

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 synchronisation 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.

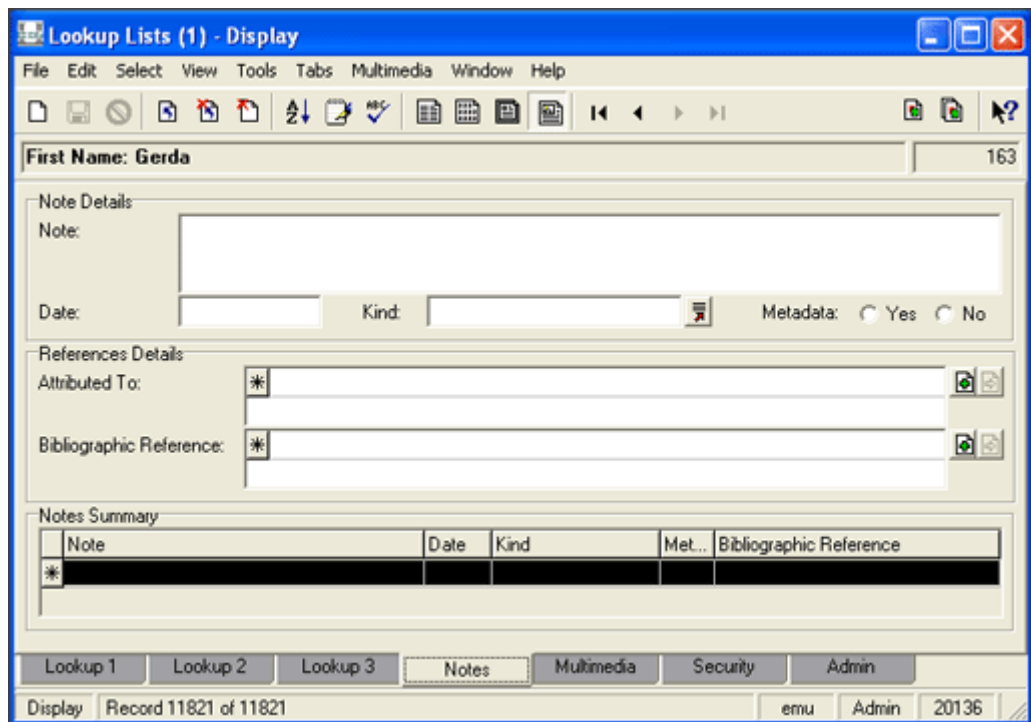
The Lookup List Server, `lutserver`, provides the functionality required to keep the Lookup List table in sync with values used within the EMu System. The addition of the server removes the need to rebuild the Lookup List tables on a nightly (or otherwise) basis.

Lookup List module

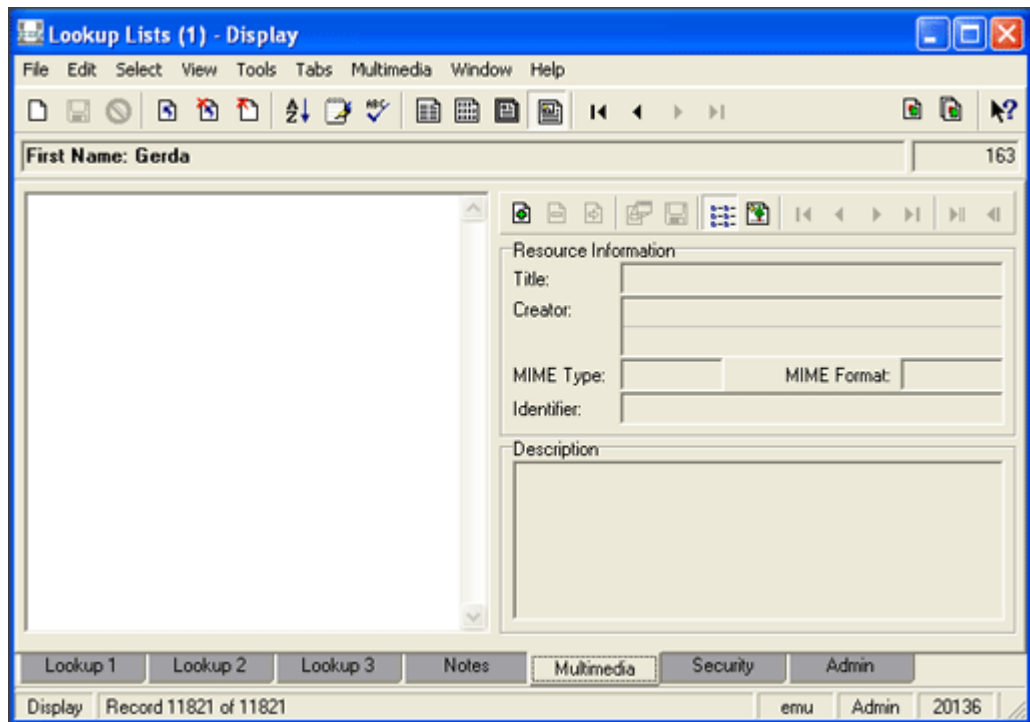
Until the release of the new Lookup List maintenance facilities in EMu 4.1, users could not interact with the Lookup List module and expect their changes to be preserved. The module itself was simplified to the point where only Lookup List specific information was stored. Fields available in all other 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:

