



EMu Documentation

Statistics

Document Version 1

EMu Version 4.0



Contents

SECTION 1	Statistics Facility	3
	Overview	3
	Statistics Module	4
	Reporting	7
	Periodic Tasks	10
	emuperiodic	11
	Tasks	13
	Creating a new period	18
	Regenerate missing data	19
SECTION 2	Appendix A - KE::Statistics perl module	21
	Name	21
	Synopsis	22
	Description	23
	KE::Statistics::Session	24
	Methods	25
	KE::Statistics::ResultSet	27
	Methods	28
	KE::Statistics::Date	29
	Methods	30
	KE::Statistics::Statistics	33
	Methods	34
	Bugs	36
	See Also	37

SECTION 1

Statistics Facility

Overview

As institutions continue with their EMu implementations the question of statistical analysis of system operations and data content inevitably arises. System administrators and managers require reports showing the number and type of operations performed on a per user basis, e.g. the number of insertions into the Catalogue module on a daily basis for the past month listed by user. The answer to this request is found in the records in the Audit module. In order to produce the information in a reportable format it is necessary to perform a number of searches of the Audit information and collate the results into a spreadsheet, which can then be graphed or tabulated. The process may be quite time consuming and tedious, and if the same information is required again at a future date, the same steps need to be repeated to get the same results.

EMu 4.0.01 introduces a Statistics facility that allows statistical information to be generated on a regular basis (hourly, daily, weekly or monthly) and stored in the Statistics module for later use. System administrators and managers need only search the Statistics module to locate the information they require and then produce a report (Excel Pivot table) from which tables and graphs may be generated.

The Statistics facility consists of two parts:

- **Statistics Module**

The Statistics module contains records with computed statistical values. Each record contains one value, a floating point number, that represents the result of a statistical criteria. For example, a value of 10 may indicate the number of records inserted by user `james` into the Catalogue module on 17 February 2009. A standard EMu module interface is provided to the Statistics module. An Excel report is supplied that presents the records in a Pivot table for further manipulation.

- **Periodic Tasks**

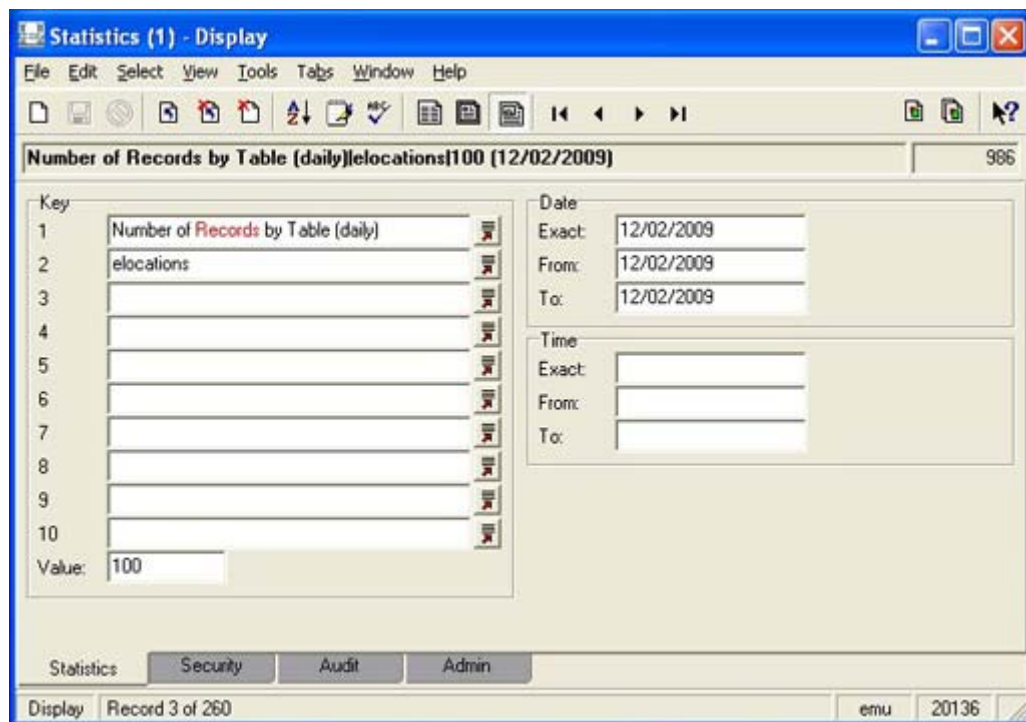
In order to provide useful statistical information it is necessary to have statistic records generated at regular intervals, removing the need for information to be obtained manually. The Periodic Tasks facility implements a framework in which individual tasks (scripts) can be placed and executed on a regular basis. It is the purpose of the tasks to generate statistical records by examining the various system reports and data within an EMu implementation. Periodic tasks can be run on an hourly, daily, weekly or monthly basis. It is possible to add new periods (e.g. fortnightly) if required.

Statistics Module

EMu 4.0.01 sees the addition of the Statistics module. Designed to hold statistical data, the module stores one statistical value per record. The value is computed by a task, which is charged with creating the record. Administrators can search the module to retrieve sequences of records used to produce reports.

The module consists of a Statistics tab that contains all the information about the statistical data. The other tabs are:

- Security - controls access to the data.
- Audit - lists auditable operations performed on the record.
- Admin - contains record creation and modification dates/times.



The Statistics tab stores three discrete pieces of information:

