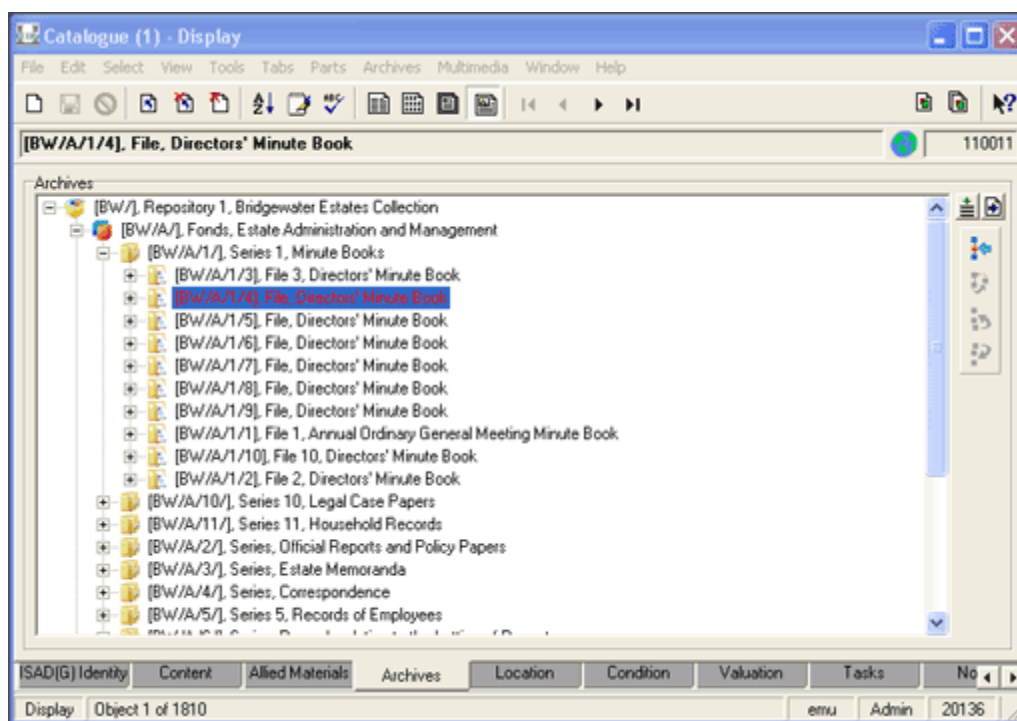
SECTION 1

Archive View

EMu has provided support for Archives since version 3.2.04 (November 2007), enabling users to choose from a number of archive formats:

- EAD - Encoded Archival Description
- ISAD(G) - General International Standard Archival Description
- Custom - User specified archival description

In order to help visualize the layout of an archive, a tab was added to the Catalog module containing a tree view of the structure. This Archives tab displays the hierarchy of the archive with icons used to differentiate between the various levels (fonds, series, etc.). A typical example is shown below:



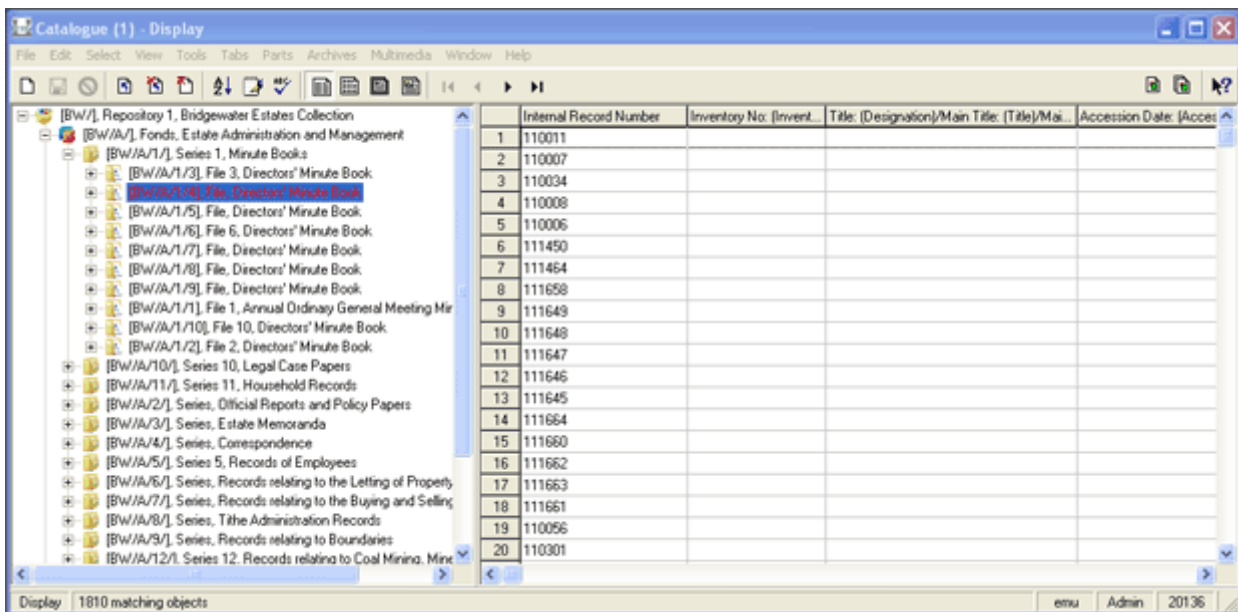
The Archives tab provides functionality to relocate the current record within the hierarchy. For example, the current record (highlighted in blue in the screenshot) can be drag-and-dropped to another position in the hierarchy, a move that shifts the record and all its children to the new location.

Although this support for archives is sufficient for viewing the structure of the hierarchy for the current record, it does not provide a simple mechanism for relocating other records within the hierarchy (i.e. records other than the current record), nor does it allow the hierarchy to be restructured easily. In order to address these two shortcomings and provide a constant visual reminder of the archive hierarchy, EMu 4.1 introduces the Archive View facility.

With the Archive View facility:

- An Archive View can be displayed in a panel to the left of the record display (similar to the Shortcuts View). Shown below.
The Archive View (on the left) displays the entire Archive tree for the current record (selected or displaying on the right of the module window).
- A menu option is used to display/hide the Archive View.
- The archive tree displayed in the Archive View has the same layout as used on the Archives tab.
- The current record is highlighted in the Archive View on the left of the module window. Fields appearing in Details View on the right (the record display) may be edited.
- Selecting a record in the Archive View, other than the current record, displays details of the selected record on the right of the module window. The record may be edited.
- Any record may be drag-and-dropped in the Archive View to allow the archive hierarchy to be adjusted.

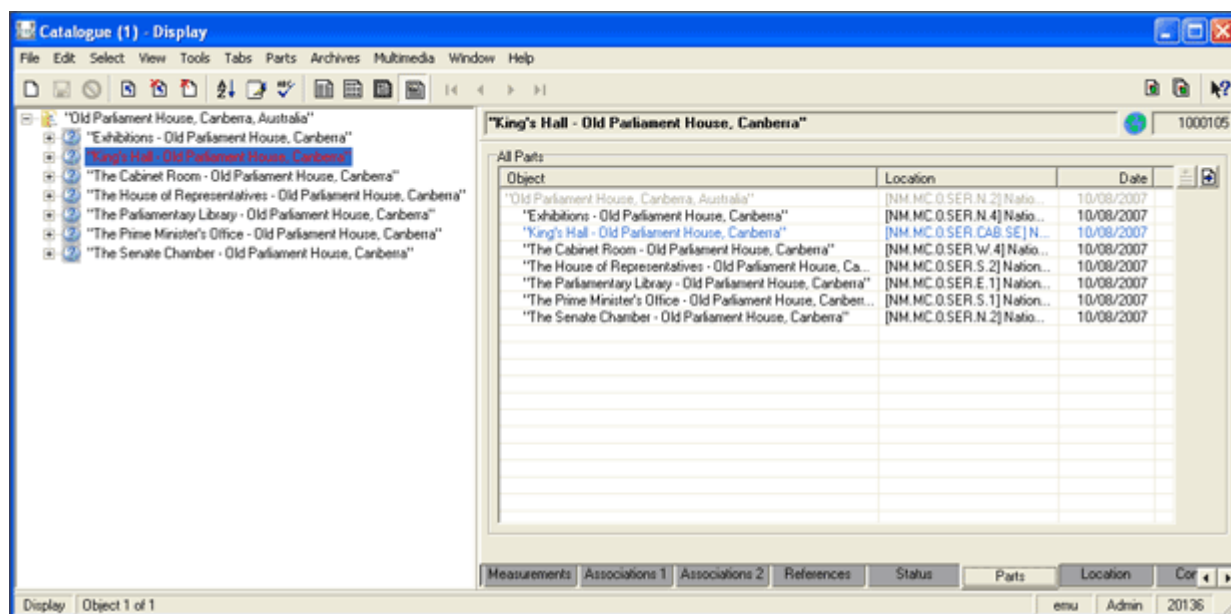
The image below shows the Archive View enabled (on the left of the module window) while displaying Archive records (on the right):



Multi-part objects

As long as the Archives tab is enabled for a Catalog, the Archive View functionality can be used to display the hierarchy for multi-part objects.

The Parts tab displays the association between objects within a collection using a list type view. The Archive View displays the same hierarchy in a tree view. The image below shows a multi-part object with Archive View enabled:



Enable / disable Archive View



The Archive View functionality is only enabled for Catalogs that have the Archives tab enabled. To use this functionality for multi-part objects, it is also necessary for the Archives tab to be enabled.

To enable / disable Archive View:

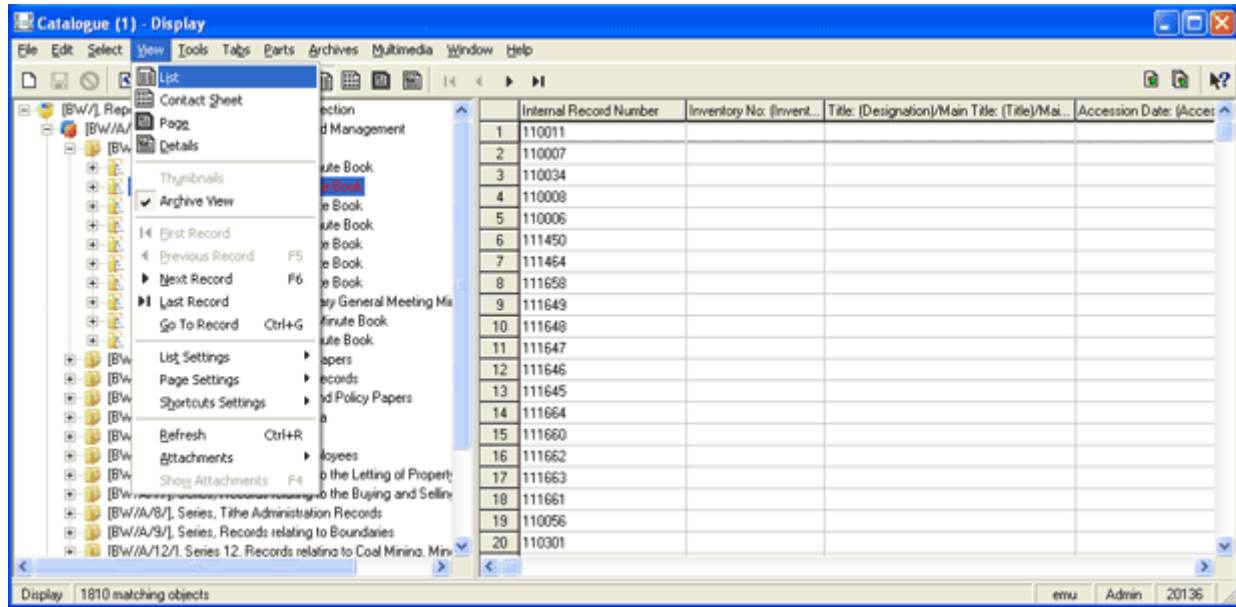
1. In the Catalog module, search for or otherwise list a group of records.
2. Select **View>Archive View** in the Menu bar

-OR-

Use the keyboard shortcut, ALT+V+C.

This Menu option toggles between displaying/hiding the Archive View panel. If the panel was hidden, it will now display, and vice versa.

When active, a tick displays beside the Archive View Menu option:



The Archive View setting (enable/disable) is retained between EMU sessions: when the setting is enabled, it will remain so until it is disabled.



Using Archive View



With Archive View enabled, it is possible to:

- Select a new record to display.
- Move records around in the archive hierarchy.

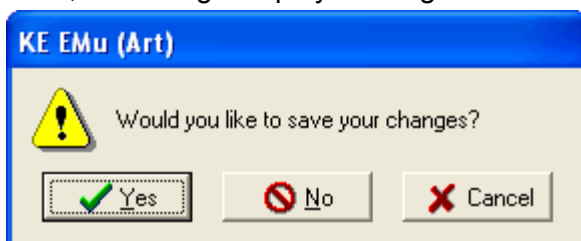
Selecting records

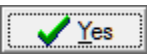


To select records:

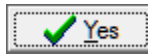

1. In the Catalog module, search for or otherwise list a group of records.
2. Enable Archive View (page 4).
3. In the Archive View tree, locate the record to display and select it by clicking any part of the record's text.

 The  plus icon to the left of an entry is used to show all child records.

If the current record is being edited when you select another record in the Archive View, a message displays asking whether the changes should be saved:



- Select  to save the changes and display the newly selected Archive record.
- Select  to lose the changes and display the newly selected Archive record.
- Select  to cancel the display of the newly selected Archive record.

If  /  was selected, the Archive record is displayed to the right of the Archive View panel.

The record is shown in the current mode (Details, List, Contact Sheet, etc.) and may be edited.

If the record selected was not one of the records returned at Step 1, it is added to the list of records.

