OTRS 2.3 - Developer Manual

OTRS 2.3 - Developer Manual

Christian Schöpplein, Richard Kammermeyer, Stefan Rother, Thomas Raith, Burchard Steinbild, Andre Mindermann, Christopher Kuhn, Martin Edenhofer

Table of Contents

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.

Formatting

TAB: We use 4 spaces. Examples for braces:

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

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

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. @TicktIDs or $Output or BuildQueueView()
```

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 1.13 2007/07/02 14:30:53 tr Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/gpl.txt.
# --
```

The following line is updated by the CVS:

```
# $Id: codesyntax.xml,v 1.13 2007/07/02 14:30:53 tr Exp $
```

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
```

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

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
```

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"

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

Perldoc

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

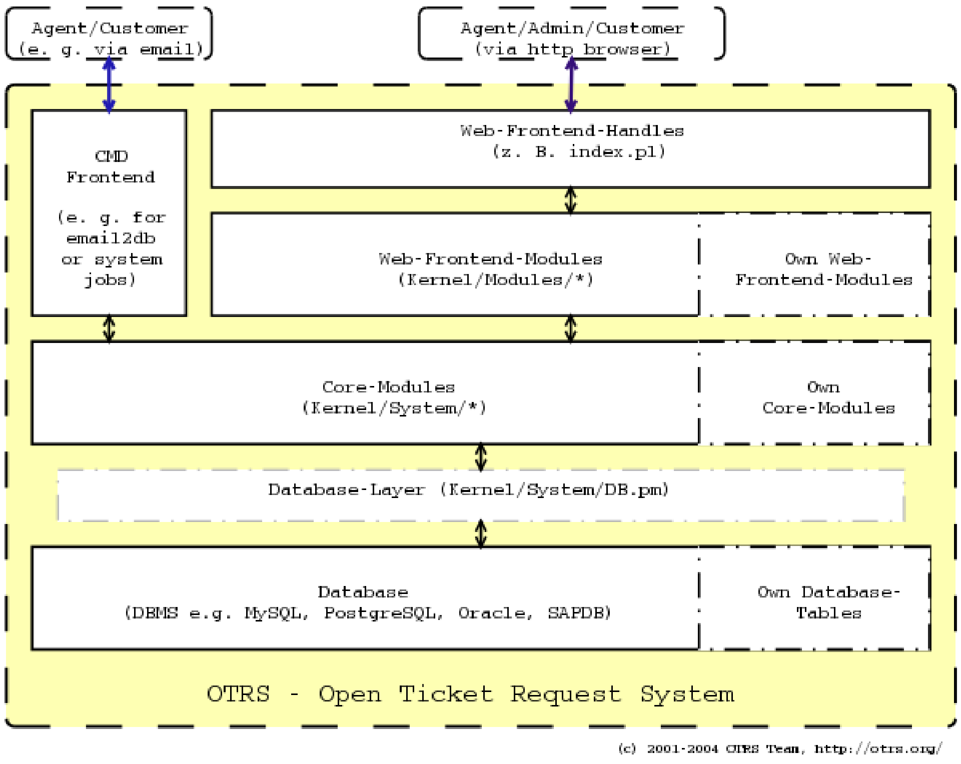
```
    =item SystemTime2TimeStamp()

    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" if the date is today (if
 needed).

        my $TimeStamp = $TimeObject->SystemTime2TimeStamp(
            SystemTime => $SystemTime,
            Type => 'Short',
        );
    =cut
```

Length of lines

Please see that a line of code is not langer then 80 charactars.

Chapter 3. Architecture

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



Directories

Directory	Description
bin/	CMD programmes
bin/cgi-bin/	web handle
bin/fcgi-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

Directory	Description
var/httpd/htdocs/images/Standard/	icons and pictures
scripts/	misc
scripts/test/	test files
scripts/sample/	sample files

Files

.pl = Perl

.pm = Perl Modul

.dtl = Dynamic Template Language (html template file)

.dist = Default Templates of Files

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/

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 []

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/

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.

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

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 of 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',
};
```

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.

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',
);
```

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 a config items here an example:

```
<ConfigItem Name="Ticket::Hook" Required="1" Valid="1">
    <Description Lang="en">The identifyer 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.

Types of XML Config Variables

The XML config settings support various types of variables.

String

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

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

Textarea

A config element for multiline text.

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

Options

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

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

Array

With this config element arrays can be displayed.