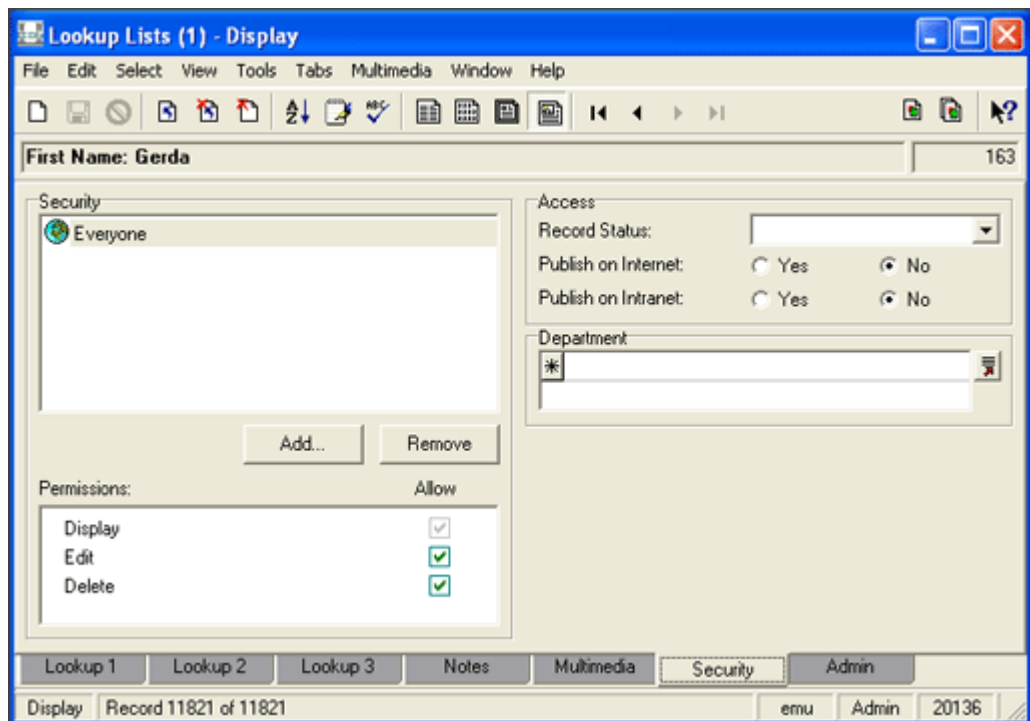
Notes



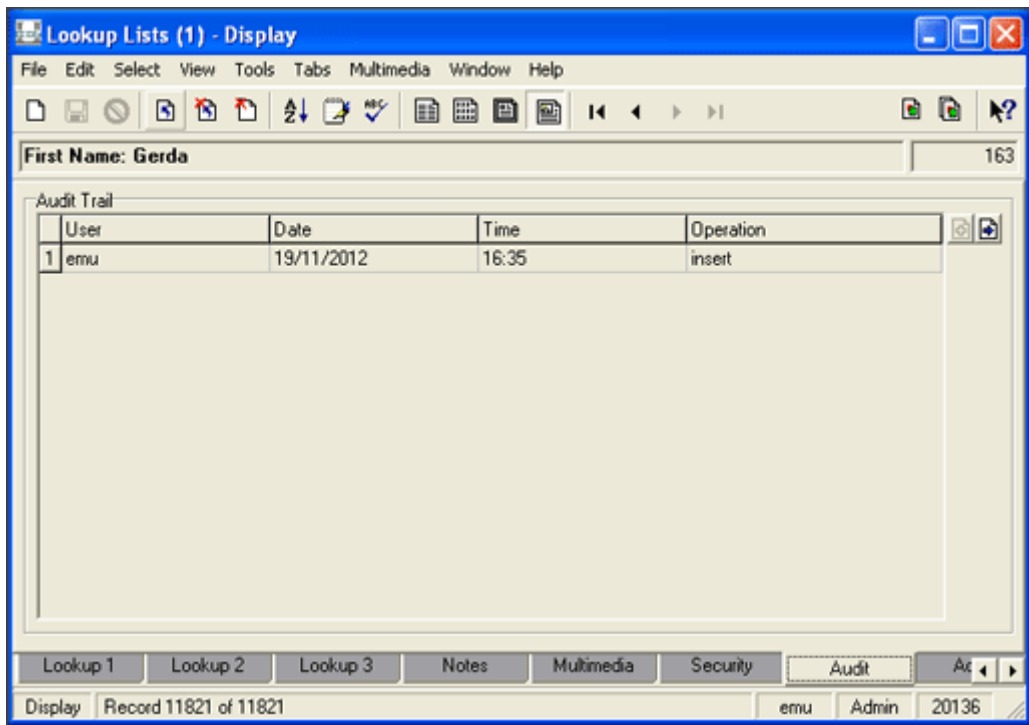
Multimedia



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 synchronised • 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	12
	KE::Export usage	17
