Auditing Facility

Overview

During the day to day operation of EMu there may be times when an administrator or user wants details about who changed a record, when it was changed and what changes were made. An auditing facility creates *audit trails*, a sequence of records that describe the operations performed by a user and the time at which the changes were made. These records generally include the:

- operation performed (e.g. insertion, update, etc.)
- module affected (e.g. eparties)
- name of the user
- date and time
- IRN of the record affected
- data specific to the operation performed (e.g. fields changed for an update)
- database sub-system that performed the operation

EMu has provided simple auditing facilities since it was first released. In particular, every record in EMu includes the name of the person who last changed it, along with the date and time of the modification. Similar information is stored about the creation of the record. These details are located on the Admin tab of each module (usually the last tab in a module).

The EMu database manager also provides simple audit trails on a per module basis. These audit trails can be accessed via the `texaudit` command. In general, this level of auditing has not been used as it requires manual configuration and the level of information produced is restricted to operations performed rather than to data specific changes (i.e. what fields changed values when this record was updated).

EMu 3.2.04 sees the introduction of a fully integrated auditing facility. The facility includes a new module, Audit Trails (eaudit), which contains complete audit trails for all operations registered for monitoring. There are five levels for which audit records may be generated:

- **change** (insertions, updates, deletions)
- **search** (queries)
- **display** (viewed, sorted, reported)
- **login** (first access of a module in current session)
- **all** (all database operations)

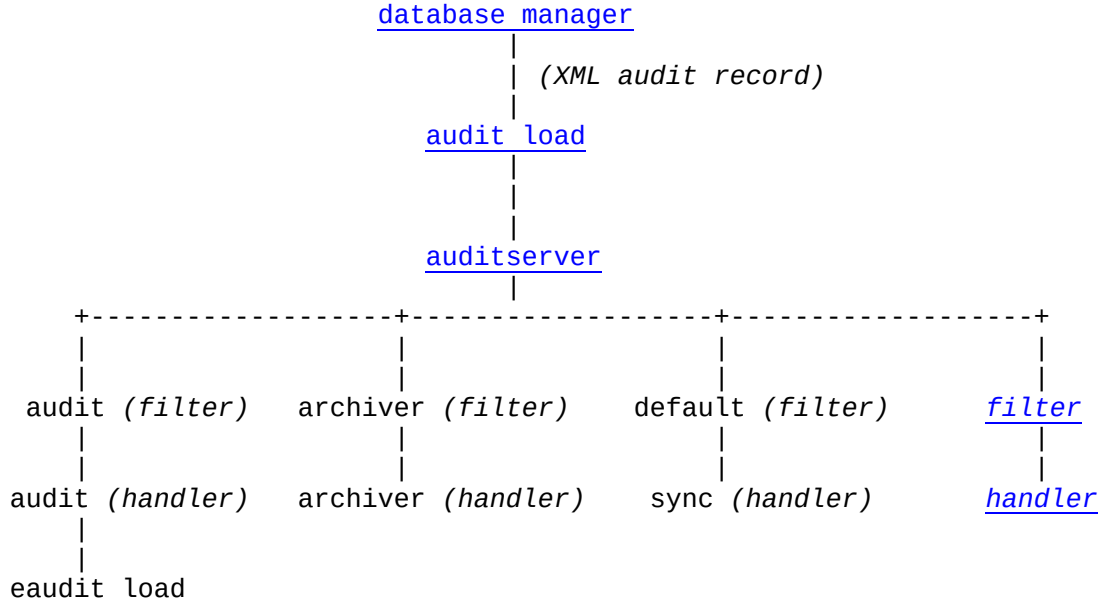
Each of these may be set on a per module basis, which means that it is possible to monitor all changes to records in the Parties module, for instance, while Multimedia records may be audited for changes, searches and records viewed.

Every EMu module now contains an Audit tab that lists all audit trail records for the current record. Each record details the operation performed, who performed it and the date on which it was performed. From this tab it is also possible to view the complete audit trail record in the Audit Trails module.

The new facility is extensible, which allows other services that require access to detailed information about record changes to take advantage of its framework. In particular two other services use the audit framework to monitor record modifications. The first is the **archiver** service, which allows raw XML audit trail records to be stored in a file. The second is the **sync** service, which is used to ensure that dependent records are updated when the master record is changed.

The Auditing system

The Auditing system comprises several components that together provide the services necessary to record audit trails as well as a data stream of record changes that other services use to monitor data. The diagram below illustrates the components that make up the Auditing system:



database manager

The EMu database manager carries out all operations on the underlying database tables. Each table can have a set of auditing options enabled that detail what operations should produce audit records. These options are defined in the **opts** file located in the database directory. For example, the Parties module **opts** file can be found at **data/eparties/opts**. As an example, if record insertions, updates and deletions were audited, the **opts** file would be similar to:

```
xmlaudit=on
xmlauditpath=/home/ke/emu/artdemo/loads/audit
xmlauditoptions=update;updatehistory;insert;delete;tempinsert;
tempdelete;tempupdate;tempmove
```

The `xmlaudit` option simply enables auditing for the table. Each audit record is an XML entry that is added to the file defined by the `xmlauditpath` option. The `xmlauditoptions` entry lists all the table operations that should be audited. The list shown audits all record insertions updates and deletions. Administrators do not need to understand the contents of the **opts** file as it is maintained and modified by the `emuaudit` command, which is discussed in the next section.