Drag and drop

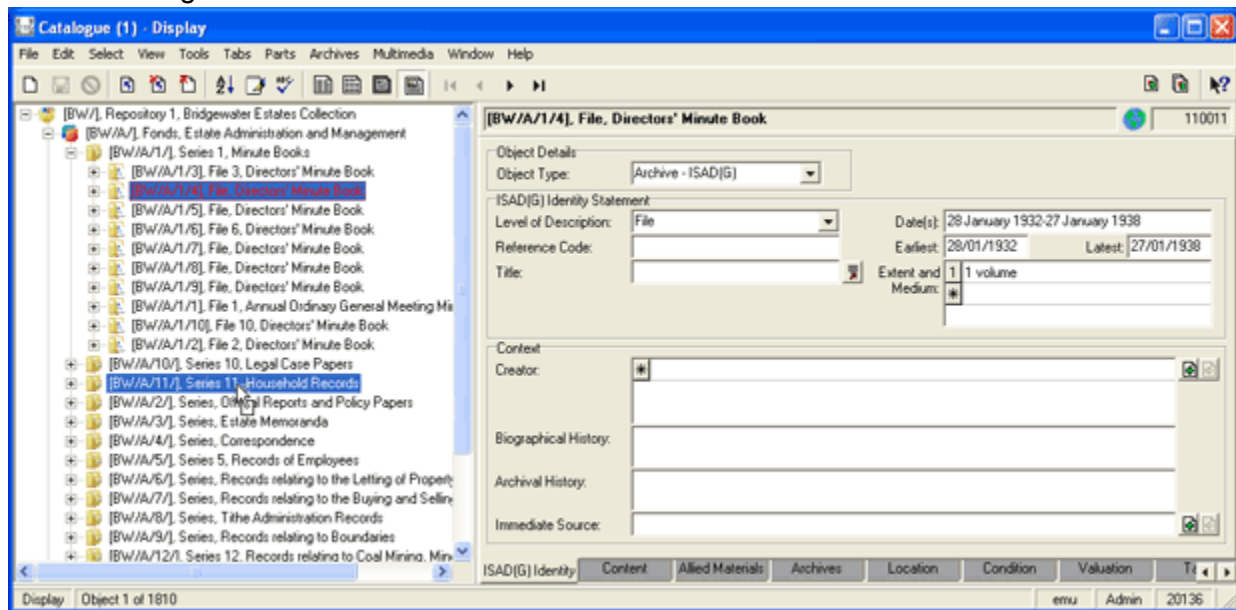
It is possible to drag and drop a record from one position to another using the Archive View tree. The functionality is similar to that provided on the Archives tab:

1. In the Catalog module, search for or otherwise list a group of records.
Moving a record in the Tree View is a matter of clicking a record, dragging it and dropping it on to another record in the Tree.
2. Expand the Archive View tree to display the destination record.

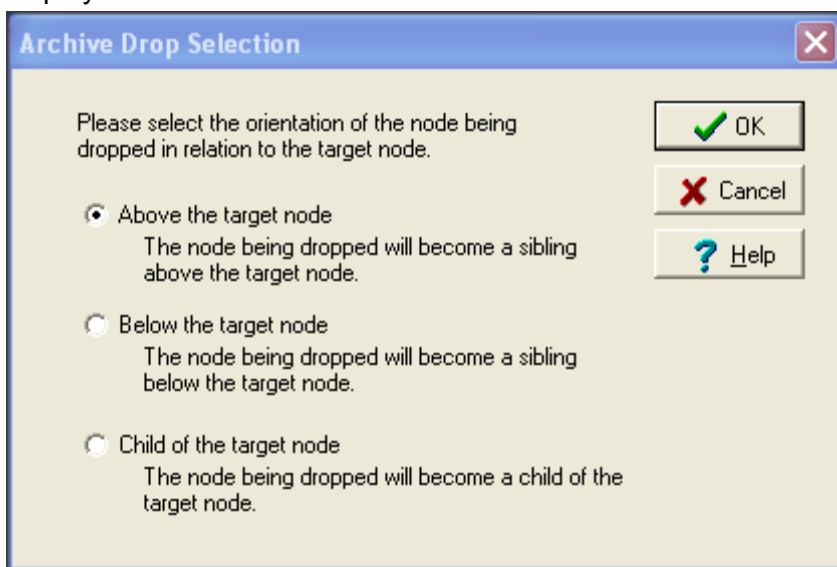


Once the drag operation begins, the tree cannot be expanded.


3. Select the record to be moved by clicking it and holding down the left mouse button.
4. Drag the record to the destination record and release the left button:



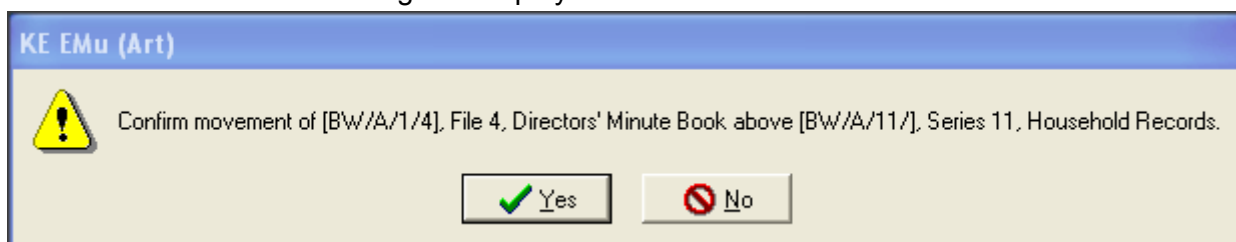
If the archive has user-defined ordering enabled (see the Archive|User Defined Ordering Registry entry in the EMu Help), the Archive Drop Selection dialog box displays:



5. Select an option from the Archive Drop Selection box.

 If user-defined ordering has not been specified, the moved record is always placed as a child of the destination record.

A confirmation dialog box displays:



Once confirmed, the dragged record is moved to the location specified.

Unlike the Archives tab drag and drop facility (which requires that the current record is saved before any change in the hierarchy is made), as soon as the operation is completed using the Archive View functionality (Step 5 above), the change to the hierarchy is complete.

Index

A

Archive View • 1

D

Drag and drop • 6

E

Enable / disable Archive View • 4, 5

M

Multi-part objects • 3

S

Selecting records • 5

U

Using Archive View • 5



EMu Documentation

Collection Descriptions module

Document Version 1.1

EMu Version 4.1



www.kesoftware.com

© 2012 KE Software. All rights reserved.

Contents

SECTION 1	Overview	1
SECTION 2	Collection Descriptions module tabs	3
	Description	4
	Significance	6
	External	7
	Parties	9
	Thesaurus	10
	Authority	11
	Sub Collection	12
	Hierarchy	17
	Associations	20
SECTION 3	Links to other modules	21
	Index	23

SECTION 1

Overview

EMu 4.1 introduces the Collection Descriptions module (CDM) which, as its name suggests, holds high level descriptive details about collections. A collection is very broadly any group of related or grouped items. The relationship may be items collected on an expedition; paintings from a particular period or by a specific artist, and so on.

As with information held in the Narratives module, collection level information is particularly useful for use on the web, and the CDM has been designed to facilitate web access. One opportunity presented by this module is cross-institutional searching of collection level details. Institutions which choose to participate can allow their CDM records to be harvested and made available for searching via their own website and that of any other participating institution.

This document provides a set of screen shots of each tab in the Collection Description module and a description of the fields on each tab.

SECTION 2

Collection Descriptions module tabs

Collection level details are recorded across nine tabs:

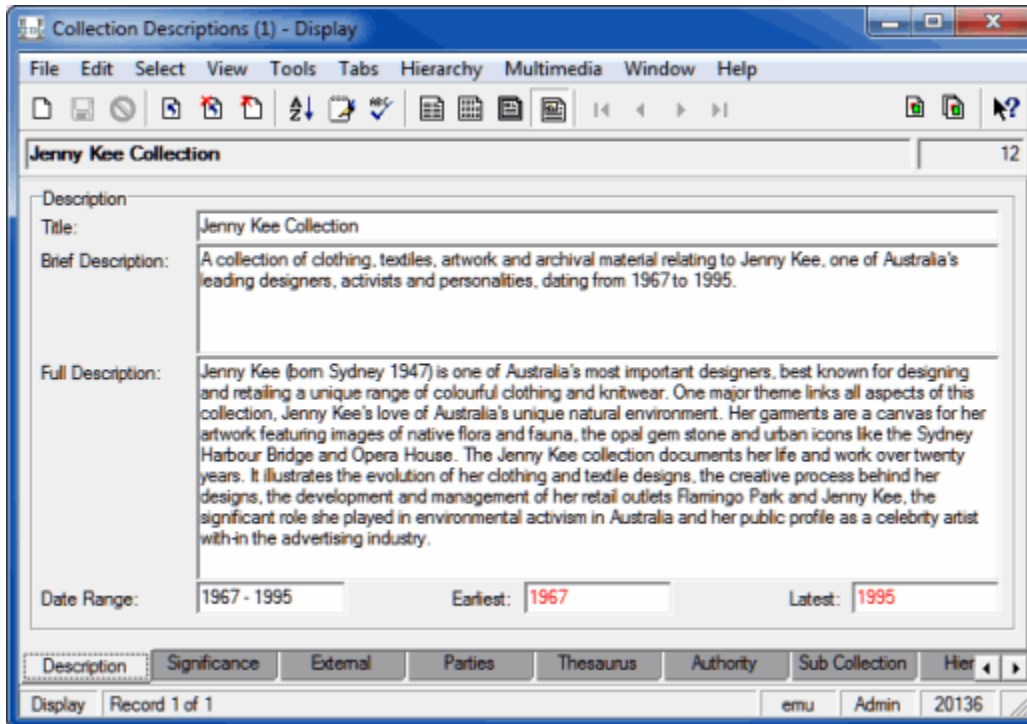
- Description (page 4)
- Significance (page 6)
- External (page 7)
- Parties (page 9)
- Thesaurus (page 10)
- Authority (page 11)
- Sub Collection (page 12)
- Hierarchy (page 17)
- Associations (page 20)

The following standard EMu tabs are also include in the CDM:

- Tasks
- Notes
- Multimedia
- Security
- Audit
- Admin

Description

Description



The Description tab holds a title for the collection, an abstract, detailed description, and dates covered by the collection.

The fields are:

Fields	Description
<i>Title</i>	The collection title, e.g. Jenny Kee Collection.
<i>Brief Description</i>	A brief summary or abstract describing the collection. The <i>Brief Description</i> is of particular use when presenting the collection on a website: when a search is performed, this abstract can be presented to the searcher as part of the search results.
<i>Full Description</i>	A detailed description of the collection.

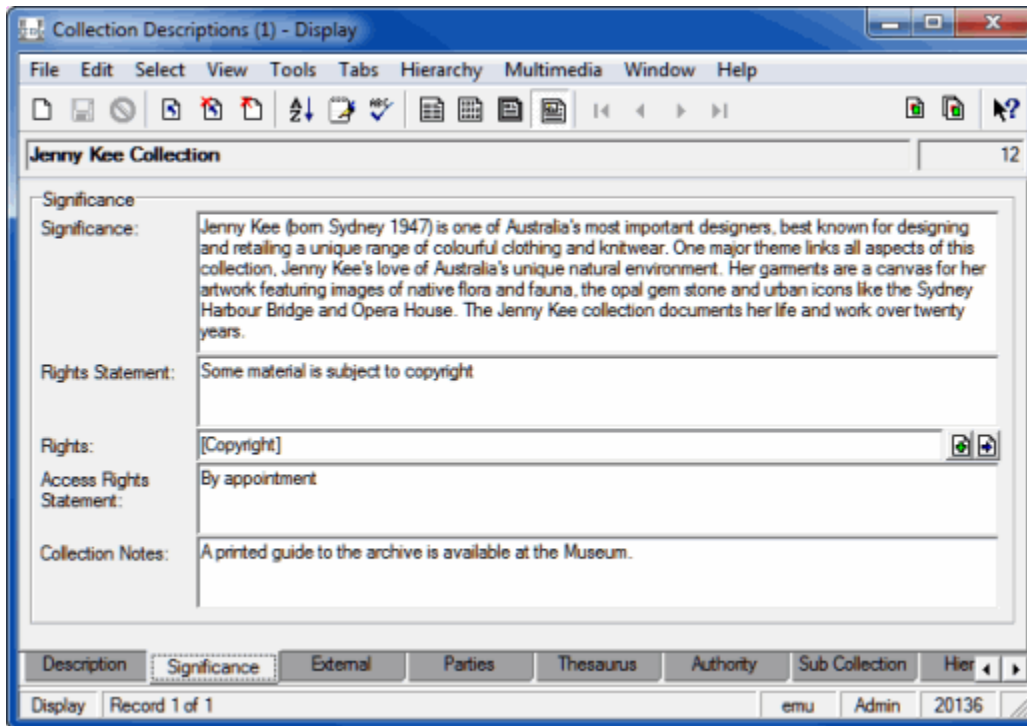
Description*Date Range*

The period covered by the collection. For example, the start and end dates of an expedition on which a group of items was collected; the period in which a group of paintings was painted, etc.

The three date fields operate in the same way as any other date range fields in EMu: enter a date, date range or circa date in the *Date Range* field and the *Earliest* and *Latest* fields are updated automatically (and can be edited as required).

Significance

Description



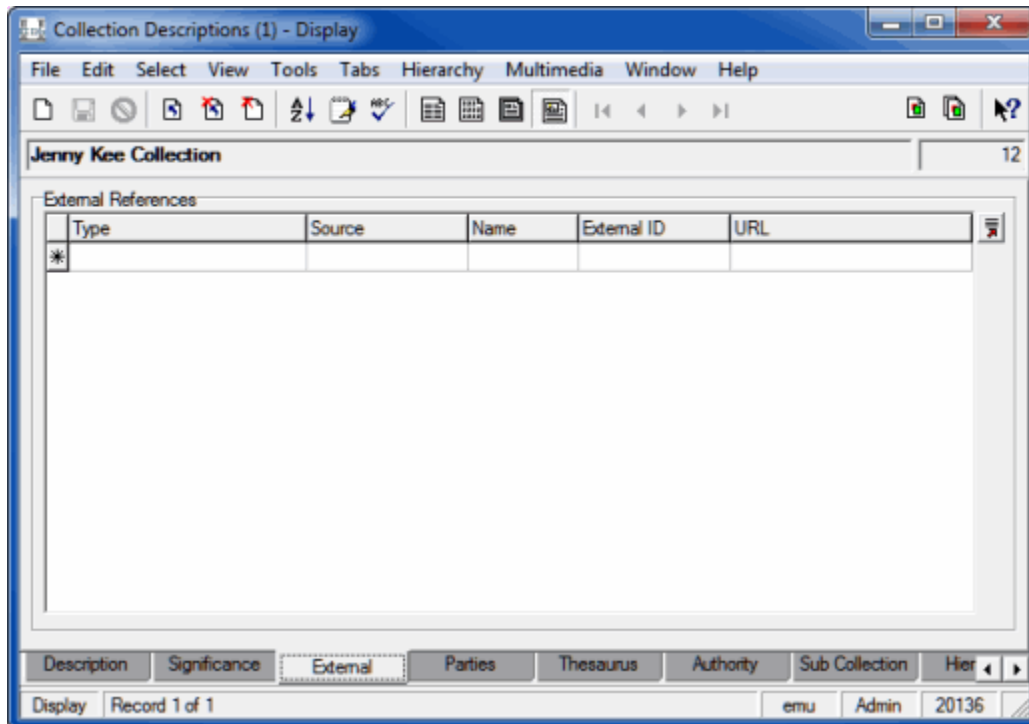
The Significance tab holds a general statement about the importance of the collection, details about any rights that apply to the collection, and any notes / comments specifically about the collection that do not fit anywhere else in the module.

The fields are:

Fields	Description
<i>Significance</i>	A statement about the collection's significance / importance.
<i>Rights Statement</i>	A statement concerning any rights that apply to the collection, e.g. <i>Some material is subject to copyright.</i>
<i>Rights</i>	A link to a record in the Rights module with details about the rights pertaining to this collection.
<i>Access Rights Statement</i>	A statement about access rights to the collection, e.g. <i>By appointment.</i>
<i>Collection Notes</i>	Similar to the Notes tab, but intended to hold any notes / comments specifically about the collection that do not fit elsewhere in the module.

