



OTRS
Open Technology
Real Services

Documentation

OTRS 2.4 - Developer Manual

Aitutaki Beach (Cook Island) Edition

Build Date:

2009-03-06

OTRS 2.4 - Developer Manual

Copyright © 2003-2009 OTRS AG

René Bakker, Hauke Böttcher, Stefan Bedorf, Shawn Beasley, Jens Bothe, Udo Bretz, Martin Edenhofer, Manuel Hecht, Christopher Kuhn, André Mindermann, Henning Oschwald, Thomas Raith, Stefan Rother, Burchard Steinbild

This work is copyrighted by OTRS AG.

You may copy it in whole or in part as long as the copies retain this copyright statement.

UNIX is a registered trademark of X/Open Company Limited. Linux is a registered trademark of Linus Torvalds.

MS-DOS, Windows, Windows 95, Windows 98, Windows NT, Windows 2000, Windows XP, Windows 2003 and Windows Vista are registered trademarks of Microsoft Corporation. Other trademarks and registered trademarks are: SUSE and YaST of SUSE Linux GmbH, Red Hat and Fedora are registered trademarks of Red Hat, Inc. Mandrake is a registered trademark of MandrakeSoft, SA. Debian is a registered trademark of Software in the Public Interest, Inc. MySQL and the MySQL Logo are registered trademarks of MySQL AB.

All trade names are used without the guarantee for their free use and are possibly registered trade marks.

OTRS AG essentially follows the notations of the manufacturers. Other products mentioned in this manual may be trademarks of the respective manufacturer.



Table of Contents

1. Introduction	1
2. Coding Style Guide	2
1. Formatting	2
2. Naming	2
3. Source Code Header and Charset	2
4. Version Comments	3
5. Special comments	3
6. Restrictions for some functions	3
7. Perldoc	4
8. Length of lines	4
9. Core-Objects	4
9.1. Objects and their allocation	4
9.2. Using of the MainObject	4
9.3. Using of the EncodeObject	4
3. Architecture	5
1. Directories	5
2. Files	6
3. Core Modules	6
4. Frontend Handle	6
5. Frontend Modules	6
6. CMD Frontend	7
7. Database	7
4. Config Mechanism	8
1. Default Config	8
2. Custom Config	8
3. Accessing Config Options	8
4. XML Config Options	9
4.1. Types of XML Config Variables	9
4.2. String	10
4.3. Textarea	10
4.4. Options	10
4.5. Array	10
4.6. Hash	10
4.7. Hash with SubArray, SubHash	10
4.8. FrontendModuleReg (NavBar)	11
4.9. FrontendModuleReg (NavBarModule)	11
5. Database Mechanism	13
1. How it works	13
1.1. SQL	13
1.2. XML	14
2. Database Drivers	16
3. Supported Databases	16
6. Log Mechanism	17
1. Use and Syntax	17
2. Example	17
7. Module Format	18
1. Core Modules	18
1.1. Log Module	18
1.2. Output Filter	20
1.3. Stats Module	21
2. Frontend Modules	33

2.1. Dashboard Module	33
3. Old Module Descriptions	36
3.1. Auth Module	36
3.2. Notify Module	36
3.3. Navigation Module	38
3.4. Frontend Modules	38
3.5. Core Modules	41
3.6. Customer Auth Module	42
3.7. Customer User Module	42
3.8. Customer Navigation Module	44
3.9. Stats Module	44
3.10. Ticket Modules	46
8. Templates	54
1. Formatting	54
1.1. Comment	54
1.2. \$Data{""}	54
1.3. \$QData{""}	55
1.4. \$LQData{""}	55
1.5. \$Env{""}	55
1.6. \$QEnv{""}	55
1.7. \$Quote{""}	55
1.8. \$Text{""}	56
1.9. \$JSText{""}	56
1.10. \$Config{""}	56
1.11. \$Include{""}	56
1.12. Block	56
1.13. set	57
1.14. if	57
1.15. system-call	58
2. Example	58
9. Layout	59
1. CSS Style	59
1.1. View	59
1.2. Edit/New/Search	60
1.3. Overview/Search Result	61
1.4. Delete	62
2. Images	62
3. Special CSS Definitions	62
10. Language Translations	64
1. How it works	64
1.1. Default Framework Translation File	64
1.2. Frontend Translation File	65
1.3. Custom Translation File	65
2. Add a new default framework translation	66
11. Object Basics	67
1. Object Options	67
2. Search Options	67
3. Config Naming	67
4. Config File	67
5. NavBar Settings	68
6. Screen flow	70
12. Development Environment	71
1. Framework checkout (CVS)	71
2. Linking Expansion Modules	71

3. Necessary Actions after Linking	71
13. Writing an OTRS module for a new object	72
1. What we want to write	72
2. Default Config File	72
3. Frontend Module	73
4. Core Module	74
5. dtl Template File	74
6. Language File	75
7. Summary	75
14. Package Management	76
1. Package Distribution	76
1.1. Package Repository Index	76
2. Package Commands	76
2.1. Install	76
2.2. Uninstall	76
2.3. Upgrade	77
2.4. List	77
15. Package Building	78
1. Package Spec File	78
1.1. Name	78
1.2. Version	78
1.3. Framework	78
1.4. Vendor	78
1.5. URL	78
1.6. License	78
1.7. ChangeLog	79
1.8. Description	79
1.9. BuildHost	79
1.10. BuildDate	79
1.11. PackageRequired	79
1.12. ModuleRequired	79
1.13. OS (^M)	79
1.14. Filelist	80
1.15. DatabaseInstall	80
1.16. DatabaseUpgrade	80
1.17. DatabaseReinstall	80
1.18. DatabaseUninstall	81
1.19. IntroInstall	81
1.20. IntroUninstall	81
1.21. IntroReinstall	81
1.22. IntroUpgrade	81
1.23. CodeInstall	82
1.24. CodeUninstall	82
1.25. CodeReinstall	82
1.26. CodeUpgrade	83
2. Example .sopm	83
3. Package Build	84
4. Package Life Cycle - Install/Upgrade/Uninstall	84
16. Unit Tests	85
1. Creating a test file	85
2. Testing	85
3. True()	86
4. False()	86
5. Is()	86

A. Additional Ressources	87
1. OTRS.org	87
2. Online API Library	87
3. Developer Mailing List	87
4. Commercial Support	87



Chapter 1. Introduction

OTRS is a multi-platform web application framework which was originally developed for a trouble ticket system. It supports different web servers and databases.

This manual shows how to develop your own OTRS modules and applications based on the OTRS styleguides.



Chapter 2. Coding Style Guide

In order to preserve the consistent development of the OTRS project, we have set up a few guidelines regarding style.

1. Formatting

TAB: We use 4 spaces. Examples for braces:

```
if ($Condition) {
    Foo();
}
else {
    Bar();
}

while ($Condition == 1) {
    Foo();
}
```

2. Naming

Names and comments are written in English. Variables, Objects and Methods must be descriptive nouns or noun phrases with the first letter set upper case.

e. g. @TicketIDs or \$Output or BuildQueueView()

3. Source Code Header and Charset

Attach the following header to each and every source file. Source files are saved in Charset ISO-8859-1.

```
# --
# (file name) - a short description what it does
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: codesyntax.xml,v 1.26 2010-05-10 18:50:54 bes Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --
```

Executable files (*.pl) have a special header.

```
#!/usr/bin/perl -w
# --
# (file name) - a short description what it does
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: codesyntax.xml,v 1.26 2010-05-10 18:50:54 bes Exp $
# --
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU AFFERO General Public License as published by
# the Free Software Foundation; either version 3 of the License, or
# any later version.
```



```
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU Affero General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
# or see http://www.gnu.org/licenses/agpl.txt.
# --
```

The following line is updated by the CVS:

```
# $Id: codesyntax.xml,v 1.26 2010-05-10 18:50:54 bes Exp $
```

4. Version Comments

Some functions may not be available in the current framework version. Thus, sometimes settings have to be changed when a new module and framework version will be released. If you use version comments, you can search for information (e.g. using `grep`) on what must be changed when a new version is published. The keyword for version comments is 'FRAMEWORK' For perl use '#' and for xml use the xml comments.

```
e.g. for perl-code
# FRAMEWORK-2.1: the function ID2UserName is first available in OTRS 2.1
```

5. Special comments

The only ways to create special comments are the following ways. Example 1 - especially for subactions of frontend modules

```
# -----#
# here starts a special area
# -----#
```

Example 2 - especially for customizing standard OTRS files

```
# --- customizing for bsi
```

6. Restrictions for some functions

Some functions are not useful in every script. Please pay attention to the following restrictions.

- don't use "die" and "exit" in .pm-files
- don't use the "Dumper" function in released files
- don't use "print" in .pm files
- use OTRS specific function "SystemTime2Date" instead of "localtime"

7. Perldoc

Every function which could be used outside of its package must have a perldoc. It should look like the following example.

```
=item SystemTime2TimeStamp()  
returns a time stamp in "yyyy-mm-dd 23:59:59" format.  
  
    my $TimeStamp = $TimeObject->SystemTime2TimeStamp(  
        SystemTime => $SystemTime,  
    );  
  
If you need the short format "23:59:59" for dates that are "today",  
pass the Type parameter like this:  
  
    my $TimeStamp = $TimeObject->SystemTime2TimeStamp(  
        SystemTime => $SystemTime,  
        Type       => 'Short',  
    );  
  
=cut
```

8. Length of lines

Please see that a line of code is not longer than 100 characters.

9. Core-Objects

9.1. Objects and their allocation

In OTRS many objects are available. But it is not allowed to use every object in each script. Please note the following definitions

- don't use the LayoutObject in core modules
- don't use the ParamObject in core modules
- don't use the DBObject in frontend modules

9.2. Using of the MainObject

Information about the MainObject

- initialize the MainObject in the basic .pl-file
- in .pm files only pass it to the next Object you initialize
- don't use the Perl "require" function any more

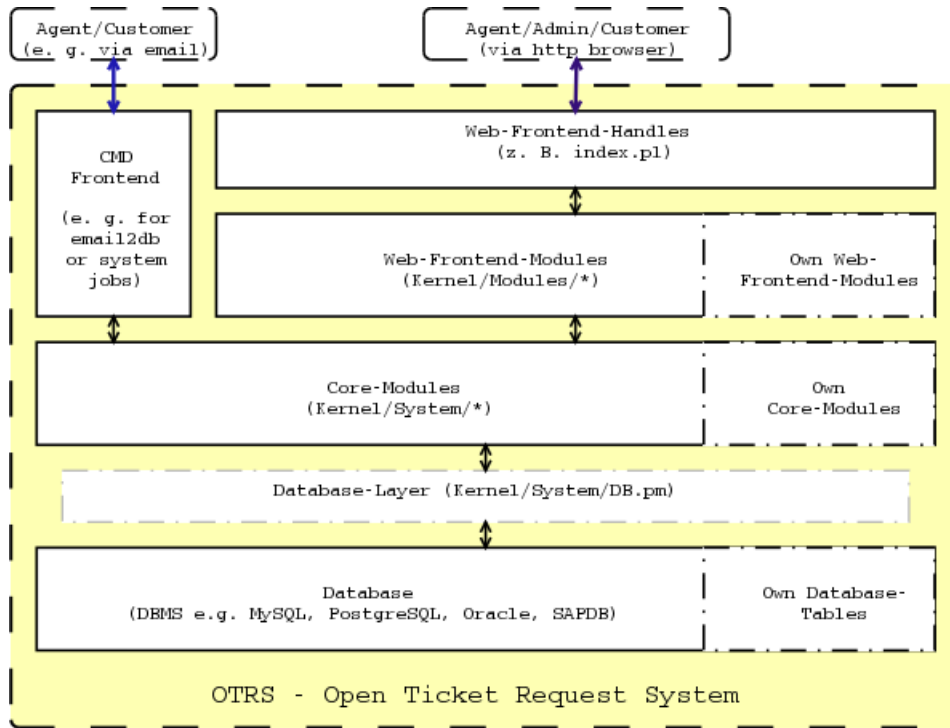
9.3. Using of the EncodeObject

Information about the EncodeObject

- initialize the EncodeObject in the basic .pl-file
- in .pm files only pass it to the next Object you initialize

Chapter 3. Architecture

The OTRS framework is modular. The following picture shows the basic layer architecture of OTRS.



(c) 2001-2004 OTRS Team, <http://otrs.org/>

1. Directories

Directory	Description
bin/	CMD programmes
bin/cgi-bin/	web handle
bin/cgi-bin/	fast cgi web handle
Kernel	modules
Kernel/Config/	config
Kernel/Config/Files	config files
Kernel/Language	language translation
Kernel/System/	core modules, e.g. Log, Ticket...
Kernel/Modules/	frontend modules, e.g. QueueView...
Kernel/Output/HTML/	html templates
var/	variable data
var/log	logfiles
var/cron/	cron files
var/httpd/htdocs/	htdocs directory with index.html
var/httpd/htdocs/css/Standard/	css-style-sheets
var/httpd/htdocs/images/Standard/	icons and pictures

Directory	Description
var/httpd/htdocs/js/	java script files
var/httpd/htdocs/images/yui/	several yui versions
scripts/	misc
scripts/test/	test files
scripts/sample/	sample files

2. Files

- .pl = Perl
- .pm = Perl Modul
- .dtl = Dynamic Template Language (html template file)
- .dist = Default Templates of Files

3. Core Modules

Core modules are located under `$OTRS_HOME/Kernel/System/*`. This layer is for the logical work. Core modules are used to handle system routines like "lock ticket" and "create ticket". A few main core modules are:

- Kernel::System::Config (to access config options)
- Kernel::System::Log (to log into OTRS log backend)
- Kernel::System::DB (to access the database backend)
- Kernel::System::Auth (to check a user authentication)
- Kernel::System::User (to manage users)
- Kernel::System::Group (to manage groups)
- Kernel::System::Email (for sending emails)

For more information, see: <http://dev.otrs.org/>

4. Frontend Handle

The interface between the browser, web server and the frontend modules. A frontend module can be used via the http-link.

<http://localhost/otrs/index.pl?Action=Modul>

5. Frontend Modules

Frontend modules are located under `"$OTRS_HOE/Kernel/Modules/*.pm"`. There are two public functions in there - "new()" and "run()" - which are accessed from the Frontend Handle (e.g. index.pl).

"new()" is used to create a frontend module object. The Frontend Handle provides the used frontend module with the basic framework objects. These are, for example: ParamObject (to get formular params), DBObject (to use existing databse connects), LayoutObject (to

use templates and other html layout functions), ConfigObject (to access config settings), LogObject (to use the framework log system), UserObject (to get the user functions from the current user), GroupObject (to get the group functions).

For more information on core modules see: <http://dev.otrs.org/>

6. CMD Frontend

The CMD (Command) Frontend is like the Web Frontend Handle and the Web Frontend Module in one (just without the LayoutObject) and uses the core modules for some actions in the system.

7. Database

The database interface supports different databases.

For tables on database relation see otrs-database.png: <ftp://ftp.otrs.org/pub/otrs/misc/>



Chapter 4. Config Mechanism

1. Default Config

There are different default config files. The main one, which comes with the framework, is:

Kernel/Config/Defaults.pm

This file should be left untouched as it is automatically updated on framework updates. There is also a sub directory where you can store the default config files for your own modules. These files are used automatically.

The directory is located under:

`$OTRS_HOME/Kernel/Config/Files/*.pm`

And could look as follows:

Kernel/config/Files/Calendar.pm

```
# module reg and nav bar
$self->{'Frontend::Module'}->{'AgentCalendar'} = {
  Description => 'Calendar',
  NavBarName => 'Ticket',
  NavBar => [
    {
      Description => 'Calendar',
      Name => 'Calendar',
      Image => 'calendar.png',
      Link => 'Action=AgentCalendar',
      NavBar => 'Ticket',
      Prio => 5000,
      AccessKey => 'c',
    },
  ],
};

# show online customers
$self->{'Frontend::NotifyModule'}->{'80-ShowCalendarEvents'} = {
  Module => 'Kernel::Output::HTML::NotificationCalendar',
};
```

2. Custom Config

If you want to change a config option, copy it to

Kernel/Config.pm

and set the new option. This file will be read out last and so all default config options are overwritten with your settings.

This way it is easy to handle updates - you just need the Kernel/Config.pm.

3. Accessing Config Options

You can read and write (for one request) the config options via the core module "Kernel::Config". The config object is a base object and thus available in each Frontend Module.

If you want to access a config option:

```
my $ConfigOption = $Self->{ConfigObject}->Get('Prefix::Option');
```

If you want to change a config option at runtime and just for this one request/process:

```
$Self->{ConfigObject}->Set(  
    Key => 'Prefix::Option'  
    Value => 'SomeNewValue',  
);
```

4. XML Config Options

XML config files are located under:

```
$OTRS_HOME/Kernel/Config/Files/*.xml
```

Each config file has the following layout:

```
<?xml version="1.0" encoding="utf-8" ?>  
<otrs_config version="1.0" init="Changes">  
  
    <!-- config items will be here -->  
  
</otrs_config>
```

The "init" attribute describes where the config options should be loaded. There are different levels available and will be loaded/overloaded in the following order: "Framework" (for framework settings e. g. session option), "Application" (for application settings e. g. ticket options), "Config" (for extensions to existing applications e. g. ITSM options) and "Changes" (for custom development e. g. to overwrite framework or ticket options).

If you want to add config options, here is an example:

```
<ConfigItem Name="Ticket::Hook" Required="1" Valid="1">  
    <Description Lang="en">The identifier for a ticket. The default is Ticket#.</Description>  
    <Description Lang="de">Ticket-Identifikator. Als Standard wird Ticket# verwendet.</  
Description>  
    <Group>Ticket</Group>  
    <SubGroup>Core::Ticket</SubGroup>  
    <Setting>  
        <String Regex="">Ticket#</String>  
    </Setting>  
</ConfigItem>
```

If "required" is set to "1", the config variable is included and cannot be disabled.

If "valid" is set to "1", the config variable is active. If it is set to "0", the config variable is inactive.

The config variable is defined in the "setting" element.

4.1. Types of XML Config Variables

The XML config settings support various types of variables.