The XML below was generated when a search for a first name of `Fred` and a surname of `Smith` was performed:

```
<audit>
        <prog>texserver</prog>
        <module>eparties</module>
        <date>2007-11-15 10:28:56</date>
        <user>emu</user>
        <op>query</op>
        <data>
                <ident>#1195082936.198211</ident>
                <querystr>select all from eparties where true and
((((NamFirst contains &amp;apos;Fred&amp;apos;)))) and ((((NamLast contains
&amp;apos;Smith&amp;apos;))))</querystr>
                <matchcount>0</matchcount>
        </data>
</audit>
```

The `<data>` element contains information that is specific to the operation type. In the case of a query, this is the query statement executed, the number of matches found and an identifier for this query that can be used to match up audit records produced when a matching record is viewed.

All tables have the same `xmlauditpath` entry, ensuring that all XML audit records are placed in the same file. The file used is **loads/audit/audit.xml**.

audit load

A new background load has been added to handle XML audit information placed in **loads/audit/audit.xml**. The load is called **audit** and is controlled via the `emuload` command. When the **audit** load is started it executes the `auditserver` command, which manages the contents of the XML audit file **loads/audit/audit.xml**. Administrators can stop the processing of the **audit.xml** file (although data will continue to accumulate) by executing:

```
emuload stop audit
```

The load can be restarted using:

```
emuload start audit
```

auditserver

When the **audit** background load commences it starts **auditserver** running. The **auditserver** is found at **utils/auditserver**. It is this server that does all the processing of the XML audit records. The server reads one XML audit record at a time and passes the record to a number of *filters*. These filters are located in **etc/audit/filters** (or **local/etc/audit/filters** for localised filters). The filters decide whether a *handler* should do something with the audit record, or whether the record should be ignored. If a filter indicates that the record should be processed, the corresponding *handler* (located in **etc/audit**) is passed the XML audit record to process the record based on its particular function (e.g. produce an audit trail record, archive the record for later examination, etc.).

An example may make the process a little clearer. Consider the audit trail *filter* (**etc/audit/filters/audit.pl**) and *handler* (**etc/audit/audit.pl**). When **auditserver** reads an XML audit record it invokes the `Filter()` function inside the *filter*, which indicates whether the record should be processed. If processing is required, **auditserver** passes the XML audit record to the `Audit()` function inside the *handler*, which builds a data record suitable for loading into the **eaudit** background load.

When **auditserver** starts it scans the **etc/audit** and **local/etc/audit** directory to locate all handlers (files that end with a .pl extension). For each handler it checks whether a filter exists in **etc/audit/filters** (called `Filter()`) and **local/etc/audit/filters** (called `LocalFilter()`). If

a filter with the same name as the handler is found, it is also registered. If a filter is not found, a default filter is used that always indicates the XML audit record is to be processed.

filter

A *filter* is a perl script that is invoked by **auditserver** to determine whether an XML audit record should be passed onto a handler, or whether it should be ignored. The filter consists of a single function called `Filter()`, which returns a non-zero value if the XML audit record is to be ignored.

handler

When an XML audit record is approved for handling, **auditserver** invokes the handler by calling the `Audit()` function. The function uses the XML audit record to perform any processing required. In the case of the **archiver** handler, the XML audit record is written to a file for archiving.

emuaudit

The **emuaudit** command is used to control the types of operations that are audited by the database manager. All auditing is performed by the database manager on the EMu server. As the database manager handles all access methods to the data, audit information is generated regardless of the access method used. For example, if a module has **search** auditing operations enabled, audit records are generated for searches performed via the EMu Windows client, the EMu Web interface or the **texql** command line processor.

The **emuaudit** command is located on the EMu server and can only be accessed by user **emu**, that is the EMu System Administrator account.

To determine the operations audited for a given module, run **emuaudit** followed by the name of the table. For example to check the Loans module (eloans table) settings use:

```
emuaudit eloans
eloans                    change
```

The `change` setting indicates that auditing for the Loans module is enabled for operations that alter records (that is insertions, updates and deletions). To list all modules settings, enter `emuaudit` without supplying a module name.

```
emuaudit
eaccessionlots            change
ebibliography             change
ecatalogue                search,change
econdition                change
econservation             change
edocuments                change
eevents                   change
efieldhelp                change
egroups                   change
einsurance                change
einternal                 change
eloans                    change
elocations                change
emovements                change
emultimedia               change
enarratives               change
eparties                  all
eregistry                 change
erights                   change
etemplate                 change
ethesaurus                change
```

```
evaluations                change
```

Notice that both ecatalogue and eparties are auditing more than just changes made to records. The list of available operations for auditing is:

- `change`
- `search`
- `display`
- `login`
- `all`

These operations are explained below.

change

The `change` operation audits any activity that results in a record being created, modified or deleted. This is the default operation for all modules and is used to track who has made changes to records. It is not possible to disable this operation (although the loading of these records into the Audit Trails module can be turned off if you do not want to track record changes). When a record is modified, a complete list of all fields before and after the change is generated. In the case of an insertion, only the new value is listed, and for a delete the old value is recorded.

The corresponding **xmlauditoptions** for the database manager are:

```
update;updatehistory;insert;delete;tempinsert;tempdelete;tempupdate;
tempmove
```

search

If `search` operations are enabled for auditing, audit records containing the *texql* statement used to perform the search are generated. By monitoring `search` audit records, administrators can determine what fields are being searched and what search terms are used. With this information it is possible to adjust database indexing to provide faster searching (e.g. by enabling NULL searching) or to reduce the indexing overhead by removing indexing from fields that are not searched on a regular basis.