External

Description



The External tab holds details about any *reference* to the collection that is not recorded in EMu: that is, there is no Parties or Thesaurus record describing the reference that can be linked to.

See the Parties (page 9) and Thesaurus (page 10) tabs for details about *internal* references to the collection.

Amongst other things, an external *reference* to the collection might be:

- A person / organization / group considered to be an authority / expert on the collection.
- A source of information of some description about the collection, such as a website.
- A place of significance to the collection.

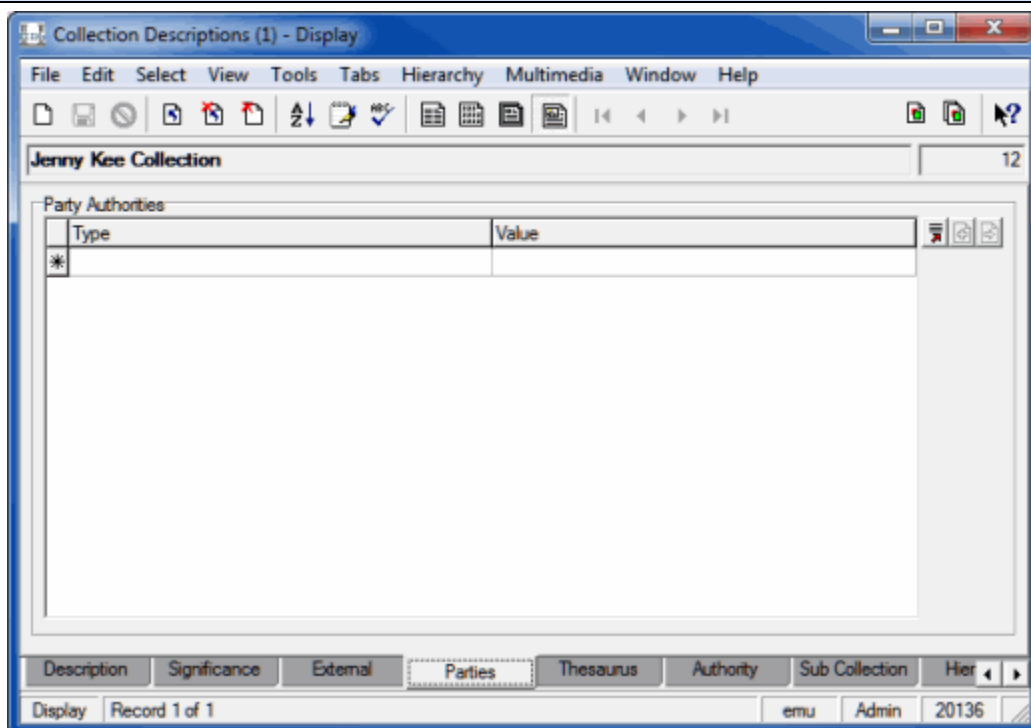
Description

The fields are:

Fields	Description
<i>Type</i>	<p>The type of external reference.</p> <p>Lookup List. Types are defined by an institution to suit its needs and could be:</p> <ul style="list-style-type: none">• Person• Cultural Group• Organisation• Subject• Place• Book• Article
<i>Source</i>	<p>The source of the external reference. A source could be a database (such as Getty), a website, a book, etc., in fact any medium which references / describes the collection.</p> <p>Lookup List. Sources are defined by an institution to suit its needs and could be:</p> <ul style="list-style-type: none">• People Australia• Local• APT• ScOT• Geoscience Australia• Getty
<i>Name</i>	<p>A keyword descriptor of the external reference (the name of a person / organization / group, book, website, etc.).</p>
<i>External ID</i>	<p>If the <i>Source</i> is an external database for instance, such as Getty, this is the external source's reference (code / identifier) for <i>Name</i>.</p>
<i>URL</i>	<p>A web address to the external reference, perhaps pointing to <i>Name</i> in the <i>Source</i>.</p>

Parties

Description



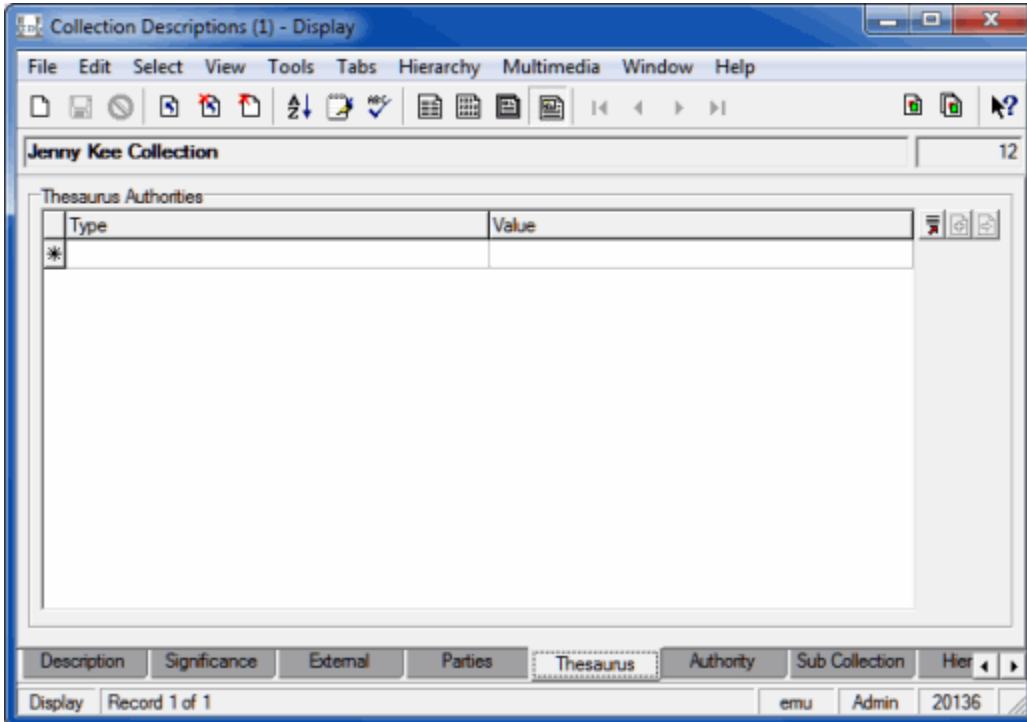
The Parties tab holds details of any people or organizations recorded in the EMu Parties module recognized as an authority / expert on the collection.

The fields are:

Fields	Description
<i>Type</i>	The type of Parties reference. Lookup List. Types are defined by an institution to suit its needs and could be: <ul style="list-style-type: none"> • Person • Organization • Cultural Group
<i>Value</i>	Link to a record in the EMu Parties module for a person or organization recognized as an authority on the collection.

Thesaurus

Description

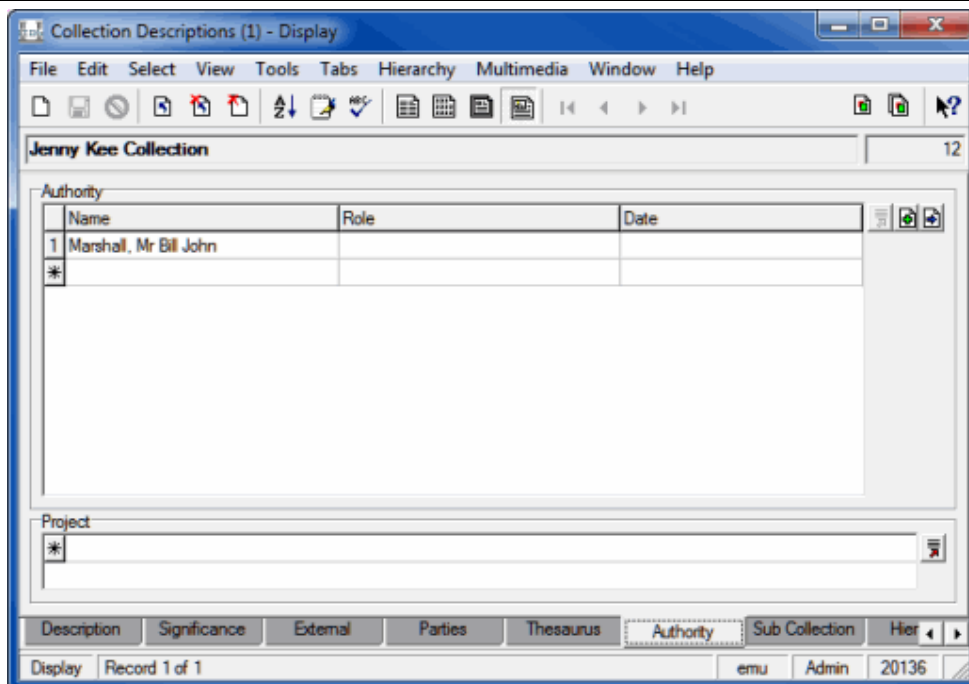


The Thesaurus tab holds any *reference* to the collection recorded in EMu's Thesaurus module. The fields are:

Fields	Description
<i>Type</i>	The type of Thesaurus reference. Lookup List. Types are defined by an institution to suit its needs and could be: <ul style="list-style-type: none"> • Object Name • Subject • Location
<i>Value</i>	Link to a record in the EMu Thesaurus module with some relevance to the collection.

Authority

Description



The Authority tab holds details about your organization's:

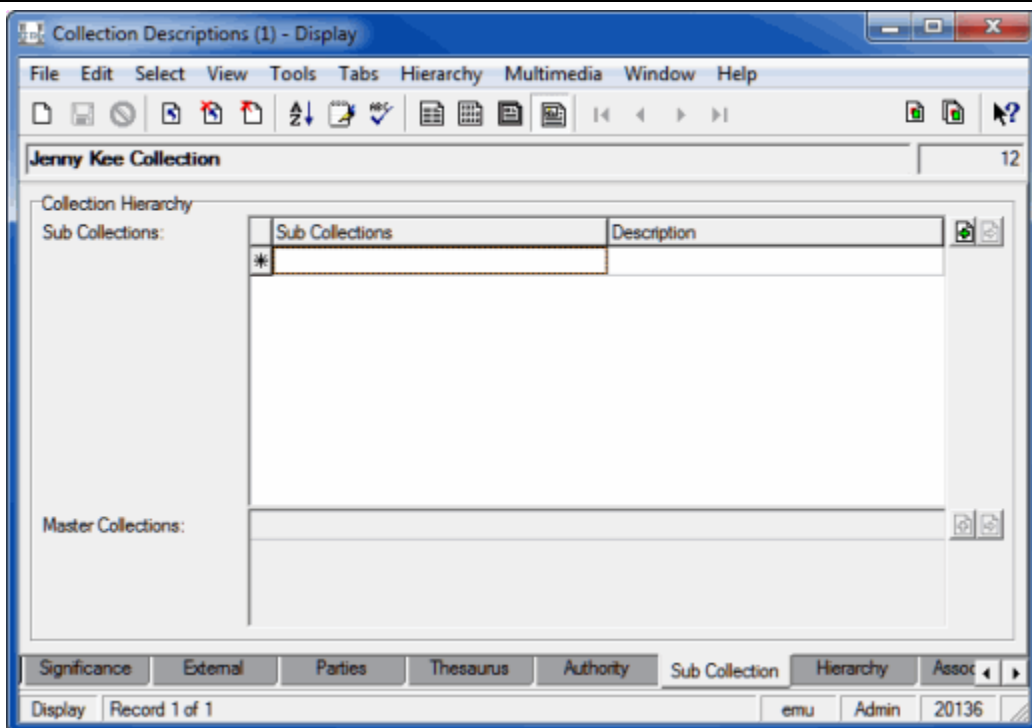
- EMu users with responsibility for maintaining the current record (EMu users who created and / or modified the current record).
- Staff with some responsibility for the collection itself.

The fields are:

Group	Fields	Description
<i>Authority</i>	<i>Name</i>	Link to a staff member's EMu record.
	<i>Role</i>	Lookup List. Roles are defined by an institution to suit its needs and could be: <ul style="list-style-type: none"> • Author • Editor • Curator • Researcher
	<i>Date</i>	The date that the record was created or modified.
<i>Project</i>		Lookup List of projects related to the collection.

Sub Collection

Description

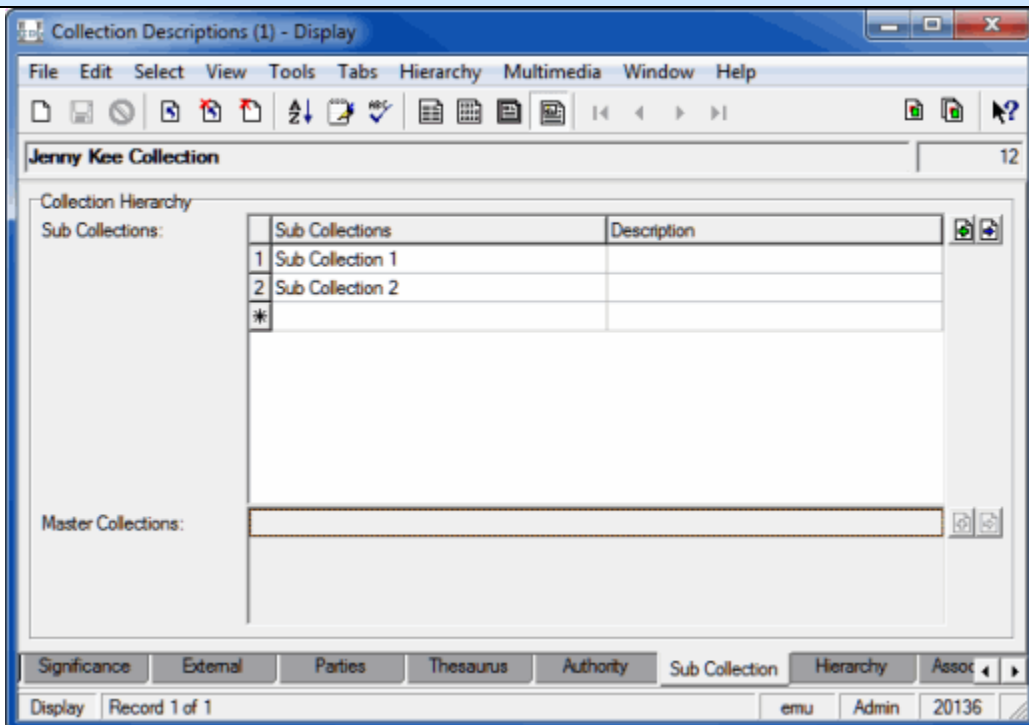


The Sub Collection, Hierarchy and Associations tabs describe relationships between collections in your institution.