- **Keys and Value**

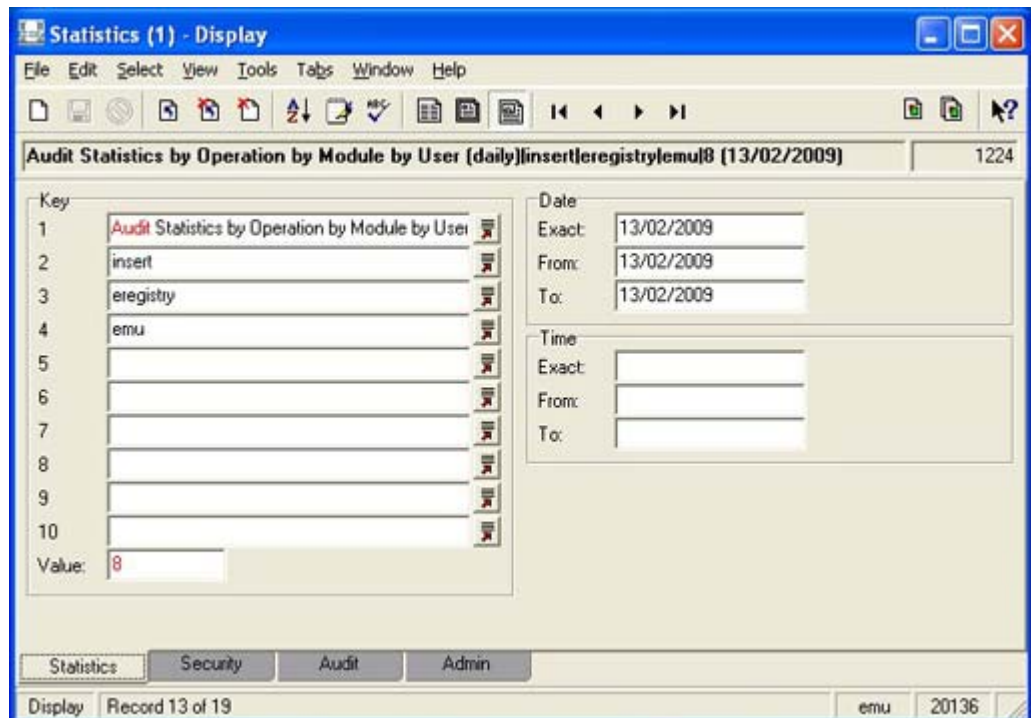
The Keys describe the type of statistical value stored in the record. A record consists of a number of hierarchical keys in which each level defines a variable piece of information for the statistic generated. The top level is reserved for the type of record. In the image above the first *Key* has a value of Number of Records By Table (daily). Three pieces of information are contained within the title:

- The *Value* of the record is a record count (Number of Records).
- The record count is generated on a per table basis (by Table).
- The *Value* is generated daily (daily).

The second *Key* (elocations) indicates the table for which the record count applies. Thus, the record above contains the number of records in the elocations table generated daily. In general, the title of the record should use the word *by* to indicate what variables are contained within the record. For

example, a title of Audit Statistics by Operation by Module by User (daily) would indicate that the record contains a count of the number of audit operations (insertions, edits, deletions, etc.) on a per table basis for each individual user. The *Value* represents the number of operations on a daily basis. Given this title, *Key 2* would contain the audit operation type, *Key 3* the table name and *Key 4* the user name.

The *Value* is a floating point number containing the numeric value defined by the Keys. In most instances the *Value* is an integer, however if averages are computed, the fractional part may be required.



- **Dates**

Three dates are provided: depending on the period of the statistical record, some or all of them may be filled:

- *Exact* - filled for data that is gathered within a single day (daily and hourly).
- *From* - the commencement date for the period. A commencement date should always be supplied.
- *To* - the completion date for the period. A completion date should always be supplied. If the period is a day or less, the commencement and completion dates are the same as the *Exact* date.

The date fields are used to define the day or range of days covered by the statistical value. The values are very useful when performing searches to gather statistical information for reporting.

- **Times**



Three times are provided: depending on the period of the statistical record, some or all of them may be filled:

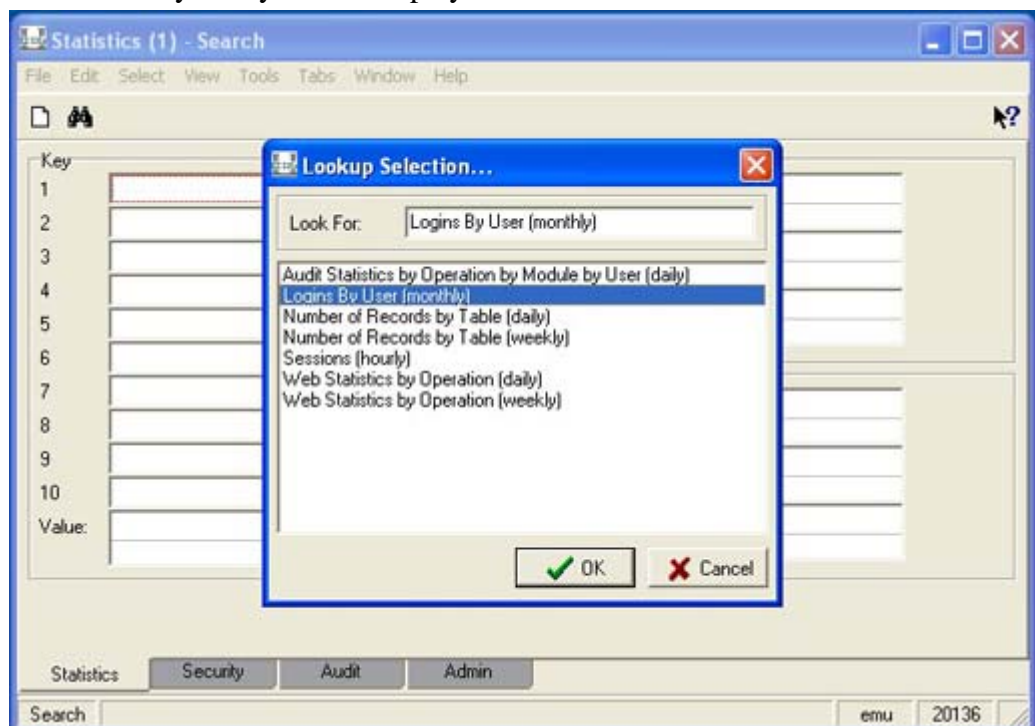
- *Exact* - filled for data that is gathered at a single point in time. Some hourly records represent a value at a fixed point in time, e.g. the number of users accessing the system. As this value represents the count at a fixed point in time, the *Exact* time field should be filled.
- *From* - the commencement time for the period. A commencement time should be supplied for tasks that are within a day (e.g. hourly).
- *To* - the completion time for the period. A completion time should be supplied for tasks that are within a day.

The time fields are used to define the point in time or range of time covered by the statistical value. If the value period is a day or longer, the time fields should be left empty. The values are very useful when performing searches to gather statistical information for reporting.