The corresponding **xmlauditoptions** for the database manager are:

```
query
```

display

Using the `display` operation, audit records are created every time a record is accessed. A record is accessed when it is viewed by a user, involved in a sort operation or used as part of a report. The audit record contains an *identifier* that can be used to link the record displayed with the search that found the record.

The corresponding **xmlauditoptions** for the database manager are:

```
display
```

login

When a user accesses a module for the first time the database server must *login* to the underlying table. The *login* process must occur before the module can access the data. If the `login` operation is enabled, audit records are created each time a user first accesses a module within a session. An audit record is also created when the user *logsout* of the module, just before terminating the session. Using this operation it is possible for administrators to calculate the time periods that a user has accessed the system.

The corresponding **xmlauditoptions** for the database manager are:

```
login;logout;badlogin
```

all

The `all` operation may be used to generate audit records for every activity performed on a given module. Not only are all of the above operations audited, but system activities (e.g. module reindexing) are also recorded. An administrator can use the `all` operation to get a complete picture of all usage of a module. This setting should be used with caution as it can generate a large number of audit records.

Audit service

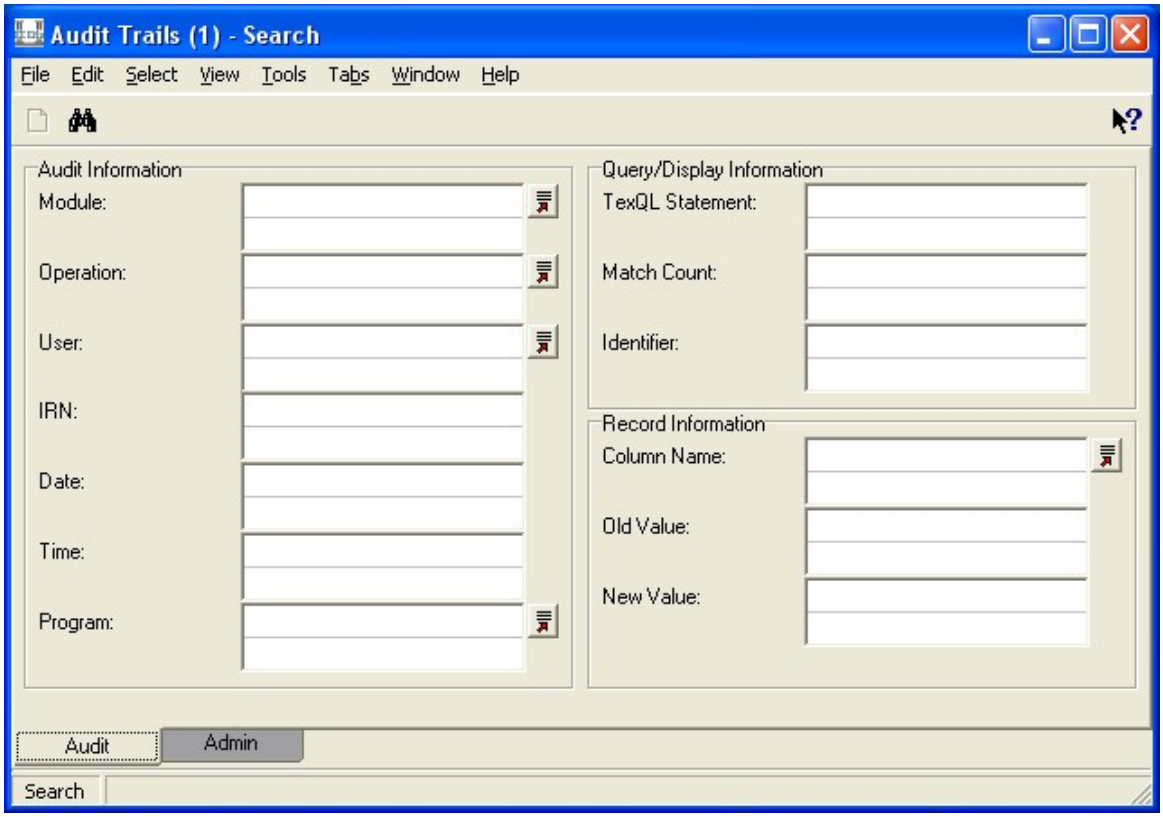
The primary purpose of the new auditing facility is to allow administrators and users to monitor access to records. This monitoring is achieved through the production of audit trail records that detail what operation occurred on what record, by whom and on what date. The audit service is charged with the generation of audit trail records from the XML audit records produced by the database manager.

Audit Trails Module

The Audit Trails module has been added to EMu. The module contains one record per audited operation. With this module, administrators and users can search audit trail records and produce reports detailing activities on a given record or operations performed by a given user. In fact any EMu command may be used to view and manipulate audit trail records. Since auditing information reflects activities that occurred in the past, all records are read-only.

The Audit Trail module is accessed from the Command Centre:

When invoked, the Audit query tab contains all fields that can be queried in audit trail records:



Most of the search fields are self explanatory; however, some may require explanation:

Identifier: (Query/Display Information)

An *Identifier* is a unique string generated for "query" audit trail records. Each record returned by a search and viewed will have an *Identifier* value set to the same value as the query. By searching the *Identifier* field with this value it is possible to obtain a list of all records that were viewed as the result of a particular search.

Column Name: (Record Information)

This field can be used to obtain a list of all the columns that have changed value. For an insertion this is all columns that received a value; for an update it is all columns that changed value; and for a deletion it is all columns that had a value.