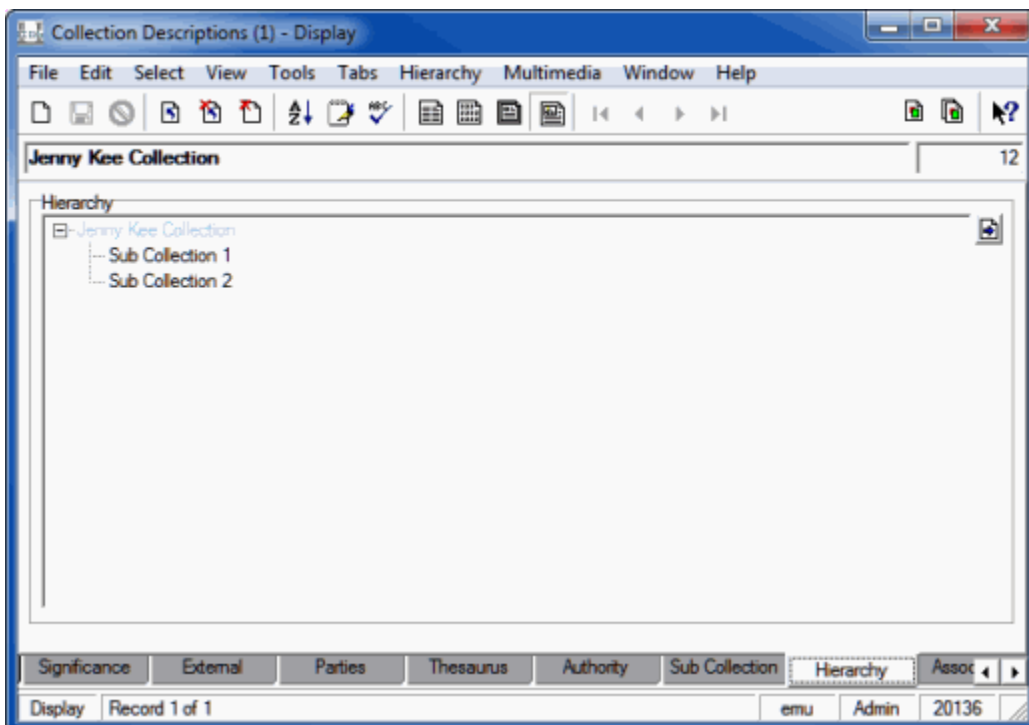
The Sub Collection and Hierarchy tabs work together to describe a hierarchy of collections. As the name suggests, the Sub Collection tab lists any other CDM record that is a sub collection (child) of the current collection (parent).

This screenshot shows two sub collections of the Jenny Kee Collection:

Description

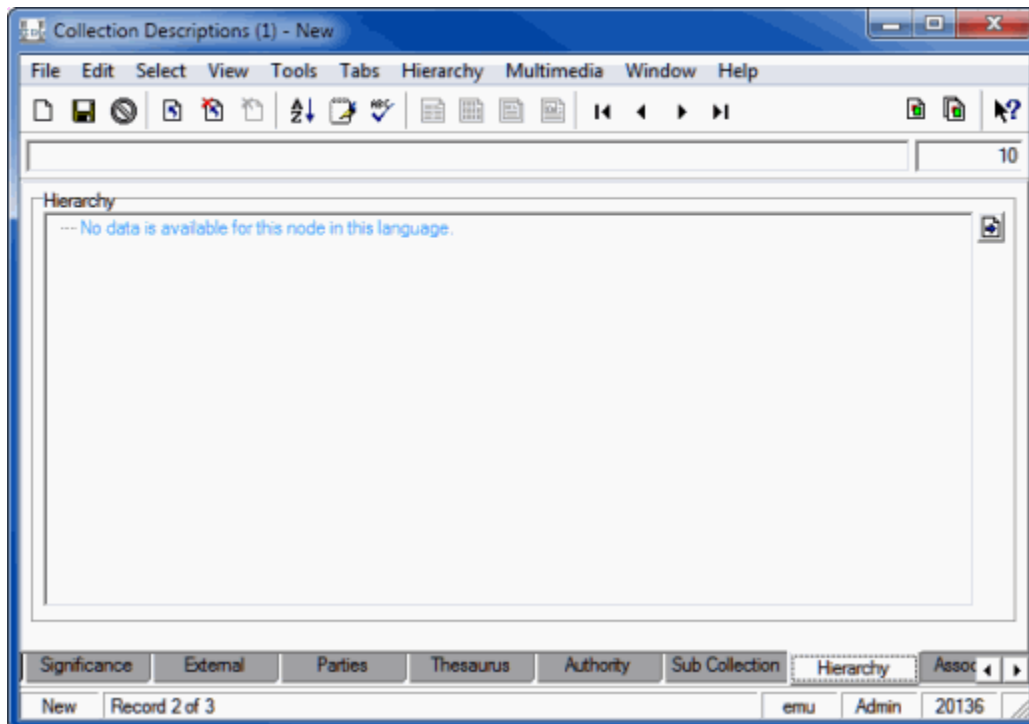


On the Hierarchy tab, the hierarchy of collections is displayed as a tree with the parent collection at the top:



Hierarchy

Description



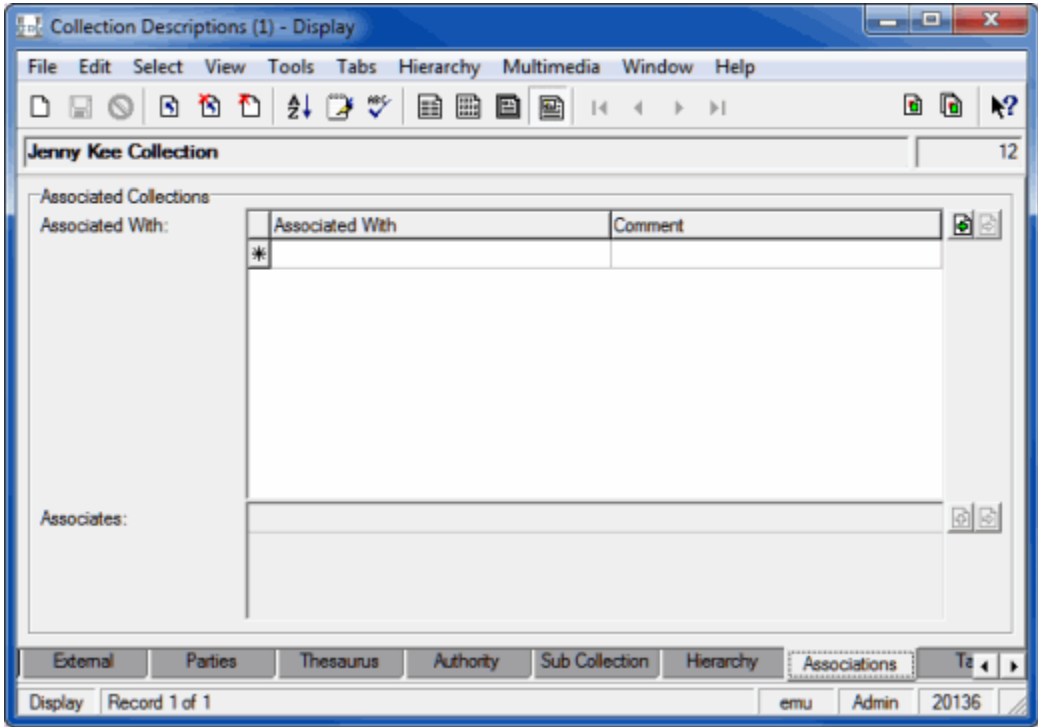
The Sub Collection, Hierarchy and Associations tabs describe relationships between collections in your institution.

The Sub Collection and Hierarchy tabs work together to describe a hierarchy of collections - see Sub Collection (page 12) for details.

The CDM includes a Hierarchy menu which can be used to update the Hierarchy:

Associations

Description



The Sub Collection, Hierarchy and Associations tabs describe relationships between collections in your institution.

The Associations tab holds details about collections that are associated with the current collection but which do not form part of its hierarchy.

The fields are:

Fields	Description
<i>Associated With</i>	Link to an associated Collection Descriptions record.
<i>Comment</i>	A comment about the nature of the association.
<i>Associates</i>	If one or more Collection Descriptions records link to the current record on the Associations tab, those records will be listed here.

SECTION 3

Links to other modules

As we have seen, a CDM record can link to other records in the same module that:

- Form a hierarchy of collections (on the Sub Collection (page 12) and Hierarchy (page 17) tabs).
- Identify other associations between collections (on the Associations (page 20) tab).

Links are also made to records in the Collection Descriptions module from the:

- Catalog module - for example, using the *Related Collection Descriptions* field on the Collections tab.
- Narratives module - for example, using the *Collection Descriptions: (Related Collection Descriptions)* field on the Collections tab.

Index

A

Associations • 3, 21, 23

Authority • 3, 11

C

Collection Descriptions module tabs • 3

D

Description • 3, 4

E

External • 3, 7

H

Hierarchy • 3, 18, 23

L

Links to other modules • 23

O

Overview • 1

P

Parties • 3, 7, 9

S

Significance • 3, 6

Sub Collection • 3, 13, 18, 19, 20, 23

T

Thesaurus • 3, 7, 10



EMu Documentation

Lookup List Maintenance

Document Version 1

EMu version 4.1



Contents

SECTION 1	Overview	1
	"Dirty" Lookup Lists	3
	New Lookup Lists background service	4
	lutserver	5
	emulutsrebuild	8
	Lookup List module	12
SECTION 2	Conclusion	15
	Index	17

SECTION 1

Overview

Lookup Lists have been a part of EMu since the first version of the software. They provide a useful mechanism for terminology control and are used extensively in the EMu client. The Lookup List facility stores data in a database table called eluts. The table contains a single record for each unique Lookup List entry. The information recorded for an entry includes:

Name The name of the Lookup List. The name is used to associate a set of Lookup List entries with a particular field in the Windows client.

Values A Lookup List entry may contain a number of values, with each value being a level in a hierarchy. The values start at level zero and increase. If a Lookup List is not part of a hierarchy (that is, it does not contain multiple levels), then only value zero is set. For hierarchies, a record will exist for each level in the hierarchy.

For example, if we have a Lookup List called Location, consisting of three levels:

- Country
- State
- City

with the data:

- Australia (Country)
- Victoria (State)
- Bendigo (City)

then three Lookup List entries (records) are generated:

	Entry 1	Entry 2	Entry 3
Country	Australia	Australia	Australia
State		Victoria	Victoria
City			Bendigo

The reason for three entries is that if a user wants to view a list of all countries (by viewing the *Countries* Lookup List for instance), then all Location Lookup entries that have the first value only filled are retrieved. As users may ask for any level in the hierarchy, the three records satisfy any potential request.

Levels The number of levels filled for the current record.

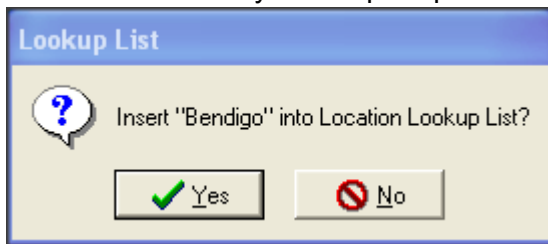
Using the example above, *Entry 1* has a `Levels` value of 1 (as only a single level is filled), while *Entry 2* has a `Levels` value of 2, etc.

- Persistent** If a Lookup List entry is not used anywhere in the system, the entry is removed from the Lookup List table. For example, if a record does not exist with a city location of `Bendigo`, then Entry 3 in the table above would be removed.
- A `Persistent` Lookup List entry is not deleted if the entry is not used.
- `Persistent` entries are used to pre-populate Lookup Lists with a known set of values, even if the values are not currently in use.
- Hidden** If a Lookup List entry is marked as `Hidden`, then the entry is shown for searches but not when inserting new values.
- For example, if the city `Bendigo` changed its name to `Sandhurst`, a new Lookup List entry is created with the city value `Sandhurst`. As the data still contains the value `Bendigo`, the Lookup List entry for this value is not removed. When users are inserting new records, we do not want the value `Bendigo` to appear in the Lookup List (as `Sandhurst` is the correct entry). However, when performing a search we still want `Bendigo` to appear in the list as there are still records containing this value. The `Hidden` attribute provides this functionality.
- The `Hidden` setting is used to phase out entries that should no longer be used.
- Used** The `Used` flag indicates whether the Lookup List entry is used anywhere within the system. If the flag is enabled, then at least one record in EMu uses the Lookup List entry's value.
- SortOrder** By default, the entries in a Lookup List are displayed in alphabetic order. It is possible to display the entries in a user specified (i.e. customized) order. The `SortOrder` value is used as the sort key for sorting the values in a Lookup List with customized ordering. The value may be numeric or alphabetic, in which case a numeric or alphabetic sort is applied respectively.
- The `SortOrder` attribute is rarely used.
- In order to implement customized sorting for a given Lookup List, the Windows client must be configured to provide the required functionality. Please contact KE Support for details.

"Dirty" Lookup Lists

There are a number of issues with the storage of Lookup List entries in the eluts database table:

1. Ideally, when deleting a record, any values in the deleted record which are in a Lookup List should be checked for uniqueness: if the values are not used in any other records, then the corresponding entry in the eluts database table should be deleted. However, the time required to perform these checks is prohibitive and in the interests of efficiency EMu does not perform them when a record is deleted. As a result Lookup List entries may exist in the eluts table where the value is not used in any records in the system.
2. If a user edits a Lookup value in a record and replaces it with a new value, when the record is saved they will be prompted to add the new value to the eluts table:



However, the old value may not be used in other records in the system. As with the first issue, the time required to perform the check would dramatically increase the time required to save a record and, again, EMu does not perform the checks in the name of efficiency.

A consequence of these issues is that the eluts table can become "dirty", containing entries that are no longer required and which users should not be seeing.

To solve the problem of "dirty" Lookup Lists, EMu rebuilds the contents of the eluts table on a nightly basis (or as defined by the system maintenance schedule). The rebuild process may be quite time consuming for sites with large numbers of records. Since the rebuild process reads the Lookup List values for all records in EMu, it can build a new version of the eluts table containing only the correct entries. Once the rebuild is complete, the eluts table is back in sync with the EMu data.

The need to rebuild the Lookup List table each night means that the Lookup Lists are offline while the rebuild takes place. It also means that there is less time to perform other nightly maintenance routines (e.g. batch updates, etc.). Another issue is that since the contents of the eluts table are replaced each night, users cannot use records in the eluts table in the way they can for any other module. For a given entry it is not possible to:

- Add notes or multimedia.
- Follow audit trails on changes made.
- Set record level security.

It is also not possible to add new values to a Lookup List by simply adding a new record to the eluts table.

New Lookup Lists background service

To reduce the nightly system maintenance and to allow the Lookup List table to be used as a regular module, EMu 4.1 has added a background service that ensures the Lookup List entries are always in sync with the records in the EMu system. The addition of the service removes the need to rebuild the Lookup Lists on a nightly basis, hence reducing system maintenance time.