```
<Setting>
    <Array>
```

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

Hash

With this config element hashes can be displayed.

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

Hash with SubArray, SubOptions, SubHash

A hash can contain content arrays, hashes or pull-down menus.

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

FrontendModuleReg (NavBar)

Module registration for Agent Interface.

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

FrontendModuleReg (NavBarModule)

Module registration for Admin Interface

```
<Setting>
    <FrontendModuleReg>
        <Group>admin</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>
```

Chapter 5. Database Mechanism

OTRS comes with a database layer that supports different databases.

How it works

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

SQL

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

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",
);
```

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.

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');
```

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 chnaged in >=2.2. Values are now used in content area (not longer in attribut Value).

INSERT

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

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>
```

DROP TABLE

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

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"/>
```

Code to process XML

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

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

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

Database Drivers

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

Supported Databases

- MySQL
- PostgreSQL
- Oracle
- MSSQL
- DB2
- MaxDB/SAPDB (experimental)

Chapter 6. Log Mechanism

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

Use and Syntax

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

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

Example

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

```
use Kernel::Config;
use Kernel::System::Log;

my $ConfigObject = Kernel::Config->new();
my $LogObject    = Kernel::System::Log->new(
    ConfigObject => $ConfigObject,
);

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

Chapter 7. Module Format

Auth Module

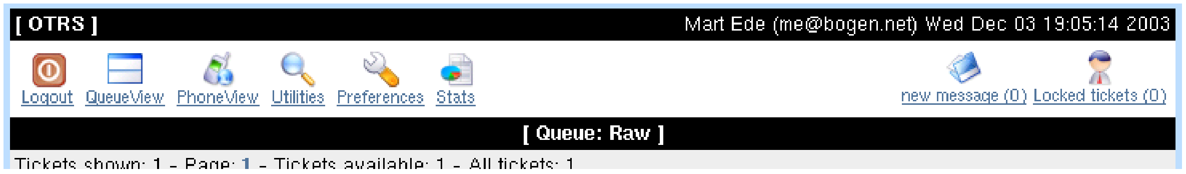
There are several agent authentication modules (DB, LDAP and HTTPBasicAuth) which come with the OTRS framework. It is also possible to develop your own authentication modules. The agent authentication modules are located under Kernel/System/Auth/*.pm. For more information about their configuration see 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:

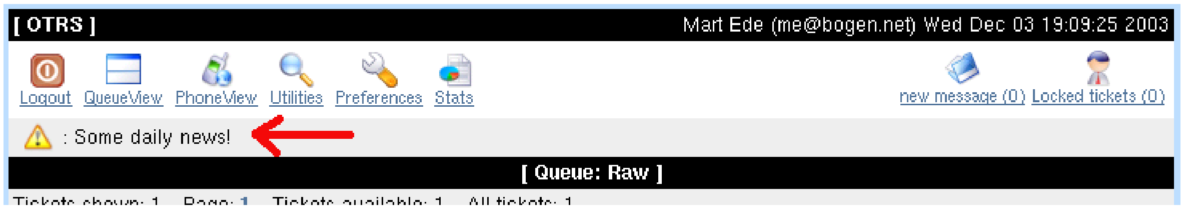
```
# --
# Kernel/System/Auth/Simple.pm - provides the simple authentication
# Copyright (C) 2001-2009 OTRS AG, http://otrs.org/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/
gpl-2.0.txt.
# --
# Note:
# available objects are: ConfigObject, LogObject and DBObject
# --

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

use strict;

sub new {
    my $Type = shift;
    my %Param = @_;
    [...]
    return $Self;
}
sub GetOption {
    my $Self = shift;
    my %Param = @_;
    # return option
    return (PreAuth => 0);
}
sub Auth {
    my $Self = shift;
    my %Param = @_;
    [...]
    if ($Authentication) {
        return $Param{User};
    }
    else {
        return;
        }
}
```