TexQL Statement: (Query/Display Information)

The *texql* query statement used to perform searches can be queried via this field. The value is a complete *texql* statement, similar to:

```
select all from eparties where true and ((((NamFirst contains 'jim'))))
```

Using a simple parser it is possible to extract information about which fields are searched and the range of values used for each field.

Old Value/New Value: (Record Information)

A list of all columns changed, followed by the original or new value respectively. The value returned includes the column name changed, followed by an XML description of the data structure and values altered. For example:

```
SummaryData: <atom>Axelrad, Axil</atom>
```

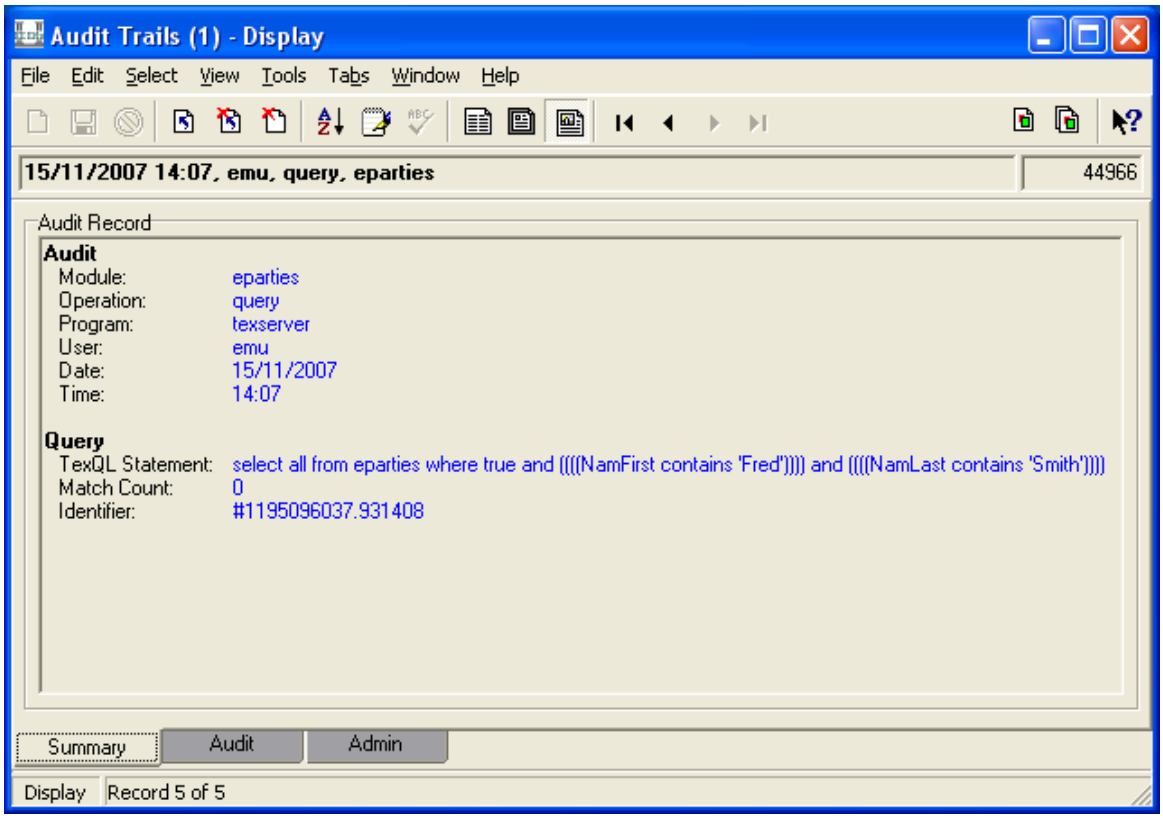
indicates that the *Summary Data* field was changed and the value is `Axelrad, Axil`. The XML description allows the structure of the data to be determined for reporting. When performing a search on these fields it is necessary to enclose the search terms within single

quotes so that they will appear in the same line. For example, to find all audit trail records where the Summary Data was changed to include `Axil`, enter:

```
'SummaryData Axil'
```