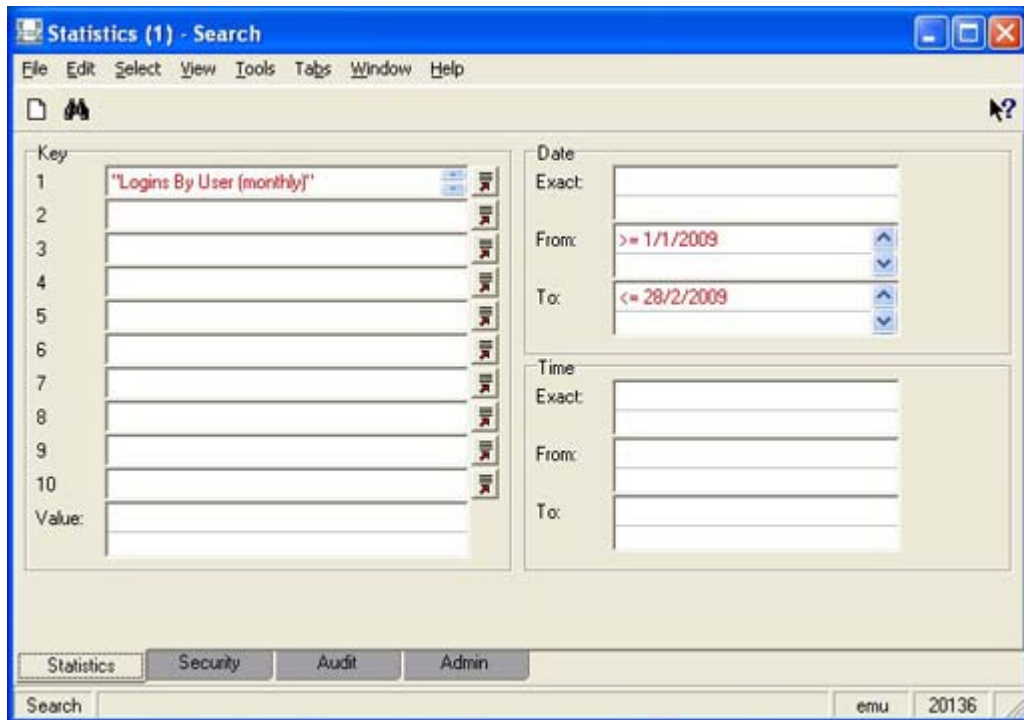
Reporting

The main reason for gathering statistical information is to produce reports. Reports may be tables of data, or more graphical representations such as charts may be used. The Statistics module provides one report only, the Excel based Statistics Pivot Table report. Before we can produce a report, it is necessary to retrieve the data on which to report. The steps below outline the process required to produce a statistical report:

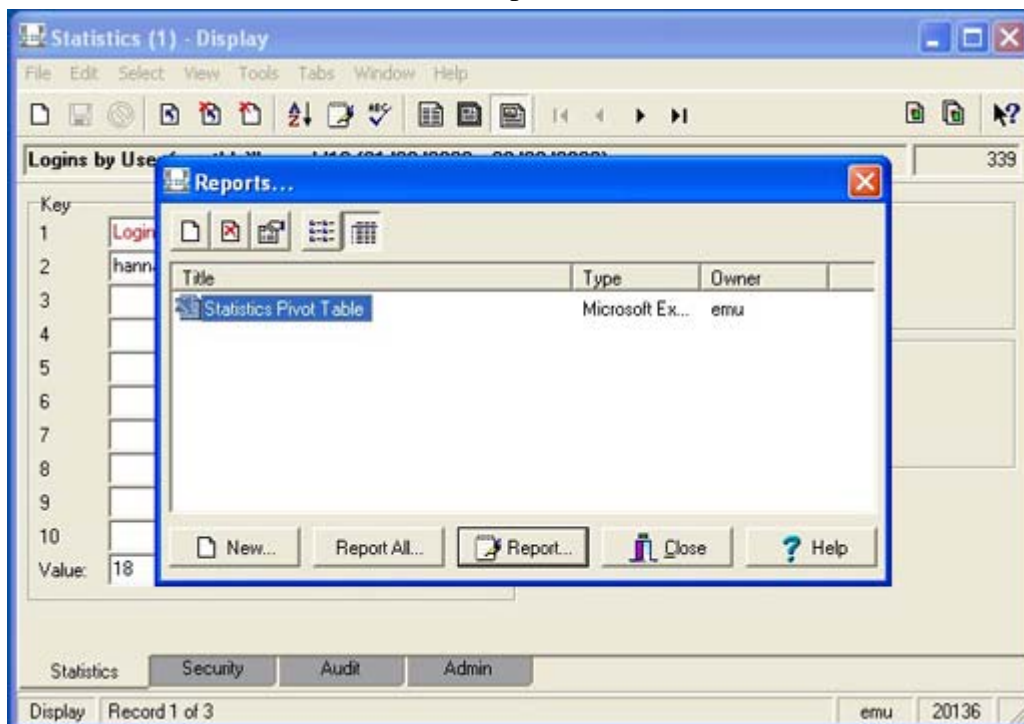
1. Open the Statistics module by selecting the **Statistics**  button in the Command Centre.
2. Select the **Lookup List**  button for *Key 1*. A list of all the statistical data maintained by the system is displayed:




3. Select the entry for the report type to be produced, e.g. **Logins by User (monthly)**.
4. If reporting on a single user or list of users as opposed to all users, the *Key 2* Lookup List could be used to select the required user names. In general, if a specific value or list of values is required for any given *Key*, the associated Lookup List can be used to select the values. If all values are to be reported, the *Key* should be left empty. In this example we want to report on the number of logins on a user basis for all users, so we leave *Key 2* empty.
5. Specify the date range for on which to report. In general this requires specifying a *From* date and a *To* date. In this example we want all records for January and February 2009:

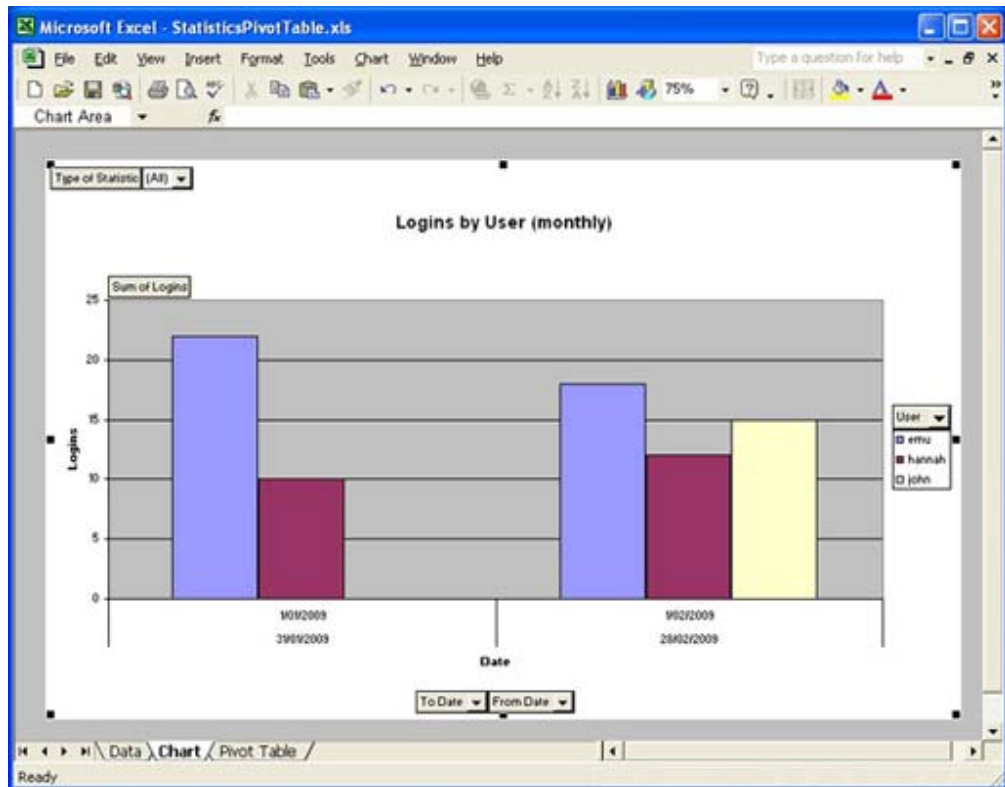


6. Perform the search to retrieve the required statistical records.
7. Select **Tools>Reports** from the Menu bar to display the Reports dialogue box and select the **Statistics Pivot Table** report:



8. Select **Report All**  to generate the report.
An Excel report will display.

 The report requires macros to be enabled so that a graph of the data can be produced.



Various tables and graphs can be produced as the data is now in a pivot table, based on the different Key values supplied.

	Type of Statistic			User		
	To Date	From Date	emu	hannah	john	
1	31/01/2009	1/01/2009	22	10	0	
2	28/02/2009	1/02/2009	18	12	15	

Once all the statistical information has been added to the Excel pivot table it is possible to manipulate any of the statistical variables by either restricting values or enabling all values. Pivot tables are extremely powerful and provide a very convenient mechanism for the production of reports with multiple statistical variables.

While Excel is the recommended tool for manipulating statistical data, it is possible to use any other reporting mechanism. If specialised output is required, it is possible to use Crystal Reports to produce the finished report. In this case it is recommended that the report is named after the type of statistical information it expects to receive.

Periodic Tasks

So far we have examined the new Statistics module, learned how to search for statistical information and considered the reporting options available. Next we explore how statistical information is generated.

In order to create useful reports, it is necessary to populate the Statistics module with meaningful records. In the simplest case it is possible to create statistic records manually by collating system information and inserting new statistic records with the required keys, dates, times and value. However it would not take long before someone forget to add the required records thus rendering the analysis incomplete.

The Periodic Tasks facility provides a framework in which tasks can be executed on a regular basis. Each task is a perl script generating one or more records for insertion into the estatistics table. At the heart of the framework is the **emuperiodic** program.

emuperiodic

The **emuperiodic** script is run at regular intervals to generate statistical information. Its primary purpose is to invoke all the task scripts for a given time period (hourly, daily, weekly, monthly). The usage message for **emuperiodic** is:

```
Usage: emuperiodic [-q] [-d yyyy:mm:dd[:HH:MM:SS]] period
```

where:

<code>-d yyyy:mm:dd[:HH:MM:SS]</code>	is the date to use for periodic tasks.
<code>-q</code>	specifies quiet mode, i.e. do not output progress.
<code>period</code>	specifies the time period for which statistical data is generated. Allowable values: <ul style="list-style-type: none"> • daily • hourly • monthly • weekly

Extra periods may be added, e.g. fortnightly, as required.

The Unix task scheduler cron is used to execute **emuperiodic** at the required intervals. The crontab entries used to invoke **emuperiodic** are:

```
#
# Run periodic tasks
#
30 * * * * /home/emu/client/bin/emurun emuperiodic hourly 2>&1 |
/home/emu/client/bin/emurun emulogger -t "KE EMu Periodic Tasks
Report" periodic
0 6 * * * /home/emu/client/bin/emurun emuperiodic daily 2>&1 |
/home/emu/client/bin/emurun emulogger -t "KE EMu Periodic Tasks
Report" periodic
30 6 * * 0 /home/emu/client/bin/emurun emuperiodic weekly 2>&1 |
/home/emu/client/bin/emurun emulogger -t "KE EMu Periodic Tasks
Report" periodic
0 7 1 * * /home/emu/client/bin/emurun emuperiodic monthly 2>&1 |
/home/emu/client/bin/emurun emulogger -t "KE EMu Periodic Tasks
Report" periodic
```

The table below shows when each instance of **emuperiodic** is executed:

Command	Executed
<code>emuperiodic hourly</code>	30 minutes past the hour being analysed.
<code>emuperiodic daily</code>	6 hours past the day being analysed.
<code>emuperiodic weekly</code>	6 hours and 30 minutes past the week being analysed, on the Sunday morning.
<code>emuperiodic monthly</code>	7 hours past the month being analysed.

All output from running periodic tasks is sent to **emulogger** which places the output into a file based on the current date (yyyy-mm-dd) in the `logs/periodic` directory. The log files provide a useful starting point if you suspect a problem with the execution of periodic tasks. As you can see, each task period is invoked **after** the time period for which it is generating statistics. The execution is delayed in order to allow any activities started in the task period to complete before the periodic tasks are run. It is also important to ensure that any system maintenance routines are not running while periodic tasks are executing, otherwise access to required tables may be denied.

When **emuperiodic** is invoked it looks for periodic tasks stored in either:

- `etc/periodic/period`

-OR-

- `local/etc/periodic/period`

where *period* is the argument supplied to **emuperiodic** (e.g. hourly). Each task is a perl script with a `.pl` (perl library) extension. If more than one task is found in the above directories, each task is executed sequentially in alphabetical order. Tasks in `local/etc/periodic` override scripts with the same name in `etc/periodic`.