Notify Module

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



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



Format:

```
# --
# Kernel/Output/HTML/NotificationMotd.pm - message of the day
# Copyright (C) 2001-2009 OTRS AG, http://otrs.org/
# --
# $Id: module-format.xml,v 1.26 2009/01/06 10:34:20 martin Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/
gpl-2.0.txt.
# --

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

use strict;

# --
sub new {
    my $Type = shift;
    my %Param = @_;
    [...]
    return $Self;
}
# --
sub Run {
    my $Self = shift;
    my %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',
};
```

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 dynamicaly
# Copyright (C) 2001-2009 OTRS AG, http://otrs.org/
# --
# $Id: module-format.xml,v 1.26 2009/01/06 10:34:20 martin Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/
gpl-2.0.txt.
# --

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

use strict;

# --
sub new {
    my $Type = shift;
    my %Param = @_;
    [...]
    return $Self;
}
# --
sub Run {
    my $Self = shift;
    my %Param = @_;
    my %Return = ();
    $Return{'0999989'} = {
            Block => 'ItemPersonal',
            Description => 'Some Desctipton',
            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',
};
```

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).

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

Format:

```
# --
# Kernel/Modules/AgentTest.pm - message of the day
# Copyright (C) 2001-2009 OTRS AG, http://otrs.org/
# --
# $Id: module-format.xml,v 1.26 2009/01/06 10:34:20 martin Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/
gpl-2.0.txt.
# --

package Kernel::Modules::AgentTest;
```

```
use strict;

# --
sub new {
    my $Type = shift;
    my %Param = @_;
    [...]
    return $Self;
}
# --
sub Run {
    my $Self = shift;
    my %Param = @_;
    [...]
    # ---------------------------------------------------------- #
    # add a new object (Note: dtl text 'New')
    # ---------------------------------------------------------- #
    if ($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 $Output;
    }
    # ---------------------------------------------------------- #
    # show overview screen
    # ---------------------------------------------------------- #
    elsif ($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.
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 curser 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.

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 [http://dev.otrs.org/]

Format:

```
# --
# Kernel/System/Backend.pm - a core module
# Copyright (C) 2001-2009 OTRS AG, http://otrs.org/
# --
# $Id: module-format.xml,v 1.26 2009/01/06 10:34:20 martin Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/
gpl-2.0.txt.
# --

package Kernel::System::Backend;

use strict;

=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;

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

=cut

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

=item SomeMethodeA()

some info about the methode

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

=cut

sub SomeMethodeA{
    my $Self = shift;
    my %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 (GPL). If you
did not receive this file, see http://www.gnu.org/licenses/
gpl-2.0.txt.

=head1 VERSION

$Revision: 1.26 $ $Date: 2009/01/06 10:34:20 $

=cut
```

Customer Auth Module

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

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) 2001-2009 OTRS AG, http://otrs.org/
# --
# $Id: module-format.xml,v 1.26 2009/01/06 10:34:20 martin Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/
gpl-2.0.txt.
# --

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

use strict;

# --
sub new {
    my $Type = shift;
    my %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.

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 notifiacation module
$Self->{'CustomerFrontend::NavBarModule'}->{'99-ABC'} = {
    Module => 'Kernel::Output::HTML::NavBarABC',
};
```

Stats Module

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

Dynamic stats

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

The following functions are required 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.

- 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.

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

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) 2001-2009 OTRS AG, http://otrs.org/
# --
# $Id: module-format.xml,v 1.26 2009/01/06 10:34:20 martin Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/
gpl-2.0.txt.
# --

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

use strict;
use warnings;

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

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

    # get common objects
    for ( keys %Param ) {
        $Self->{$_} = $Param{$_};
    }

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

    return $Self;
}

sub Param {
    my $Self = shift;
    my @Params = ();

    # get current time
    my ($s,$m,$h, $D,$M,$Y) = $Self->{TimeObject}->SystemTime2Date(
        SystemTime => $Self->{TimeObject}->SystemTime(),
    );
    # get one month bevore
    if ($M == 1) {
        $M = 12;
        $Y = $Y - 1;
    }
    else {
        $M = $M -1;
    }
    # create possible time selections
    my %Year = ();
    foreach ($Y-10..$Y+1) {
        $Year{$_} = $_;
    }
    my %Month = ();
    foreach (1..12) {
        my $Tmp = sprintf("%02d", $_);
        $Month{$_} = $Tmp;
    }
    push (@Params, {
            Frontend => 'Year',
            Name => 'Year',
            Multiple => 0,
            Size => 0,
            SelectedID => $Y,
            Data => {
                %Year,
            },
        },
    );
    push (@Params, {
            Frontend => 'Month',
            Name => 'Month',
            Multiple => 0,
            Size => 0,
            SelectedID => $M,
            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;
```