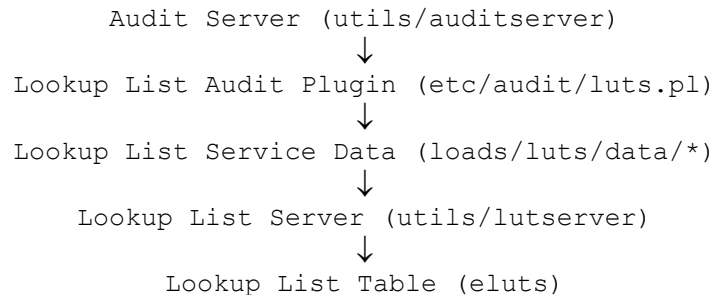
Furthermore, as the Lookup List table is no longer reloaded, the Lookup List module has been extended to include support for attributed notes (Notes tab) and multimedia (Multimedia tab). Audit trails on individual records are now maintained and Record Level Security may be used as for any other module.

Although a Lookup List rebuild program is still provided, it now applies changes to the eluts table rather than rebuilding the table completely. The introduction of the Lookup List service means that rebuilds are only required if the Lookup List table becomes corrupted, or if users accidentally delete records that are still in use.

lutserver

The server that handles updating of the eluts table in the new Lookup List service is called lutserver. lutserver runs on the EMu server machine waiting for requests to update Lookup List entries. The requests are generated by an audit trail plugin that picks up all deleted and modified records and determines what Lookup Lists need to be updated.

The process may be shown diagrammatically as:



Let's use an example to describe the steps taken by the Lookup List service to ensure Lookup Lists are kept up to date. In this example a user changes the city value in a record from `Bendigo` to `Sandhurst` and saves the record. The following steps occur:

1. When the user saves the record, the EMu server generates an XML description of the changes made to the record. The Audit Server (auditserver) loads these changes and passes them on to all registered plugins. The Lookup List Services registers the Lookup List Audit Plugin when the Audit Server is started. For our example record, the changes will show that `Bendigo` was changed to `Sandhurst` in the `City` column.
2. The Lookup List Audit Plugin looks at the changes supplied by the Audit Server. For each column changed it checks to see if a Lookup List is associated with the column. For our example, it will determine that the `City` column is associated with the `Location` Lookup List. It will then look at the two values and their associated operations. The value `Bendigo` has a delete operation (since it is being removed) and the value `Sandhurst` has an insert operation (as it is the new value). Since the Windows client was used to save the record, the Audit Plugin will ignore the insert operation as the user will have been asked to add the new entry to the `Location` Lookup List if it did not exist already. The delete operation is passed to the Lookup List Server for processing.
3. The Lookup List Audit Plugin writes a file containing the:
 - Operation performed (delete)
 - Value deleted (`Bendigo`)
 - Lookup List Name (`Location`)
 - Column changed (`City`)

The file is located in the `loads/luts/data` directory. The format of the file name is `date.time.table.irn`, where:

- `date` is an eight digit date in `yyyymmdd` format.
- `time` is a six digit time in `hhmmss` format.
- `table` is the database table in which the record was modified.
- `irn` is the key number of the record modified.

An example file name is: `20121026.133308.ecatalogue.375`.

4. The Lookup List Server fetches all files in the `loads/luts/data` directory and sorts them into date/time order. It is important that the files are processed in the same order as they were created, otherwise synchronization issues may arise. The Lookup List Server reads the contents of each file, extracting the information within. Then for each Lookup List in the file it determines the operation to apply (insert or delete):
 - For an insert operation it checks to see if the value is already in the Lookup List; if not it inserts a new value into the eluts table.
 - For a delete operation it checks whether the value is used anywhere in the EMu system for the given Lookup List name. If the value is not used, the record is deleted from the eluts table (provided it is not `Persistent`).

Once the file has been processed, it is removed. Once all the files have been processed, the Lookup List Server waits for new files to process.

The reason for so many steps is that audit plugins must be fast so that the Audit Server can process audit records quickly. To ensure the audit plugin is fast, the main Lookup List processing is removed from the audit plugin and moved to the Lookup List Server. The split ensures that the Audit Server keeps up with audit changes, even when the Lookup List Server may lag.

The Lookup List Server handles all deletion operations, that is it checks whether a Lookup List entry is still in use and if not, deletes it. It also handles insertions (that is new values) from all sources except the Windows client.

The Lookup List Server and the Registry

The Lookup List Server complies with the Lookup and Lookup Exact Registry entries:

- If a column has Lookup Exact set to `true`, then all comparisons with existing entries in the Lookup List table are performed as exact matches, i.e. the character case and punctuation must match exactly.
- If Lookup Exact is not enabled, all punctuation and character case is ignored for value comparisons. Search the EMu Help for details on the `Lookup Exact Registry` entry for more details.

The Lookup Registry entry is used to control the flags set when adding new values to a Lookup List. The table below lists each setting and examines how the Lookup List Server implements the required functionality. The Lookup Registry entry settings are generally set on a per column basis.

Setting	Description
<code>skip</code> <code>readonly</code>	The entry will not be added to the Lookup List table. If the column is part of a hierarchy, the entry is only skipped if the bottom level has this setting enabled. For example, if the <i>State</i> column has <code>skip</code> enabled, then entries with <i>Country</i> , <i>State</i> and <i>City</i> will be created, but entries with just <i>Country</i> and <i>State</i> will be skipped.
<code>readwrite</code>	The <code>readwrite</code> setting adds a new value to the Lookup List table. If an entry already exists but has the <code>Hidden</code> flag enabled, the flag will be reset (i.e. disabled). If the <code>Used</code> flag is disabled, it is enabled.
<code>autowrite</code>	The <code>autowrite</code> setting adds a new value to the Lookup List table. If an entry already exists and the <code>Used</code> flag is disabled, it is enabled.
<code>autowriteignore</code> <code>readignore</code> <code>writeignore</code>	If the entry does not exist, a new entry is created with <code>Hidden</code> enabled. If an entry already exists and the <code>Used</code> flag is disabled, it is enabled.

Finally, the Lookup List Server will not delete any entry from the Lookup List table that has `Persistent` enabled.

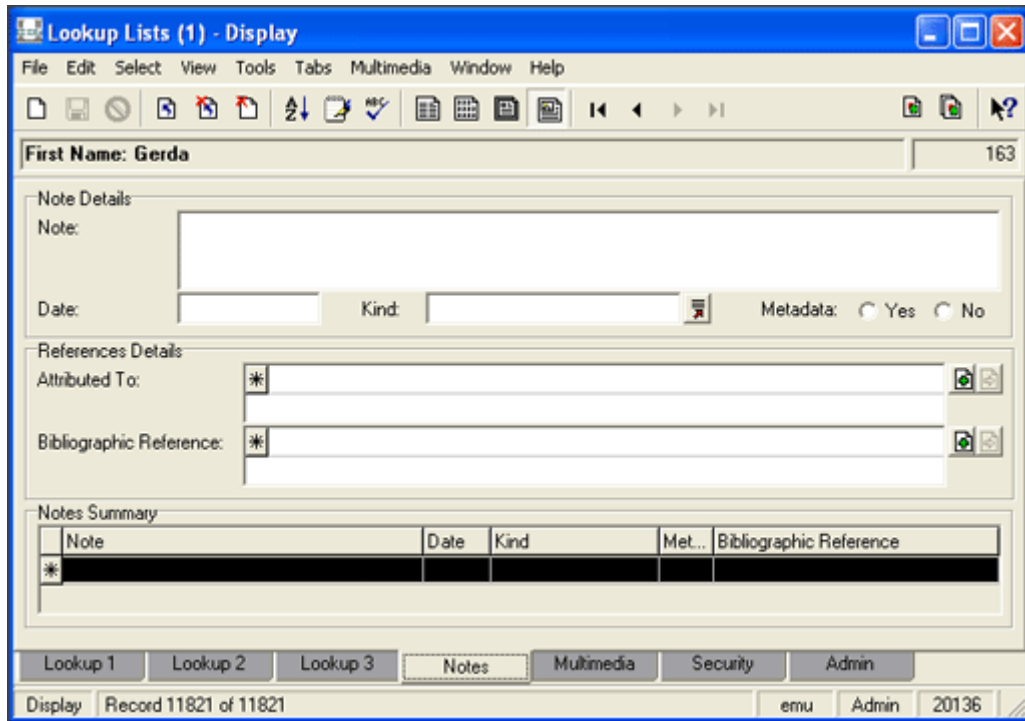
The Lookup List Server, `lutserver`, provides the functionality required to keep the Lookup List table in sync with values used within the EMu System. The addition of the server removes the need to rebuild the Lookup List tables on a nightly (or otherwise) basis.

Lookup List module

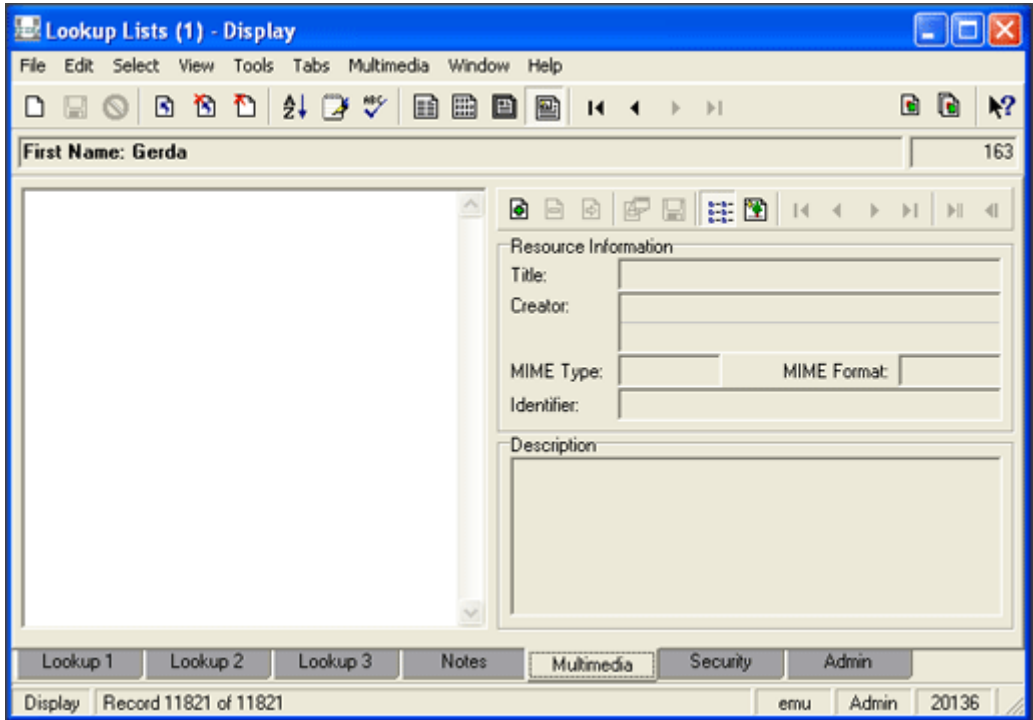
Until the release of the new Lookup List maintenance facilities in EMu 4.1, users could not interact with the Lookup List module and expect their changes to be preserved. The module itself was simplified to the point where only Lookup List specific information was stored. Fields available in all other modules were disabled. The maintenance changes introduced with EMu 4.1 mean that users can now access and use the module as they would any other module.

The following tabs are now available and functional in the Lookup Lists module:

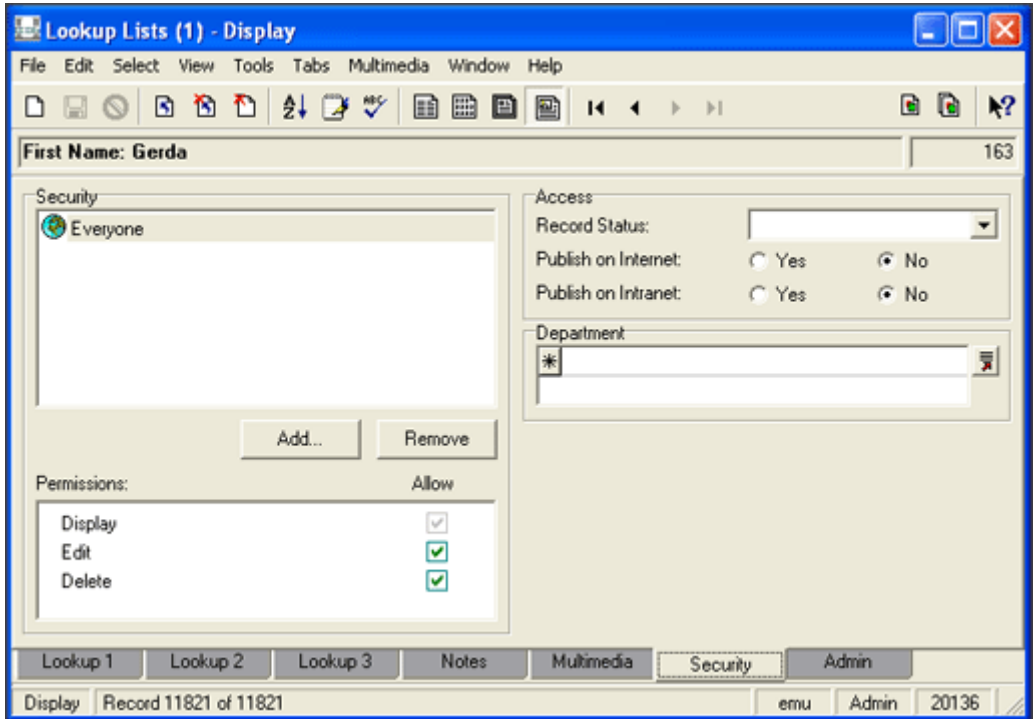
Notes



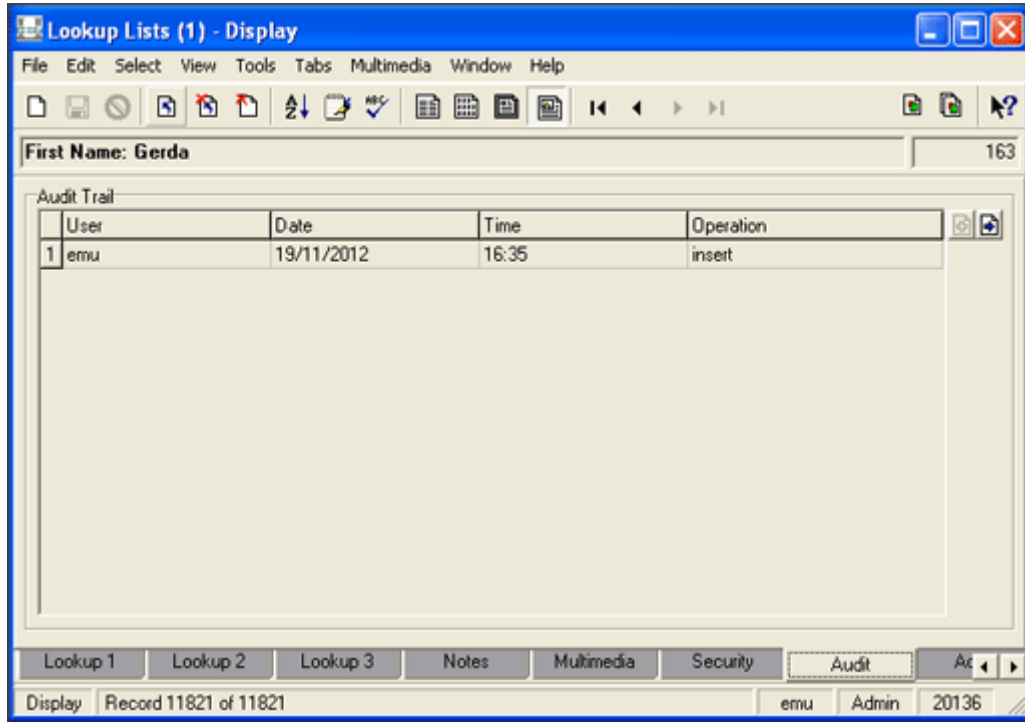
Multimedia



Security



Audit



SECTION 2

Conclusion

The changes to the Lookup List maintenance framework made in EMu 4.1 provide a number of benefits:

- Lookup Lists are always up to date. Once the last use of a Lookup List value is deleted, the entry is removed from the Lookup List table.
- The Lookup List nightly maintenance procedure is no longer required. The removal of the maintenance decreases the amount of time required by the nightly maintenance routines.
- The Lookup List module now provides the standard tabs available in all other EMu modules.
- The `emulutsrebuild` command may be used to check the consistency of the Lookup List table and apply any needed adjustments.