4.2. String

A config element for numbers and single-line strings. Checking the validity with a regex is possible. The check attribute checks elements on the file system. This contains files and directories.

```
<Setting>
  <String Regex="" Check="File"></String>
</Setting>
```

4.3. Textarea

A config element for multiline text.

```
<Setting>
  <TextArea Regex=""></TextArea>
</Setting>
```

4.4. Options

This config element offers preset values as a pull-down menu.

```
<Setting>
  <Option SelectedID="Key">
    <Item Key=""></Item>
    <Item Key=""></Item>
  </Option>
</Setting>
```

4.5. Array

With this config element arrays can be displayed.

```
<Setting>
  <Array>
    <Item></Item>
    <Item></Item>
  </Array>
</Setting>
```

4.6. Hash

With this config element hashes can be displayed.

```
<Setting>
  <Hash>
    <Item Key=""></Item>
    <Item Key=""></Item>
  </Hash>
</Setting>
```

4.7. Hash with SubArray, SubHash

A hash can contain content, arrays or hashes.


```

<Setting>
  <Hash>
    <Item Key=""></Item>
    <Item Key="">
      <Hash>
        <Item Key=""></Item>
        <Item Key=""></Item>
      </Hash>
    </Item>
    <Item Key="">
      <Array>
        <Item></Item>
        <Item></Item>
      </Array>
    </Item>
    <Item Key=""></Item>
  </Hash>
</Setting>

```

4.8. FrontendModuleReg (NavBar)

Module registration for Agent Interface.

```

<Setting>
  <FrontendModuleReg>
    <Group>group1</Group>
    <Group>group2</Group>
    <Description>Logout</Description>
    <Title></Title>
    <NavBarName></NavBarName>
    <NavBar>
      <Description>Logout</Description>
      <Name>Logout</Name>
      <Image>exit.png</Image>
      <Link>Action=Logout</Link>
      <NavBar></NavBar>
      <Type></Type>
      <Block>ItemPre</Block>
      <AccessKey>l</AccessKey>
      <Prio>100</Prio>
    </NavBar>
  </FrontendModuleReg>
</Setting>

```

4.9. FrontendModuleReg (NavBarModule)

Module registration for Admin Interface

```

<Setting>
  <FrontendModuleReg>
    <Group>admin</Group>
    <Group>admin2</Group>
    <Description>Admin</Description>
    <Title>User</Title>
    <NavBarName>Admin</NavBarName>
    <NavBarModule>
      <Module>Kernel::Output::HTML::NavBarModuleAdmin</Module>
      <Name>Users</Name>
      <Block>Block1</Block>
      <Prio>100</Prio>
    </NavBarModule>
  </FrontendModuleReg>
</Setting>

```

```
</FrontendModuleReg>  
</Setting>
```



Chapter 5. Database Mechanism

OTRS comes with a database layer that supports different databases.

1. How it works

The database layer (Kernel::System::DB) has two input options: SQL and XML.

1.1. SQL

The SQL interface should be used for normal database actions (SELECT, INSERT, UPDATE, ...). It can be used like a normal Perl DBI interface.

1.1.1. INSERT/UPDATE/DELETE

```
$Self->{DBObject}->Do(  
    SQL=> "INSERT INTO table (name, id) VALUES ('SomeName', 123)",  
);  
  
$Self->{DBObject}->Do(  
    SQL=> "UPDATE table SET name = 'SomeName', id = 123",  
);  
  
$Self->{DBObject}->Do(  
    SQL=> "DELETE FROM table WHERE id = 123",  
);
```

1.1.2. SELECT

```
my $SQL = "SELECT id FROM table WHERE tn = '123'";  
$Self->{DBObject}->Prepare(SQL => $SQL, Limit => 15);  
while (my @Row = $Self->{DBObject}->FetchrowArray()) {  
    $Id = $Row[0];  
}  
return $Id;
```

Note

Take care to use Limit as param and not in the SQL string because not all databases support LIMIT in SQL strings.

```
my $SQL = "SELECT id FROM table WHERE tn = ? AND group = ?";  
$Self->{DBObject}->Prepare(  
    SQL => $SQL,  
    Limit => 15,  
    Bind => [ $Tn, $Group ],  
);  
while (my @Row = $Self->{DBObject}->FetchrowArray()) {  
    $Id = $Row[0];  
}  
return $Id;
```

Note

Use the Bind attribute where ever you can, especially for long statements. If you use Bind you do not need the function Quote().

1.1.3. QUOTE

String:

```
my $QuotedString = $Self->{DBObject}->Quote("It's a problem!");
```

Integer:

```
my $QuotedInteger = $Self->{DBObject}->Quote('123', 'Integer');
```

Number:

```
my $QuotedNumber = $Self->{DBObject}->Quote('21.35', 'Number');
```

Note

Please use the Bind attribute instead of Quote() where ever you can.

1.2. XML

The XML interface should be used for INSERT, CREATE TABLE, DROP TABLE and ALTER TABLE. As this syntax is different from database to database, using it makes sure that you write applications that can be used in all of them.

Note

The <Insert> has changed in >=2.2. Values are now used in content area (not longer in attribut Value).

1.2.1. INSERT

```
<Insert Table="some_table">
  <Data Key="id">1</Data>
  <Data Key="description" Type="Quote">exploit</Data>
</Insert>
```

1.2.2. CREATE TABLE

Possible data types are: BIGINT, SMALLINT, INTEGER, VARCHAR (Size=1-1000000), DATE (Format: yyyy-mm-dd hh:mm:ss) and LONGBLOB.

```
<TableCreate Name="calendar_event">
  <Column Name="id" Required="true" PrimaryKey="true" AutoIncrement="true" Type="BIGINT"/>
  <Column Name="title" Required="true" Size="250" Type="VARCHAR"/>
  <Column Name="content" Required="false" Size="250" Type="VARCHAR"/>
  <Column Name="start_time" Required="true" Type="DATE"/>
  <Column Name="end_time" Required="true" Type="DATE"/>
  <Column Name="owner_id" Required="true" Type="INTEGER"/>
```

```

<Column Name="event_status" Required="true" Size="50" Type="VARCHAR"/>
<Index Name="calendar_event_title">
  <IndexColumn Name="title"/>
</Index>
<Unique Name="calendar_event_title">
  <UniqueColumn Name="title"/>
</Unique>
<ForeignKey ForeignTable="users">
  <Reference Local="owner_id" Foreign="id"/>
</ForeignKey>
</TableCreate>

```

1.2.3. DROP TABLE

```

<TableDrop Name="calendar_event"/>

```

1.2.4. ALTER TABLE

The following shows an example of add, change and drop columns.

```

<TableAlter Name="calendar_event">
  <ColumnAdd Name="test_name" Type="varchar" Size="20" Required="1"/>

  <ColumnChange NameOld="test_name" NameNew="test_title" Type="varchar" Size="30"
  Required="1"/>

  <ColumnChange NameOld="test_title" NameNew="test_title" Type="varchar" Size="100"
  Required="0"/>

  <ColumnDrop Name="test_title"/>

  <IndexCreate Name="index_test3">
    <IndexColumn Name="test3"/>
  </IndexCreate>

  <IndexDrop Name="index_test3"/>

  <UniqueCreate Name="uniq_test3">
    <UniqueColumn Name="test3"/>
  </UniqueCreate>

  <UniqueDrop Name="uniq_test3"/>
</TableAlter>

```

The next shows an example how to rename a table.

```

<TableAlter NameOld="calendar_event" NameNew="calendar_event_new"/>

```

1.2.5. Code to process XML

```

my @XMLARRAY = @{$Self->ParseXML(String => $XML)};

my @SQL = $Self->{DBObject}->SQLProcessor(
  Database => \@XMLARRAY,
);
push(@SQL, $Self->{DBObject}->SQLProcessorPost());

for (@SQL) {
  $Self->{DBObject}->Do(SQL => $_);
}

```

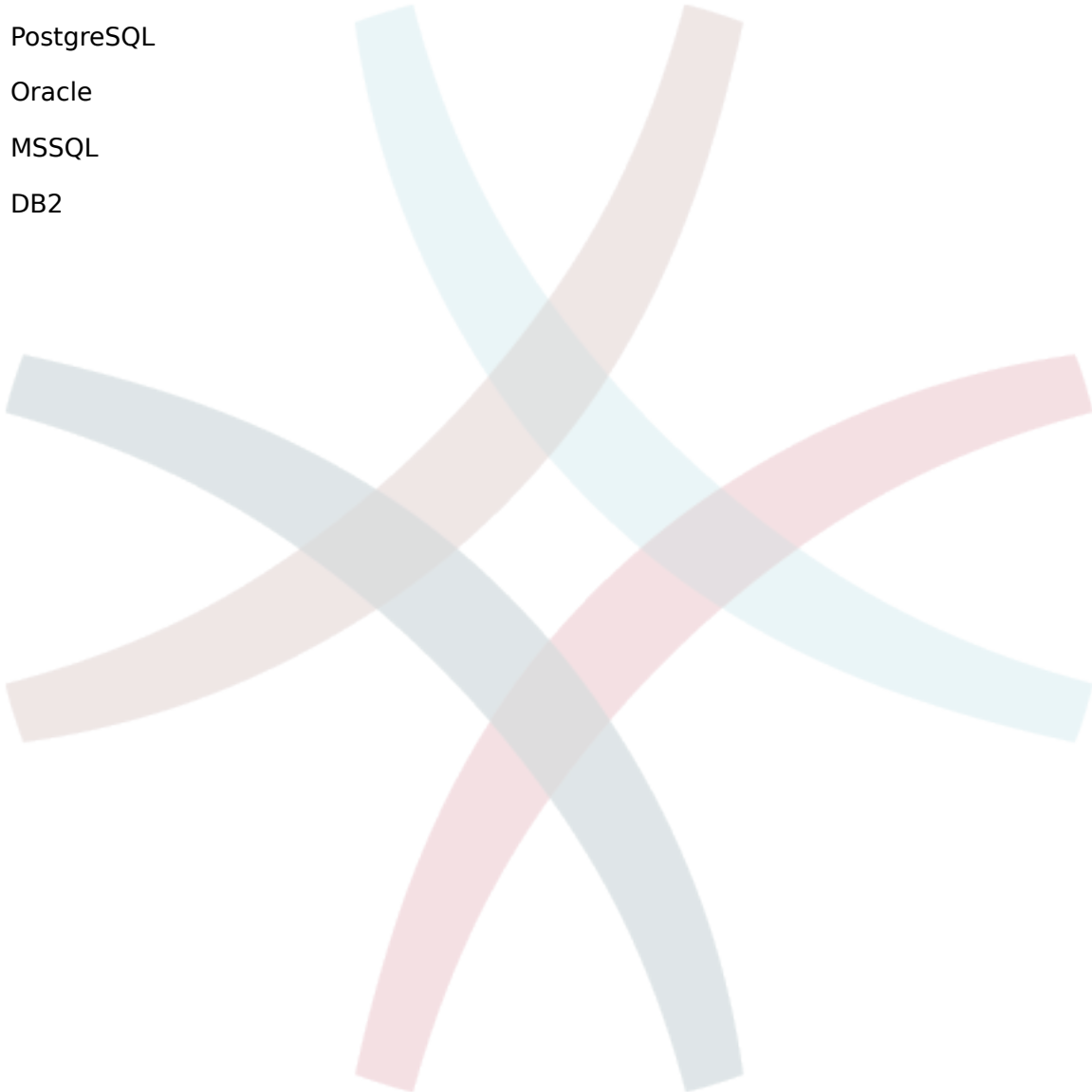
```
}
```

2. Database Drivers

The database drivers are located under `$OTRS_HOME/Kernel/System/DB/*.pm`.

3. Supported Databases

- MySQL
- PostgreSQL
- Oracle
- MSSQL
- DB2



Chapter 6. Log Mechanism

OTRS comes with a log backend that can be used for application logging and debugging.

1. Use and Syntax

All module layers have ready-made Log Objects which can be used by

```
$Self->{LogObject}->Log(  
    Priority => 'error',  
    Message => 'Need something!',  
);
```

2. Example

The following example shows how to use the log mechanism without a module layer.

```
use Kernel::Config;  
use Kernel::System::Encode;  
use Kernel::System::Log;  
  
my $ConfigObject = Kernel::Config->new();  
my $EncodeObject = Kernel::System::Encode->new(  
    ConfigObject => $ConfigObject,  
);  
my $LogObject = Kernel::System::Log->new(  
    ConfigObject => $ConfigObject,  
);  
  
$Self->{LogObject}->Log(  
    Priority => 'error',  
    Message => 'Need something!',  
);
```

Chapter 7. Module Format

1. Core Modules

1.1. Log Module

There is a global log interface for OTRS that provides the possibility to create own log backends.

Writing an own logging backend is as easy as reimplementing the `Kernel::System::Log::Log()` method.

1.1.1. Code example: `Kernel::System::Log::CustomFile`

In this small example, we'll write a little file logging backend which works similar to `Kernel::System::Log::File`, but prepends a string to each logging entry.

```
# --
# Kernel/System/Log/CustomFile.pm - file log backend
# Copyright (C) 2001-2011 OTRS AG, http://otrs.org/
# --
# $Id: log.xml,v 1.4.2.1 2011-01-19 08:39:11 mg Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::Log::CustomFile;

use strict;
use warnings;

use vars qw($VERSION);
$VERSION = qw($Revision: 1.4.2.1 $) [1];

umask "002";

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # get needed objects
    for (qw(ConfigObject EncodeObject)) {
        if ( $Param{$_} ) {
            $Self->{$_} = $Param{$_};
        }
        else {
            die "Got no $_!";
        }
    }

    # get logfile location
    $Self->{LogFile} = '/var/log/CustomFile.log';

    # set custom prefix
    $Self->{CustomPrefix} = 'CustomFileExample';

    # Fixed bug# 2265 - For IIS we need to create a own error log file.
    # Bind stderr to log file, because iis do print stderr to web page.
    if ( $ENV{SERVER_SOFTWARE} && $ENV{SERVER_SOFTWARE} =~ /^microsoft\-iis/i ) {
        if ( !open STDERR, '>>', $Self->{LogFile} . '.error' ) {
```



```

        print STDERR "ERROR: Can't write $Self->{LogFile}.error: $!";
    }
}

return $Self;
}

sub Log {
    my ( $Self, %Param ) = @_;

    my $FH;

    # open logfile
    if ( !open $FH, '>>', $Self->{LogFile} ) {

        # print error screen
        print STDERR "\n";
        print STDERR " >> Can't write $Self->{LogFile}: $! <<\n";
        print STDERR "\n";
        return;
    }

    # write log file
    $Self->{EncodeObject}->SetIO($FH);
    print $FH '[' . localtime() . '];'
    if ( lc $Param{Priority} eq 'debug' ) {
        print $FH "[Debug][$Param{Module}][$Param{Line}] $Self->{CustomPrefix}
$Param{Message}\n";
    }
    elsif ( lc $Param{Priority} eq 'info' ) {
        print $FH "[Info][$Param{Module}] $Self->{CustomPrefix} $Param{Message}\n";
    }
    elsif ( lc $Param{Priority} eq 'notice' ) {
        print $FH "[Notice][$Param{Module}] $Self->{CustomPrefix} $Param{Message}\n";
    }
    elsif ( lc $Param{Priority} eq 'error' ) {
        print $FH "[Error][$Param{Module}][$Param{Line}] $Self->{CustomPrefix}
$Param{Message}\n";
    }
    else {

        # print error messages to STDERR
        print STDERR
            "[Error][$Param{Module}] $Self->{CustomPrefix} Priority: '$Param{Priority}' not
defined! Message: $Param{Message}\n";

        # and of course to logfile
        print $FH
            "[Error][$Param{Module}] $Self->{CustomPrefix} Priority: '$Param{Priority}' not
defined! Message: $Param{Message}\n";
    }

    # close file handle
    close $FH;
    return 1;
}

1;

```

1.1.2. Configuration example

To activate our custom logging module, the administrator can either set the existing configuration item "LogModule" manually to "Kernel::System::Log::CustomFile". To realize this automatically, you can provide an XML configuration file which overrides the default setting.

```
<ConfigItem Name="LogModule" Required="1" Valid="1">
  <Description Translatable="1">Set Kernel::System::Log::CustomFile as default logging
  backend.</Description>
  <Group>Framework</Group>
  <SubGroup>Core::Log</SubGroup>
  <Setting>
    <Option Location="Kernel/System/Log/*.pm"
    SelectedID="Kernel::System::Log::CustomFile"></Option>
  </Setting>
</ConfigItem>
```

1.1.3. Use case examples

Useful logging backends could be logging to a web service or to encrypted files.

1.1.4. Caveats and Warnings

Please note that Kernel::System::Log has other methods than Log() which cannot be reimplemented, for example code for working with shared memory segments and log data caching.

1.2. Output Filter

Output filters allow to modify HTML on the fly. It is best practice to use output filters instead of modifying .dtl files directly. There are three good reasons for that. When the same adaption has to be applied to several frontend modules then the adaption only has to be implemented once. The second advantage is that when OTRS is upgraded there is a chance that the filter doesn't have to be updated, when the relevant pattern has not changed. When two extensions modify the same file there is a conflict during the installation of the second package. This conflict can be resolved by using two output filters that modify the same frontend module.

There are four different kinds of output filters. They are active at different stages of the generation of HTML content.

1.2.1. FilterElementPre

The content of a template can be changed by the filter before any processing by the Layout module takes place. This kind of filter should be used in most cases. Processing instructions like `$Text{"..."}`, `$QData{"..."}` can be inserted into the template content and they will be honored by the subsequent DTL processing.

1.2.2. FilterElementPost

The content of a template can be changed after variable substitution and translation. The kind of filter should only be used when the filter needs access to translated strings or to substituted variables.

1.2.3. FilterContent

This kind of filter allows to process the complete HTML output for the request right before it is sent to the browser. This can be used for global transformations. But in real live there is rarely a need to use this kind of filter.

1.2.4. FilterText

This kind of output filter is a plugin for the method Kernel::Output::HTML::Layout::Ascii2HTML() and is only active when the parameter LinkFeature is set to 1. Thus the FilterText output filters are currently only active for the display of the body of plain text articles. Plain text articles are generated by incoming non-HTML mails and when OTRS is configured to not use the rich text feature in the frontend.

1.2.5. Code example

See package TemplateModule.

1.2.6. Configuration example

See package TemplateModule.

1.2.7. Use Cases

1.2.7.1. Show additional ticket attributes in AgentTicketZoom.

All ticket attributes are passed to the AgentTicketZoom template. Therefore it suffices to insert e.g. the instruction `$QData{"Title"}` into the content. This can be achieved with a `FilterElementPre` output filter.

1.2.7.2. Add an additional CSS file.

An additional CSS file can be added to all agent frontends with an `FilterElementPre` filter that only modifies `Header.dtl`. Therefore it suffices to insert e.g. the instruction `$QData{"Title"}` into the content. This can be achieved with a `FilterElementPre` output filter.

1.2.7.3. Show the service selection as a multi level menu.

Use a `FilterElementPost` for this feature. The list of selectable services can be parsed from the processed template output. The multi level selection can be constructed from the service list and inserted into the template content. A `FilterElementPost` output filter must be used for that.

1.2.7.4. Create links within plain text article bodies.

A biotech company uses gene names like IPI00217472 in plain text articles. A `FilterText` output filter can be used to create links to a sequence database, e.g. `http://srs.ebi.ac.uk/srsbin/cgi-bin/wgetz?-e+[IPI-acc:IPI00217472]+-vn+2`, for the gene names.

1.2.7.5. Prohibit active content

There is firewall rule that disallows all active content. In order to avoid rejection by the firewall the HTML tag `<applet>` can be filtered with an `FilterContent` output filter.