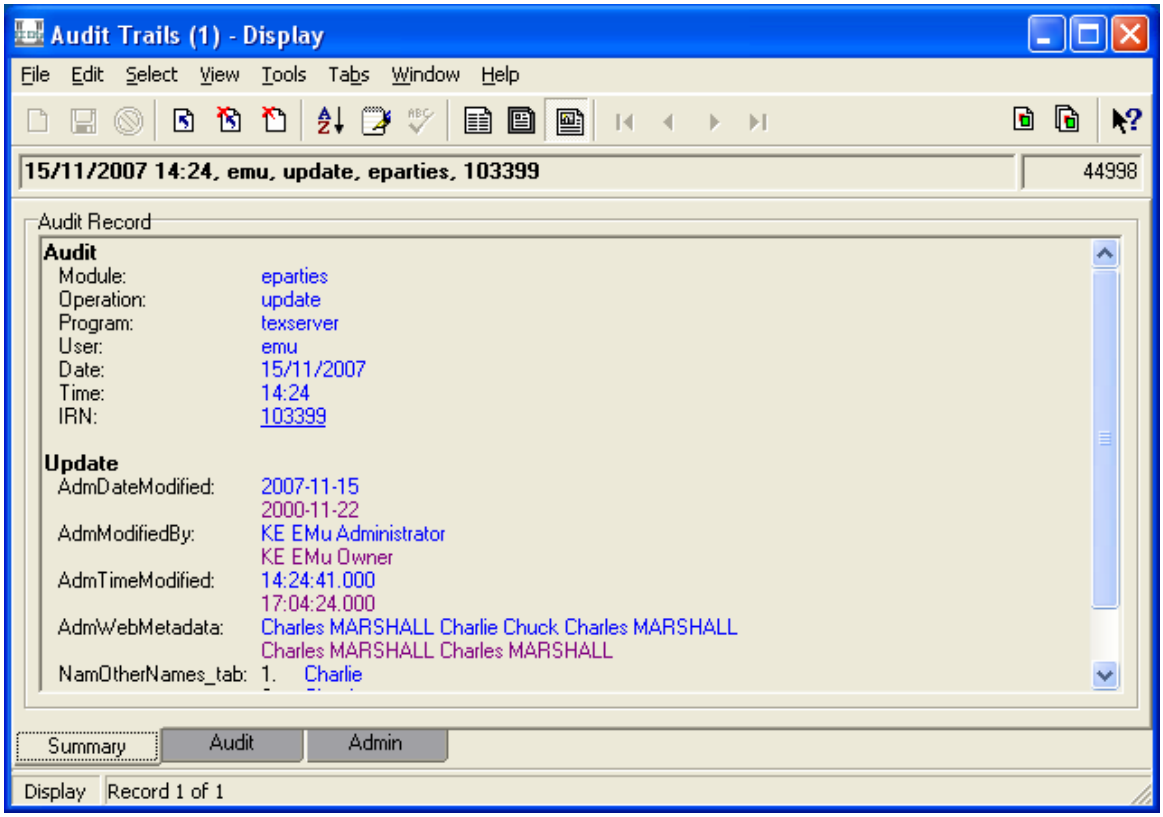
as the search term.

When audit trail records are retrieved they can be viewed using any of the standard display modes in EMu (List, Detail). A Summary tab has been added that presents the audit trail record as a single page display:

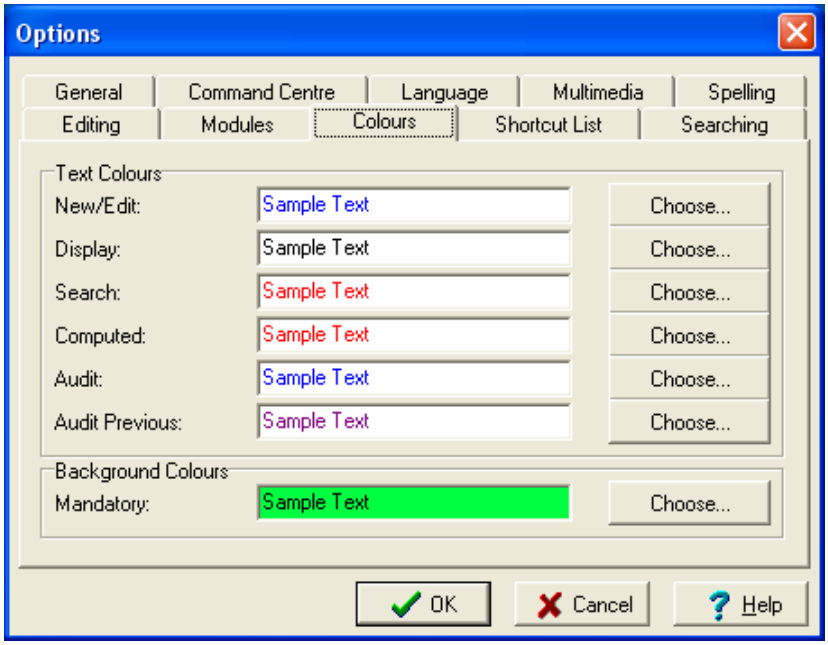


This view consists of two sections. The Audit section contains information that is common to all audit trail records. The following section contains operation specific information. In this example the query performed and the number of matching records.

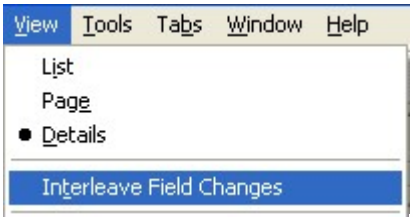
If an audit trail record is for an update operation, this view will contain the new and old values so that users can determine what fields were modified:



The old values are shown below the new values and in purple. It is possible to configure the colours used to display data using the Options dialogue box (**Tools>Options**) and selecting the **Colours** tab. **Audit** colour is used for current data, and **Audit Previous** is used for previous values:



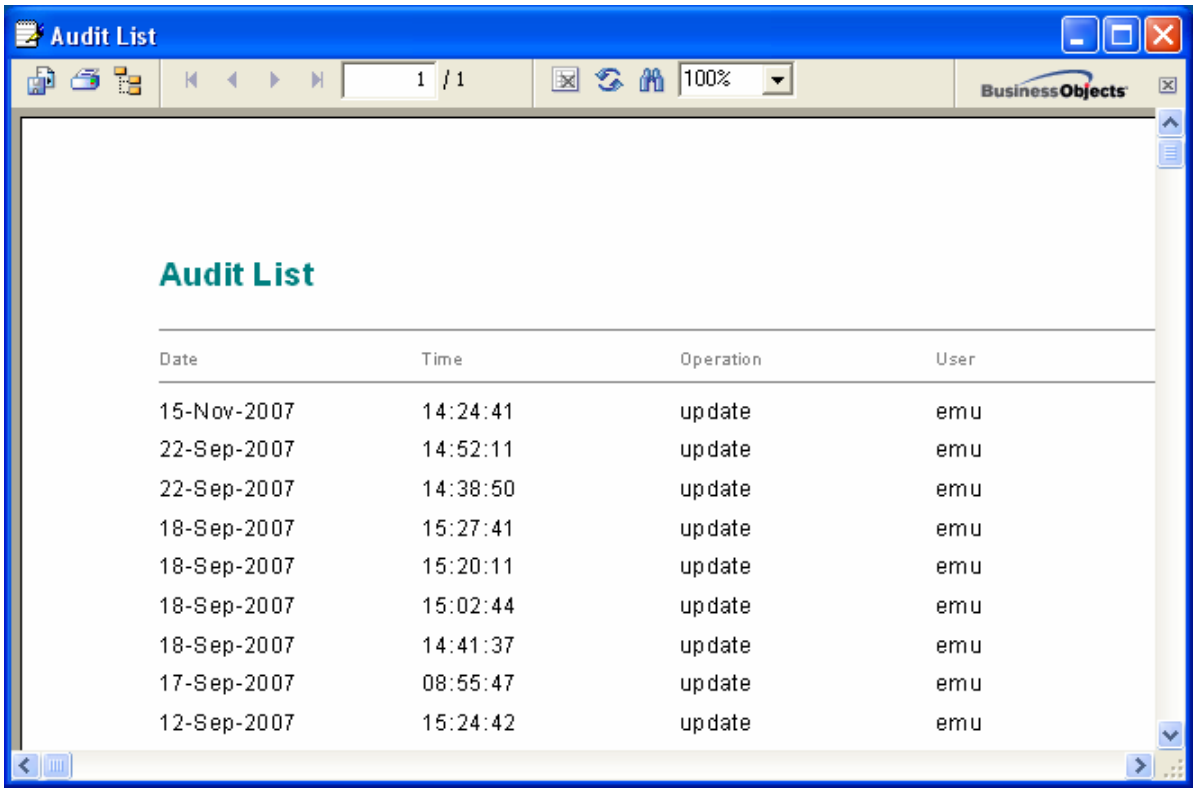
When displaying fields that contain lists of values the differences can be displayed either *interleaved*, that is each line shows the old and new value, or *consecutively*, that is all the new values are shown first, followed by all the old values. The Interleave Field Changes menu option (**View>Interleave Field Changes**) determines which view is used:



While viewing audit trail records it is possible to view the record to which the audit trail applies:

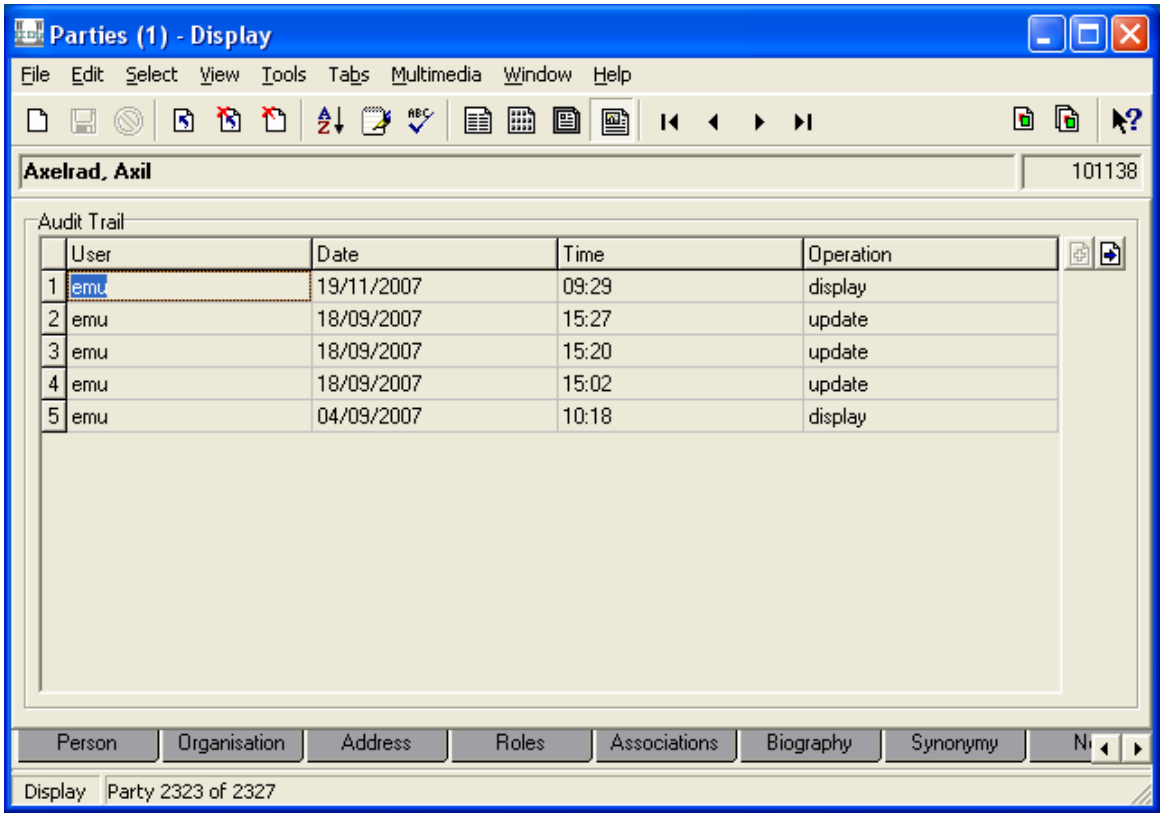
- If viewing the Summary tab, the IRN value will appear as a clickable link (the number will be underlined). If clicked, the audited record will display.
- If viewing the Audit tab, the View Attachments button next to the IRN value may be used. While on the IRN field, the View Attached menu command (**Edit>View Attached>Current Record** and **Edit>View Attached>Selected Records**) may be invoked. In the case of the Selected Records option, only records that are in the same module as the current record will be displayed.