Tasks

Each task is a perl function called by **emuperiodic** to generate statistical information. The *bare-bones* perl required for a task is:

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

#
# Copyright (c) 1998-2009 KE Software Pty Ltd
#

use strict;
use warnings;
use KE::Statistics;
no warnings 'redefine';

#
# Calculate the number of records per table.
#
sub
Periodic
{
    my $session = shift;
    my $date = shift;
    my $period = shift;

    #
    # Insert task code here
    #
}

```

emuperiodic calls the function `Periodic($session, $date, $period)` within the task script. The script then generates the statistical data and creates the required estatisctics record(s). The arguments to `Periodic()` are:

<code>\$session</code>	A <code>KE::Statistics::Session</code> object provides a connection to the back-end database environment. The object may be used to gather information to generate statistical values and to create estatisctics records.
<code>\$date</code>	A <code>KE::Statistics::Date</code> object contains the date and time at which emuperiodic was invoked. The <code>\$date</code> object is used to determine the date/time range of the statistical information for the task invoked.
<code>\$period</code>	A string that contains the name of the time period for the task being run. Typical periods include hourly, daily, weekly and monthly. Administrators may create new periods (e.g. fortnightly) as required, in which case <code>\$period</code> will contain the name of the new period.

A perl module is provided to help with the creation of estatisctics records and the generation of statistical values. The module is `KE::Statistics` and must be included in a task to gain access to its objects (via `use KE::Statistics;`). The

module provides a suite of classes to manipulate statistical information. The classes are:

`KE::Statistics::Session`
(page 24)

A `KE::Statistics::Session` object is used to gather information from the back-end server. The object may query any table or set of tables to allow statistical information to be generated. A set of methods allow information about the server environment to be gathered (e.g. list of registered users, list of tables, etc.).

`KE::Statistics::ResultSet`
(page 27)

A `KE::Statistics::ResultSet` object is returned by the `KE::Statistics::Session->search($texql)` method. The object provides access to the records returned as a result of the specified query.

`KE::Statistics::Date` (page 29)

The `KE::Statistics::Date` object makes the manipulation of dates easier. The object contains a breakdown of a date (`{year}`, `{month}`, `{day}`, `{hour}`, `{minute}` and `{second}`). A number of methods are provided that allow the date/time to be manipulated.

`KE::Statistics::Statistics`
(page 33)

The `KE::Statistics::Statistics` object is designed to provide easy insertions into the statistics table. A `Statistics` object allows the columns within a record to be set and the record written. A check is made to see if the record already exists in the table and if so an update is performed rather than an insertion. This allows periodic tasks to be re-run to refresh data without duplicate records being created.

The task script below is used to generate the number of records on a per table basis each day:

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

#
# Copyright (c) 1998-2009 KE Software Pty Ltd
#

use strict;
use warnings;
use KE::Statistics;
no warnings 'redefine';

#
# Calculate the number of records per table.
#
sub
Periodic
{
    my $session = shift;
    my $date = shift;
    my $period = shift;

    #
    # Run texlist -l and parse the results
    #
    my %data;
    for my $line (split(/\n/, `texlist -l`))
    {
        my @bits = split(/\s+/, $line);
        $data{$bits[0]} = $bits[2];
    }

    #
    # Create a statistics object we can use to insert
    # statistics records.
    #
    my $stats = $session->statistics();
    my $yesterday = $date->yesterday();
    $stats->setDate($yesterday);
    $stats->setDateFrom($yesterday);
    $stats->setDateTo($yesterday);
    $stats->setKey1("Records by Table ($period)");

    #
    # Now add the data for each type of operation
    #
    for my $table(keys %data)
    {
        $stats->setKey2($table);
        $stats->setValue($data{$table});
        $stats->write();
    }
}

1;
```

The example shows how it is possible to obtain a `KE::Statistics::Statistics` object (`$session->statistics()`) and use it to create statistics records. A point of interest is that the three Date values are set to yesterday's date. As the task is invoked 6 hours after the day has ended, it is necessary to use the date of the day before.

The task below generates statistical data about the number of audit operations performed on a per user and per table basis:

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

#
# Copyright (c) 1998-2009 KE Software Pty Ltd
#

use strict;
use warnings;
use KE::Statistics;
no warnings 'redefine';

#
# Calculate user statistics for operations.
#
sub
Periodic
{
    my $session = shift;
    my $date = shift;
    my $period = shift;

    #
    # Zero the operations count for all users of all tables.
    #
    my $data = {};
    foreach my $user (@{$session->users()})
    {
        foreach my $table (@{$session->tables()})
        {
            foreach my $operation (@{$session->
>operations($table)})
            {
                $data->{$user}->{$table}-
>{$operation} = 0;
            }
        }
    }

    #
    # Get back all the records for the supplied date.
    #
    my $yesterday = $date->yesterday();
    my $query = "select AudUser, AudOperation, AudTable from
eaudit " .
                "where AudDate = DATE" . $session->quote() .
                $yesterday->dateText() . $session->quote();
    my $results = $session->search($query);
    die ("Invalid query $query") if (! defined($results));

    #

```

```

value
    # Move through the results incrementing the appropriate
    # in the results table.
    #
    while ($results->next())
    {
        my $user = $results->text("AudUser");
        my $table = $results->text("AudTable");
        my $operation = $results->text("AudOperation");

        $data->{$user}->{$table}->{$operation}++;
    }
    $results->close();

    #
    # Create a statistics object we can use to insert
    # statistics records.
    #
    my $stats = $session->statistics();
    $stats->setDate($yesterday);
    $stats->setDateFrom($yesterday);
    $stats->setDateTo($yesterday);
    $stats->setKey1("Audit Statistics by Operation by Module
by User ($period)");

    #
    # Now move through the results table adding the
    appropriate records
    # to the statistics table.
    #
    foreach my $user (keys(%{$data}))
    {
        $stats->setKey4($user);
        foreach my $table (keys(%{$data->{$user}}))
        {
            $stats->setKey3($table);
            foreach my $operation (keys(%{$data-
>{$user}->{$table}}))
            {
                $stats->setKey2($operation);
                $stats->setValue($data->{$user}-
>{$table}->{$operation});
                $stats->write();
            }
        }
    }
}

1;

```

The above task shows how the `KE::Statistics::Session` object can be used to obtain information about the EMu environment (list of registered users, etc.) and also query tables (eaudit table). For a complete description of all the methods available in the `KE::Statistics` perl module please see Appendix A (page 21).

Creating a new period

The Periodic Tasks facility is designed to be extensible: new periods can be added as required. In this section we will add a new period that generates statistical information on a quarterly basis. The steps required are:

1. Determine a name for the period, e.g. `quarterly`.
2. Create the directory in which the quarterly tasks will be stored, e.g. `local/etc/periodic/quarterly`.
3. Add an entry to cron so that **emuperiodic** is invoked at a suitable time. The entry for `quarterly` will look like:

```
0 7 1 1,4,7,11 * /home/emu/client/bin/emurun emuperiodic
quarterly 2>&1 | /home/emu/client/bin/emurun emulogger -t "KE
EMu Periodic Tasks Report" periodic
```

4. Add the `quarterly` tasks to `local/etc/periodic/quarterly`.

Statistics generate on a quarterly basis. Note that the quarterly tasks are run at 7:00 am the day after the quarter ends.

Regenerate missing data

In some cases it may be necessary to generate statistic records for time periods that have passed, for instance periods before the Periodic Tasks facility was installed. It is possible to run **emuperiodic** using the `-d` option to specify the date passed through to the period tasks. In effect, the `-d` option makes it possible to alter the value of `$date` passed through to the `Periodic()` function. It is up to the task itself to examine the date and generate the correct information, where possible.

The date specified with the `-d` option should correspond to the date and time at which the original tasks would have been executed. For example, to run the daily tasks for 15 February 2009, the following command should be used:

```
emuperiodic -d 2009:02:16 daily
```

Notice how the date given was for the next day as this corresponds to the date on which cron would have invoked the daily tasks for 15 February 2009. By varying the date supplied it is possible to generate statistical information for periods before Periodic Tasks was installed. If a record already exists for the statistic generated, the value is simply updated.

The generation of data for previous time periods is successful only if the data for the period specified exists: it is not possible to generate auditing information if the audit records do not exist for the period specified.

SECTION 2

Appendix A - KE::Statistics perl module

The `KE::Statistics` module provides a set of objects to make the creation of tasks easier. The module is located in the `utils/KE` directory on the EMu server. The code is documented using POD (plain old documentation). The information in this Appendix was generated from the POD in the module.

Name

`KE::Statistics` - A set of objects usable by periodic scripts.

Synopsis

```
use KE::Statistics;

sub
Periodic
{
    my $session = shift;
    my $date = shift;
    my $period = shift;

    my $users = $session->users();
    my $tables = $session->tables();
    my $operations = $session->operations("eregistry");

    my $query = "Select all from eregistry where Key1 = " .
                $session->quote() . "User" . $session->quote();
    my $results = $session->search($query);

    while ($results->next())
    {
        my $key = $results->text("Key1");
        ...
    }
    $results->close();

    my $stats = $session->statistics();
    my $yesterday = $date->yesterday();
    $stats->setDate($yesterday);
    $stats->setDateFrom($yesterday);
    $stats->setDateTo($yesterday);
    $stats->setKey1("Records By Table");

    $stats->setValue("3");
    $stats->write();
}
```

Description

The `KE::Statistics` module provides a set of objects to facilitate the generation of records for the `eststatistics` table. The Periodic Tasks subsystem provides a plug-in mechanism that allows new tasks to be added to the existing framework. Each task is contained within a perl library (`.pl` file) and must contain at least one function, the `Periodic($session, $date, $period)` method.

The arguments are:

- | | |
|------------------------|---|
| <code>\$session</code> | A <code>KE::Statistics::Session</code> object provides a connection to the back-end database environment. The object may be used to gather information to generate statistical values and to create <code>eststatistics</code> records. |
| <code>\$date</code> | A <code>KE::Statistics::Date</code> object contains the date and time at which the Periodic Tasks subsystem was invoked. The <code>\$date</code> object is used to determine the date/time range of the statistical information for the task invoked. |
| <code>\$period</code> | A string that contains the name of the time period for the task being run. Typical periods include <code>hourly</code> , <code>daily</code> , <code>weekly</code> and <code>monthly</code> . Administrators may create new periods (e.g. <code>fortnightly</code>) as required, in which case <code>\$period</code> will contain the name of the new period. |

The `Periodic()` function is called by the Periodic Tasks subsystem at a scheduled time (e.g. `hourly`, `daily`, `weekly`, `monthly`, etc.) to create records in the `eststatistics` table.

The following objects are provided within the module:

KE::Statistics::Session

A `KE::Statistics::Session` object is used to gather information from the back-end server. The object may query any table or set of tables to allow statistical information to be generated. A set of methods allows information about the server environment to be gathered (e.g. list of registered users, list of tables, etc.).

As a `Session` object is provided as an argument to the `Periodic()` function, it is not necessary to create the object yourself, rather the supplied object should be used (which is efficient as only one `Session` object is used by all tasks invoked in the current execution). As the `Session` object is shared, you must not `close()` it in your task.

Methods

`new()`

```
$session = KE::Statistics::Session->new();
```

Creates a connection to the server environment. As the Periodic Tasks subsystem provides a `Session` object to your task, it is not necessary to use this method. The return value is an instance of a `Session` object.

`search($texql)`

```
$results = $session->search("count(select all from eparties)");
```

Executes a `TexQL` query statement on the server. The `$texql` argument may be any valid `TexQL` query statement. The return value is a `KE::Statistics::ResultSet` object. If the query statement is invalid, an `undef` value is returned.

`statistics()`

```
$stats = $session->statistics();
```

After your tasks have generated statistical information, it is necessary to write the data into `estatistics` records. The `KE::Statistics::Statistics` object provides a convenient object for creating `estatistics` records. The `statistics()` method returns a `Statistics` object that may be used to create the records.

`quote()`

```
$texql = "select all from eparties where NamLast contains " . $session->quote() . "Badenoff" . $session->quote();
```

When building `TexQL` statements, non-numeric values must be enclosed within *quotes*. The quote character is configurable and is set to avoid escaping characters within values. The default quote character is `\001` (`Ctrl+A`). The `quote()` method returns the current quote character.

`close()`

```
$session->close();
```

Once all communication with the server environment is complete, the connection needs to be closed so that system resources can be returned to other users. The `close()` method terminates a `Session` connection. As the Periodic Tasks subsystem handles the creation and closing of the session, you should not call this method.

`users()`

```
foreach my $user (@{$session->users()})
```

The `users()` method returns a reference to a list of registered users in the server environment. The list is built from records in the server registry (`eregistry` table).

`tables()`

```
foreach my $table (@{$session->tables()})
```

The `tables()` method returns a reference to a list of tables in the server environment. The **Table Access Registry** entry is used to build the list of tables.