1.2.8. Caveats and Warnings

Every `ElementPre` and `ElementPost` output filter is constructed and run for every Template that is needed for the current request. Thus low performance of the output filter or a large number of filters can severely degrade performance. When that becomes an issue, the construction of needed objects can be done in the `Run`-method after the checks. Thus the expensive code is run only in the relevant cases.

1.2.9. Best Practices

In order to increase flexibility the list of affected templates should be configurable in `SysConfig`.

1.2.10. Release Availability

The four kinds of output filters are available in OTRS 2.4.

1.3. Stats Module

There are two different types of internal stats modules - dynamic and static. This section describes how such stats modules can be developed.

1.3.1. Dynamic Stats

In contrast to static stats modules, dynamic statistics can be configured via the OTRS web interface. In this section a simple statistic module is developed. Each dynamic stats module has to implement these subroutines

- new
- GetObjectName
- GetObjectAttributes
- ExportWrapper
- ImportWrapper

Furthermore the module has to implement either GetStatElement or GetStatTable. And if the header line of the result table should be changed, a sub called GetHeaderLine has to be developed.

1.3.1.1. Code example

In this section a sample stats module is shown and each subroutine is explained.

```
# --
# Kernel/System/Stats/Dynamic/DynamicStatsTemplate.pm - all advice functions
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::Stats::Dynamic::DynamicStatsTemplate;

use strict;
use warnings;

use Kernel::System::Queue;
use Kernel::System::State;
use Kernel::System::Ticket;

use vars qw($VERSION);
$VERSION = qw($Revision: 1.3.2.1 $) [1];
```

This is common boilerplate that can be found in common OTRS modules. The class/package name is declared via the package keyword. Then the needed modules are 'use'd.

```
sub new {
    my ( $Type, %Param ) = @_ ;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for my $Object (
        qw(DBObject ConfigObject LogObject UserObject TimeObject MainObject EncodeObject)
    )
    {
        $Self->{$Object} = $Param{$Object} || die "Got no $Object!";
    }
}
```

```

# created needed objects
$Self->{QueueObject} = Kernel::System::Queue->new( %{$Self} );
$Self->{TicketObject} = Kernel::System::Ticket->new( %{$Self} );
$Self->{StateObject} = Kernel::System::State->new( %{$Self} );

return $Self;
}

```

new is the constructor for this statistic module. It creates a new instance of the class. According to the coding guidelines objects of other classes that are needed in this module have to be created in "new". In lines 27 to 29 the object of the stats module is created. Lines 31 to 37 check if objects that are needed in this code - either for creating other objects or in this module - are passed. After that the other objects are created.

```

sub GetObjectName {
    my ( $Self, %Param ) = @_;

    return 'Sample Statistics';
}

```

GetObjectName returns a Name for the Statistics module. This is the label that is shown in the drop down in the configuration as well as in the list of existing statistics (column "object").

```

sub GetObjectAttributes {
    my ( $Self, %Param ) = @_;

    # get state list
    my %StateList = $Self->{StateObject}->StateList(
        UserID => 1,
    );

    # get queue list
    my %QueueList = $Self->{QueueObject}->GetAllQueues();

    # get current time to fix bug#3830
    my $TimeStamp = $Self->{TimeObject}->CurrentTimestamp();
    my ($Date) = split /\s+/, $TimeStamp;
    my $Today = sprintf "%s 23:59:59", $Date;

    my @ObjectAttributes = (
        {
            Name           => 'State',
            UseAsXvalue    => 1,
            UseAsValueSeries => 1,
            UseAsRestriction => 1,
            Element        => 'StateIDs',
            Block          => 'MultiSelectField',
            Values         => \%StateList,
        },
        {
            Name           => 'Created in Queue',
            UseAsXvalue    => 1,
            UseAsValueSeries => 1,
            UseAsRestriction => 1,
            Element        => 'CreatedQueueIDs',
            Block          => 'MultiSelectField',
            Translation    => 0,
            Values         => \%QueueList,
        },
        {
            Name           => 'Create Time',

```

```

    UseAsXvalue      => 1,
    UseAsValueSeries => 1,
    UseAsRestriction => 1,
    Element          => 'CreateTime',
    TimePeriodFormat => 'DateInputFormat',    # 'DateInputFormatLong',
    Block            => 'Time',
    TimeStop         => $Today,
    Values           => {
        TimeStart => 'TicketCreateTimeNewerDate',
        TimeStop  => 'TicketCreateTimeOlderDate',
    },
  },
);

return @ObjectAttributes;
}

```

In this sample stats module, we want to provide three attributes the user can choose from: A list of queues, a list of states and a time drop down. To get the values shown in the drop down, some operations are needed. In this case call StateList and GetAllQueues.

Then the list of attributes is created. Each attribute is defined via a hashreference. You can use these keys:

- Name
the label in the web interface
- UseAsXvalue
Can this attribute be used on the x-axis
- UseAsValueSeries
Can this attribute be used on the y-axis
- UseAsRestriction
Can this attribute be used for restrictions.
- Element
the HTML fieldname
- Block
the block name in the template file (e.g. <OTRS_HOME>/Kernel/Output/HTML/Standard/AgentStatsEditXaxis.dtl)
- Values
the values shown in the attribute

Hint: If you install this sample and you configured a statistic with some queues - let's say 'queue A' and 'queue B' - then these queues are the only ones that are shown to the user when he starts the statistic. Sometimes a dynamic drop down or multiselect field is needed. In this case, you can set "SelectedValues" in the definition of the attribute:

```

{
  Name          => 'Created in Queue',

```

```

    UseAsXvalue      => 1,
    UseAsValueSeries => 1,
    UseAsRestriction => 1,
    Element          => 'CreatedQueueIDs',
    Block            => 'MultiSelectField',
    Translation      => 0,
    Values           => \%QueueList,
    SelectedValues   => [ @SelectedQueues ],
},

```

```

sub GetStatElement {
    my ( $Self, %Param ) = @_;

    # search tickets
    return $Self->{TicketObject}->TicketSearch(
        UserID      => 1,
        Result      => 'COUNT',
        Permission  => 'ro',
        Limit       => 100_000_000,
        %Param,
    );
}

```

GetStatElement gets called for each cell in the result table. So it should be a numeric value. In this sample it does a simple ticket search. The hash %Param contains information about the "current" x-value and the y-value as well as any restrictions. So, for a cell that should count the created tickets for queue 'Misc' with state 'open' the passed parameter hash looks something like this:

```

'CreatedQueueIDs' => [
    '4'
],
'StateIDs' => [
    '2'
]

```

If the "per cell" calculation should be avoided, GetStatTable is an alternative. GetStatTable returns a list of rows, hence an array of arrayreferences. This leads to the same result as using GetStatElement

```

sub GetStatTable {
    my ( $Self, %Param ) = @_;

    my @StatData;

    for my $StateName ( keys %{ $Param{TableStructure} } ) {
        my @Row;
        for my $Params ( @{ $Param{TableStructure}->{$StateName} } ) {
            my $Tickets = $Self->{TicketObject}->TicketSearch(
                UserID      => 1,
                Result      => 'COUNT',
                Permission  => 'ro',
                Limit       => 100_000_000,
                %{$Params},
            );

            push @Row, $Tickets;
        }
    }
}

```

```

    push @StatData, [ $StateName, @Row ];
  }

  return @StatData;
}

```

GetStatTable gets all information about the stats query that is needed. The passed parameters contains information about the attributes (Restrictions, attributes that are used for x/y-axis) and the table structure. The table structure is a hash reference where the keys are the values of the y-axis and their values are hashreferences with the parameters used for GetStatElement subroutines.

```

'Restrictions' => {},
'TableStructure' => {
  'closed successful' => [
    {
      'CreatedQueueIDs' => [
        '3'
      ],
      'StateIDs' => [
        '2'
      ]
    },
  ],
  'closed unsuccessful' => [
    {
      'CreatedQueueIDs' => [
        '3'
      ],
      'StateIDs' => [
        '3'
      ]
    },
  ],
},
'ValueSeries' => [
  {
    'Block' => 'MultiSelectField',
    'Element' => 'StateIDs',
    'Name' => 'State',
    'SelectedValues' => [
      '5',
      '3',
      '2',
      '1',
      '4'
    ],
    'Translation' => 1,
    'Values' => {
      '1' => 'new',
      '10' => 'closed with workaround',
      '2' => 'closed successful',
      '3' => 'closed unsuccessful',
      '4' => 'open',
      '5' => 'removed',
      '6' => 'pending reminder',
      '7' => 'pending auto close+',
      '8' => 'pending auto close-',
      '9' => 'merged'
    }
  }
],
'XValue' => {
  'Block' => 'MultiSelectField',

```



```

'Element' => 'CreatedQueueIDs',
'Name' => 'Created in Queue',
'SelectedValues' => [
  '3',
  '4',
  '1',
  '2'
],
'Translation' => 0,
'Values' => {
  '1' => 'Postmaster',
  '2' => 'Raw',
  '3' => 'Junk',
  '4' => 'Misc'
}
}
}

```

Sometimes the headers of the table have to be changed. In that case, a subroutine called `GetHeaderLine` has to be implemented. That subroutine has to return an arrayreference with the column headers as elements. It gets information about the x-values passed.

```

sub GetHeaderLine {
  my ( $Self, %Param ) = @_;

  my @HeaderLine = ( '' );
  for my $SelectedXValue ( @ { $Param{XValue}->{SelectedValues} } ) {
    push @HeaderLine, $Param{XValue}->{Values}->{$SelectedXValue};
  }

  return \@HeaderLine;
}

```

```

sub ExportWrapper {
  my ( $Self, %Param ) = @_;

  # wrap ids to used spelling
  for my $Use (qw(UseAsValueSeries UseAsRestriction UseAsXvalue)) {
    ELEMENT:
    for my $Element ( @ { $Param{$Use} } ) {
      next ELEMENT if !$Element || !$Element->{SelectedValues};
      my $ElementName = $Element->{Element};
      my $Values      = $Element->{SelectedValues};

      if ( $ElementName eq 'QueueIDs' || $ElementName eq 'CreatedQueueIDs' ) {
        ID:
        for my $ID ( @ { $Values } ) {
          next ID if !$ID;
          $ID->{Content} = $Self->{QueueObject}->QueueLookup( QueueID => $ID-
>{Content} );
        }
      }
      elsif ( $ElementName eq 'StateIDs' || $ElementName eq 'CreatedStateIDs' ) {
        my %StateList = $Self->{StateObject}->StateList( UserID => 1 );
        ID:
        for my $ID ( @ { $Values } ) {
          next ID if !$ID;
          $ID->{Content} = $StateList{ $ID->{Content} };
        }
      }
    }
  }
  return \%Param;
}

```

Configured statistics can be exported into XML format. But as queues with the same queue names can have different IDs on different OTRS instances it would be quite painful to export the IDs (the statistics would calculate the wrong numbers then). So an export wrapper should be written to use the names instead of ids. This should be done for each "dimension" of the stats module (x-axis, y-axis and restrictions).

ImportWrapper works the other way around - it converts the name to the ID in the instance the configuration is imported to.

This is a sample export:

```
<?xml version="1.0" encoding="utf-8"?>
<otrs_stats>
<Cache>0</Cache>
<Description>Sample stats module</Description>
<File></File>
<Format>CSV</Format>
<Format>Print</Format>
<Object>DeveloperManualSample</Object>
<ObjectModule>Kernel::System::Stats::Dynamic::DynamicStatsTemplate</ObjectModule>
<ObjectName>Sample Statistics</ObjectName>
<Permission>stats</Permission>
<StatType>dynamic</StatType>
<SumCol>0</SumCol>
<SumRow>0</SumRow>
<Title>Sample 1</Title>
<UseAsValueSeries Element="StateIDs" Fixed="1">
<SelectedValues>removed</SelectedValues>
<SelectedValues>closed unsuccessful</SelectedValues>
<SelectedValues>closed successful</SelectedValues>
<SelectedValues>new</SelectedValues>
<SelectedValues>open</SelectedValues>
</UseAsValueSeries>
<UseAsXvalue Element="CreatedQueueIDs" Fixed="1">
<SelectedValues>Junk</SelectedValues>
<SelectedValues>Misc</SelectedValues>
<SelectedValues>Postmaster</SelectedValues>
<SelectedValues>Raw</SelectedValues>
</UseAsXvalue>
<Valid>1</Valid>
</otrs_stats>
```

Now, that all subroutines are explained, this is the complete sample stats module.

```
# --
# Kernel/System/Stats/Dynamic/DynamicStatsTemplate.pm - all advice functions
# Copyright (C) 2001-2010 OTRS AG, http://otrs.org/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::Stats::Dynamic::DynamicStatsTemplate;

use strict;
use warnings;

use Kernel::System::Queue;
use Kernel::System::State;
```

```

use Kernel::System::Ticket;

use vars qw($VERSION);
$VERSION = qw($Revision: 1.3.2.1 $) [1];

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless( $Self, $Type );

    # check needed objects
    for my $Object (
        qw(DBObject ConfigObject LogObject UserObject TimeObject MainObject EncodeObject)
    )
    {
        $Self->{$Object} = $Param{$Object} || die "Got no $Object!";
    }

    # created needed objects
    $Self->{QueueObject} = Kernel::System::Queue->new( %{$Self} );
    $Self->{TicketObject} = Kernel::System::Ticket->new( %{$Self} );
    $Self->{StateObject} = Kernel::System::State->new( %{$Self} );

    return $Self;
}

sub GetObjectName {
    my ( $Self, %Param ) = @_;

    return 'Sample Statistics';
}

sub GetObjectAttributes {
    my ( $Self, %Param ) = @_;

    # get state list
    my %StateList = $Self->{StateObject}->StateList(
        UserID => 1,
    );

    # get queue list
    my %QueueList = $Self->{QueueObject}->GetAllQueues();

    # get current time to fix bug#3830
    my $TimeStamp = $Self->{TimeObject}->CurrentTimestamp();
    my ($Date) = split /\s+/, $TimeStamp;
    my $Today = sprintf "%s 23:59:59", $Date;

    my @ObjectAttributes = (
        {
            Name           => 'State',
            UseAsXvalue    => 1,
            UseAsValueSeries => 1,
            UseAsRestriction => 1,
            Element        => 'StateIDs',
            Block          => 'MultiSelectField',
            Values         => \%StateList,
        },
        {
            Name           => 'Created in Queue',
            UseAsXvalue    => 1,
            UseAsValueSeries => 1,
            UseAsRestriction => 1,
            Element        => 'CreatedQueueIDs',
            Block          => 'MultiSelectField',
        }
    );
}

```

```

        Translation      => 0,
        Values           => \%QueueList,
    },
    {
        Name              => 'Create Time',
        UseAsXvalue       => 1,
        UseAsValueSeries => 1,
        UseAsRestriction => 1,
        Element           => 'CreateTime',
        TimePeriodFormat => 'DateInputFormat',    # 'DateInputFormatLong',
        Block             => 'Time',
        TimeStop          => $Today,
        Values            => {
            TimeStart => 'TicketCreateTimeNewerDate',
            TimeStop  => 'TicketCreateTimeOlderDate',
        },
    },
);

return @ObjectAttributes;
}

sub GetStatElement {
    my ( $Self, %Param ) = @_;

    # search tickets
    return $Self->{TicketObject}->TicketSearch(
        UserID      => 1,
        Result      => 'COUNT',
        Permission  => 'ro',
        Limit       => 100_000_000,
        %Param,
    );
}

sub ExportWrapper {
    my ( $Self, %Param ) = @_;

    # wrap ids to used spelling
    for my $Use (qw(UseAsValueSeries UseAsRestriction UseAsXvalue)) {
        ELEMENT:
        for my $Element ( @ { $Param{$Use} } ) {
            next ELEMENT if !$Element || !$Element->{SelectedValues};
            my $ElementName = $Element->{Element};
            my $Values      = $Element->{SelectedValues};

            if ( $ElementName eq 'QueueIDs' || $ElementName eq 'CreatedQueueIDs' ) {
                ID:
                for my $ID ( @ { $Values } ) {
                    next ID if !$ID;
                    $ID->{Content} = $Self->{QueueObject}->QueueLookup( QueueID => $ID-
>{Content} );
                }
            }
            elsif ( $ElementName eq 'StateIDs' || $ElementName eq 'CreatedStateIDs' ) {
                my %StateList = $Self->{StateObject}->StateList( UserID => 1 );
                ID:
                for my $ID ( @ { $Values } ) {
                    next ID if !$ID;
                    $ID->{Content} = $StateList{ $ID->{Content} };
                }
            }
        }
    }
    return \%Param;
}

```

```

sub ImportWrapper {
    my ( $Self, %Param ) = @_;

    # wrap used spelling to ids
    for my $Use (qw(UseAsValueSeries UseAsRestriction UseAsXvalue)) {
        ELEMENT:
        for my $Element ( @{$Param{$Use}} ) {
            next ELEMENT if !$Element || !$Element->{SelectedValues};
            my $ElementName = $Element->{Element};
            my $Values      = $Element->{SelectedValues};

            if ( $ElementName eq 'QueueIDs' || $ElementName eq 'CreatedQueueIDs' ) {
                ID:
                for my $ID ( @{$Values} ) {
                    next ID if !$ID;
                    if ( $Self->{QueueObject}->QueueLookup( Queue => $ID->{Content} ) ) {
                        $ID->{Content}
                        = $Self->{QueueObject}->QueueLookup( Queue => $ID->{Content} );
                    }
                    else {
                        $Self->{LogObject}->Log(
                            Priority => 'error',
                            Message => "Import: Can' find the queue $ID->{Content}!"
                        );
                        $ID = undef;
                    }
                }
            }
            elsif ( $ElementName eq 'StateIDs' || $ElementName eq 'CreatedStateIDs' ) {
                ID:
                for my $ID ( @{$Values} ) {
                    next ID if !$ID;

                    my %State = $Self->{StateObject}->StateGet(
                        Name => $ID->{Content},
                        Cache => 1,
                    );
                    if ( $State{ID} ) {
                        $ID->{Content} = $State{ID};
                    }
                    else {
                        $Self->{LogObject}->Log(
                            Priority => 'error',
                            Message => "Import: Can' find state $ID->{Content}!"
                        );
                        $ID = undef;
                    }
                }
            }
        }
    }
    return \%Param;
}
1;

```

1.3.1.2. Configuration example

```

<?xml version="1.0" encoding="iso-8859-1" ?>
<otrs_config version="1.0" init="Config">
  <ConfigItem Name="Stats::DynamicObjectRegistration###DynamicStatsTemplate" Required="0"
    Valid="1">
    <Description Lang="en">Here you can decide if the common stats module may generate
    stats about the number of default tickets a requester created.</Description>
  </ConfigItem>
</otrs_config>

```

```

    <Description Lang="de">Hier können Sie festlegen, ob das Statistik-Modul auch
    allgemeine Statistiken über die Anzahl von "default"-Tickets, die von Benutzern erzeugt
    wurden, generieren darf.</Description>
    <Group>Framework</Group>
    <SubGroup>Core::Stats</SubGroup>
    <Setting>
      <Hash>
        <Item Key="Module">Kernel::System::Stats::Dynamic::DynamicStatsTemplate</Item>
      </Hash>
    </Setting>
  </ConfigItem>
</otrs_config>

```

1.3.1.3. Use case examples

Use cases.