The changes to the Lookup List maintenance mechanism are the first of a series of changes designed to decrease the amount of time EMu requires to complete its maintenance runs.

Index

C

Check Lookup List table is synchronized • 11

Conclusion • 15

E

emulutsrebuild • 8

L

Lookup List module • 12

lutserver • 5

N

New Lookup Lists background service • 4

O

Overview • 1

R

Rebuild all Lookup Lists • 11

Rebuild the Location and Admin Names Lookup List • 11

T

The Lookup List Server and the Registry • 7



EMu Documentation

The After Export facility

Document Version 1.3

EMu Version 4.1



Contents

SECTION 1	Overview	1
SECTION 2	Setting an After Export Command	3
SECTION 3	Developing an After Export script	7
	The After Export script	7
	Creating an After Export script	13
	KE::Export usage	19
	Index	31

SECTION 1

Overview

The Scheduled Exports facility introduced with EMu 4.0.0.2 provides a mechanism for exporting data out of EMu on a regular or adhoc basis. Exports scheduled on a regular basis are executed by the server without any user intervention. When an export is complete, a record is added to the Exports module with:

- The results of the export
- A list of the files generated
- Any associated errors

The user must then retrieve the record from the Exports module to access the results and the exported data.

Introduced with EMu 4.1 the After Export facility allows a command to be executed once an export has completed. Amongst other things, the command can:

- Email the export files to a list of users.
- Email the results of the export to a list of users.
- Copy the export files to another machine behind a secure firewall.
- Copy the export files over the internet via a secure transfer mechanism.
- Send an SMS to a list of telephone numbers.

In fact, an After Export command may perform any number of tasks as it has full access to the Exports record generated. The command runs on the EMu server allowing access to the full facilities offered by the server. The After Export facility is designed to allow new commands to be added within the existing framework. In order to simplify the creation of new commands, the `KE::Export` perl module is provided; this incorporates much of the functionality required by an After Export command.

SECTION 2

Setting an After Export Command

A scheduled export is configured using the Export Properties dialog in a module (by selecting **Tools>Export** from the module Menu bar). An After Export command is added to a scheduled export using the After Export tab of the Export Properties dialog:

- If an After Export command has been defined, it can be selected from the *Command* drop list on the After Export tab.
- A command may require values to be provided by a user in order to run; if so, input boxes for the required values will display on the After Export tab when the command is selected from the *Command* drop list.

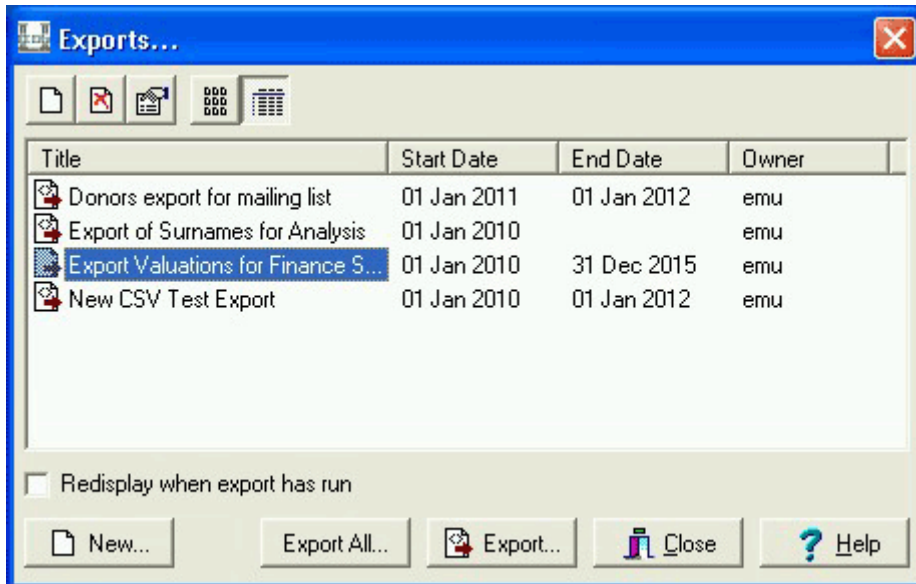
In EMu:

1. Open any module.
2. Search for or otherwise list a group of records.
3. Select **Tools>Export** in the Menu bar

-OR-

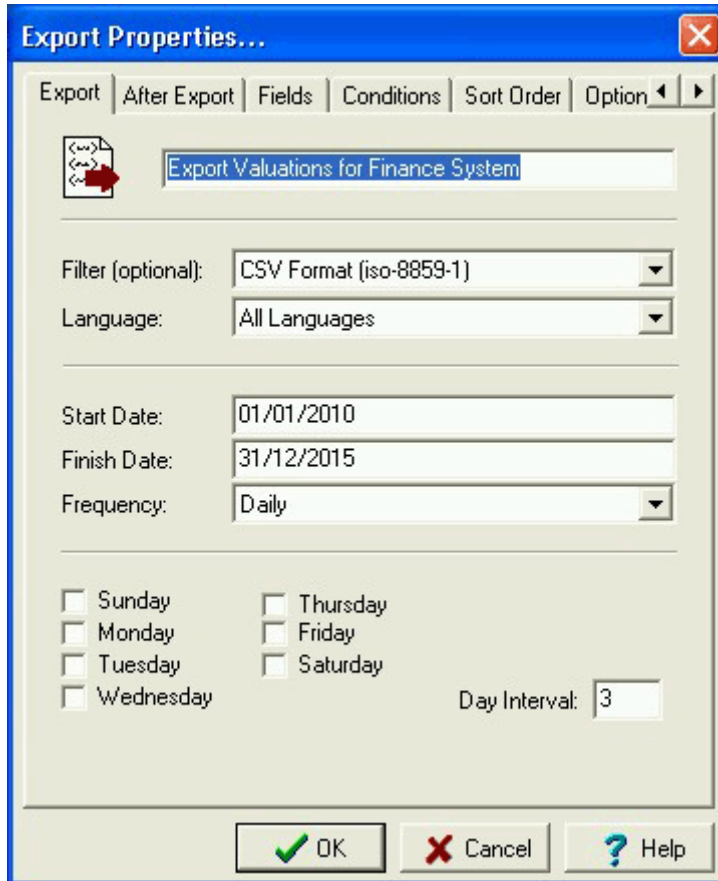
Use the keyboard shortcut, ALT+T+X.

The Exports dialog displays with a list of scheduled exports for the current module:

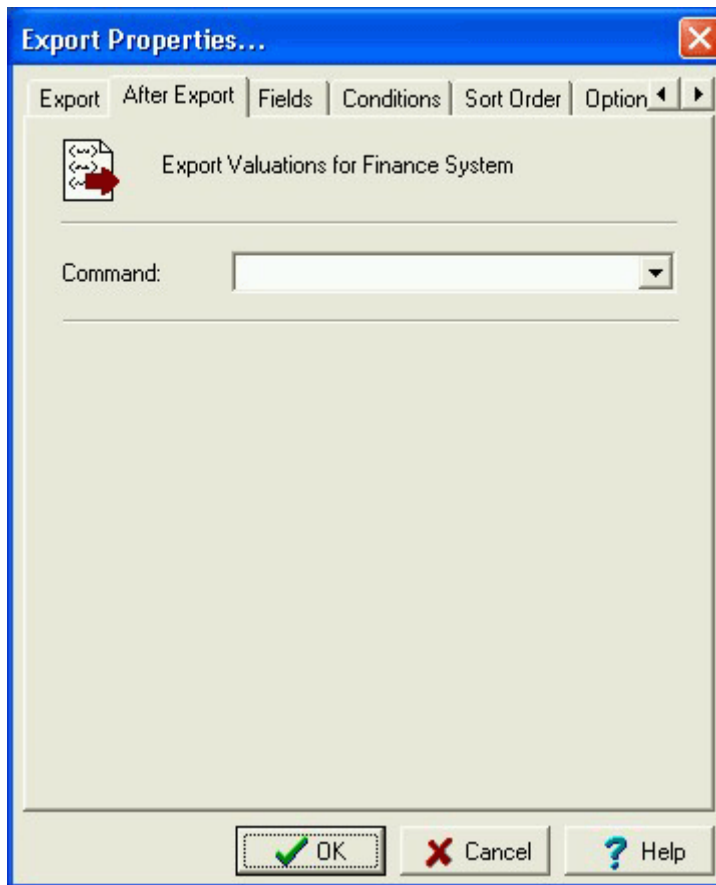


4. Select a scheduled export and click .

The Export Properties dialog displays:

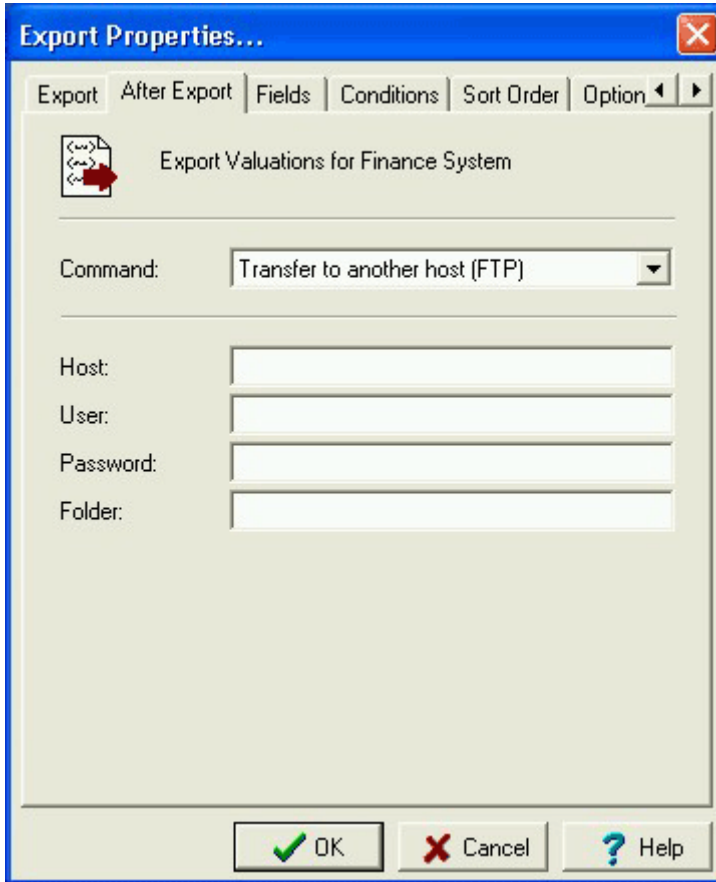



5. Select the **After Export** tab:

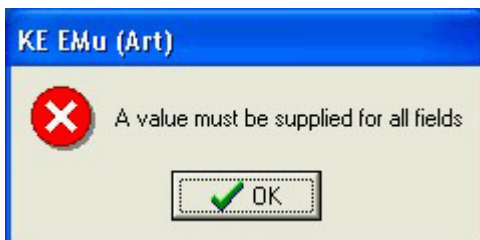


6. From the *Command* drop list, select the command to be executed.

If the command requires values to be provided in order to run, input boxes for the parameters will display on the After Export tab:



7. Enter values for each of the fields and when you're done, click . If a parameter was not provided, an error will display:



If all parameters have a value, the After Export command is saved and will be executed next time the scheduled export is run.

SECTION 3

Developing an After Export script

The After Export script

The Schedules module holds a record for each scheduled export defined in any module.

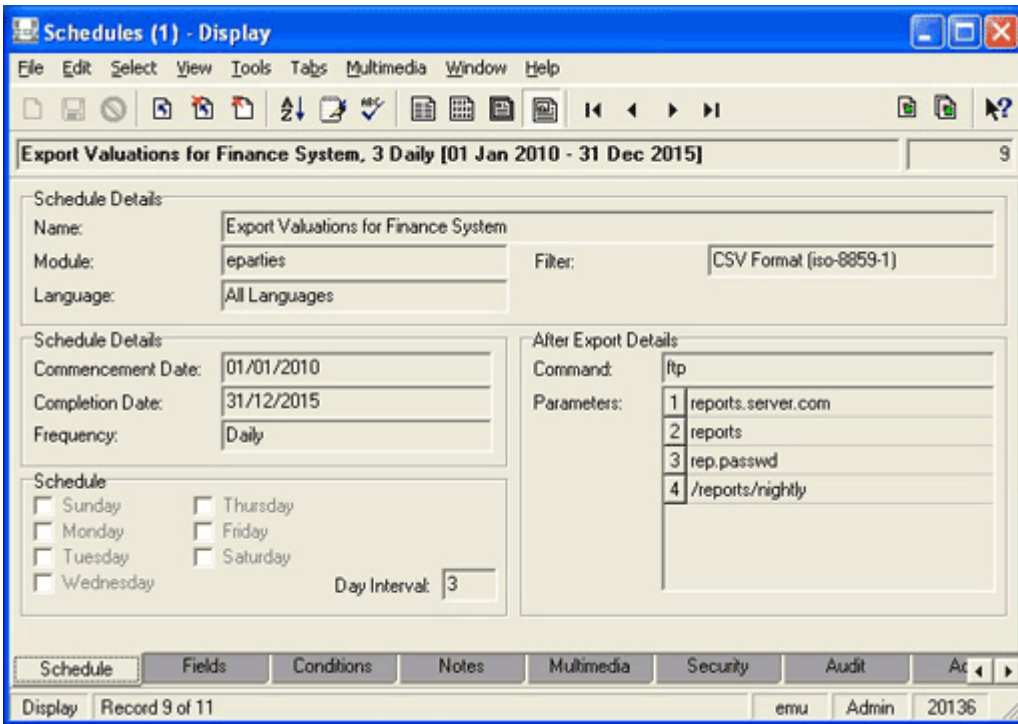
When an After Export command is added to a scheduled export, a script located on the EMu server is executed after the scheduled export is run. In the Schedules record for the scheduled export:

- *Command: (After Export Details)* holds the name of the script to be executed.
- *Parameters: (After Export Details)* lists the parameters required by the script.