	Index	27

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 dialogue 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 dialogue:

- 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.

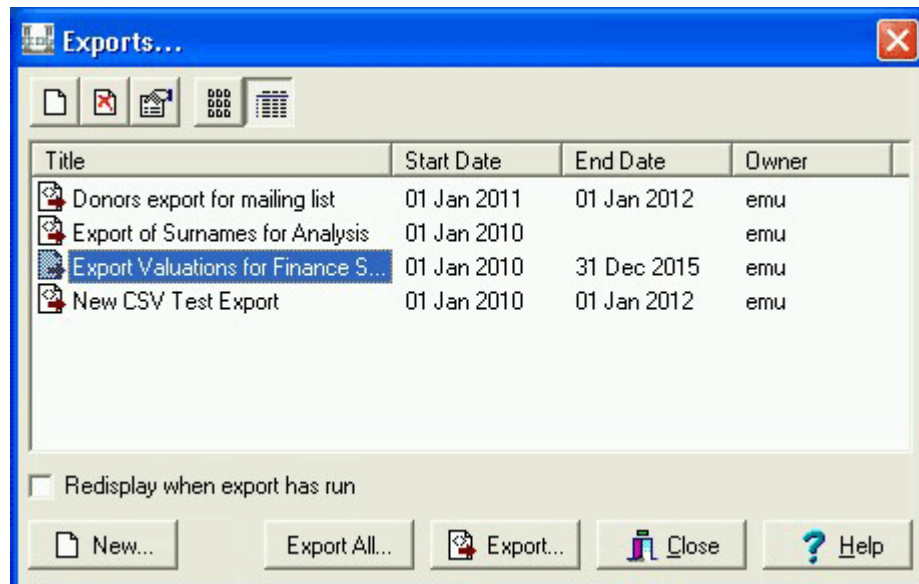
In EMu:

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

-OR-

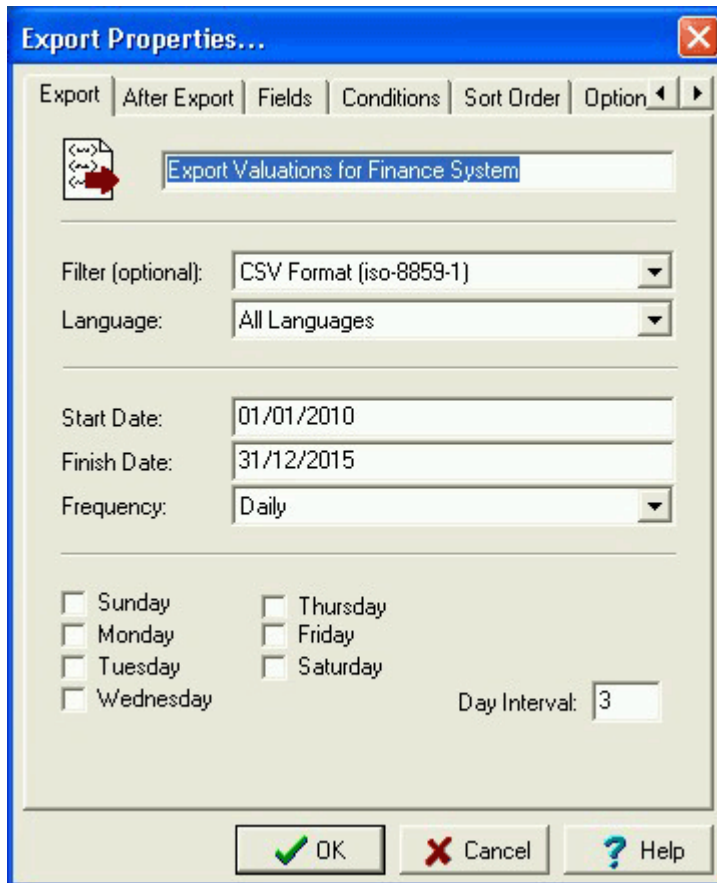
Use the keyboard shortcut, ALT+T+X.

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

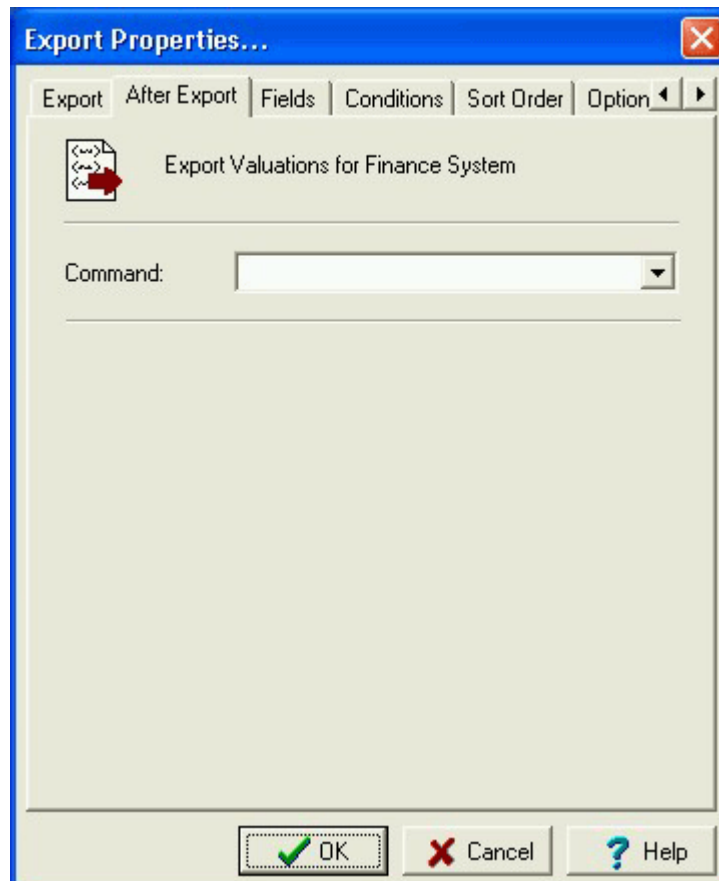


4. Select a scheduled export and click .

The Export Properties dialogue displays:

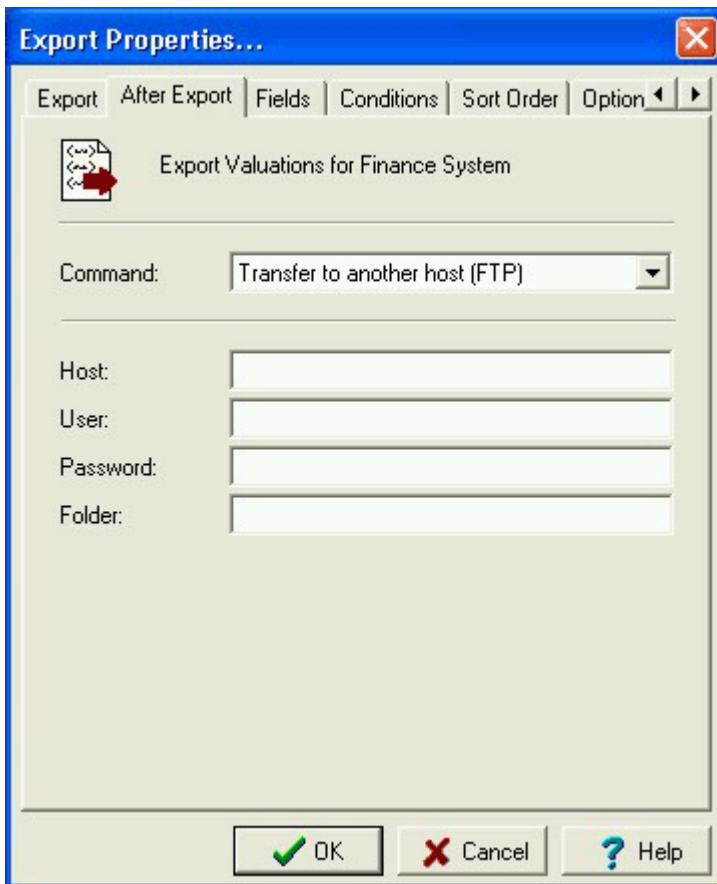


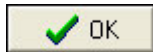
5. Select the **After Export** tab:

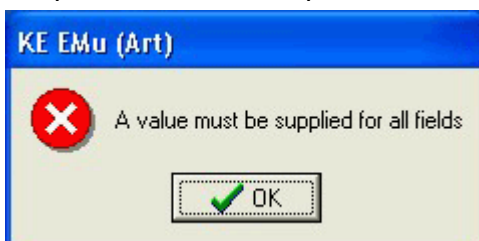


6. From the *Command* drop list, select the command to be executed.

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

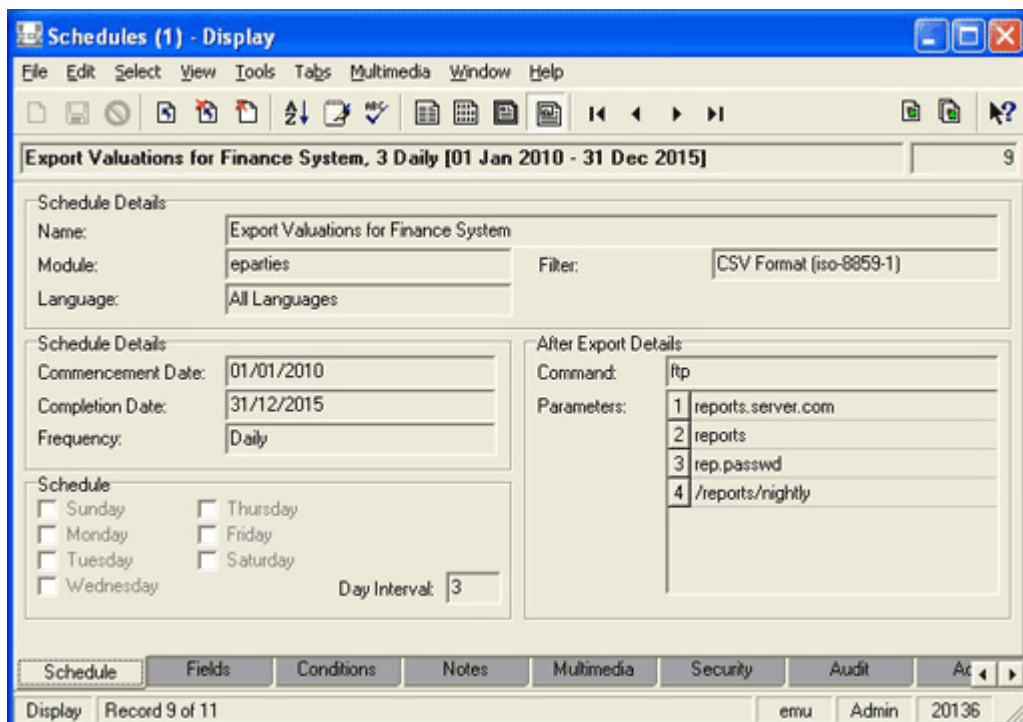
The Schedules module holds a record for each scheduled export defined in any module.