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;
```

Ticket Modules

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) 2001-2009 OTRS AG, http://otrs.org/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/
gpl-2.0.txt.
# --
# Generates auto increment ticket numbers like ss.... (e. g. 1010138,
 1010139, ...)
# --

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

use strict; use vars qw($VERSION);

$VERSION = '$Revision: 1.26 $';
$VERSION =~ s/^\$.*:\W(.*)\W.+?$/$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;
```

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 examplary module to match emails and set X-OTRS-Headers (see doc/X-OTRS-Headers.txt for more info).

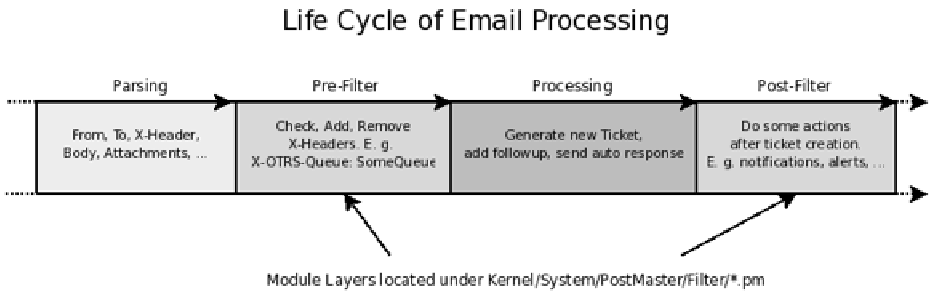
Kernel/Config.pm:

```
# 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) 2001-2009 OTRS AG, http://otrs.org/
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/
gpl-2.0.txt.
# --

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

use strict;
use vars qw($VERSION);

$VERSION = '$Revision: 1.26 $';
$VERSION =~ s/^\$.*:\W(.*)\W.+?$/$1/;

sub new {
    my $Type = shift;
    my %Param = @_;

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

    $Self->{Debug} = $Param{Debug} || 0;
    # get needed opbjects
    foreach (qw(ConfigObject LogObject DBObject)) {
        $Self->{$_} = $Param{$_} || die "Got no $_!";
    }

    return $Self;
}

sub Run {
    my $Self = shift;
    my %Param = @_;
    # get config options
    my %Config = ();
    my %Match = ();
    my %Set = ();
    if ($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 $MatchedNot = 0;
    foreach (sort keys %Match) {
        if ($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) {
       foreach (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
Parsing
Pre-Filter
Processing
Post-Filter
From, To, X-Header, Body, Attachments, ...
Check, Add, Remove X-Headers. E. g. X-OTRS-Queue: SomeQueue
Generate new Ticket, add followup, send auto response
Do some actions after ticket creation. E. g. notifications, alerts, ...
Module Layers located under Kernel/System/PostMaster/Filter/*.pm

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) 2001-2009 OTRS AG, http://otrs.org/
# --
# $Id: module-format.xml,v 1.26 2009/01/06 10:34:20 martin Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/
gpl-2.0.txt.
# --

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

use strict;

use vars qw($VERSION);
$VERSION = '$Revision: 1.26 $';
$VERSION =~ s/^\$.*:\W(.*)\W.+?$/$1/;

# --
sub new {
    my $Type = shift;
    my %Param = @_;

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

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

    return $Self;
}
# --
sub Run {
    my $Self = shift;
    my %Param = @_;
    # check needed stuff
    if (!$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;
```

Ticket Event Module

Do do some actions on ticket events, just write your own ticket event module.

These modules are located under "Kernel/System/Ticket/Event/*.pm". For default modules see the admin manual. You just need two functions: new() and Run():

The following example shows you how to unlock a ticket after a move action.