1.3.1.4. Caveats and Warnings

If you have a lot of cells in the result table and the GetStatElement is quite complex, the request can take a long time.

1.3.1.5. Release Availability

Dynamic stat modules are available since OTRS 2.0.

1.3.2. Static Stats

The subsequent paragraphs describe the static stats.

1.3.2.1. Use case examples

Use cases.

1.3.2.2. Caveats and Warnings

Caveats and Warnings for static stats.

1.3.2.3. Release Availability

Static stat modules are available since OTRS 1.3.

1.3.2.4. Using old static stats

Standard OTRS versions 1.3 and 2.0 already facilitated the generation of stats. Various stats for OTRS versions 1.3 and 2.0 which have been specially developed to meet customers' requirements can be used in more recent versions too.

The files must merely be moved from the Kernel/System/Stats/ path to Kernel/System/Stats/Static/. Additionally the package name of the respective script must be amended by "::Static".

The following example shows how the first path is amended.

```
package Kernel::System::Stats::AccountedTime;
```

```
package Kernel::System::Stats::Static::AccountedTime;
```

2. Frontend Modules

2.1. Dashboard Module

Dashboard module to display statistics in the form of a line graph.

```
# --
# Kernel/Output/HTML/DashboardTicketStatsGeneric.pm - message of the day
# Copyright (C) 2001-2011 OTRS AG, http://otrs.org/
# --
# $Id: dashboard.xml,v 1.6.2.1 2011-01-19 08:39:11 mg Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::Output::HTML::DashboardTicketStatsGeneric;

use strict;
use warnings;

use vars qw($VERSION);
$VERSION = qw($Revision: 1.6.2.1 $) [1];

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {%Param};
    bless( $Self, $Type );

    # get needed objects
    for (
        qw(Config Name ConfigObject LogObject DBObject LayoutObject ParamObject TicketObject
        UserID)
    )
    {
        die "Got no $_!" if !$Self->{$_};
    }

    return $Self;
}

sub Preferences {
    my ( $Self, %Param ) = @_;

    return;
}

sub Config {
    my ( $Self, %Param ) = @_;

    my $Key = $Self->{LayoutObject}->{UserLanguage} . '-' . $Self->{Name};
    return (
        %{ $Self->{Config} },
        CacheKey => 'TicketStats' . '-' . $Self->{UserID} . '-' . $Key,
    );
}

sub Run {
    my ( $Self, %Param ) = @_;
```

```

my %Axis = (
  '7Day' => {
    0 => { Day => 'Sun', Created => 0, Closed => 0, },
    1 => { Day => 'Mon', Created => 0, Closed => 0, },
    2 => { Day => 'Tue', Created => 0, Closed => 0, },
    3 => { Day => 'Wed', Created => 0, Closed => 0, },
    4 => { Day => 'Thu', Created => 0, Closed => 0, },
    5 => { Day => 'Fri', Created => 0, Closed => 0, },
    6 => { Day => 'Sat', Created => 0, Closed => 0, },
  },
);

my @Data;
my $Max = 1;
for my $Key ( 0 .. 6 ) {

  my $TimeNow = $Self->{TimeObject}->SystemTime();
  if ($Key) {
    $TimeNow = $TimeNow - ( 60 * 60 * 24 * $Key );
  }
  my ( $Sec, $Min, $Hour, $Day, $Month, $Year, $WeekDay )
    = $Self->{TimeObject}->SystemTime2Date(
      SystemTime => $TimeNow,
    );

  $Data[$Key]->{Day} = $Self->{LayoutObject}->{LanguageObject}->Get(
    $Axis{'7Day'}->{$WeekDay}->{Day}
  );

  my $CountCreated = $Self->{TicketObject}->TicketSearch(

    # cache search result 20 min
    CacheTTL => 60 * 20,

    # tickets with create time after ... (ticket newer than this date) (optional)
    TicketCreateTimeNewerDate => "$Year-$Month-$Day 00:00:00",

    # tickets with created time before ... (ticket older than this date) (optional)
    TicketCreateTimeOlderDate => "$Year-$Month-$Day 23:59:59",

    CustomerID => $Param{Data}->{UserCustomerID},
    Result      => 'COUNT',

    # search with user permissions
    Permission => $Self->{Config}->{Permission} || 'ro',
    UserID    => $Self->{UserID},
  );
  $Data[$Key]->{Created} = $CountCreated;
  if ( $CountCreated > $Max ) {
    $Max = $CountCreated;
  }

  my $CountClosed = $Self->{TicketObject}->TicketSearch(

    # cache search result 20 min
    CacheTTL => 60 * 20,

    # tickets with create time after ... (ticket newer than this date) (optional)
    TicketCloseTimeNewerDate => "$Year-$Month-$Day 00:00:00",

    # tickets with created time before ... (ticket older than this date) (optional)
    TicketCloseTimeOlderDate => "$Year-$Month-$Day 23:59:59",

    CustomerID => $Param{Data}->{UserCustomerID},
    Result      => 'COUNT',

    # search with user permissions

```



```

    Permission => $Self->{Config}->{Permission} || 'ro',
    UserID => $Self->{UserID},
  );
  $Data[$Key]->{Closed} = $CountClosed;
  if ( $CountClosed > $Max ) {
    $Max = $CountClosed;
  }
}

@Data = reverse @Data;
my $Source = $Self->{LayoutObject}->JSON(
  Data => \@Data,
);

my $Content = $Self->{LayoutObject}->Output(
  TemplateFile => 'AgentDashboardTicketStats',
  Data => {
    %{ $Self->{Config} },
    Key => int rand 99999,
    Max => $Max,
    Source => $Source,
  },
);

return $Content;
}
1;

```

To use this module add the following to the Kernel/Config.pm and restart your webserver (if you use mod_perl).

```

<ConfigItem Name="DashboardBackend###0250-TicketStats" Required="0" Valid="1">
  <Description Lang="en">Parameters for the dashboard backend. "Group" are used to restricted
  access to the plugin (e. g. Group: admin;group1;group2;). "Default" means if the plugin is
  enabled per default or if the user needs to enable it manually. "CacheTTL" means the cache
  time in minutes for the plugin.</Description>
  <Description Lang="de">Parameter für das Dashboard Backend. "Group" ist verwendet um
  den Zugriff auf das Plugin einzuschränken (z. B. Group: admin;group1;group2;). ""Default"
  bedeutet ob das Plugin per default aktiviert ist oder ob dies der Anwender manuell machen
  muss. "CacheTTL" ist die Cache-Zeit in Minuten nach der das Plugin erneut aufgerufen wird.</
  Description>
  <Group>Ticket</Group>
  <SubGroup>Frontend::Agent::Dashboard</SubGroup>
  <Setting>
    <Hash>
      <Item Key="Module">Kernel::Output::HTML::DashboardTicketStatsGeneric</Item>
      <Item Key="Title">7 Day Stats</Item>
      <Item Key="Created">1</Item>
      <Item Key="Closed">1</Item>
      <Item Key="Permission">rw</Item>
      <Item Key="Block">ContentSmall</Item>
      <Item Key="Group"></Item>
      <Item Key="Default">1</Item>
      <Item Key="CacheTTL">45</Item>
    </Hash>
  </Setting>
</ConfigItem>

```

2.1.1. Caveats and Warnings

An excessive number of days or individual lines may lead to performance degradation.

2.1.2. Release Availability

from 2.4.0

3. Old Module Descriptions

Please remove these old sections if newer ones were created.

3.1. Auth Module

There are several agent authentication modules (DB, LDAP and HTTPBasicAuth) which come with the OTRS framework. It is also possible to develop your own authentication modules. The agent authentication modules are located under Kernel/System/Auth/*.pm. For more information about their configuration see the admin manual. Following, there is an example of a simple agent auth module. Save it under Kernel/System/Auth/Simple.pm. You just need 3 functions: new(), GetOption() and Auth(). Return the uid, then the authentication is ok.

Format:

```
# --
# Kernel/System/Auth/Simple.pm - provides the simple authentication
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.39 2010-05-10 18:54:49 bes Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::Auth::Simple;

use strict;
use warnings;

sub new {
    my ( $Type, %Param ) = @_;
    [...]
    return $Self;
}

sub GetOption {
    my ( $Self, %Param ) = @_;
    # return option
    return (PreAuth => 0);
}

sub Auth {
    my ( $Self, %Param ) = @_;
    [...]
    if ($Authentication) {
        return $Param{User};
    }
    else {
        return;
    }
}
}
```

3.2. Notify Module

With agent notification modules you can inform agents about new information. A normal navigation looks like the following screen shot.

[OTRS] Mart Ede (me@bogen.net) Wed Dec 03 19:05:14 2003

Logout QueueView PhoneView Utilities Preferences Stats new message (0) Locked tickets (0)

[Queue: Raw]

Tickets shown: 1 - Page: 1 - Tickets available: 1 - All tickets: 1

With this exemplary module you can do something like the following.

[OTRS] Mart Ede (me@bogen.net) Wed Dec 03 19:09:25 2003

Logout QueueView PhoneView Utilities Preferences Stats new message (0) Locked tickets (0)

⚠ : Some daily news! ←

[Queue: Raw]

Tickets shown: 1 - Page: 1 - Tickets available: 1 - All tickets: 1

Format:

```
# --
# Kernel/Output/HTML/NotificationMotd.pm - message of the day
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.39 2010-05-10 18:54:49 bes Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::Output::HTML::NotificationMotd;

use strict;
use warnings;

# --
sub new {
    my ( $Type, %Param ) = @_;
    [...]
    return $Self;
}
# --
sub Run {
    my ( $Self, %Param ) = @_;
    return $Self->{LayoutObject}->Notify(Info => 'Some daily news!' );
}
# --
1;
```

To use this module add the following to the Kernel/Config.pm and restart your webserver (if you use mod_perl).

```
# Frontend::NotifyModule - module name (50-Motd)
$Self->{'Frontend::NotifyModule'}->{'50-Motd'} = {
    Module => 'Kernel::Output::HTML::NotificationMotd',
};
```

3.3. Navigation Module

In this module layer you can create dynamic navigation bar items with dynamic content (Name and Description). Navigation Module are located under Kernel/Output/HTML/NavBar*.pm.

Format:

```
# --
# Kernel/Output/HTML/NavBarABC.pm - shows a navbar item dynamically
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.39 2010-05-10 18:54:49 bes Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::Output::HTML::NavBarABC;

use strict;
use warnings;

# --
sub new {
    my ( $Type, %Param ) = @_;
    [...]
    return $Self;
}
# --
sub Run {
    my ( $Self, %Param ) = @_;
    my %Return = ();
    $Return{'0999989'} = {
        Block      => 'ItemPersonal',
        Description => 'Some Descripton',
        Name       => 'Text',
        Image      => 'new-message.png',
        Link       => 'Action=AgentMailbox&Subaction=New',
        AccessKey  => 'j',
    };
    return %Return;
}
# --
1;
```

To use this module add the following code to the Kernel/Config.pm and restart your webserver (if you use mod_perl).

```
[Kernel/Config.pm]
# agent interface notification module
$self->{'Frontend::NavBarModule'}->{'99-ABC'} = {
    Module => 'Kernel::Output::HTML::NavBarABC',
};
```

3.4. Frontend Modules

Frontend Modules are located under "\$OTRS_HOME/Kernel/Modules/*.pm". There are two public functions in there - new() and run() - which are accessed from the Frontend Handle (e. g. index.pl). "new()" is used to create a frontend module object. The Frontend Handle

provides the used frontend module with the basic framework object. These are, for example: ParamObject (to get formular params), DBObject (to use existing database connects), LayoutObject (to use templates and other html layout functions), ConfigObject (to access config settings), LogObject (to use the framework log system), UserObject (to get the user functions from the current user), GroupObject (to get the group functions), MainObject (to get main functions like 'Require') and EncodeObject (for the charset encoding).

For more information on core modules see <http://dev.otrs.org/>

Format:

```
# --
# Kernel/Modules/AgentTest.pm - message of the day
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.39 2010-05-10 18:54:49 bes Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::Modules::AgentTest;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.39 $) [1];

# --
sub new {
    my ( $Type, %Param ) = @_;
    [...]
    return $Self;
}
# --
sub Run {
    my ( $Self, %Param ) = @_;
    [...]
    # ----- #
    # add a new object (Note: dtl text 'New')
    # ----- #
    if ( $Self->{Subaction} eq 'Add' ) {
        my $Output = '';
        my %Frontend = ();
        [...]
        # add add block
        $Self->{LayoutObject}->Block(
            Name => 'Add',
            Data => { %Param, %Frontend },
        );
        # build output
        $Output .= $Self->{LayoutObject}->Header( Area => 'Agent', Title => "Test" );
        $Output .= $Self->{LayoutObject}->NavigationBar();
        $Output .= $Self->{LayoutObject}->Output(
            Data => { %Param, %Frontend },
            TemplateFile => 'AgentTest',
        );
        $Output .= $Self->{LayoutObject}->Footer();
        return $Output;
    }
    # ----- #
    # show overview screen
    # ----- #
}
```

```

elseif ($Self->{Subaction} eq 'Overview') {
    # add overview block
    $Self->{LayoutObject}->Block(
        Name => 'Overview',
        Data => {%Param, %Frontend},
    );
    # build output
    $Output .= $Self->{LayoutObject}->Header(Area => 'Agent', Title => "Test");
    $Output .= $Self->{LayoutObject}->NavigationBar();
    $Output .= $Self->{LayoutObject}->Output(
        Data => {%Param, %Frontend},
        TemplateFile => 'AgentTest',
    );
    $Output .= $Self->{LayoutObject}->Footer();
    return $Output;
}
# ----- #
# show error screen
# ----- #
return $Self->{LayoutObject}->ErrorScreen(Message => "Invalid Subaction process!");
}
# --
1;

```

You also need a module registration for frontend modules. Define read only groups with the "GroupRo" and read/write groups with the 'Group' param (see table below for details). You can define navigation bar icons via the "NavBar" param, too (see table below for details).

```

[Kernel/Config.pm]
$Self->{'Frontend::Module'}->{'AgentTest'} = {
    Group => ['admin'],
    GroupRo => ['test', 'admin'],
    Description => 'A test Module',
    NavBarName => 'Ticket',
    NavBar => [
        {
            Description => 'Test Module',
            Name => 'Test',
            Image => 'stats.png',
            Link => 'Action=AgentTest',
            NavBar => 'Ticket',
            Prio => 85,
        },
    ],
};

```

You can access this frontend module via http (browse) with the Action param = Module or over the navigation bar.

<http://localhost/otrs/index.pl?Action=AgentTest>

Description of Frontend::Module options:

Key	Description
Group	An ARRAY reference of rw groups of this module.
GroupRo	An ARRAY reference of ro groups of this module.
Description	Module description, just for internal use - not shown in the user interface.

Key	Description
NavBarName	NavBar context name of this module.

Description of NavBar (icon points) options:

Key	Description
Description	The description of the icon which is shown in the navbar after the cursor is pointed on it.
Name	The icon name shown in the navbar.
Image	The icon image shown in the navbar.
Link	The link behind the icon in the navbar.
NavBar	Only shown this icon in this NavBar context.
Prio	Sort prio of the icon in the navbar.

3.5. Core Modules

Core modules are located under \$OTRS_HOME/Kernel/System/*. This layer is for the logical work. The modules are used to handle system routines like "lock ticket" and "create ticket". These modules always need pod (Perl Documentation).

A few common core modules are: Log (Kernel::System::Log); Ticket (Kernel::System::Ticket), Auth (Kernel::System::Auth), User (Kernel::System::User), Email (Kernel::System::Email).

For more information on the core modules see <http://dev.otrs.org>

Format:

```
# --
# Kernel/System/Backend.pm - a core module
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.39 2010-05-10 18:54:49 bes Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::Backend;

use strict;
use warnings;

=head1 NAME

Kernel::System::Log - global log interface

=head1 SYNOPSIS

All log functions.

=head1 PUBLIC INTERFACE

=over 4

=item new()
```

```

create a backend object

use Kernel::Config;
use Kernel::System::Backend;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.39 $) [1];

my $ConfigObject = Kernel::Config->new();
my $BackendObject = Kernel::System::Backend->new(ConfigObject => $ConfigObject);

=cut

sub new {
    my ( $Type, %Param ) = @_;
    [...]
    return $Self;
}

=item SomeMethodA()

some info about the method

$BackendObject->SomeMethodA(ParamA => 'error', ParamB => "Need something!");

=cut

sub SomeMethodA{
    my ( $Self, %Param ) = @_;
    [...]
    return 1;
}
1;

=head1 TERMS AND CONDITIONS

This software is part of the OTRS project (http://otrs.org/).

This software comes with ABSOLUTELY NO WARRANTY. For details, see
the enclosed file COPYING for license information (AGPL). If you
did not receive this file, see http://www.gnu.org/licenses/agpl.txt.

=head1 VERSION

$Revision: 1.39 $ $Date: 2010-05-10 18:54:49 $

=cut

```

3.6. Customer Auth Module

The same as "Agent Auth Modules" but the module location is Kernel/System/CustomerAuth/*.pm.

3.7. Customer User Module

This module layer can be used as a bridge between your customer source system and OTRS. Thus it is possible to use your customer data directly for your data ware house (read only and read/write).

Format:

```

# --
# Kernel/System/CustomerUser/ABC.pm - a customer data backend

```



```
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.39 2010-05-10 18:54:49 bes Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::CustomerUser::ABD;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.39 $) [1];

# --
sub new {
    my ( $Type, %Param ) = @_ ;
    [...]
    return $Self;
}
# --
sub CustomerName {
    [...]
    return $Name;
}
# --
sub CustomerSearch {
    [...]
    return %Result;
}
# --
sub CustomerUserList {
    [...]
    return %List;
}
# --
sub CustomerIDs {
    [...]
    return @CustomerIDs;
}
# --
sub CustomerUserDataGet {
    [...]
    return %Data;
}
# --
sub CustomerUserAdd {
    [...]
    return 1
}
# --
sub CustomerUserUpdate {
    [...]
    return 1;
}
# --
sub SetPassword {
    [...]
    return 1;
}
}
1;
```

To use this module, see the admin manual.

3.8. Customer Navigation Module

In this module layer you can create dynamic navigation bar items with dynamic content (Name and Description).

The format is the same as in the Navigation Module.

Just the config setting key is different. To use this module, add the following to the Kernel/Config.pm and restart your webserver (if you use mod_perl).

```
[Kernel/Config.pm]
# customer notification module
$self->{'CustomerFrontend::NavBarModule'}->{'99-ABC'} = {
    Module => 'Kernel::Output::HTML::NavBarABC',
};
```

3.9. Stats Module

There are two different types of internal stats modules - dynamic and static

3.9.1. Dynamic stats

If you create a dynamic stats file, you can configurate your own statistics via the web frontend.