A set of standard reports is available for producing listings and detailed views of audit trail records. An example list report is shown below:



Audit Tab

Every module in EMu has an Audit tab when viewing records in Details view. The tab displays the complete audit trail history for the current record as stored in the Audit Trails module. Using the View Attachment button or the View Attached menu command (**Edit>View Attached>Current Record** and **Edit>View Attached>Selected Records**), the full audit trail record(s) can be viewed in the Audit Trails module. The content of the Audit tab is generated each time the tab is viewed:



Disabling auditing

The new audit service may generate a large number of audit trail records for very active systems. In some instances it may not be necessary to generate audit trail records for all modules.

It is not possible to disable auditing on a module completely using the **emuaudit** command as the system will always produce XML audit records for any record changes. These records are used by other auditing services (e.g. **sync** service) to monitor system changes.

To disable the loading of audit trails records for a module add:

```
AUDIT=no
```

to the **emuoptions** file under the database directory (**data/*tablename***). It may be necessary to create the **emuoptions** file if it does not exist; thus to disable auditing of the Parties module add:

```
AUDIT=no
```

to **data/eparties/emuoptions**. When an entry is added to the **emuoptions** file, the EMu client must be restarted before it will take effect.

It is also possible to disable the loading of all audit trail records by removing the Audit Trails table (eaudit). If EMu cannot find the eaudit table, the **audit** service will ignore any XML audit records generated.

Archiver service

Another service that uses the new auditing facility is the **archiver**. All XML audit records generated are passed to the **archiver** service to determine whether a copy should be placed in a file for archiving. The service allows administrators to keep a backup copy of all XML audit records (which can be used for reference or supplying to other software as input).

To enable the **archiver** service the directory **logs/audit** must exist on the EMu server. By default, the directory does not exist when EMu is installed and must be created by an administrator. When the directory is first created the auditing sub-system must be restarted via:

```
emuload stop audit
emuload start audit
```

All XML audit records for a day are placed in a single file under the **logs/audit** directory. The file name is in the form **yyyy-mm-dd**. Once all the entries for a day have been added, the file is compressed (with gzip) and renamed to **yyyy-mm-dd.gz**. To view a compressed archive file use:

```
gzcat  yyyy-mm-dd.gz
```

The archiver can be disabled by either renaming the **logs/audit** directory or removing it. Once disabled the auditing sub-system must be restarted via:

```
emuload stop audit
emuload start audit
```

Sync service

The **sync** service is used to ensure that all records that extract data from another record are up to date. For example, a Catalogue record may copy the *Full Name* field from the Parties module into the *Creator's Name* field for a work of art. The copy is used to allow high speed searching on the *Creator's Name* field in the Catalogue. If the record in the Parties module is changed so that the value of the *Full Name* field changes, all Catalogue records that copy in the *Full Name* value must be updated to the new value.

A table is maintained automatically in **etc/syncmap** and is used by the **sync** service to determine which fields need to be updated when a field is modified. In the example above the map would contain:

```
eparties    NamFullName    ecatalogue    CreCreatorRef_tab
```

This indicates that when the *NamFullName* field in the eparties table is changed, the *CreCreatorRef_tab* field in ecatalogue table is checked to see if the value needs to be updated.

The **sync** service is implemented as a background load and is controlled by the **emuload** command. In the day to day working of the system the **sync** service must be running to ensure that records are always up to date.

Adding a new service

The new auditing system is a general purpose facility designed to provide a stream of XML audit records to registered services. These services examine the XML audit records and take actions appropriate for the service they provide. In the case of the **archiver** service the XML audit record is copied to a file; the **audit** service creates audit trail records for loading into the Audit Trail module (eaudit); and the **sync** service looks for data that has changed and updates records that copy the modified data.

It is also possible to add client specific services by creating a new handler and installing it in the appropriate location on the EMu server. Client specific handlers are placed in the **local/etc/audit** directory in a file with a .pl extension. The name of the file should reflect the service and the code must be written using perl.

The service file must contain a function called `Audit()`, which is used to process the XML audit record. The function does not return a value. A sample stub is provided below:

```
#!/usr/bin/perl
```

```
#
#  Archive the audit XML into a log file.
#
sub
Audit
{
        my $tree = shift;
        my $columns = shift;
        my $xml = shift;

        #
        #   Process the XML audit record for this service
        #
}
```