Kernel/Config.pm:

```
$Self->{'Ticket::EventModulePost'}->{'99-ForceUnlockOnMove'} =  {
    Module => 'Kernel::System::Ticket::Event::ForceUnlock'
};
```

Format:

```
# --
# Kernel/System/Ticket/Event/ForceUnlook.pm - unlock ticket
# Copyright (C) 2001-2009 OTRS AG, http://otrs.org/
# --
# $Id: module-format.xml,v 1.26 2009/01/06 10:34:20 martin Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/
gpl-2.0.txt.
# --

package Kernel::System::Ticket::Event::ForceUnlock;

use strict;

use vars qw($VERSION);
$VERSION = '$Revision: 1.26 $';
$VERSION =~ s/^\$.*:\W(.*)\W.+?$/$1/;

# --
sub new {
    my $Type = shift;
    my %Param = @_;

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

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

    return $Self;
}
# --
sub Run {
    my $Self = shift;
    my %Param = @_;
    # check needed stuff
    foreach (qw(TicketID Event Config)) {
      if (!$Param{$_}) {
        $Self->{LogObject}->Log(Priority => 'error', Message => "Need
 $_!");
        return;
      }
    }
    if ($Param{Event} eq 'MoveTicket') {
        $Self->{TicketObject}->LockSet(
            TicketID => $Param{TicketID},
            Lock => 'unlock',
            SendNoNotification => 1,
            UserID => 1,
        );
    }
    return 1;
}
# --
1;
```

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;

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 to 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.

Formatting

TAB: We use two spaces for every new open tag. 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>
```

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
```

$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.

$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"}.
```

$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>
```

$Env{""}

Env is an environment variable which is usable in more templates at a time. $Data{""} is just availabe 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
```

$QEnv{""}

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

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

$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"}
```

$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"}"}
```

$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"}'
```

$Include{""}

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

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

Yet another examlpe:

```
<html>
<head>
    <tilte>Some Title</tilte>
# include css.dtl file
$Include{"css"}
</head>
```

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>
        <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
foreach (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
    },
);
```

set

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

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

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"; }>
```

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.

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{""}.

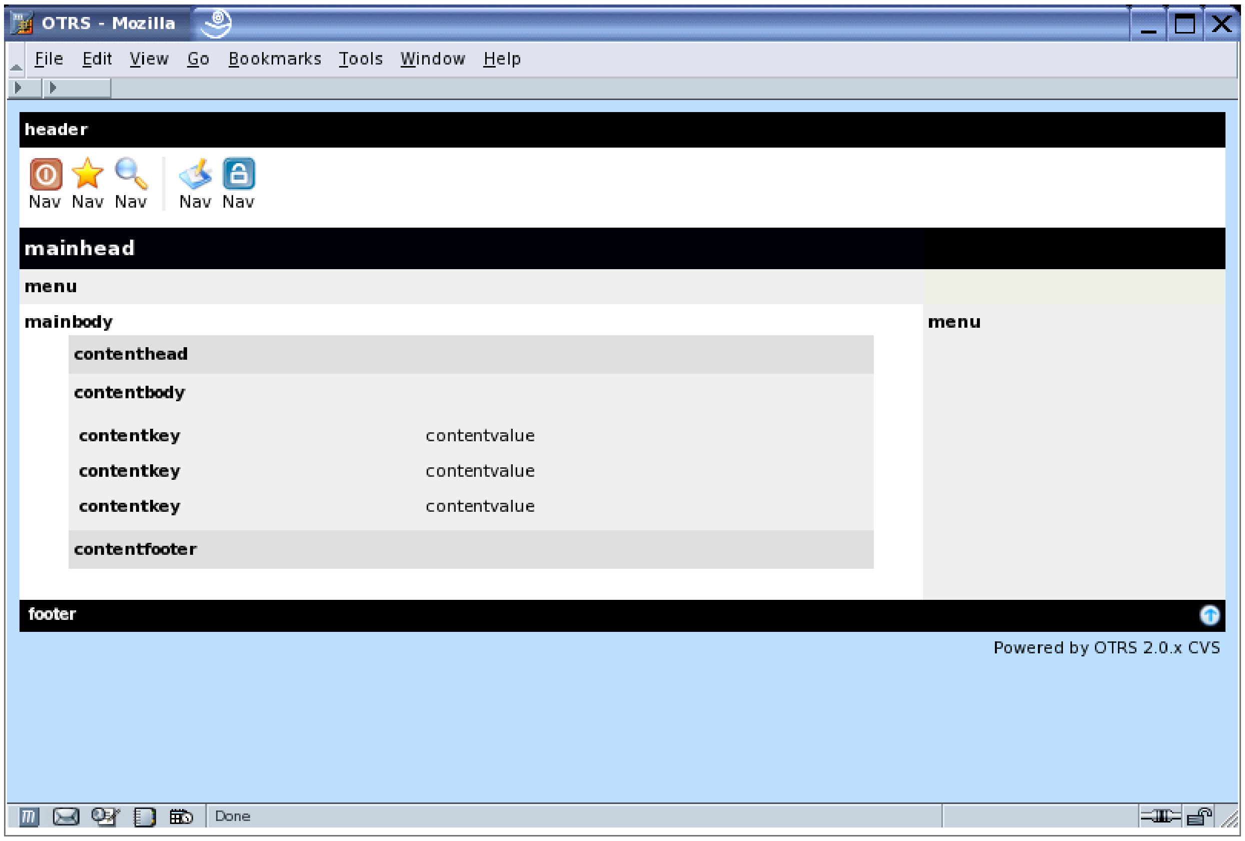
Example

```
# 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"}
```