The following functions are required (or optional) to get a functional script.

- sub new
to generate a new instance
- sub GetObjectName
gives the name of this dynamic stats module to the web frontend.
- sub GetObjectAttributes
generates an array with all possible elements of this dynamic stats module.
- sub GetStatElement
provides the content of a cell. Required if you do not use the function GetStatTable(). For example, have a look at Kernel/System/Stats/Dynamic/Ticket.pm.
- sub GetStatTable
provides the content of a table. Required if you do not use the function GetStatElement(). For example, have a look at Kernel/System/Stats/Dynamic/TicketList.pm.
- sub ExportWrapper
converts the internal stats xml into a commonly usable format. For example 'queue id' into 'queue name'.
- sub ImportWrapper
undoes the changes of the ExportWrapper during stats import.
- sub GetHeaderLine

provides a individual sorted header line. Optional. Only if needed.

For example, have a look at Kernel/System/Stats/Dynamic/Ticket.pm:

3.9.2. Static stats

A static stats module is easy to make - you just need two functions. Param() to get all usable params for the stats and Run() to return the stats result.

Kernel/System/Stats/Static/SomeExampleStats.pm:

```
# --
# Kernel/System/Stats/Static/SomeExampleStats.pm - stats module
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.39 2010-05-10 18:54:49 bes Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::Stats::Static::SomeExampleStats;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.39 $) [1];

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = { %Param };
    bless( $Self, $Type );

    # check all needed objects
    for (qw(ConfigObject EncodeObject LogObject DBObject)) {
        die "Got no $_" if ( !$Self->{$_} );
    }

    return $Self;
}

sub Param {
    my $Self = shift;

    # get current time
    my ($s,$m,$h, $D,$M,$Y) = $Self->{TimeObject}->SystemTime2Date(
        SystemTime => $Self->{TimeObject}->SystemTime(),
    );
    # get one month before
    my $SelectedYear = $M == 1 ? $Y - 1 : $Y;
    my $SelectedMonth = $M == 1 ? 12 : $M - 1;

    # create possible time selections
    my %Year = map { $_ => $_; } ( $Y - 10 .. $Y );
    my %Month = map { $_ => sprintf( "%02d", $_ ); } ( 1 .. 12 );

    my @Params = (
        {
            Frontend => 'Year',
            Name => 'Year',
```

```

        Multiple => 0,
        Size => 0,
        SelectedID => $SelectedYear,
        Data => {
            %Year,
        },
    },
    {
        Frontend => 'Month',
        Name => 'Month',
        Multiple => 0,
        Size => 0,
        SelectedID => $SelectedMonth,
        Data => {
            %Month,
        },
    },
);
return @Params;
}

sub Run {
    my ( $Self, %Param ) = @_;

    my $Title = "$Param{Year}-$Param{Month}";

    my @HeadData = ('Col A', 'Col B', 'Col C');

    my @Data = (
        [1, 4, 7],
        [6, 1, 4],
        [9, 4, 6],
    );

    return ([ $Title ], [ @HeadData ], @Data );
}

1;

```

3.9.3. Using old static stats

Standard OTRS versions 1.3 and 2.0 already facilitated the generation of stats. Various stats for OTRS versions 1.3 and 2.0 which have been specially developed to meet customers' requirements can be used in more recent versions too.

The files must merely be moved from the Kernel/System/Stats/ path to Kernel/System/Stats/Static/. Additionally the package name of the respective script must be amended by "::Static".

The following example shows how the first path is amended.

```
package Kernel::System::Stats::AccountedTime;
```

```
package Kernel::System::Stats::Static::AccountedTime;
```

3.10. Ticket Modules

3.10.1. Ticket Number Module

If you want to create your own ticket number format, just create your own ticket number module. These modules are located under "Kernel/System/Ticket/Number/*.pm". For

default modules see the admin manual. You just need 2 functions: CreateTicketNr() and GetTNByString():

Format:

```
# --
# Ticket/Number/Simple.pm - a ticket number auto increment generator
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.39 2010-05-10 18:54:49 bes Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::Ticket::Number::Simple;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.39 $) [1];

sub CreateTicketNr {
    my $Self = shift;
    my $JumpCounter = shift || 0;
    # get needed config options
    [...]
    return $Tn;
}
# --
sub GetTNByString {
    my $Self = shift;
    my $String = shift || return;
    [...]
    return $Tn;
}
1;
```

3.10.2. Ticket PostMaster Module

PostMaster modules are used during the PostMaster process. There are two kinds of PostMaster modules. PostMasterPre (used after parsing an email) and PostMasterPost (used after an email is processed and in the database) modules.

If you want to create your own postmaster filter, just create your own module. These modules are located under "Kernel/System/PostMaster/Filter/*.pm". For default modules see the admin manual. You just need two functions: new() and Run():

The following is an exemplary module to match emails and set X-OTRS-Headers (see doc/X-OTRS-Headers.txt for more info).

Kernel/Config.pm:

```
# Job Name: 1-Match
# (block/ignore all spam email with From: noreply@)
$self->{'PostMaster::PreFilterModule'}->{'1-Example'} = {
    Module => 'Kernel::System::PostMaster::Filter::Example',
    Match => {
        From => 'noreply@',
    },
},
```

```

    Set => {
      'X-OTRS-Ignore' => 'yes',
    },
  };

```

Format:

```

# --
# Kernel/System/PostMaster/Filter/Example.pm - a postmaster filter
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.39 2010-05-10 18:54:49 bes Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::System::PostMaster::Filter::Example;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.39 $) [1];

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless ( $Self, $Type);

    $Self->{Debug} = $Param{Debug} || 0;

    # get needed objects
    for (qw(ConfigObject EncodeObject LogObject DBObject)) {
        $Self->{$_} = $Param{$_} || die "Got no $_!";
    }

    return $Self;
}

sub Run {
    my ( $Self, %Param ) = @_;
    # get config options
    my %Config = ();
    my %Match = ();
    my %Set = ();
    if ($Param{JobConfig} && ref($Param{JobConfig}) eq 'HASH') {
        %Config = %{$Param{JobConfig}};
        if ($Config{Match}) {
            %Match = %{$Config{Match}};
        }
        if ($Config{Set}) {
            %Set = %{$Config{Set}};
        }
    }
    # match 'Match => ???' stuff
    my $Matched = '';
    my $MatchedNot = 0;
    for (sort keys %Match) {
        if ($Param{GetParam}->{$_} &&& $Param{GetParam}->{$_} =~ /$Match{$_}/i) {
            $Matched = $1 || '1';
            if ($Self->{Debug} > 1) {

```

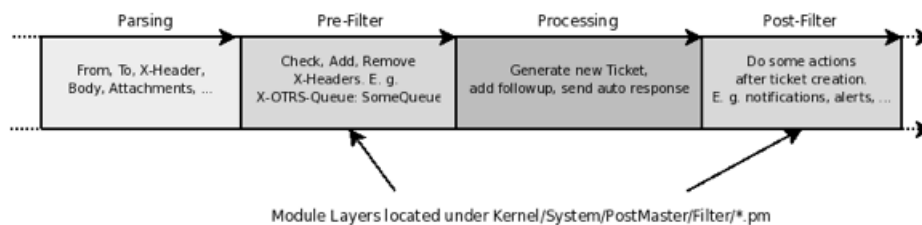
```

        $Self->{LogObject}->Log(
            Priority => 'debug',
            Message => "'$Param{GetParam}->{$_}' =~ /$Match{$_}/i matched!",
        );
    }
}
else {
    $MatchedNot = 1;
    if ($Self->{Debug} > 1) {
        $Self->{LogObject}->Log(
            Priority => 'debug',
            Message => "'$Param{GetParam}->{$_}' =~ /$Match{$_}/i matched NOT!",
        );
    }
}
}
# should I ignore the incoming mail?
if ($Matched && !$MatchedNot) {
    for (keys %Set) {
        if ($Set{$_} =~ /\[\*\*\*\]/i) {
            $Set{$_} = $Matched;
        }
        $Param{GetParam}->{$_} = $Set{$_};
        $Self->{LogObject}->Log(
            Priority => 'notice',
            Message => "Set param '$_' to '$Set{$_}' (Message-ID: $Param{GetParam}-
>{'Message-ID'}) ",
        );
    }
}
return 1;
}
1;

```

The following image shows you the email processing flow.

Life Cycle of Email Processing



3.10.3. Ticket Menu Module

To add links in the ticket menu, just use ticket menu modules.

If you want to create your own ticket menu link, just create your own module. These modules are located under "Kernel/Output/HTML/TicketMenu*.pm". For default modules see the admin manual. You just need two functions: new() and Run():

The following example shows you how to show a lock or a unlock ticket link.

Kernel/Config.pm:

```

# for ticket zoom menu
$Self->{'Ticket::Frontend::MenuModule'}->{'100-Lock'} = {

```

```

    Action => 'AgentTicketLock',
    Module => 'Kernel::Output::HTML::TicketMenuLock',
    Name   => 'Lock'
};

# for ticket preview menu
$self->{'Ticket::Frontend::PreMenuModule'}->{'100-Lock'} = {
    Action => 'AgentTicketLock',
    Module => 'Kernel::Output::HTML::TicketMenuLock',
    Name   => 'Lock'
};

```

Format:

```

# --
# Kernel/Output/HTML/TicketMenuLock.pm
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: module-format.xml,v 1.39 2010-05-10 18:54:49 bes Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::Output::HTML::TicketMenuLock;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.39 $) [1];

# --
sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {};
    bless ( $Self, $Type);

    # get needed objects
    for (qw(ConfigObject LogObject DBObject LayoutObject UserID TicketObject)) {
        $Self->{$_} = $Param{$_} || die "Got no $_!";
    }

    return $Self;
}

# --
sub Run {
    my ( $Self, %Param ) = @_;
    # check needed stuff
    if (!$Param{Ticket}) {
        $Self->{LogObject}->Log(Priority => 'error', Message => 'Need Ticket!');
        return;
    }

    # check permission
    if ($Self->{TicketObject}->LockIsTicketLocked(TicketID => $Param{TicketID})) {
        my $AccessOk = $Self->{TicketObject}->OwnerCheck(
            TicketID => $Param{TicketID},
            OwnerID => $Self->{UserID},
        );
        if (!$AccessOk) {
            return $Param{Counter};
        }
    }
}

```



```

    }
}

$self->{LayoutObject}->Block(
    Name => 'Menu',
    Data => { },
);
if ($Param{Counter}) {
    $self->{LayoutObject}->Block(
        Name => 'MenuItemSplit',
        Data => { },
    );
}
if ($Param{Ticket}->{Lock} eq 'lock') {
    $self->{LayoutObject}->Block(
        Name => 'MenuItem',
        Data => {
            %{$Param{Config}},
            %{$Param{Ticket}},
            %Param,
            Name => 'Unlock',
            Description => 'Unlock to give it back to the queue!',
            Link => 'Action=AgentTicketLock&Subaction=Unlock&TicketID=
$QData{"TicketID"}',
        },
    );
}
else {
    $self->{LayoutObject}->Block(
        Name => 'MenuItem',
        Data => {
            %{$Param{Config}},
            %Param,
            Name => 'Lock',
            Description => 'Lock it to work on it!',
            Link => 'Action=AgentTicketLock&Subaction=Lock&TicketID=
$QData{"TicketID"}',
        },
    );
}
$Param{Counter}++;
return $Param{Counter};
}
# --
1;

```

3.10.4. Ticket Event Module

Ticket event modules are running right after a ticket action takes place. Per convention these modules are located in the directory "Kernel/System/Ticket/Event". An ticket event module needs only the two functions new() and Run(). The method Run() receives at least the parameters Event, UserID, and TicketID. For the events TicketFreeTextUpdate and TicketFreeTimeUpdate, the parameter Counter is also passed to the Run() method, in order to identify which free field was updated. The article related handler functions also receives the ArticleID as parameter.

3.10.4.1. Code example

See Kernel/System/Ticket/Event/EventModulePostTemplate.pm in the package TemplateModule.

3.10.4.2. Configuration example

See Kernel/Config/Files/EventModulePostTemplate.xml in the package TemplateModule.

3.10.4.3. Use Cases

3.10.4.3.1. A ticket should be unlocked after a move action.

This standard feature has been implemented with the ticket event module `Kernel::System::Ticket::Event::ForceUnlock`. When this feature is not wanted, then it can be turned off by unsetting the SysConfig entry `Ticket::EventModulePost###910-ForceUnlockOnMove`.

3.10.4.3.2. Perform extra cleanup action when a ticket is deleted.

A customized OTRS might hold non-standard data in additional database tables. When a ticket is deleted then this additional data needs to be deleted. This functionality can be achieved with a ticket event module listening to 'TicketDelete' events.

3.10.4.3.3. New tickets should be twittered.

A ticket event module listening to 'TicketCreate' can send out tweets.

3.10.4.4. Caveats and Warnings

No caveats are known.

3.10.4.5. Release Availability

Ticket events have been available in OTRS since OTRS 2.0.

Ticket Events for OTRS 2.0: TicketCreate, TicketDelete, TicketTitleUpdate, TicketUnlockTimeoutUpdate, TicketEscalationStartUpdate, MoveTicket, SetCustomerData, TicketFreeTextSet, TicketFreeTimeSet, TicketPendingTimeSet, LockSet, StateSet, OwnerSet, TicketResponsibleUpdate, PrioritySet, HistoryAdd, HistoryDelete, TicketAccountTime, TicketMerge, ArticleCreate, ArticleFreeTextSet, ArticleUpdate, ArticleSend, ArticleBounce, SendAgentNotification, SendCustomerNotification, SendAutoResponse, ArticleFlagSet;

Ticket Events for OTRS 2.1 and higher: TicketCreate, TicketDelete, TicketTitleUpdate, TicketUnlockTimeoutUpdate, TicketEscalationStartUpdate, TicketQueueUpdate (MoveTicket), TicketCustomerUpdate (SetCustomerData), TicketFreeTextUpdate (TicketFreeTextSet), TicketFreeTimeUpdate (TicketFreeTimeSet), TicketPendingTimeUpdate (TicketPendingTimeSet), TicketLockUpdate (LockSet), TicketStateUpdate (StateSet), TicketOwnerUpdate (OwnerSet), TicketResponsibleUpdate, TicketPriorityUpdate (PrioritySet), TicketSubscribe, TicketUnsubscribe, HistoryAdd, HistoryDelete, TicketAccountTime, TicketMerge, ArticleCreate, ArticleFreeTextUpdate (ArticleFreeTextSet), ArticleUpdate, ArticleSend, ArticleBounce, ArticleAgentNotification (SendAgentNotification), ArticleCustomerNotification (SendCustomerNotification), ArticleAutoResponse (SendAutoResponse), ArticleFlagSet, ArticleFlagDelete;

Ticket Events for OTRS 2.4: TicketCreate, TicketDelete, TicketTitleUpdate, TicketUnlockTimeoutUpdate, TicketQueueUpdate(MoveTicket), TicketTypeUpdate, TicketServiceUpdate, TicketSLAUpdate, TicketCustomerUpdate (SetCustomerData), TicketFreeTextUpdate, TicketFreeTimeUpdate, TicketPendingTimeUpdate (TicketPendingTimeSet), TicketLockUpdate (LockSet), TicketStateUpdate (StateSet), TicketOwnerUpdate (OwnerSet), TicketResponsibleUpdate, TicketPriorityUpdate (PrioritySet), HistoryAdd, HistoryDelete, TicketAccountTime, TicketMerge, ArticleCreate, ArticleFreeTextUpdate (ArticleFreeTextSet), ArticleUpdate, ArticleSend, ArticleBounce, ArticleAgentNotification (SendAgentNotification), ArticleCustomerNotification (SendCustomerNotification), ArticleAutoResponse(SendAutoResponse), ArticleFlagSet, ArticleFlagDelete;

3.10.5. More Modules

The Agent Ticket Permission Modules (Kernel/System/Ticket/Permission/) contain functions to verify an agent's authorisation to access a ticket.

The Customer Ticket Permission Modules (Kernel/System/Ticket/CustomerPermission/) contain functions to verify a customer's authorisation to access a ticket.

The Article Module (Kernel/System/Ticket/Article.pm) facilitates the readout and generating of ticket articles.

More modules and their descriptions are listed under <http://dev.otrs.org/>



Chapter 8. Templates

The .dtl files are about 70% plain html and just used from frontend modules (Kernel/Modules/*.pm). The .dtl files are located under:

```
$OTRS_HOME/Kernel/Output/HTML/Standard/*.dtl
```

The usable dtl tags and syntax are described below.

1. Formatting

We use four spaces per indentation level. Examples:

```
<table>
  <tr>
    <td>Key</td>
    <td>Value</td>
  </tr>
  <tr>
    <td>aaa</td>
    <td>bbb</td>
  </tr>
</table>
```

```
<form action = "index.pl">
  <input type="text" value="">
  <input type="text" value="">
  <table>
    <tr>
      <td>Key1</td>
      <td>Value1</td>
    </tr>
    <tr>
      <td>Key2</td>
      <td>Value2</td>
    </tr>
  </table>
  <input type="submit">
</form>
```

1.1. Comment

The dtl comment starts with a # at the beginning of a line and will not be shown in the html output.

```
# this can be a comment in the dtl file
```

1.2. \$Data{""}

If data params are given to the templates, these params can be printed to the template.

```
The name of the author is $Data{"Name"}.
```

Warning

If the value of the param Name includes html tags, these tags are also shown.