In the Schedules record below, the script executed when the scheduled export is complete is called `ftp`. It has four parameters:

- `reports.server.com` (host name)
- `reports` (user name)
- `rep.passwd` (password)

- `/reports/nightly` (location)



The script file is located in one of the following directories on the EMu server:

- `local/etc/exports/after/table`
- `local/etc/exports/after`
- `etc/exports/after/table`
- `etc/exports/after`

where *table* is the name of the table (module) on which the export is performed. The table name is indicated in the *Module: (Schedule Details)* field (as shown above). To locate the script, the system looks for a file called `ftp` (in this case) in each of the directories listed, from first to last. When the file is found, the script stored in it is executed.

The hierarchy of directories listed above allows customized versions of existing scripts to be added by simply placing a file with the same name as the existing script into one of the `local` directories.

An After Export script is called by one of two methods:

1. In the first method, a script is called when a user selects a command from the *Command* drop list on the After Export tab of the Export Properties dialog.



In this case, the EMu client calls the script to get the title and list of parameters required by the command: the title displays in the *Command* drop list and the parameters are displayed below it. The EMu client calls the script with one argument, an option indicating which language to use to display the script's title and parameters. The usage is:

```
script -lnum
```

where *num* is a number corresponding to the language to use:

Number	Language
0	English
1	French
2	English (American)
3	Spanish
4	German
5	Italian
6	Dutch
7	Danish
8	Polish
9	Norwegian
10	Swedish
11	Greek
12	Arabic
13	Hebrew
14	French (Canadian)
15	Finish

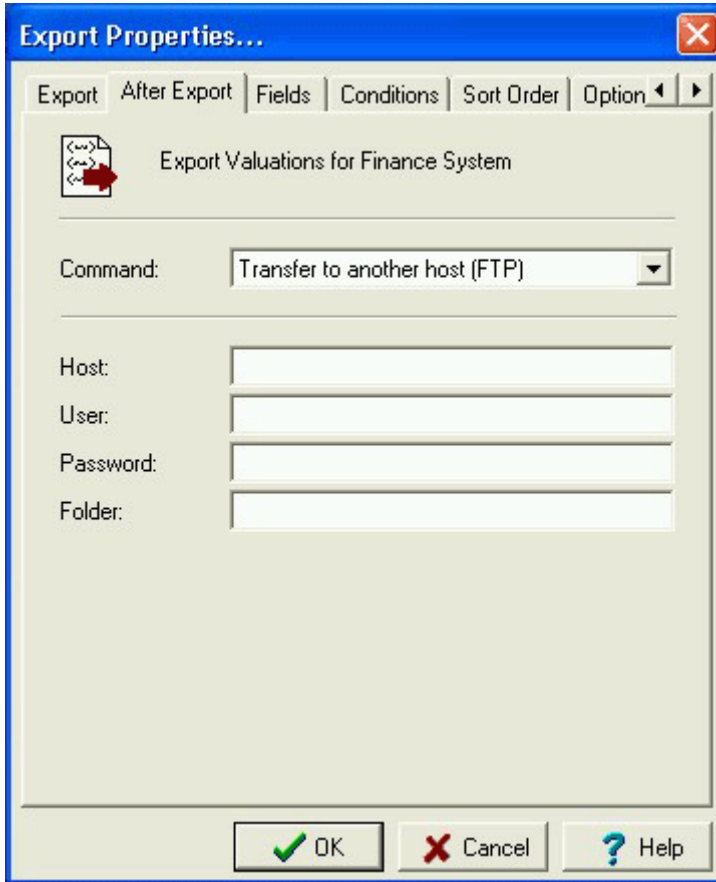


The *l* in `script -lnum` is a lowercase L.

For example, running `ftp - 10` produces:

```
Transfer to another host (FTP)
Host:
User:
Password:
Folder:
```

The first line is displayed in the *Command* drop list and the remaining lines are displayed as prompts for input boxes:



If the command is not supported by the local machine, then no output should be generated (the command should be hidden in the *Command* drop list) and a non-zero exit status returned.

2. In the second method, the back end `emuexport` command calls the After Export script after a scheduled export has run and a record with the results of the export has been created in the Exports module. In this instance the script is called with the IRN (Internal Record Number) of the record created in the Exports (`eexports`) module. The usage is:

```
script exportirn
```

The script should use *exportirn* to access the Exports (`eexports`) record and perform whatever activities the script was designed to do (e.g. send an email, copy files, etc.). If an error occurs while processing the export, an error message should be printed and a non-zero exit status returned. If the script completes successfully, a zero exit status should be returned.

These two methods are explained in detail in *Creating an After Export script* (page 13).

The perl code below is a sample `ftp` script:

```
#!/usr/bin/perl

#
# Copyright (c) 1998-2011 KE Software Pty Ltd
#

use strict;
use KE::Export;

#
# Parameters for ftp.
#
my $prompts =
{
    0 => [
        "Transfer to another host (FTP)",
        "Host:",
        "User:",
        "Password:",
        "Folder:"
    ]
};

#
# Check whether ftp is supported on this machine.
#
my $ftp = KE::Export::Ftp->new();
if (! $ftp->IsSupported())
{
    exit 1;
}

#
# Parse the arguments.
#
my $export = KE::Export->new();
if (! $export->ParseArgs(\@ARGV))
{
    exit 1;
}

#
# List parameters if required.
#
if ($export->ListParameters($prompts))
{
    exit 0;
}

#
# Do the transfer with the required arguments.
#
my $status = $ftp->Execute
(
```

```
host      =>      $export->GetData('Parameters_tab')->[0],
user      =>      $export->GetData('Parameters_tab')->[1],
password  =>      $export->GetData('Parameters_tab')->[2],
destination =>      $export->GetData('Parameters_tab')->[3],
filelist  =>      $export->GetData('FileName_tab')
);
#
# Send back the error status.
#
exit $status;
```

This script uses the perl `KE::Export` module which provides most of the required functionality.

In essence the script:

1. Checks whether the server provides FTP support. If not, it returns a non-zero exit status (i.e. 1).
2. Parses the arguments to determine which version of the script has been called.

If the arguments are invalid, a non-zero exit status is returned once again.

-OR-

If a valid `-l num` argument was given and the script was called using the first method described above, the script prints out the title and parameters for language `num`. Then exits with a zero exit status (indicating success).

-OR-

If the script was called using the second method described above, the file transfer is performed using ftp. An ftp object, passed the required parameters, transfers the files. The status of the ftp object is used for the exit status, where a non-zero value indicates the transfer failed, and a zero indicates success.

For a complete description of the functionality provided by the `KE::Export` module see *KE::Export usage*.

Creating an After Export script

The script for an After Export command must handle the two usage cases where the script is either called with:

- `-l num` to provide the command title and list of parameters
- OR-
- with the IRN of an eexports record to perform what the script is required to do

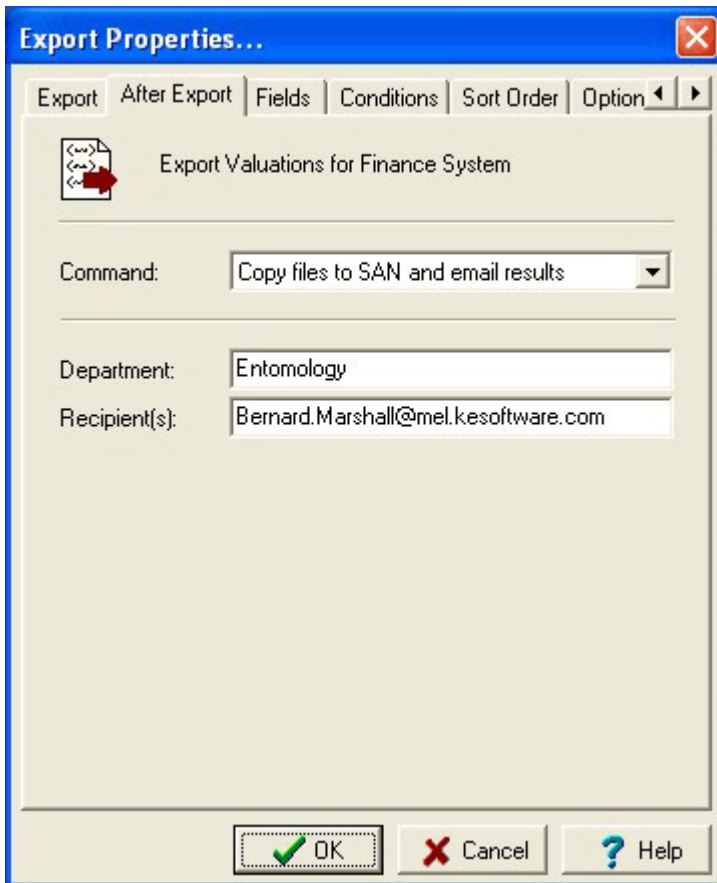
In order to make script writing easier, a perl module `KE::Export` is provided that encompasses most of the functionality needed by an After Export script. It is recommended that you use the module to cut down development time. For a complete description of the functionality provided by the `KE::Export` module see *KE::Export usage*.

To demonstrate how a new command could be put together we'll write a script that copies the output files from an export into a location on a SAN drive (mounted as `/exports`) based on the date of the export and a user specified department and then send an email to a user supplied address.

The first part of the script involves setting up the title and parameters. Two parameters are required, the first is the department under which to file the export files and the second is the email addresses to notify once the files have been copied:

```
#
# Parameters for copy and email notification
#
my $prompts =
{
    0      => [
        "Copy files to SAN and email results",
        "Department:",
        "Recipient(s):"
    ]
};
```

The first string is the title to show in the *Command* drop list, and the following two parameters allow the department and email addresses to be entered. The resulting After Export tab is:



Next check to make sure the server supports both the copying of files and emailing of messages. If support for either is not provided, the command should be hidden in the *Command* drop list. An exit status of 1 indicates the command is not supported:

```
#
# Check whether email and copy are supported on this machine.
#
my $copy = KE::Export::Copy->new();
my $email = KE::Export::Email->new();
if (! $email->IsSupported() || ! $copy->IsSupported())
{
    exit 1;
}
```

Once we have confirmed the required functionality is supported, we check that the supplied arguments are valid. If they are not, exit with an error status of 1:

```
#
# Parse the arguments.
#
my $export = KE::Export->new();
if (! $export->ParseArgs(\@ARGV))
{
    exit 1;
}
```

Since the supplied arguments are acceptable, look for the `-1num` case. If the arguments match, the prompts defined above are printed out and we exit with a successful status of 0:

```
#
# List parameters if required.
#
if ($export->ListParameters($prompts))
{
    exit 0;
}
```

If we get to here, we must process the `eexports` record whose IRN was supplied as the argument. First build up the full path to the folder in which we want to store the export files. We use the `FileRunDate` column on the `eexports` record to get the date on which the export was run, and the first entry in the `Parameters_tab` column list to get the department entered by the user. Once we have the destination path, make sure the folder exists (using `mkdir`). If the folder cannot be created, exit with an error status of 1:

```
#
# Build up the destination directory and make sure it exists.
#
my $date = $export->GetData('FileRunDate');
my $department = $export->GetData('Parameters_tab')->[0];
my $destination = "/tmp/$department/$date";
if (system("mkdir -p '$destination'") != 0)
{
    exit 1;
}
```

It is now time to copy the export files to the destination folder. We use a `KE::Export::Copy` object to perform the transfer. The `FileName_tab` column contains a list of all the files created by the export:

```
#
# Do the copy with the required arguments.
#
my $status = $copy->Execute
(
    destination => $destination,
    filelist    => $export->GetData('FileName_tab')
);
```

Now build up the body of the email message to send to the recipients. Include the name of the export, the date on which it ran, the folder into which the export files were copied and indicate whether the transfer was successful:

```
#  
# Build up the email message to send  
#  
my $body = "The files from export \" .  
           $export->GetData('ScheduleRef:eschedule:Name') .  
           "\", run on $date have been copied to $destination.\n" .  
           "The copy " . $status ? "failed" : "succeeded.\n";
```

Finally, send the email message to all the recipients. Use the second parameter to extract the email addresses of the recipients. The email's subject is the name of the scheduled export:

```
#  
# Do the email with the required arguments.  
#  
my $status = $email->Execute  
(  
    recipients => $export->GetData('Parameters_tab')->[1],  
    subject    => $export->GetData('ScheduleRef:eschedule:Name'),  
    body       => $body  
);
```

Return the status of the email object, indicating whether the emails were sent correctly or not:

```
#  
# Send back the error status.  
#  
exit $status;
```

The complete script is:

```
#!/usr/bin/perl

#
# Copyright (c) 1998-2011 KE Software Pty Ltd
#

use strict;
use KE::Export;

#
# Parameters for copy and email notification
#
my $prompts =
{
    0 => [
        "Copy files to SAN and email results",
        "Department:",
        "Recipient(s):"
    ]
};

#
# Check whether email and copy are supported on this machine.
#
my $copy = KE::Export::Copy->new();
my $email = KE::Export::Email->new();
if (! $email->IsSupported() || ! $copy->IsSupported())
{
    exit 1;
}

#
# Parse the arguments.
#
my $export = KE::Export->new();
if (! $export->ParseArgs(\@ARGV))
{
    exit 1;
}

#
# List parameters if required.
#
if ($export->ListParameters($prompts))
{
    exit 0;
}

#
# Build up the destination directory and make sure it exists.
#
my $date = $export->GetData('FileRunDate');
my $department = $export->GetData('Parameters_tab')->[0];
my $destination = "/tmp/$department/$date";
```

```

if (system("mkdir -p '$destination'") != 0)
{
    exit 1;
}

#
# Do the copy with the required arguments.
#
my $status = $copy->Execute
(
    destination    =>    $destination,
    filelist       =>    $export->GetData('FileName_tab')
);

#
# Build up the email message to send.
#
my $body = "The files from export \"\" .
           $export->GetData('ScheduleRef:eschedule:Name') .
           "\", run on $date have been copied to $destination.\n" .
           "The copy " . ($status ? "failed" : "succeeded") . ".\n";

#
# Do the email with the required arguments.
#
my $status = $email->Execute
(
    recipients     =>    $export->GetData('Parameters tab')->[1],
    subject        =>    $export->GetData('ScheduleRef:eschedule:Name'),
    body           =>    $body
);

#
# Send back the error status.
#
exit $status;

```

The script should be placed in the directory `local/etc/exports/after` as it is a customized script. The file name of the script is not important, but something like `copyemail` would be appropriate. Remember to change the file permissions so it can be executed (i.e. `chmod 755 copyemail`). Your script is now ready for use.