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 customised 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 dialogue. 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



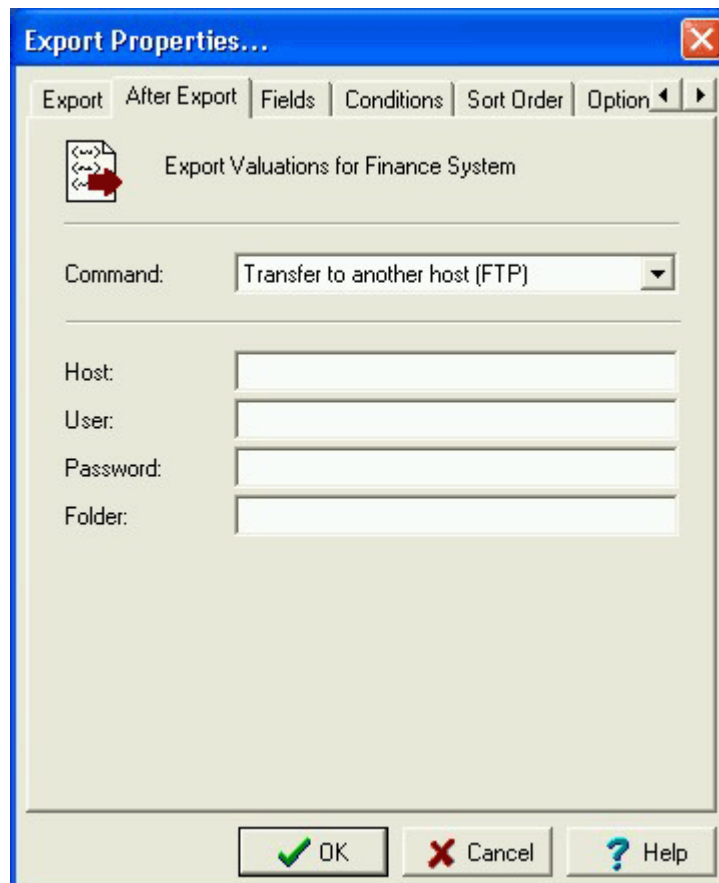


The `l` in `script -l num` is a lowercase L.

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.

- 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 12).

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 `-lnum` 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

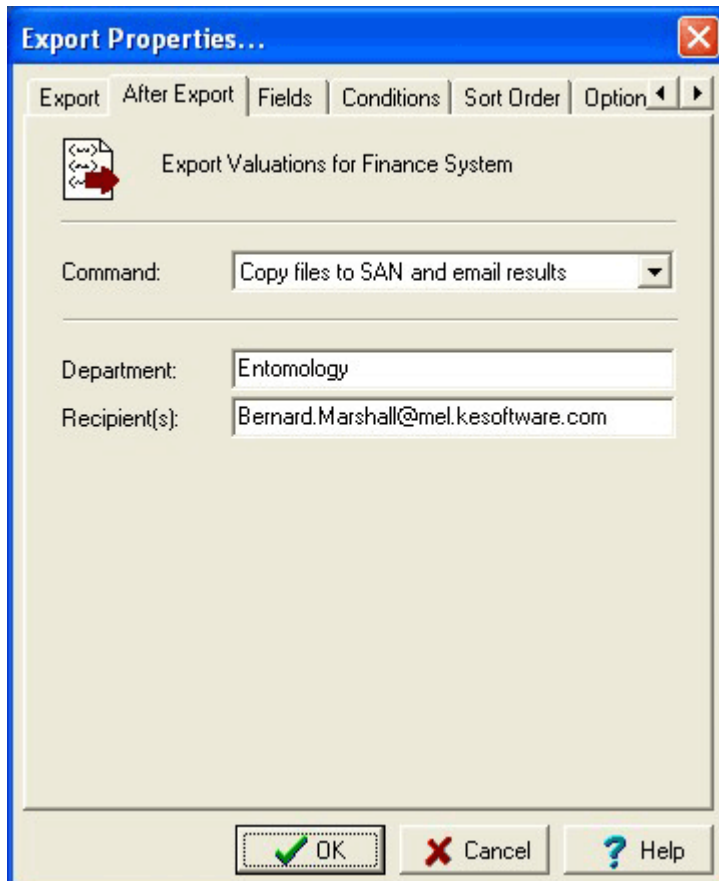
In order to make script writing easier, a perl module `KE::Export` is provided that encompasses most of the functionality needed by an After Export script. It is recommended that you use the module to cut down development time. For a complete description of the functionality provided by the `KE::Export` 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 `-lnum` 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 customised 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 • 9, 12

D

Developing an After Export script • 7

K

KE
Export usage • 17

O

Overview • 1

S

Setting an After Export Command • 3

T

The After Export script • 7