1.3. \$QData{""}

The same as \$Data{""} but the value if the param Name is html quoted (safe).

```
The name of the author is $QData{"Name"}.
```

It's also possible to cut the value. If, for example, you just want to show 20 characters of a variable (result will be "SomeName[..]"), use the following:

```
The first 20 characters of the author's name: $QData{"Name","20"}.
```

1.4. \$LQData{""}

The same as \$QData{""} but with link encoding. This means for example that a space will be a + (e. g. for "a href").

```
<a href="$Env{"Baselink"}&Location=$LQData{"File"}>$QData{"File","110"}</a>
```

1.5. \$Env{""}

Env is an environment variable which is usable in more templates at a time. \$Data{""} is just available for one template.

```
The current user name is: $Env{"Userfirstname"}
```

Some other common variables are:

```
$Env{"SessionID"} --> the current session id
$Env{"Time"} --> the current time e. g. Thu Dec 27 16:00:55 2001
$Env{"CGIHandle"} --> the current CGI handle e. g. index.pl
$Env{"UserCharset"} --> the current site charset e. g. iso-8859-1
$Env{"Baselink"} --> the baselink --> index.pl?SessionID=...
$Env{"UserFirstname"} --> e. g. Dirk $Env{"UserFirstname"}
$Env{"UserLogin"} --> e. g. mgg@x11.org
$Env{"UserIsGroup[users]"} = Yes --> user groups (useful for own links)
$Env{"UserIsGroup[admin]"} = Yes $Env{"Action"} --> the current action
```

1.6. \$QEnv{""}

QEnv is an html quoted environment variable (like Env but quoted).

```
The current user name is: $QEnv{"Userfirstname"}
```

1.7. \$Quote{""}

A tag to quote html strings.

```
Show no html tags, quote this: $Quote{"<hr><b>some text</b></hr>"}
```

It's also possible to cut the value. If, for example, you just want to show 20 characters of a variable (result will be "SomeMess[..]"), use the following:

```
Show no html tags, quote this and show just 20 characters:
```

```
$Quote{"<hr><b>some text</b></hr>","20"}
```

1.8. `$Text{""}`

This tag translates the string (based on the user selected language).

```
Translate this text: $Text{"Help"}
Translate this text and insert the given data: $Text{"Change %s settings", "$Data{"Type"}"}
```

1.9. `$JSText{""}`

The same as `$Text{""}` but with java script quoting. This function quote the ' in java script code with \. Otherwise the code is broken.

```
Translate this text: $JSText{"Help"}
Translate this text and insert the given data: $JSText{"Change %s settings", "$Data{"Type"}"}
```

1.10. `$Config{""}`

With this tag you can use config variables in the template. For example the Kernel/config.pm:

```
[Kernel/Config.pm]
# FQDN
# (Full qualified domain name of your system.)
$self->{FQDN} = 'otrs.example.com';
# AdminEmail
# (Email of the system admin.)
$self->{AdminEmail} = 'admin@example.com';
[...]
```

And the use in the dtl template:

```
The hostname is '$Config{"FQDN"}'
The admin email address is '$Config{"AdminEmail"}'
```

1.11. `$Include{""}`

If you want to include other dtl templates into a template, use the include tag:

```
# include Copyright.dtl
$Include{"Copyright"}
```

1.12. `Block`

The block tag can be used to define blocks. These blocks can be used several times by the frontend module with different data params.

```
<table>
<!-- dtl:block:Row -->
  <tr>
    <td valign="top" width="15%">
      <b>
        $Text{"$Data{"Key"}"}:
      </b>
    </td>
  </tr>
</table>
```

```

    </b>
  </td>
  <td width="85%">
    <div title="$QData{Value}">
      $QData{Value,"160"}
    </div>
  </td>
</tr>
<!-- dtl:block:Row -->
</table>

```

The html code can be displayed in template blocks in the frontend:

```

# get article
my %Article = $Self->{TicketObject}->ArticleGet(
  ArticleID => $ArticleID,
);

# file blocks
for (qw(From To Cc Subject)) {
  if ($Article{$_}) {
    $Self->{LayoutObject}->Block(
      Name => 'Row',
      Data => {
        Key => $_,
        Value => $Article{$_},
      },
    );
  }
}

# process template
my $HTMLOutput = $Self->{LayoutObject}->Output(
  TemplateFile => 'AgentZoomBody',
  Data => {
    %Article
  },
);

```

1.13. set

With this tag you can set a variable in the dtl template.

```
<dtl set $Data{"Test"} = "Some Text">
```

Note

This template function can still be found in several OTRS templates but will be removed in future versions of OTRS. The removal is necessary as a sequence control in the templates is not desired. The function should thus not be used any more.

1.14. if

It's also possible to use a really "simple" (ne|eq|=~|!~) if condition.

```
<dtl if ($Text{"Lock"} eq "Lock") { $Data{"Language"} = "en"; }>
```

Or with a regexp.

```
<dtl if ($Text{"Lock"} =~ "/text/i") { $Data{"Lala"} = "Matched"; }>
```

Note

This template function can still be found in several OTRS templates but will be removed in future versions of OTRS. The removal is necessary as a sequence control in the templates is not desired. The function should thus not be used any more.

1.15. system-call

If you want the output of a system call back to a dtl variable.

```
# execute system call
<dtl system-call $Data{"uptime"} = "uptime">
# print
The output of 'uptime' is: $Data{"uptime"}
```

Yet another example:

```
# execute system call
<dtl system-call $Data{"procinfo"} = "procinfo | head -n1 ">
# print
The output of 'procinfo' is: $Data{"procinfo"}
```

Usable to set: `$Data{""}`, `$Env{""}` and `$Config{""}`.

Note

This template function can still be found in several OTRS templates but will be removed in future versions of OTRS. The removal is necessary as a sequence control in the templates is not desired. The function should thus not be used any more.

2. Example

```
# set variable
<dtl set $Data{"Test1"} = "English">

# print variable
Echo: $Data{"Test1"}
```

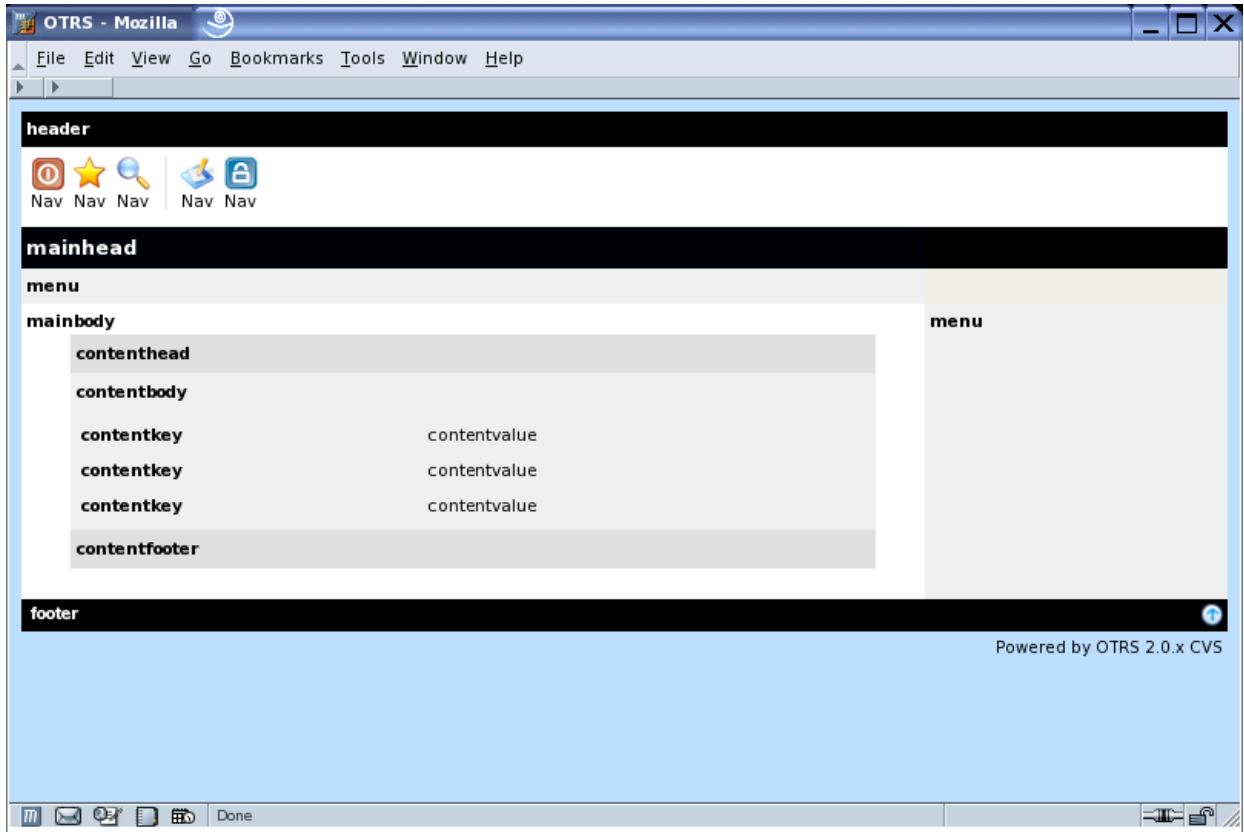
```
# condition
<dtl if ($Text{"Lock"} ne "Lock") { $Data{"Test2"} = "Not English!"; }>

# print result
Result: $Data{"Test1"}
```

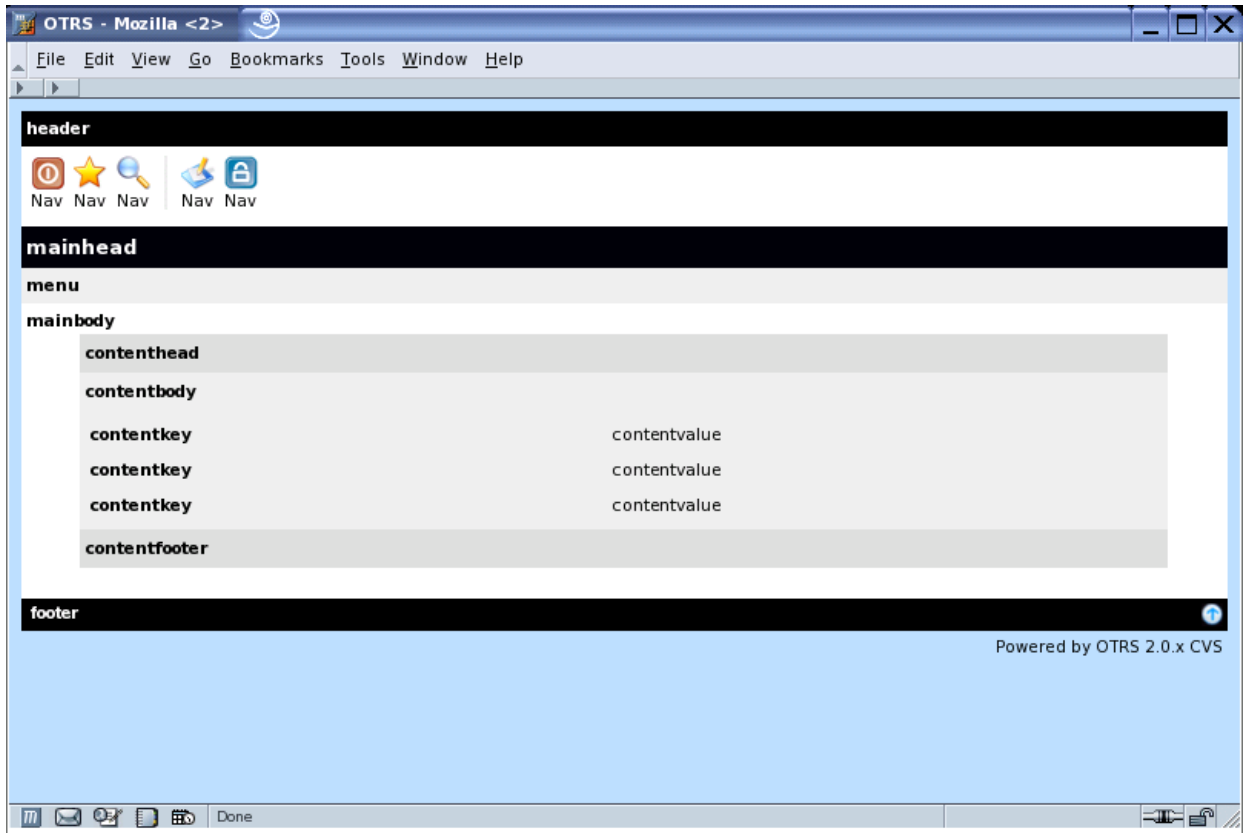

Chapter 9. Layout

1. CSS Style

1.1. View

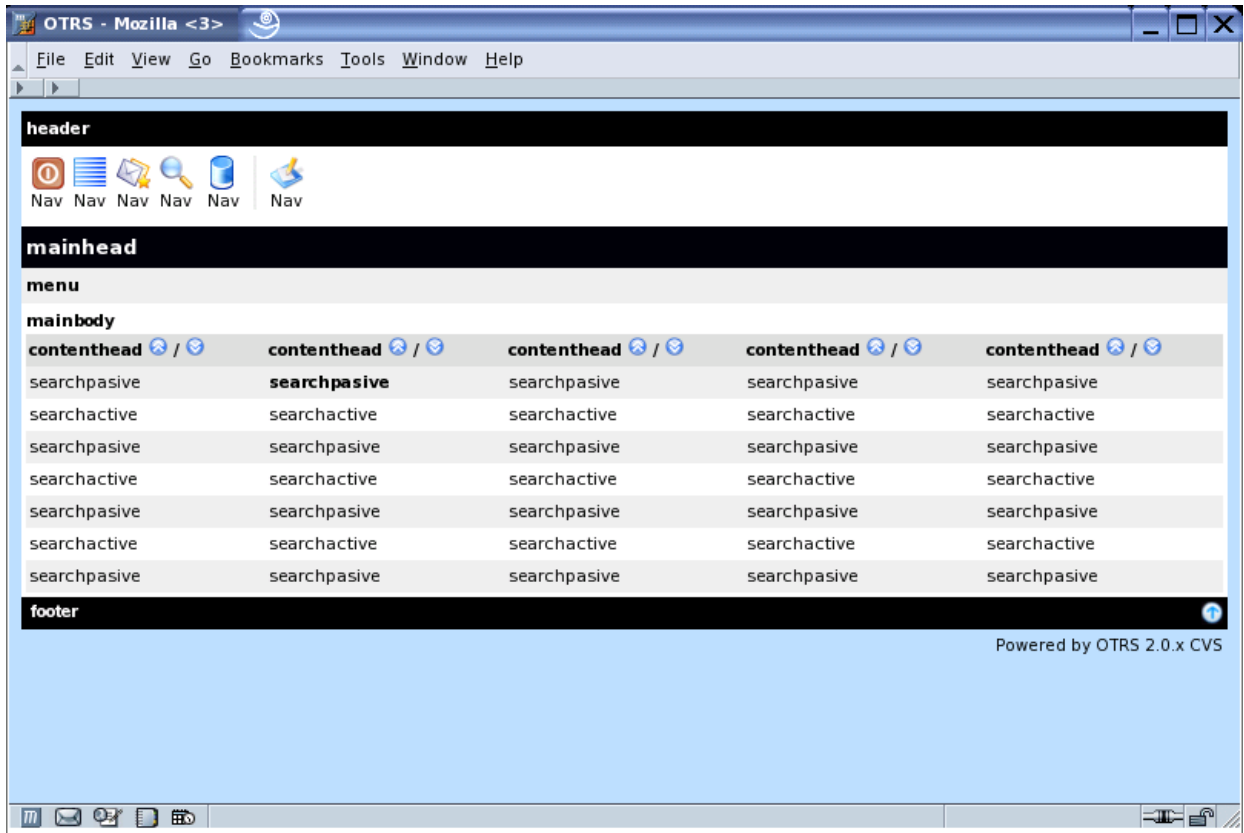


1.2. Edit/New/Search



The default width for the content tables is 700px and should be maintained on all pages.

1.3. Overview/Search Result

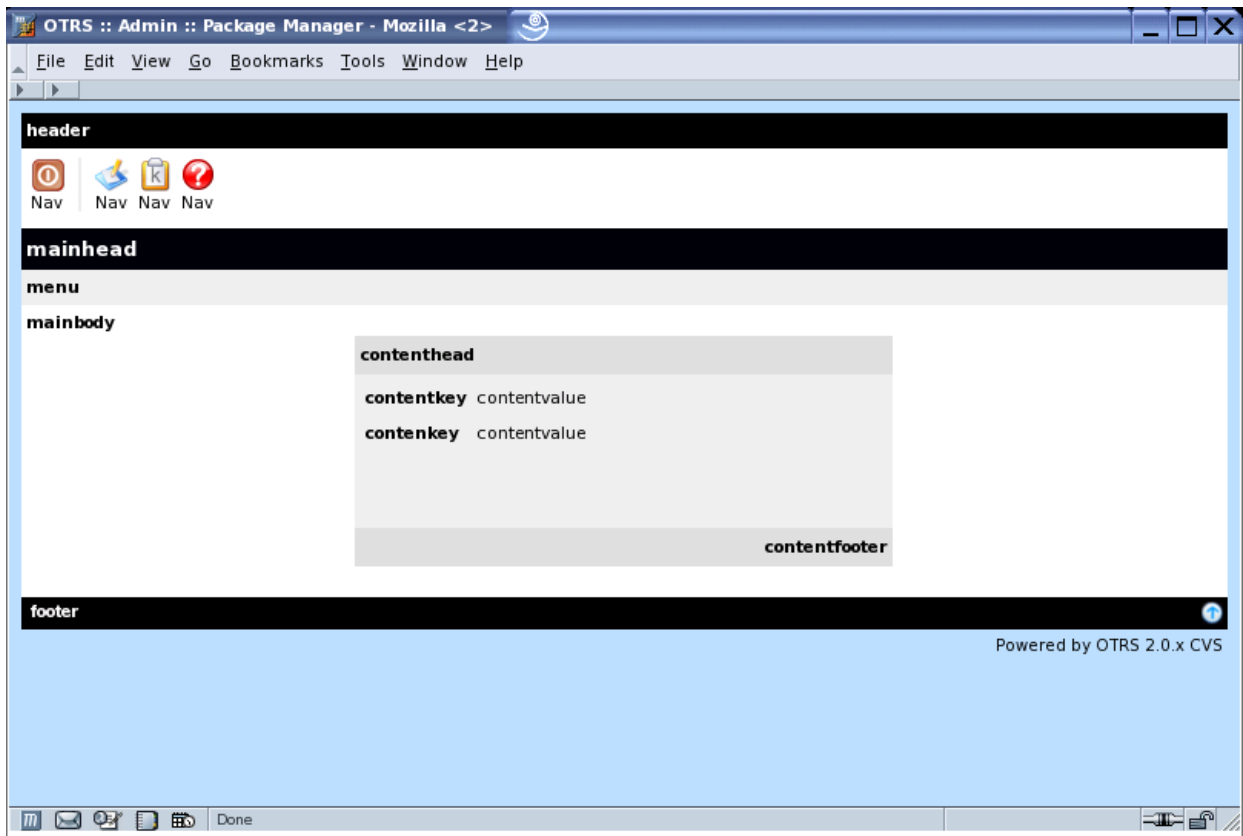


The screenshot shows a Mozilla browser window displaying the OTRS search results page. The page layout includes a header with navigation icons, a main menu, a table of search results, and a footer with the OTRS version.

contenthead	contenthead	contenthead	contenthead	contenthead
searchpasive	searchpasive	searchpasive	searchpasive	searchpasive
searchactive	searchactive	searchactive	searchactive	searchactive
searchpasive	searchpasive	searchpasive	searchpasive	searchpasive
searchactive	searchactive	searchactive	searchactive	searchactive
searchpasive	searchpasive	searchpasive	searchpasive	searchpasive
searchactive	searchactive	searchactive	searchactive	searchactive
searchpasive	searchpasive	searchpasive	searchpasive	searchpasive

Powered by OTRS 2.0.x CVS

1.4. Delete



2. Images

The image format is png. The image size for the navigation icons is 22x22 pixels. All other icons are 16x16 pixels.

Images are named after their function. For example new.png or search.png.