```
operations($table)
```

```
    foreach my $operation (@{$session->operations("eparties")})
```

The `operations()` method returns a reference to a list of audit operations enabled for the table supplied in the `$table` argument. The list returned is populated by operations defined by `texaudit`. Use `texaudit -h` to get a complete list of the available operations.

KE::Statistics::ResultSet

A `KE::Statistics::ResultSet` object is returned by the `KE::Statistics::Session->search($texql)` (page 25) method. The object provides access to the records returned as a result of the specified query. Once you have finished dealing with the `ResultSet` object, it is necessary to `close()` it so that system resources can be returned to other users.

Methods

`new()`

```
$results = KE::Statistics::ResultSet->new($cursor)
```

A `ResultSet` object provides a convenient mechanism for dealing with a query cursor (`$cursor`). The cursor is returned by the server environment when a search has completed. As a `ResultSet` object is returned by `KE::Statistics::Session->search($texql)` (page 25), it is not necessary to create instances of `ResultSet` objects.

`next()`

```
while ($results->next())
```

When a `ResultSet` object is returned by `KE::Statistics::Session->search($texql)` (page 25), the current record is positioned before the first record. The `next()` method moves the current record position to the next matching record. A value of 0 is returned if you are past the last matching record, otherwise 1 is returned.

`text($column)`

```
$value = $results->text("NamLast");
```

The `text()` method returns the value for the column specified by `$column` argument. The value is returned as a string. If the column does not exist, an `undef` value is returned.

`close()`

```
$results->close();
```

Once you have finished with the records in the `ResultSet` object, you should `close()` (page 25) the object so that server resources are returned to users. If you do not close a `ResultSet` object, it will be closed by the Periodic Tasks subsystem once it has completed processing all tasks.

KE::Statistics::Date

In order to make the manipulation of dates easier, the `KE::Statistics::Date` object is provided. The object contains a breakdown of a date (`{year}`, `{month}`, `{day}`, `{hour}`, `{minute}` and `{second}`). A number of methods are provided that allow the date/time to be manipulated.

To help with arithmetic manipulation of dates the `julianNumber()` (page 30) method is provided to return the Julian date (see http://en.wikipedia.org/wiki/Julian_day). The integer part of the floating point number returned represents the day number, while the fractional part encodes the time within the day. Normal arithmetic may be applied to the number. The `julianDate()` (page 30) method is used to convert a Julian date to a `Date` (page 33) object. For example, the following code could be used to find the date three days back from today:

```
$now = KE::Statistics::Date->new();
$then = KE::Statistics::Date->julianDate($now->julianNumber() -
3);
```

Subtracting two Julian dates will result in the number of days, hours, minutes and seconds between them:

```
$diff = $now - $then;
```

When using dates with `TexQL` query statements, always specify the date in ODBC format (`yyyy-mm-dd`). The `dateText()` (page 30) method provides the value in the correct format. Similarly, time values should be specified using a 24 hour clock (`HH:MM:SS`). The `timeText()` (page 30) method provides the value formatted correctly.

Methods

```
new($year, $month, $day, $hour, $minutes, $seconds)
```

```
$date = KE::Statistics::Date->new();  
$date = KE::Statistics::Date->new(2009, 02, 11);  
$date = KE::Statistics::Date->new(2009, 02, 11, 16, 55, 02);
```

The `new()` (page 25) method creates a new instance of a `Date` (page 33) object. Up to six arguments may be provided to initialise the `Date` object with a given date and/or time. If any arguments are missing, the component for the current date/time is used. Thus, calling `new()` without any arguments provides a `Date` object with the current date and time.

```
clone()
```

```
$newdate = $date->clone();
```

The `clone()` method creates a copy of a `Date` object initialised with the same date/time as the calling `Date` object.

```
yesterday()
```

```
$yesterday = $date->yesterday();
```

Returns a new `Date` object initialised with yesterday's date. The value is 24 hours before the calling `Date` object; that is, the time component is not changed.

```
lastHour()
```

```
$newdate = $date->lastHour();
```

Returns a new `Date` object initialised with the date/time one hour before the date/time of the calling `Date` object.

```
lastSecond()
```

```
$newdate = $date->lastSecond();
```

Returns a new `Date` object initialised with the date/time one second before the date/time of the calling `Date` object.

```
lastWeek()
```

```
$newdate = $date->lastWeek();
```

Returns a new `Date` object initialised with the date/time one week before the date/time of the calling `Date` object.

```
lastMonth()
```

```
$newdate = $date->lastMonth();
```

Returns a new `Date` object initialised with the date/time one month before the date/time of the calling `Date` object. If the resulting date is past the end of the month, the last day of the month is used.

```
set($year, $month, $day, $hour, $minute, $second)
```

```
$date->set(2010, 12, 14); $date->set(undef, undef, undef, 0, 0, 0);
```

The `set()` method allows any component of a `Date` object to be assigned a value. If `undef` is provided for a component, the component's current value is maintained. If a component is missing, a value of `undef` is assumed.


```
compare($date)
```

```
if ($date1->compare($date2) == 0)
```

The `compare()` method compares two `Date` objects for equality. The return value can be used to determine the equality of the objects:

```
-1 - date argument is lower than date object
0 - date argument is same as Date object
+1 - date argument is greater than Date object
```

```
compareDate($date)
```

```
if ($date1->compareDate($date2) == 0)
```

The `compareDate()` method compares two `Date` objects for equality at the date level. The time component is ignored. The return value can be used to determine the equality of the object's dates:

```
-1 - date argument is lower than date object
0 - date argument is same as date object
+1 - date argument is greater than Date object
```

```
compareTime($date)
```

```
if ($date1->compareTime($date2) == 0)
```

The `compareTime()` method compares two `Date` objects for equality at the time level. The date component is ignored. The return value can be used to determine the equality of the object's times:

```
-1 - date argument is lower than date object
0 - date argument is same as date object
+1 - date argument is greater than date object
```

```
dateText()
```

```
$texql = "select all from eaudit where AudDate = DATE" .
    $session->quote() . $date->dateText() . $session->quote();
```

The `dateText()` method returns a text representation of the object's date in ODBC format (`yyyy-mm-dd`). The value is suitable for `DATE` values in `TexQL` queries regardless of the date format used on the server.

```
timeText()
```

```
$texql = "select all from eaudit where AudTime = TIME" .
    $session->quote() . $date->timeText() . $session->quote();
```

The `timeText()` method returns a text representation of the object's time in ODBC format (`HH:MM:SS`). The value is suitable for `TIME` values in `TexQL` queries regardless of the time format used on the server.

```
julianNumber($date)
```

```
$julian = $date->julianNumber();
```

The return value of `julianNumber()` is a floating point number representing the day number in the integer part and the time (in 1/86400th of a second) in the fractional part. Note that the Julian number for a day represents midday for the given day. Any time before midday will have an integer value one less than any time after midday. To

get the Julian number for any time within a day it is necessary to add 0.5 before calling `int()`. Thus:

```
$daynumber = int($date->julianNumber() + 0.5);
```

returns the Julian day number. See http://en.wikipedia.org/wiki/Julian_day for details.