The `Audit()` function receives three arguments. The first is a simple tree representation of the XML audit record. A sample listing of the tree for a changed Parties module record looks like:

```
$tree => {
          'date' => {
                      'content' => '2007-11-19 12:47:50'
                    },
          'prog' => {
                      'content' => 'texload'
                    },
          'user' => {
                      'content' => 'emu'
                    },
          'data' => {
                      'atom' => [
                                  {
                                    'name' => 'PlaRbgFeaturesHazards',
                                    'new' => {}
                                  },
                                  {
                                    'name' => 'irn',
                                    'new' => {
                                               'content' => '252'
                                             }
                                  },
                                  {
                                    'name' => 'SummaryData',
                                    'new' => {
                                               'content' =>
'04/10/2007 Excellent (MARSHALL, Dr Charles John)',
                                               'computed' => 'yes'
                                             },
                                    'old' => {
                                               'content' =>
'04/10/2007 Excellent (MARSHALL, Mr Charles James)'
                                             },
                                    'modified' => 'yes'
                                  },
                                  {
                                    'name' => 'ConCatalogueRef',
                                    'new' => {
                                               'content' => '1000029'
                                             }
                                  },
                                . . .
                                ]
                    },
```

```
          'op' => {
                    'content' => 'update'
                  },
          'key' => {
                     'atom' => {
                                  'content' => '252'
                                }
                   },
          'module' => {
                        'content' => 'econdition'
                      }
        };
```

The second argument is a reference to a hash of the column names within the `data` section of the XML audit record. This can be used to look up the value of a column directly. A sample listing looks like:

```
$columns => {
          'SummaryData' => {
                             'name' => 'SummaryData',
                             'new' => {
                                        'content' => '04/10/2007
Excellent (MARSHALL, Dr Charles James)',
                                        'computed' => 'yes'
                                      },
                             'old' => {
                                        'content' => '04/10/2007
Excellent (MARSHALL, Dr Charles John)'
                                      },
                             'modified' => 'yes'
                           },
          'NamFirst' => {
                          'name' => 'NamFirst',
                          'new' => {
                                     'content' => 'Charles'
                                   }
                        },
          'NamMiddle' => {
                           'name' => 'NamMiddle',
                           'new' => {
                                      'content' => 'James'
                                    },
                           'old' => {
                                      'content' => 'John'
                                    },
                           'modified' => 'yes'
                         },
                       .  .  .
        };
```

The third argument is a reference to a list containing the complete XML audit record in text form, with one line per list index:

```
$xml => [
          '<audit>
',
          '     <prog>texload</prog>
',
          '     <module>econdition</module>
',
          '     <key>
',
```

```
            '              <atom>252</atom>
',
            '     </key>
',
            '     <date>2007-11-19 13:23:15</date>
',
            '     <user>emu</user>
',
            '     <op>update</op>
',
            '     <data>
',
',
            '              <atom name="SummaryData" modified="yes">
',
            '                       <old>04/10/2007 Excellent (MARSHALL,
Dr Charles James)</old>
',
            '                       <new computed="yes">04/10/2007
Excellent (MARSHALL, Dr Charles John)</new>
',
            '              </atom>
',
            '              <atom name="NamMiddle" modified="yes">
',
            '                       <old>James</old>
',
            '                       <new>John</new>
',
            '              </atom>
                     . . .
            '     </data>
',
            '</audit>
'
        ];
```

The `Audit()` function must not change any of the supplied arguments as they are not local copies (required for efficiency reasons). Also the handler itself must be reasonably efficient as it is part of a chain of services that is called for every XML audit record generated. The standard services distributed with EMu can be found in **etc/audit**.

Overriding filters

Many of the standard services (e.g. **audit** and **archiver**) separate the code used to perform the service (the handler) from the code used to determine whether the service should be used at all (the filter). The reason for this separation is that administrators may want to override the filter, allowing them to set new criteria for when a service should process a record.

For example, an administrator may only be interested in creating audit trail records for records that have been changed (that is insertions, updates and deletions), however they would not only like to archive changes but searches as well.

In order to provide the correct XML audit records both the `change` and `search` audit levels are required. To generate the correct XML audit records from the database manager, run:

```
emuaudit -o change,search
eaccessionlots            search,change
ebibliography             search,change
ecatalogue                search,change
econdition                search,change
```

```
econservation             search,change
edocuments                search,change
eevents                   search,change
efieldhelp                search,change
egroups                   search,change
einsurance                search,change
einternal                 search,change
eloans                    search,change
elocations                search,change
emovements                search,change
emultimedia               search,change
enarratives               search,change
eparties                  search,change
eregistry                 search,change
erights                   search,change
etemplate                 search,change
ethesaurus                search,change
evaluations               search,change
```

The next step is to write a `LocalFilter()` that determines whether the XML audit record should be processed. In this case we need to check whether the audit operation was "insert", "update" or "delete".

The `LocalFilter()` function must exist in **local/etc/audit/filters/audit.pl** as it is overriding the standard filter found in **etc/audit/filters/audit.pl**. If a standard filter does not exist, a local filter can still be defined by adding a file to **local/etc/audit/filters** where the filename is the same name as the audit service file.

The code below could be used to implement the local filter:

```
#!/usr/bin/perl

use     strict;
use     warnings 'all';

#
#  Get definition for the "standard" filter function.
#
require "$ENV{EMUPATH}/etc/audit/filters/audit.pl";

#
#  Only allow insertions, updates and deletions to be audited.
#
sub
LocalFilter
{
        my $tree = shift;
        my $columns = shift;
        my $xml = shift;
        my $operation;

        #
        #  Check whether the "standard" filter will allow the record
        #  to be processed.
        #
        return(1) if (Filter($tree, $columns, $xml) != 0);

        #
        #  Check for "insert", "update" and "delete".
        #
        $operation = $tree->{op}->{content};
        return($operation ne "insert" && $operation ne "update" &&
```

```
            $operation ne "delete");
}

1;
```

Notice that the standard filter `Filter()` should be called from within your `LocalFilter()` if you want standard filtering to apply. The arguments to the `Filter()` and `LocalFilter()` functions are the same as for the `Audit()` function defined for an audit service.