Default images are:

- new.png (for creating)
- search.png (for searching)
- overview.png (for an overview)
- logout.png (logout)

3. Special CSS Definitions

All html pages in OTRS use a readymade CSS declaration that is part of every header. Should the style sheets not contain the needed declaration, additional sheets must be defined as an extra css block in the html source code instead of in an html element.

Examples from the Calendar Module:

[...]

```
<!-- dtl:block:Overview -->
<style type="text/css">
<!--
    .sunday {
        font-family:Geneva,Helvetica,Arial,sans-serif;
        font-size:10pt;
        vertical-align:top;
        background-color:#ffe0e0;
        color:#000000;
    }
    .saturday {
        font-family:Geneva,Helvetica,Arial,sans-serif;
        font-size:10pt;
        vertical-align:top;
        background-color:#ffe0e0;
        color:#000000;
    }
}
[.]
//-->
</style>

<table border="0" width="100%" cellspacing="0" cellpadding="3">
<tr>
    <td class="mainhead">
        $Env{"Box0"}$Text{"Calendar"}$Env{"Box1"}
    </td>
</tr>
[.]
```

Chapter 10. Language Translations

The OTRS framework allows for different languages to be used in the frontend.

1. How it works

There are three different translation file types which are used in the following order. If a word/sentence is redefined in a translation file, the latest definition will be used.

1. Default Framework Translation File

```
Kernel/Language/$Language.pm
```

2. Frontend Module Translation File

```
Kernel/Language/$Language_$FrontendModule.pm
```

3. Custom Translation File

```
Kernel/Language/$Language_Custom.pm
```

1.1. Default Framework Translation File

The Default Framework Translation File includes the basic translations. The following is an example of a Default Framework Translation File.

Format:

```
# --
# Kernel/Language/de.pm - provides de language translation
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: language-translations.xml,v 1.16 2010-05-10 18:50:54 bes Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --
package Kernel::Language::de;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.16 $) [1];

# --
sub Data {
    my $Self = shift;

    # $$START$$

    # possible charsets
    $Self->{Charset} = ['iso-8859-1', 'iso-8859-15', ];
    # date formats (%A=WeekDay;%B=LongMonth;%T=Time;%D=Day;%M=Month;%Y=Year;)
    $Self->{DateFormat} = '%D.%M.%Y %T';
    $Self->{DateFormatLong} = '%A %D %B %T %Y';
    $Self->{DateFormatShort} = '%D.%M.%Y';
    $Self->{DateInputFormat} = '%D.%M.%Y';
    $Self->{DateInputFormatLong} = '%D.%M.%Y - %T';
```

```

$self->{Translation} = {
# Template: AAABase
'Yes' => 'Ja',
'No' => 'Nein',
'yes' => 'ja',
'no' => 'kein',
'Off' => 'Aus',
'off' => 'aus',Kernel/Language/$Language_Custome.pm
};
# $$STOP$$
return 1;
}
# --
1;

```

1.2. Frontend Translation File

The Frontend Translation File is used for a specific frontend module. If, for example, the frontend module "Kernel/Modules/AgentCalendar.pm", also <http://otrs.example.com/otrs/index.pl?Action=AgentCalendar>, is used, the Frontend Translation File "Kernel/Language/de_Agentcalendar.pm" is used, too.

Format:

```

# --
# Kernel/Language/de_AgentCalendar.pm - provides de language translation
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: language-translations.xml,v 1.16 2010-05-10 18:50:54 bes Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::Language::de_AgentCalendar;

use strict;
use warnings;

sub Data {
    my $Self = shift;

    $Self->{Translation}->{'CW'} = 'KW';
    $Self->{Translation}->{'Today'} = 'heute';
    $Self->{Translation}->{'Tomorrow'} = 'Morgen';
    $Self->{Translation}->{'1 St. May'} = 'Erster Mai';
    $Self->{Translation}->{'Christmas'} = 'Weihnachten';
    $Self->{Translation}->{'Silvester'} = 'Silvester';
    $Self->{Translation}->{'New Year\'s Eve!'} = 'Neu Jahr!';
    $Self->{Translation}->{'January'} = 'Januar';
    $Self->{Translation}->{'February'} = 'Februar';
    return 1;
}
1;

```

1.3. Custom Translation File

The Custom Translation File is read out last and so its translation which will be used. If you want to add your own wording to your installation, create this file for your language.

Format:

```
# --
# Kernel/Language/xx_Custom.pm - provides xx custom language translation
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: language-translations.xml,v 1.16 2010-05-10 18:50:54 bes Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --
package Kernel::Language::xx_Custom;

use strict;
use warnings;

use vars qw(@ISA $VERSION);
$VERSION = qw($Revision: 1.16 $) [1];

# --
sub Data {
    my $Self = shift;

    # $$START$$

    # own translations
    $Self->{Translation}->{'Lock'} = 'Lala';
    $Self->{Translation}->{'Unlock'} = 'Lulu';

    # $$STOP$$
    return 1;
}
# --
1;
```

2. Add a new default framework translation

If you want to translate the OTRS framework into a new language, you have to follow these five steps:

1. Take the current German translation (Kernel/Language/de.pm) from CVS (<http://cvs.otrs.org>). Use the German version because this is always up to date.
2. Change the package name (e.g. "package Kernel::Language::de;" to "package Kernel::Language::fr;") and translate each word/sentence.
3. Add the new language translation to the framework by adding it to your Kernel/Config.pm.

```
$Self->{DefaultUsedLanguages}->{fr} = 'French';
```

4. If you use mod_perl, restart your webserver and the new language will be shown in your preferences selection.
5. Send the new translation file to mailing list "i18n at otrs.org" - Thanks!

Chapter 11. Object Basics

This chapter describes the basics of a new object (e. g. a ticket, faq, calendar, ...) and how the environment should look like.

1. Object Options

An object (e.g. a ticket, faq, calendar, ...) should at least have the following options (named after their function) in the application and in the database.

Application	database naming
ObjectID	object_id
Number	number
Title	title
...	...
StateID	state_id
WordAndWord	word_and_word
...	...
Created	created
CreatedBy	created_by
Changed	changed
ChangedBy	changed_by

2. Search Options

A search over free text fields should support:

- a normal search "thomas" (always)
- an and condition like "thomas+raith" (if possible)
- an or condition like "thomas||raith" (if possible)

3. Config Naming

Config naming should be with a leading prefix, the object name like this:

```
$Self->{"Object::Option"} = 1234;
```

Config-Hashes should be named with the same name as in the .dtl. For example:

```
$Self->{"Object::CategoryList"} -> $Data{"CategoryList"}
```

The config order should be global setting followed by detail settings.

4. Config File

An object should have a unique config file which should be located under \$OTRS_HOME/Kernel/Config/Files/*.pm. For example:

```
# module reg and nav bar
$self->{'Frontend::Module'}->{'AgentFileManager'} = {
  Description => 'Web File Manager',
  NavBarName => 'FileManager',
  NavBar => [
    {
      Description => 'A web file manager',
      Name => 'File-Manager',
      Image => 'filemanager.png',
      Link => 'Action=AgentFileManager',
      NavBar => 'FileManager',
      Prio => 5000,
      AccessKey => 'f',
    },
  ],
};

# browse/download root directory
$self->{"FileManager::Root"} = '/home/';

# trash directory
$self->{"FileManager::Trash"} = "/home/Trash/";
```

Description of the Config Preferences:

Element	description
NavBarName	module name
Group/GroupRo	group access authorization
Name	name of the link button
Image	image for the link button
Link	URI
NavBar	module name (correlation)
Prio	prio in the button list
AccessKey	short key (key + ALT) for quick access over the keyboard.

5. NavBar Settings

A NavBar item should look like the following example:

NavBarPoint	Prio	AccessKey	Image
Overview	100	o	overview.png
New	200	n	new.png
Search	300	s	search.png
Delete	400	d	delete.png
Import	500	-	import.png
Setting	900	-	module_setting.png

Menu functions - generic, used by any application module

NavBarPoint	Prio	AccessKey	Image
Logout	10	l	logout.png

NavBarPoint	Prio	AccessKey	Image
Preferences	0	p	preferences.png
New Messages	999989	m	new-messages.png
Locked Tickets	9999999	k	personal.png

Menu functions - global, always used

NavBarPoint	Prio	AccessKey	Image
Ticket	200	t	ticket.png
Incident (SIRIOS-Project)	2000	i	incident.png
Advisory (SIRIOS-Project)	2100	d	advisory.png
ShortAdvisory (SIRIOS-BSI-specific)	2150	z	advisory_short.png
VirusWarning (SIRIOS-BSI-specific)	2300	x	viruswarning.png
FreeTextMessage (SIRIOS-BSI-specific)	2400	y	freetextmessage.png
Vulnerability (SIRIOS-Project)	2500	v	vulnerability.png
Artefact (SIRIOS-Project)	2600	r	artefactdb.png
WebWatcher (SIRIOS-Project)	2700	z	webwatcher.png
IDMEFConsole (SIRIOS-Project)	2800	-	idmef_console.png
WID-Authoring (WID-Project)	2900	-	wid_authoring.png
WID-Portal-Admin-User (WID-Project)	2910	-	wid_portal_admin_user.png
WID-Portal-Admin-Group (WID-Project)	2920	-	wid_portal_admin_group.png
ITSMService (OTRS::ITSM)	3100	-	itsm_service.png
ITSMConfigItem (OTRS::ITSM)	3200	-	itsm_configitem.png
ITSMLocation (OTRS::ITSM)	3300	-	itsm_location.png
TimeAccounting	6000	-	time_accounting.png
ContentManager	7050	-	contentmanager.png
Calendar	8000	c	calendar.png
FileManager	8100	f	filemanager.png
WebMail	8200	w	webmail.png

NavBarPoint	Prio	AccessKey	Image
FAQ	8300	q	help.png
Call	8400	-	call.png
Stats	8500	-	stats.png
CustomerDB	9000	-	folder_yellow.png
CustomerCompanyDB	9100	-	folder_yellow.png
Admin	10000	a	admin.png

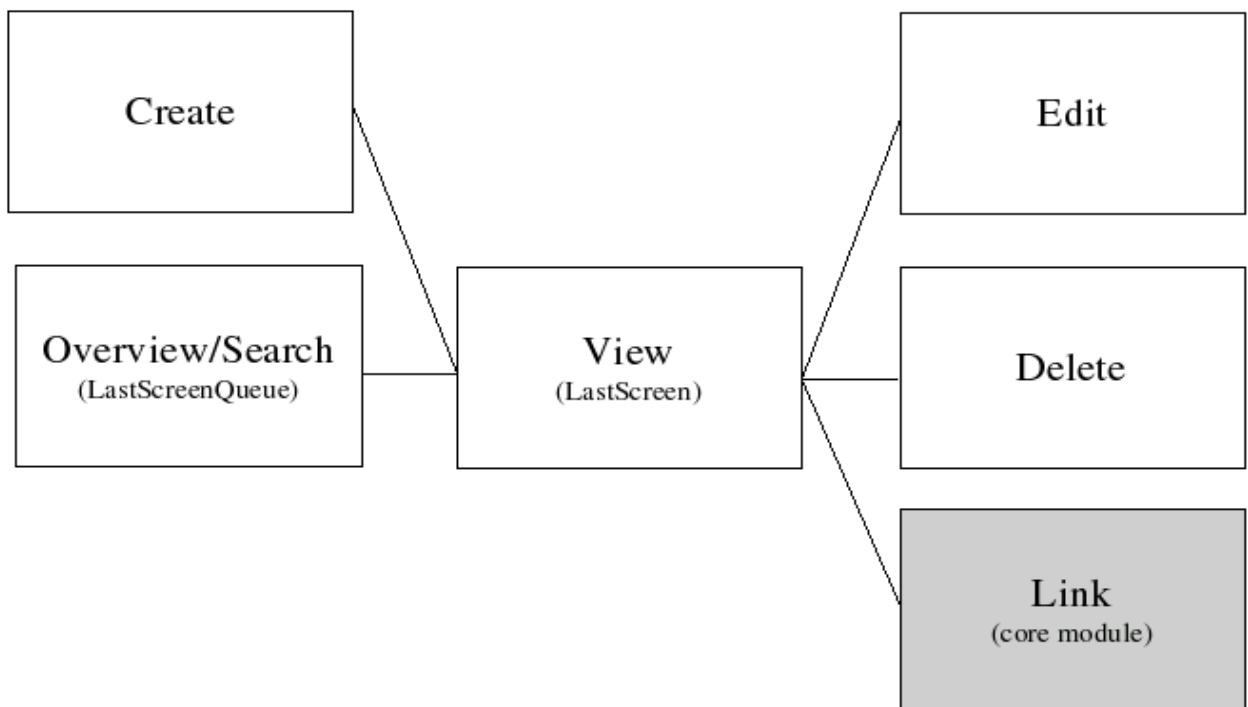
Table 3: Menu Applications -default application modules

Note

AccessKey "g" is also reserved for the submission of forms.

6. Screen flow

An object module should have the following process flow:



Chapter 12. Development Environment

To facilitate the writing of OTRS expansion modules, the creation of a development environment is necessary.

1. Framework checkout (CVS)

First of all a directory must be created in which the modules can be stored. Then switch to the new directory using the command line and check them out of OTRS 2.4 or the CVS head by using the following command (loginpassword: cvs):

```
shell> cvs -d :pserver:anonymous@cvs.otrs.org:/home/cvs login
```

```
shell> cvs -z3 -d :pserver:anonymous@cvs.otrs.org:/home/cvs co otrs
```

```
shell> cvs -z3 -d :pserver:anonymous@cvs.otrs.org:/home/cvs co -r rel-2_4 otrs
```

OTRS expansion modules can be checked out following the same routine. Check out the "module-tools" module, too for your development environment. It contains a number of useful tools.

To enable the CVS-OTRS it is necessary to configure it on the Apache web server and to create the Config.pm. Then the Installer.pl can be executed. The basic system is ready to run now.

2. Linking Expansion Modules

A clear separation between OTRS and the modules is necessary for proper developing. Particularly when using a CVS, a clear separation is crucial. In order to facilitate the OTRS access to the files, links must be created. This is done by a script in the directory module tools (to get this tools, check out the CVS module "module-tools"). Example: Linking the Calendar Module:

```
shell> ~/src/module-tools/link.pl ~/src/Calendar/ ~/src/otrs/
```

Whenever new files are added, they must be linked as described above.

To remove links from OTRS enter the following command:

```
shell> ~/src/module-tools/remove_links.pl ~/src/otrs/
```

3. Necessary Actions after Linking

As soon as the linking is completed, the Sysconfig must be run to register the module in OTRS. Required users, groups and roles must be created manually and access authorizations must be defined. If an additional databank table is required, this must be created manually, too. If an OPM package exists, the SQL commands can be read out to create the tables. Example:

```
shell> cat Calendar.sopm | bin/xml2sql.pl -t mysql
```

Chapter 13. Writing an OTRS module for a new object

In this chapter, the writing of a new OTRS module is illustrated on the basis of a simple small programme. Necessary prerequisite is an OTRS development environment as specified in the chapter of the same name.

1. What we want to write

We want to write a little OTRS module that displays the text 'Hello World' when called up. First of all we must build the directory /Hello World for the module in the developer directory. In this directory, all directories existent in OTRS can be created. Each module should at least contain the following directories:

Kernel/

Kernel/System/

Kernel/Modules/

Kernel/Output/HTML/Standard/

Kernel/Config/

Kernel/Config/Files/

Kernel/Language/

2. Default Config File

The creation of a module registration facilitates the display of the new module in OTRS. Therefore we create a file '/Kernel/System/Config/Files/HelloWorld.xml'. In this file, we create a new config element. The impact of the various settings is described in the chapter 'Config Mechanism'.

```
<?xml version="1.0" encoding="UTF-8" ?>
<otrs_config version="1.0" init="Application">
  <ConfigItem Name="Frontend::Module###AgentHelloWorld" Required="1" Valid="1">
    <Description Lang="en">FrontendModuleRegistration for HelloWorld modul.</Description>
    <Description Lang="de">FrontendModulRegistration für das HelloWorld Modul.</
Description>
    <Group>HelloWorld</Group>
    <SubGroup>AgentFrontendModuleRegistration</SubGroup>
    <Setting>
      <FrontendModuleReg>
        <Title>HelloWorld</Title>
        <Group>users</Group>
        <Description>HelloWorld</Description>
        <NavBarName>HelloWorld</NavBarName>
        <NavBar>
          <Description>HelloWorld</Description>
          <Name>HelloWorld</Name>
          <Image>overview.png</Image>
          <Link>Action=AgentHelloWorld</Link>
          <NavBar>HelloWorld</NavBar>
          <Type>Menu</Type>
          <Prio>8400</Prio>
          <Block>ItemArea</Block>
        </NavBar>
      </FrontendModuleReg>
    </Setting>
  </ConfigItem>
</otrs_config>
```

```

        </FrontendModuleReg>
    </Setting>
</ConfigItem>
</otrs_config>

```

3. Frontend Module

After creating the links and executing the Sysconfig, a new module with the name 'HelloWorld' is displayed. When calling it up, an error message is displayed as OTRS cannot find the matching frontend module yet. This is the next thing to be created. To do so, we create the following file:

```

# --
# Kernel/Modules/AgentHelloWorld.pm - frontend modul
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: writing-otrs-application.xml,v 1.22 2010-05-10 18:50:54 bes Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

package Kernel::Modules::AgentHelloWorld;

use strict;
use warnings;

use Kernel::System::HelloWorld;

sub new {
    my ( $Type, %Param ) = @_;

    # allocate new hash for object
    my $Self = {%Param};
    bless ( $Self, $Type);

    # check needed objects
    for (qw(ParamObject DBObject TicketObject LayoutObject LogObject QueueObject ConfigObject
EncodeObject MainObject)) {
        if ( !$Self->{$_} ) {
            $Self->{LayoutObject}->FatalError( Message => "Got no $_!" );
        }
    }

    # create needed objects
    $Self->{HelloWorldObject} = Kernel::System::HelloWorld->new(%Param);

    return $Self;
}

sub Run {
    my ( $Self, %Param ) = @_;
    my %Data = ();

    $Data{HelloWorldText} = $Self->{HelloWorldObject}->GetHelloWorldText();

    # build output
    my $Output = $Self->{LayoutObject}->Header(Title => "HelloWorld");
    $Output .= $Self->{LayoutObject}->NavigationBar();
    $Output .= $Self->{LayoutObject}->Output(
        Data => \%Data,
        TemplateFile => 'AgentHelloWorld',
    );
}

```

```
$Output .= $Self->{LayoutObject}->Footer();  
return $Output;  
}  
1;
```

4. Core Module

Next, we create the file for the core module `"/HelloWorld/Kernel/System/HelloWorld.pm"` with the following content:

```
# --  
# Kernel/System/HelloWorld.pm - core modul  
# Copyright (C) (year) (name of author) (email of author)  
# --  
# $Id: writing-otrs-application.xml,v 1.22 2010-05-10 18:50:54 bes Exp $  
# --  
# This software comes with ABSOLUTELY NO WARRANTY. For details, see  
# the enclosed file COPYING for license information (AGPL). If you  
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.  
# --  
  
package Kernel::System::HelloWorld;  
  
use strict;  
use warnings;  
  
sub new {  
    my ( $Type, %Param ) = @_;  
  
    # allocate new hash for object  
    my $Self = {};  
    bless ( $Self, $Type );  
  
    return $Self;  
}  
  
sub GetHelloWorldText {  
    my ( $Self, %Param ) = @_;  
  
    return 'Hello World';  
}  
  
1;
```

5. dtl Template File

The last thing missing before the new module can run is the relevant template. Thus, we create the following file:

```
# --  
# Kernel/Output/HTML/Standard/AgentHelloWorld.dtl - overview  
# Copyright (C) (year) (name of author) (email of author)  
# --  
# $Id: writing-otrs-application.xml,v 1.22 2010-05-10 18:50:54 bes Exp $  
# --  
# This software comes with ABSOLUTELY NO WARRANTY. For details, see  
# the enclosed file COPYING for license information (AGPL). If you  
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.  
# --  
<!-- start form -->
```



```
<table border="0" width="100%" cellspacing="0" cellpadding="3">
  <tr>
    <td class="mainhead">
      $Env{"Box0"}$Text{"Overview"}: $Text{"HelloWorld"}$Env{"Box1"}
    </td>
  </tr>
  <tr>
    <td class="mainbody">
      <br>
      $Text{"$QData{"HelloWorldText"}"}!<br>
      <br>
      <br>
    </td>
  </tr>
</table>
<!-- end form -->
```

The module is working now and displays the text 'Hello World' when called up.