```
julianDate($number)
```

```
$date = KE::Statistics::Date->julianDate($number);
```

A `Date` object is returned containing the date and time expressed by the Julian date number passed as an argument. The `julianDate()` method provides a mechanism for getting a `Date` object after some date numeric arithmetic has been performed.

```
weekDay()
```

```
$day = ("Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"][$date->weekDay()];
```

Returns the numeric day of the week for the given `Date` object, where 0 = Sunday, 6 = Saturday.

KE::Statistics::Statistics

The `KE::Statistics::Statistics` object is designed to provide easy insertions into the `estatistics` table. A `Statistics` object allows the columns within a record to be set and the record written. A check is made to see if the record already exists in the table and if so an update is performed rather than an insertion. This allows periodic tasks to be re-run to refresh data without duplicate records being created.

An `estatistics` record consists of four main components:

- Keys* A hierarchy of up to ten Keys may be specified. The Keys are used to define the variables used to arrive at the statistical value. The first Key should contain a title defining what the statistical value is. For example:
- ```
Key1: Audit Statistics by Operation by Module by User
 (daily)
Key2: delete
Key3: elocations
Key4: bill
```
- allows you to determine from `Key1` what information is being stored. The next three Keys are the variables (*Operation, Module, User*) available. The above convention should be used so users can easily locate records within the `estatistics` table.
- Date*      Three date fields exist in `estatistics`: *DateExact*, *DateFrom* and *DateTo*. The *DateExact* field is filled if the statistical value represents a period of a day or less (that is daily or hourly), otherwise it is left empty. The *DateFrom* and *DateTo* fields should always be filled with the commencement and completion dates respectively.
- Time*      Three time fields exist in `estatistics`: *TimeExact*, *TimeFrom* and *TimeTo*. The *TimeExact* field is filled if the statistical value represents a single point of time in a day, otherwise it is left empty. If the period is a range of time in a day, the *TimeFrom* and *TimeTo* fields should be filled with the commencement and completion times respectively.
- Value*      The *Value* is the statistical datum associated with the set of defined Keys for the given date and/or time. For example, a *Value* of 10 with the above Keys would indicate user `bill` has deleted 10 location records for the specified date/time period.

When completing an `estatistics` record, the appropriate fields should be filled based on the period the value covers.

## Methods

```
new($session)
```

```
 $stats = KE::Statistics::Statistics->new($session);
```

A new instance of a `Statistics` object tied to the supplied `Session` (`$session` (page 23)) is created. You should not create instances of a `Statistics` object directly, rather `$session->statistics()` (page 25) should be used as this ties the created object to the session.

```
setDate($date)
```

```
 $stats->setDate($date);
```

Sets the *DateExact* column in *estaticistics* to the value of the `Date` object supplied. The *DateExact* column should be filled if the statistic record is for a particular day (that is, daily) or a time range within a day (that is, hourly).

```
setDateFrom($date)
```

```
 $stats->setDateFrom($date);
```

Sets the *DateFrom* column in *estaticistics* to the value of the `Date` object supplied. The *DateFrom* column should always be filled. It contains the starting date for the statistics period.

```
setDateTo($date)
```

```
 $stats->setDateTo($date);
```

Sets the *DateTo* column in *estaticistics* to the value of the `Date` object supplied. The *DateTo* column should always be filled. It contains the finishing date for the statistics period.

```
setTime($date)
```

```
 $stats->setTime($date);
```

Sets the *TimeExact* column in *estaticistics* to the value of the `Date` object supplied. The *TimeExact* column should only be filled if the statistic record is for a single point in time, otherwise the column should be left empty.

```
setTimeFrom($date)
```

```
 $stats->setTimeFrom($date);
```

Sets the *TimeFrom* column in *estaticistics* to the value of the `Date` object supplied. The *TimeFrom* column contains the starting time for the statistics period. It should only be filled for statistic records for a single point in time, or a time range (that is hourly).

```
setTimeTo($date)
```

```
 $stats->setTimeTo($date);
```

Sets the *TimeTo* column in *estaticistics* to the value of the `Date` object supplied. The *TimeTo* column contains the completion time for the statistics period. It should only be filled for statistic records for a single point in time, or a time range (that is hourly).

```
setKey1($value)
```

```
 $stats->setKey1($value);
```

Sets the *Key1* column in *estaticistics* to the value supplied.

```
setKey2($value)
```

```
 $stats->setKey2($value);
```

Sets the *Key2* column in estatictics to the value supplied.

```
setKey3($value)
```

```
 $stats->setKey3($value);
```

Sets the *Key3* column in estatictics to the value supplied.

```
setKey4($value)
```

```
 $stats->setKey4($value);
```

Sets the *Key4* column in estatictics to the value supplied.

```
setKey5($value)
```

```
 $stats->setKey5($value);
```

Sets the *Key5* column in estatictics to the value supplied.

```
setKey6($value)
```

```
 $stats->setKey6($value);
```

Sets the *Key6* column in estatictics to the value supplied.

```
setKey7($value)
```

```
 $stats->setKey7($value);
```

Sets the *Key7* column in estatictics to the value supplied.

```
setKey8($value)
```

```
 $stats->setKey8($value);
```

Sets the *Key8* column in estatictics to the value supplied.

```
setKey9($value)
```

```
 $stats->setKey9($value);
```

Sets the *Key9* column in estatictics to the value supplied.

```
setKey10($value)
```

```
 $stats->setKey10($value);
```

Sets the *Key10* column in estatictics to the value supplied.

```
setValue($value)
```

```
 $stats->setValue($value);
```

Sets the *Value* column in estatictics to the value supplied. The statistical value is a floating point number.

```
write()
```

The `write()` method saves the data in the `Statistics` object to the estatictics table. If a record already exists with the same Keys, dates and times, the value is updated, otherwise a new record is created.

---

## Bugs

Encoding dates as a Julian number with the time as the fractional component can lead to issues when subtracting dates, as the result represents the number of days, hours, minutes and seconds between the two dates. If you need to find the number of days between two dates, it is necessary to clear the time component before applying the subtraction:

```
$date1->set(undef, undef, undef, 0, 0, 0);
$date2->set(undef, undef, undef, 0, 0, 0);
$days = abs($date2->julianNumber() - $date1->julianNumber());
```

While this not a bug, it is something to keep in mind when manipulating Julian date numbers.

---

## See Also

For a complete description of how Julian date numbers are generated and used see:

[http://en.wikipedia.org/wiki/Julian\\_day](http://en.wikipedia.org/wiki/Julian_day)