KE::Export usage

The `KE::Export` perl package provides a number of very useful objects that simplify the process of creating an After Export script. The package file is located in `utils/KE/Export.pm` and is documented fully. To view the documentation use `pod2text Export.pm` (assuming you are in the `utils/KE` directory). For your convenience the documentation is reproduced here:

NAME

`KE::Export` - A set of objects usable by After Export scripts

SYNOPSIS

```
use KE::Export;
my $prompts =
{
    0    => [
        'Copy to another folder (CP)',
        'Folder:'
    ]
};

my $copy = KE::Export::Copy->new();
if (! $copy->IsSupported())
{
    exit 1;
}

my $export = KE::Export->new();
if (! $export->ParseArgs(\@ARGV))
{
    exit 1;
}
if ($export->ListParameters($prompts))
{
    exit 0;
}

my $status = $copy->Execute
(
    destination => $export->GetData('Parameters_tab')->[0],
    filelist    => $export->GetData('FileName_tab')
);

exit $status;
```

DESCRIPTION

The "`KE::Export`" module provides a set of objects to make the implementation of After Export commands easier. An After Export command may be registered with a scheduled export through the "After

Export" tab in the Export Properties dialogue box. If an After Export command is registered with a scheduled export, the command is executed once the export phase is complete.

The After Export command provides a mechanism for dealing with the exported data after it is generated. In particular the command may:

- * Email the results of the export to a list of users.
- * Email the export files to a list of users.
- * Copy the export files onto another machine.
- * Send an SMS to a list of telephone numbers.

An After Export command corresponds to a script located in one of the following directories on the server machine:

```
local/etc/exports/I<table>/after
local/etc/exports/after
etc/exports/I<table>/after
etc/exports/after
```

The directories are examined in the order specified above to locate the required script. Using this mechanism it is possible to override a script provided with the system with a custom built one. To do so just add your custom script, with the same name as the script you are overriding, to a directory listed above the directory in which the system script is located. For example, if you want to override the system "ftp" script stored in etc/export/after/ftp, you would place your script in the file local/etc/export/after/ftp.

Each After Export script may be invoked in two ways. These are:

```
"script -l*num*"
```

where *num* is the language number to use when outputting the parameters. This form of the script is expected to print out the *title* of the script to use in the drop list on the "After Export" tab in the client and a list of required parameter prompts, one per line. For example, the "ftp" script invoked by "ftp -l0" (to print out the title and parameters in English) produces:

```
Transfer to another host (FTP)
```

```
Host:
```

```
User:
```

```
Password:
```

where the first line is displayed in the "Command:" drop list on the "After Export" properties tab and the remaining lines are shown as input boxes below the "Command:" drop list with the text used as the prompt. The user must specify each of the parameters before a valid After Export command may be saved.

The language number supplied via the "-l" option determines the language in which the output should appear. The registered language numbers are:



- 0 - English
- 1 - French
- 2 - English (American)
- 3 - Spanish
- 4 - German
- 5 - Italian
- 6 - Dutch
- 7 - Danish
- 8 - Polish
- 9 - Norwegian
- 10 - Swedish
- 11 - Greek
- 12 - Arabic
- 13 - Hebrew
- 14 - French (Canadian)
- 15 - Finnish

"script *exportirn*"

The second way of invoking a script is to supply the irn (Internal Record Number) of the record in the "eexports" table on which the script is to operate. In this case the script needs to perform what is deemed its duty. For example, in the case of the "ftp" script, the export files produced will be FTPed to another host, The username, password and destination on the remote host are used to make the transfer.

The normal life cycle of a "KE::Export" object is:

- 1 Create the object (via "new()").
- 2 Parse any script parameters to determine which of the two uses of the script is appropriate (via "ParseArgs()").
- 3 Output the title and parameters if the script was invoked with the first usage (via "ListParameters()").
- 4 Extract the parameters and execute the required functionality if the script was invoked with the second usage (via "GetData()").

For examples of how to use the "KE::Export" module please review the After Export scripts installed on the server machine with the default installation.

KE::Export

A "KE::Export" object provides a wrapper around a set of utility functions. These functions are designed to simplify the process of writing After Export scripts by encapsulating standard functionality.

In particular, the following facilities are offered:

- * A standard way of parsing the script's arguments to determine which type of invocation was used. See `ParseArgs()`.
- * A standard mechanism for outputting the script parameters when invoked with the `-l` option. See `ListParameters()`.
- * A mechanism for determining the languages supported by the server. See `Languages()`.
- * A means of extracting data from the underlying batch record and its associated schedule record. In fact, any links from the batch record may be followed to retrieve data from other modules. See `GetData()`.

Each After Export script should use a `"KE::Export"` object to simplify access to the underlying export record.

Methods

`new()`

```
$export = KE::Batch->new();
```

Creates an object that provides access to the underlying export record. The object also provides access to helper functions that simplify After Export scripts. In order to release all resources associated with a `"KE::Export"` object (for example, a connection to a server) `"undef"` should be assigned to the object variable once the object is no longer required.

`GetData($colname)`

```
$data = $export->GetData('StartDate');  
$host = $export->GetData('Parameters_tab')->[0];  
$filelist = $export->GetData('FileList_tab');  
$name = $export->GetData('ScheduleRef:eschedule:Name');
```

Retrieves the data for the given column. The `$colname` argument may be the name of any column in the `"eexports"` table. The value returned is consistent with the kind of data stored in the column. An atomic column returns a string, a table returns a reference to a list of strings and a nested table returns a reference to a list where each element is itself a list of strings.

It is also possible to access columns in other modules by specifying the name of the link field in `"eexports"` followed by the table name and column in the linked table. Each component is separated by a colon. There is no limit to the number of components in the column name for linked fields. For example, the column name `"ScheduleRef:eschedule:Name"` uses the link from `"eexports"` to `"eschedule"` (via column `ScheduleRef`) to access the `Name` field in the `"eschedule"` table. In other words, the name of the scheduled export is retrieved.

The `"Parameters_tab"` column provides access to the command parameters entered for the After Export command. The `"FileList_tab"` column provides a list of all the files generated by the export process.



Languages ()

```
$langlist = $export->Languages ();
```

Retrieves the list of languages supported by the server. The list consists of a string of semi-colon separated language numbers. For example, the string "0;1" indicates the server supports English and French, where English is the primary language. The "Languages()" call is used by After Export commands where text is generated as part of the script. The text must be output in languages supported by the server.

The "Languages()" function returns the value of the "System|Setting|Language|Supported" Registry entry.

ListParameters(\$prompts)

```
my $prompts =
{
    0 => [
        "Email export results (SMTP)",
        "Recipient(s):"
    ]
};

if ($export->ListParameters($prompts))
{
    exit 0;
}
```

Prints out the title and parameters for the After Export command. If the "ParseArgs()" call determined the list of parameters should be printed (via the "-lnum" option), then the title and parameters for the given language number are printed and the call returns 1. If the "-lnum" option was not specified, the call returns 0.

The \$prompts argument is a reference to a hash, where the key is the language number and the value is a reference to a list of strings. The first string is the title and subsequent strings are parameters. The title string is shown in the Command drop list, and the parameters strings are shown below the Command drop list with a corresponding data entry field where values may be specified.

ParseArgs (\@ARGV)

```
if (! ParseArgs (\@ARGV))
{
    exit 1;
}
```

Parses the on-line arguments determining whether the parameters are to be printed (as "-lnum" was supplied) or the command executed (the eexports irn was supplied).

If invalid options are found, a usage message is printed and 0 is returned. If the arguments are correct, 1 is returned.

KE::Export::Command

The "KE::Export::Command" class is the base class for all After Export commands. It consists of two methods each After Export command must implement. The first is "IsSupported()" which returns 1 if the After Export command is supported by the server. Some commands may require special software or certain perl modules to be installed before they can be used. The second method is "Execute()", which performs the command itself.

This class should not be called directly from within After Export scripts, rather a sub-class should be created and the IsSupported() and Execute() methods overridden.

Methods

new()

```
$export = KE::Batch::Command->new();  
$export = KE::Batch::Command->new(debug => 1);
```

Creates an object used to execute an After Export command. The "new()" method should not be called directly, rather sub-classed versions should be used. Debugging may be enabled by setting the named argument "debug" to a non-zero value.

Execute()

```
$status = $command->Execute()
```

Executes the After Export command. The "Execute()" method implements the functionality required by the After Export command. For example, if the command is to email the resulting export files to a user, the method must perform the actual emailing and attaching of export files.

The method should return 0 if the command was successful, otherwise 1 should be returned.

Each sub-class must override this method to implement the functionality specific to the sub-class's command.

IsSupported()

```
$support = $command->IsSupported();
```

Determines whether the After Export command is supported by the server. An After Export command may have dependencies on a number of programs or perl modules. The "IsSupported()" method checks each dependency is available, and if so returns 1, otherwise 0. A return value of 0, removes the After Export command from the Command drop list in the client.

KE::Export::Sftp

The "KE::Export::Sftp" class allows secure file transfer (SFTP) to be used to copy the export files to another machine. The "sftp" functionality provided by the "scp" command set is used for the file transfers. The file is encrypted during the copy ensuring privacy of data.



Methods

```
Execute()
    $sftp = KE::Export::Sftp->new();
    $status = $sftp->Execute
    (
        host      =>  'other.machine',
        user      =>  'username',
        password  =>  'passwd',
        destination =>  '/exports/nightly/',
        filelist  =>  $export->GetData('FileList_tab')
    );
```

Performs a secure copy of the export files generated to another host. A number of named arguments are available, all are mandatory:

host

The host name of the machine onto which the export files are to be copied.

user

The user name to use to log in to the remote machine.

password

The password to use to log in to the remote machine.

destination

The directory in which the export files are to be placed. The directory must exist.

filelist

A reference to a list containing the files to be transferred to the remote machine.

"Execute()" returns 0 if the transfers succeeded, otherwise 1 is returned. If an error occurs, it is written to stdout.

IsSupported()

```
$sftp = KE::Export::Sftp->new();
$status = $sftp->IsSupported();
```

Indicates whether the server has the necessary dependencies installed to provide secure file transfer. A return value of 0 implies the server does not support secure file transfer, while a value of 1 implies it does.

KE::Export::Ftp

The "KE::Export::Ftp" class allows file transfer (FTP) to be used to copy the export files to another machine. The data transferred is not encrypted while in transit. As the password is sent to the server without any encryption, that is as clear text, "KE::Export::Ftp" should be used only within internal networks, behind a secure firewall. FTP may offer superior throughput to SFTP.

Methods

```
Execute()  
    $ftp = KE::Export::Ftp->new();  
    $status = $ftp->Execute  
    (  
        host          => 'other.machine',  
        user          => 'username',  
        password      => 'passwd',  
        destination  => '/exports/nightly/',  
        filelist      => $export->GetData('FileList_tab')  
    );
```

Performs a copy of the export files generated to another host. A number of named arguments are available, all are mandatory:

host

The host name of the machine onto which the export files are to be copied.

user

The user name to use to log in to the remote machine.

password

The password to use to log in to the remote machine.

destination

The directory in which the export files are to be placed. The directory must exist.

filelist

A reference to a list containing the files to be transferred to the remote machine.

"Execute()" returns 0 if the transfers succeeded, otherwise 1 is returned. If an error occurs, it is written to stdout.

IsSupported()

```
$ftp = KE::Export::Ftp->new();  
$status = $ftp->IsSupported();
```

Indicates whether the server has the necessary dependencies installed to provide file transfer support. A return value of 0 implies the server does not support file transfer, while a value of 1 implies it does.

KE::Export::Scp

The "KE::Export::Scp" class allows secure copy (SCP) to be used to transfer the export files to another machine. The data transferred is encrypted while in transit. An SCP connection may be formed by either supplying a password, or not. If a password is not supplied, X509 based certificates may be used removing the need for a password. In general, X509 based connections are preferred as a password does not need to be stored in the command script.



Methods

```
Execute()
  $scp = KE::Export::Scp->new();
  $status = $scp->Execute
  (
    host      =>  'other.machine',
    user      =>  'username',
    password  =>  'passwd',
    destination =>  '/exports/nightly/',
    filelist  =>  $export->GetData('FileList_tab')
  );
```

Performs a copy of the export files generated to another host. A number of named arguments are available, most are mandatory:

host

The host name of the machine onto which the export files are to be copied. A host name must be supplied.

user

The user name to use to log in to the remote machine. A user name must be supplied.

password

The password to use to log in to the remote machine. If a password is not supplied, an X509 certificate based connection is attempted.

destination

The directory in which the export files are to be placed. The directory must exist. A destination must be supplied.

filelist

A reference to a list containing the files to be transferred to the remote machine. A list of files to transfer must be supplied.

"Execute()" returns 0 if the transfers succeeded, otherwise 1 is returned. If an error occurs, it is written to stdout.

IsSupported()

```
$scp = KE::Export::Scp->new();
$status = $scp->IsSupported();
```

Indicates whether the server has the necessary dependencies installed to provide secure file copying. A return value of 0 implies the server does not support file copying, while a value of 1 implies it does.

KE::Export::Copy

The "KE::Export::Copy" class allows the export files to be copied to another location on the same machine. The command may also be used to copy the export files onto a file system mounted on the same machine (e.g. a SAMBA share).

Methods

```
Execute()  
    $copy = KE::Export::Copy->new();  
    $status = $copy->Execute  
    (  
        destination => '/exports/nightly/',  
        filelist     => $export->GetData('FileList_tab')  
    );
```

Performs a copy of the export files generated to another location on the same host. Two named arguments are available, both are mandatory:

destination

The directory in which the export files are to be placed. The directory must exist.

filelist

A reference to a list containing the files to be copied to another location.

"Execute()" returns 0 if the copy succeeded, otherwise 1 is returned. If an error occurs, it is written to stdout.

IsSupported()

```
$copy = KE::Export::Copy->new();  
$status = $copy->IsSupported();
```

Indicates whether the server has the necessary dependencies installed to provide file copying. A return value of 0 implies the server does not support file copying, while a value of 1 implies it does.

KE::Export::Email

The "KE::Export::Email" class allows an email notification to be sent to a list of email addresses when a scheduled export is complete. If the list of export files is provided, the files are sent as an attachment to the notification email, otherwise just the result of the export is emailed.

Methods

```
Execute()  
    $email = KE::Export::Email->new();  
    $status = $email->Execute  
    (  
        recipients => 'user1@abc.com, user2@def.com',  
        filelist    => $export->GetData('FileList_tab')  
    );
```

Emails the results of a scheduled export to a list of users. If the export files are supplied, via the filelist named parameter, the export files are attached to the email. Two named arguments are available, one of which is mandatory:

recipients



A comma separated list of email addresses defining users who should receive the results of the scheduled export. At least one recipient must be supplied.

filelist

A reference to a list containing the files to be attached to the email message. If a filelist is not supplied, only the results are emailed to each user.

"Execute()" returns 0 if the email succeeded, otherwise 1 is returned. If an error occurs, it is written to stdout.

IsSupported()

```
$email = KE::Export::Email->new();  
$status = $email->IsSupported();
```

Indicates whether the server has the necessary dependencies installed to provide email services. A return value of 0 implies the server does not support emailing, while a value of 1 implies it does.

Index

C

Creating an After Export script • 10, 13

D

Developing an After Export script • 7

K

KE
Export usage • 19

O

Overview • 1

S

Setting an After Export Command • 3

T

The After Export script • 7