6. Language File

If the text 'Hello World' is to be translated into German, a language file for this language must be created: '/HelloWorld/Kernel/Language/de_AgentHelloWorld.pm'. Example:

```
package Kernel::Language::de_AgentHelloWorld;

use strict;
use warnings;

sub Data {
  my $Self = shift;

  $Self->{Translation}->{'Hello World'} = 'Hallo Welt';

  return 1;
}
1;
```

7. Summary

The example given above shows that it is not too difficult to write a new module for OTRS. It is important, though, to make sure that the module and file name are unique and thus do not interfere with the framework or other expansion modules. When a module is finished, an OPM package must be generated from it (see chapter 'Package Building').

Chapter 14. Package Management

The OPM (OTRS Package Manager) is a mechanism to distribute software packages for the OTRS framework via http, ftp or file upload.

For example, the OTRS project offers OTRS modules like a calendar, a file manager or web mail in OTRS packages via online repositories on our ftp servers. The packages can be managed (install/upgrade/uninstall) via the admin interface.

1. Package Distribution

If you want to create an OPM online repository, just tell the OTRS framework where the location is. Then you will have a new select option in the admin interface.

```
[Kernel/Config.pm]

# Package::RepositoryList
# (repository list)
$self->{'Package::RepositoryList'} = {
    'ftp://ftp.example.com/packages/' => '[Example-Repository]',
};

[...]
```

1.1. Package Repository Index

In your repository, create an index file for your OPM packages. OTRS just reads this index file and knows what packages are available.

```
shell> bin/opm.pl -a index -d /path/to/repository/ > /path/to/repository/otrs.xml
shell>
```

2. Package Commands

You can use the following OPM commands over the admin interface or over bin/opm.pl to manage admin jobs for OPM packages.

2.1. Install

Install OPM packages.

- Web: <http://localhost/otrs/index.pl?Action=AdminPackageManager>
- CMD:

```
shell> bin/opm.pl -a install -p /path/to/package.opm
```

2.2. Uninstall

Uninstall OPM packages.

- Web: <http://localhost/otrs/index.pl?Action=AdminPackageManager>
- CMD:

```
shell> bin/opm.pl -a uninstall -p /path/to/package.opm
```

2.3. Upgrade

Upgrade OPM packages.

- Web: <http://localhost/otrs/index.pl?Action=AdminPackageManager>
- CMD:

```
shell> bin/opm.pl -a upgrade -p /path/to/package.opm
```

2.4. List

List all OPM packages.

- Web: <http://localhost/otrs/index.pl?Action=AdminPackageManager>
- CMD:

```
shell> bin/opm.pl -a list
```

Chapter 15. Package Building

If you want to create an OPM package (.opm) you need to create a spec file (.sopm) which includes the properties of the package.

1. Package Spec File

The OPM package is XML based. You can create/edit the .sopm via a text or xml editor. It contains meta data, a file list and database options.

1.1. Name

The package name (required).

```
<Name>Calendar</Name>
```

1.2. Version

The package version (required).

```
<Version>1.2.3</Version>
```

1.3. Framework

The required framework version (2.4.x means e.g. 2.4.1 or 2.4.9) (required).

```
<Framework>2.4.x</Framework>
```

Can also be used several times.

```
<Framework>2.4.x</Framework>  
<Framework>2.3.x</Framework>  
<Framework>2.2.x</Framework>
```

1.4. Vendor

The package vendor (required).

```
<Vendor>OTRS AG</Vendor>
```

1.5. URL

The vendor URL (required).

```
<URL>http://otrs.org/</URL>
```

1.6. License

The license of the package (required).

```
<License>GNU AFFERO GENERAL PUBLIC LICENSE Version 3, November 2007</License>
```

1.7. ChangeLog

The package change log (optional).

```
<ChangeLog Version="1.1.2" Date="2007-02-15 18:45:21">Added some feature.</ChangeLog>  
<ChangeLog Version="1.1.1" Date="2007-02-15 16:17:51">New package.</ChangeLog>
```

1.8. Description

The package description in different languages (required).

```
<Description Lang="en">A web calendar.</Description>  
<Description Lang="de">Ein Web Kalender.</Description>
```

1.9. BuildHost

This will be filled in automatically by OPM (auto).

```
<BuildHost>?</BuildHost>
```

1.10. BuildDate

This will be filled in automatically by OPM (auto).

```
<BuildDate>?</BuildDate>
```

1.11. PackageRequired

Packages that must be installed beforehand (optional). If PackageRequired is used, a version of the required package must be specified.

```
<PackageRequired Version="1.0.3">SomeOtherPackage</PackageRequired>  
<PackageRequired Version="5.3.2">SomeotherPackage2</PackageRequired>
```

1.12. ModuleRequired

Perl modules that must be installed beforehand (optional).

```
<ModuleRequired Version="1.03">Encode</ModuleRequired>  
<ModuleRequired Version="5.32">MIME::Tools</ModuleRequired>
```

1.13. OS (^M)

Required OS (optional).

```
<OS>linux</OS>  
<OS>darwin</OS>  
<OS>mswin32</OS>
```

1.14. Filelist

This is a list of files included in the package (optional).

```
<Filelist>
  <File Permission="644" Location="Kernel/Config/Files/Calendar.pm"/>
  <File Permission="644" Location="Kernel/System/CalendarEvent.pm"/>
  <File Permission="644" Location="Kernel/Modules/AgentCalendar.pm"/>
  <File Permission="644" Location="Kernel/Language/de_AgentCalendar.pm"/>
</Filelist>
```

1.15. DatabaseInstall

Database entries that have to be created when a package is installed (optional).

```
<DatabaseInstall>
  <TableCreate Name="calendar_event">
    <Column Name="id" Required="true" PrimaryKey="true" AutoIncrement="true" Type="BIGINT"/>
    <Column Name="title" Required="true" Size="250" Type="VARCHAR"/>
    <Column Name="content" Required="false" Size="250" Type="VARCHAR"/>
    <Column Name="start_time" Required="true" Type="DATE"/>
    <Column Name="end_time" Required="true" Type="DATE"/>
    <Column Name="owner_id" Required="true" Type="INTEGER"/>
    <Column Name="event_status" Required="true" Size="50" Type="VARCHAR"/>
  </TableCreate>
</DatabaseInstall>
```

You also can choose `<DatabaseInstall Type="post">` or `<DatabaseInstall Type="pre">` to define the time of execution separately (post is default). For more info see chapter "Package Life Cycle".

1.16. DatabaseUpgrade

Information on which actions have to be performed in case of an upgrade (subject to version tag), (optional). Example (if already installed package version is below 1.3.4 (e. g. 1.2.6), defined action will be performed):

```
<DatabaseUpgrade>
  <TableCreate Name="calendar_event_involved" Version="1.3.4">
    <Column Name="event_id" Required="true" Type="BIGINT"/>
    <Column Name="user_id" Required="true" Type="INTEGER"/>
  </TableCreate>
</DatabaseUpgrade>
```

You also can choose `<DatabaseUpgrade Type="post">` or `<DatabaseUpgrade Type="pre">` to define the time of execution separately (post is default). For more info see chapter "Package Life Cycle".

1.17. DatabaseReinstall

Information on what actions have to be performed if the package is reinstalled, (optional).

```
<DatabaseReinstall></DatabaseReinstall>
```

You also can choose `<DatabaseReinstall Type="post">` or `<DatabaseReinstall Type="pre">` to define the time of execution separately (post is default). For more info see chapter "Package Life Cycle".

1.18. DatabaseUninstall

Uninstall (if a package gets uninstalled), (optional).

```
<DatabaseUninstall>  
  <TableDrop Name="calendar_event" />  
</DatabaseUninstall>
```

You also can choose `<DatabaseUninstall Type="post">` or `<DatabaseUninstall Type="pre">` to define the time of execution separately (post is default). For more info see chapter "Package Life Cycle".

1.19. IntroInstall

To show a "pre" or "post" install introduction in installation dialog.

```
<IntroInstall Type="post" Lang="en" Title="Some Title">  
Some Info formatted in dtl/html....  
</IntroInstall>
```

You can also use the "Format" attribute to define if you want to use "html" (which is default) or "plain" to use automatically a "`<pre></pre>`" tag wenn intro is shown (to use the new lines and spaces of the content).

1.20. IntroUninstall

To show a "pre" or "post" uninstall introduction in uninstallation dialog.

```
<IntroUninstall Type="post" Lang="en" Title="Some Title">  
Some Info formatted in dtl/html....  
</IntroUninstall>
```

You can also use the "Format" attribute to define if you want to use "html" (which is default) or "plain" to use automatically a "`<pre></pre>`" tag wenn intro is shown (to use the new lines and spaces of the content).

1.21. IntroReinstall

To show a "pre" or "post" reinstall introduction in reinstallation dialog.

```
<IntroReinstall Type="post" Lang="en" Title="Some Title">  
Some Info formatted in dtl/html....  
</IntroReinstall>
```

You can also use the "Format" attribute to define if you want to use "html" (which is default) or "plain" to use automatically a "`<pre></pre>`" tag wenn intro is shown (to use the new lines and spaces of the content).

1.22. IntroUpgrade

To show a "pre" or "post" upgrade introduction in upgrading dialog.

```
<IntroUpgrade Type="post" Lang="en" Title="Some Title">  
Some Info formatted in dtl/html....
```

```
</IntroUpgrade>
```

You can also use the "Format" attribute to define if you want to use "html" (which is default) or "plain" to use automatically a "<pre></pre>" tag wenn intro is shown (to use the new lines and spaces of the content).

1.23. CodeInstall

To execute perl code if the package is installed (optional).

```
<CodeInstall>
# example
if (1) {
    print STDERR "Some info to STDERR\n";
}
# log example
$self->{LogObject}->Log(
    Priority => 'notice',
    Message => "Some Message!",
)
# database example
$self->{DBObject}->Do(SQL => "SOME SQL");
</CodeInstall>
```

You also can choose <CodeInstall Type="post"> or <CodeInstall Type="pre"> to define the time of execution separately (post is default). For more info see chapter "Package Life Cycle".

1.24. CodeUninstall

To execute perl code if the package is uninstalled (optional). On "pre" or "post" time of package uninstallation.

```
<CodeUninstall>
# example
if (1) {
    print STDERR "Some info to STDERR\n";
}
</CodeUninstall>
```

You also can choose <CodeUninstall Type="post"> or <CodeUninstall Type="pre"> to define the time of execution separately (post is default). For more info see chapter "Package Life Cycle".

1.25. CodeReinstall

To execute perl code if the package is reinstalled (optional).

```
<CodeReinstall>
# example
if (1) {
    print STDERR "Some info to STDERR\n";
}
</CodeReinstall>
```

You also can choose <CodeReinstall Type="post"> or <CodeReinstall Type="pre"> to define the time of execution separately (post is default). For more info see chapter "Package Life Cycle".

1.26. CodeUpgrade

To execute perl code if the package is upgraded (subject to version tag), (optional). Example (if already installed package version is below 1.3.4 (e. g. 1.2.6), defined action will be performed):

```
<CodeUpgrade Version="1.3.4">
  # example
  if (1) {
    print STDERR "Some info to STDERR\n";
  }
</CodeUpgrade>
```

You also can choose `<CodeUpgrade Type="post">` or `<CodeUpgrade Type="pre">`; to define the time of execution separately (post is default).

2. Example .sopm

This is a whole example spec file.

```
<?xml version="1.0" encoding="utf-8" ?>
<otrs_package version="1.0">
  <Name>Calendar</Name>
  <Version>0.0.1</Version>
  <Framework>2.4.x</Framework>
  <Vendor>OTRS AG</Vendor>
  <URL>http://otrs.org/</URL>
  <License>GNU GENERAL PUBLIC LICENSE Version 2, June 1991</License>
  <ChangeLog Version="1.1.2" Date="2007-02-15 18:45:21">Added some feature.</ChangeLog>
  <ChangeLog Version="1.1.1" Date="2007-02-15 16:17:51">New package.</ChangeLog>
  <Description Lang="en">A web calendar.</Description>
  <Description Lang="de">Ein Web Kalender.</Description>
  <IntroInstall Type="post" Lang="en" Title="Thank you!">Thank you for choosing the Calendar module.</IntroInstall>
  <IntroInstall Type="post" Lang="de" Title="Vielen Dank!">Vielen Dank fuer die Auswahl des Kalender Modules.</IntroInstall>
  <BuildDate>?</BuildDate>
  <BuildHost>?</BuildHost>
  <Filelist>
    <File Permission="644" Location="Kernel/Config/Files/Calendar.pm"></File>
    <File Permission="644" Location="Kernel/System/CalendarEvent.pm"></File>
    <File Permission="644" Location="Kernel/Modules/AgentCalendar.pm"></File>
    <File Permission="644" Location="Kernel/Language/de_AgentCalendar.pm"></File>
    <File Permission="644" Location="Kernel/Output/HTML/Standard/AgentCalendar.dtl"></File>
    <File Permission="644" Location="Kernel/Output/HTML/NotificationCalendar.pm"></File>
    <File Permission="644" Location="var/httpd/htdocs/images/Standard/calendar.png"></File>
  </Filelist>
  <DatabaseInstall>
    <TableCreate Name="calendar_event">
      <Column Name="id" Required="true" PrimaryKey="true" AutoIncrement="true" Type="BIGINT"/>
      <Column Name="title" Required="true" Size="250" Type="VARCHAR"/>
      <Column Name="content" Required="false" Size="250" Type="VARCHAR"/>
      <Column Name="start_time" Required="true" Type="DATE"/>
      <Column Name="end_time" Required="true" Type="DATE"/>
      <Column Name="owner_id" Required="true" Type="INTEGER"/>
      <Column Name="event_status" Required="true" Size="50" Type="VARCHAR"/>
    </TableCreate>
  </DatabaseInstall>
  <DatabaseUninstall>
    <TableDrop Name="calendar_event"/>
  </DatabaseUninstall>
</otrs_package>
```

3. Package Build

To build an .opm package from the spec opm.

```
shell> bin/opm.pl -a build -p /path/to/example.sopm
writing /tmp/example-0.0.1.opm
shell>
```

4. Package Life Cycle - Install/Upgrade/Uninstall

The following image shows you how the life cycle of a package instalation/upgrade/uninstallation works in the backend step by step.

Life Cycle of Package Install/Upgrade/Uninstall



Chapter 16. Unit Tests

OTRS provides unit tests for core modules.

1. Creating a test file

The test files are stored in .t files under /scripts/test/*.t. For example the file /scripts/test/Calendar.t for the Calendar Module.

A test file consists of the function call of the function to be tested and the analysis of the return value. Example (/scripts/test/Calendar.t):

```
# --
# Calendar.t - Calendar
# Copyright (C) (year) (name of author) (email of author)
# --
# $Id: test-mechanism.xml,v 1.9 2010-05-10 18:50:54 bes Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (AGPL). If you
# did not receive this file, see http://www.gnu.org/licenses/agpl.txt.
# --

use strict;
use warnings;
use utf8;

use vars qw($Self);

use Kernel::System::User;
use Kernel::System::CalendarEvent;

$Self->{UserObject} = Kernel::System::User->new(%{$Self});
$Self->{EventObject} = Kernel::System::CalendarEvent->new(%{$Self}, UserID => 1);

my $EventID = $Self->{EventObject}->EventAdd(
    Title => 'Some Test',
    StartTime => '1977-10-27 20:15',
    EndTime => '1977-10-27 21:00',
    State => 'public',
    UserIDs => [1],
);

$Self->True(
    $EventID,
    'EventAdd()',
);

[..]
```

2. Testing

To check your tests, just use "bin/UnitTest.pl -n Calendar" to use /scripts/test/Calendar.t.

```
shell:/opt/otrs> bin/UnitTest.pl -n Calendar
+-----+
/opt/otrs/scripts/test/Calendar.t:
+-----+
ok 1 - EventAdd()
=====
Product:   OTRS 2.0.x CVS
```

```
Test Time: 0 s
Time:      2006-04-02 12:58:37
Host:     yourhost.example.com
Perl:     5.8.7
OS:       linux
TestOk:   1
TestNotOk: 0
```

```
=====
shell:/opt/otrs&gt;
```

3. True()

This function tests whether the return value of the function 'EventAdd()' in the variable \$EventID is valid.

```
$Self-&gt;True(
    $EventID,
    'EventAdd()',
);
```

4. False()

This function tests whether the return value of the function 'EventAdd()' in the variable \$EventID is invalid.

```
$Self-&gt;False(
    $EventID,
    'EventAdd()',
);
```

5. Is()

This function tests whether the variables \$A and \$B are equal.

```
$Self-&gt;Is(
    $A,
    $B,
    'Test Name',
);
```

Appendix A. Additional Resources

1. OTRS.org

The OTRS project website with source code, documentation and news is available at:

<http://otrs.org/>

2. Online API Library

The OTRS developer API documentation is available at:

<http://dev.otrs.org/>

3. Developer Mailing List

The OTRS developer mailing list is available at:

<http://lists.otrs.org/>

4. Commercial Support

For business assistance (support, consulting and training) please contact the commercial part of OTRS, the OTRS AG.

OTRS AG
Europaring 4
94315 Straubing (Germany)
Phone: +49 (0)9421 1862 760
Fax: +49 (0)9421 1862 769
Web: <http://otrs.com/>