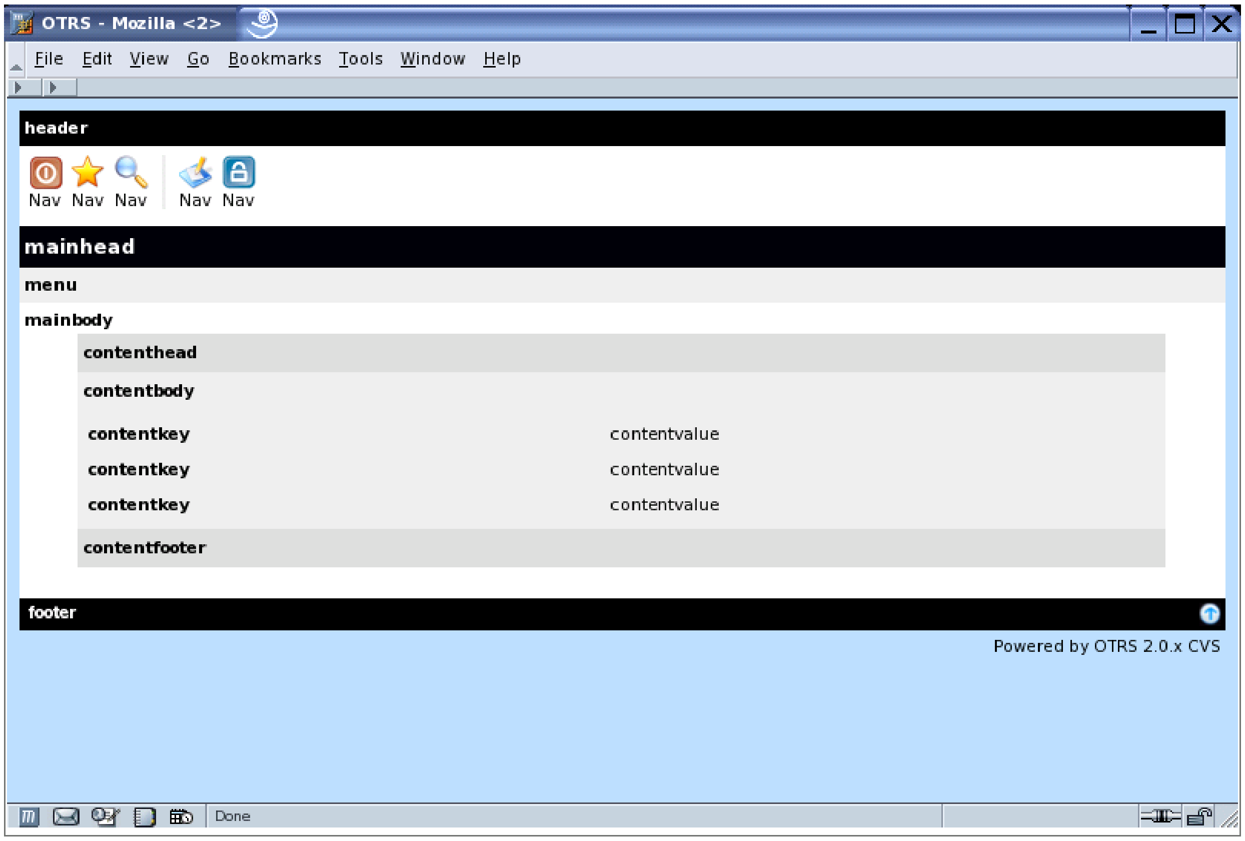
Chapter 9. Layout

CSS Style

View



Edit/New/Search



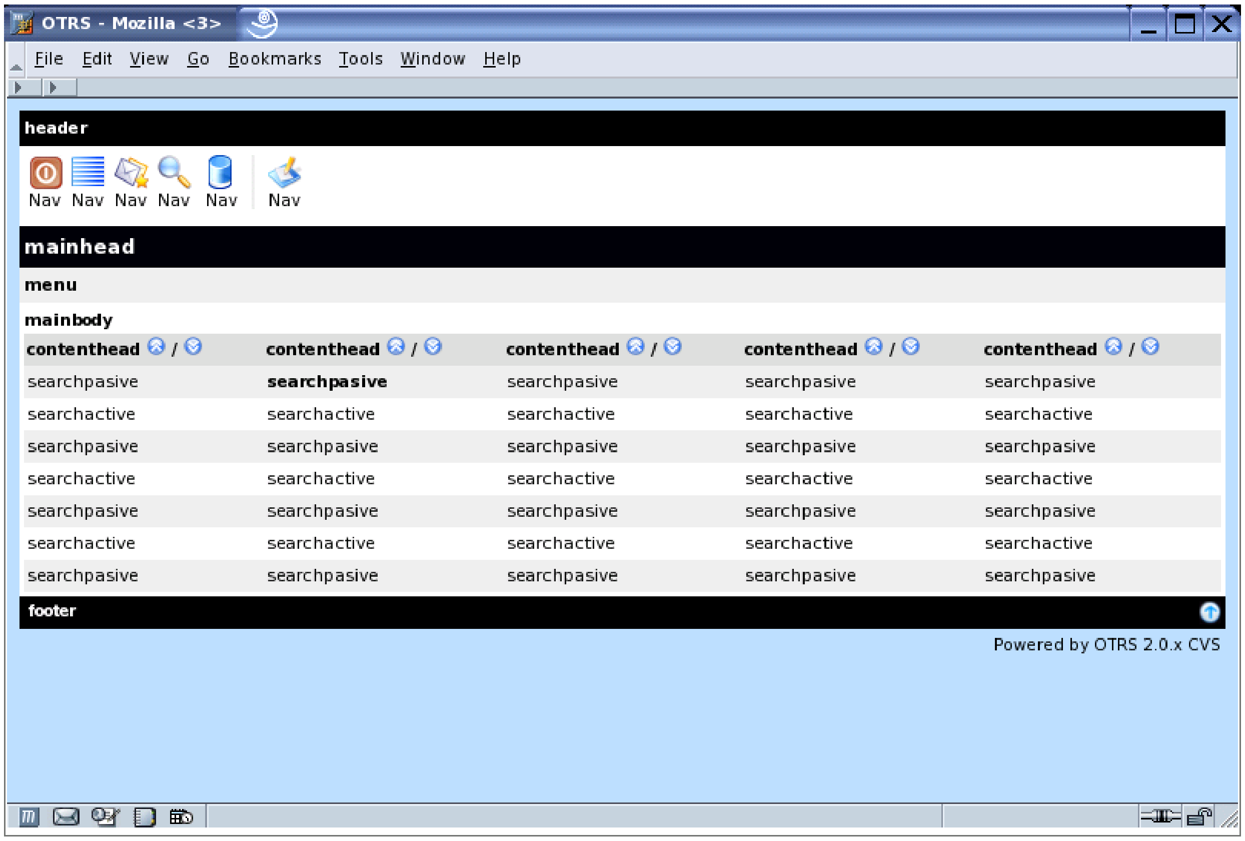
contentkey	contentvalue
contentkey	contentvalue
contentkey	contentvalue

contentfooter

Powered by OTRS 2.0.x CVS

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

Overview/Search Result



header

mainhead

menu

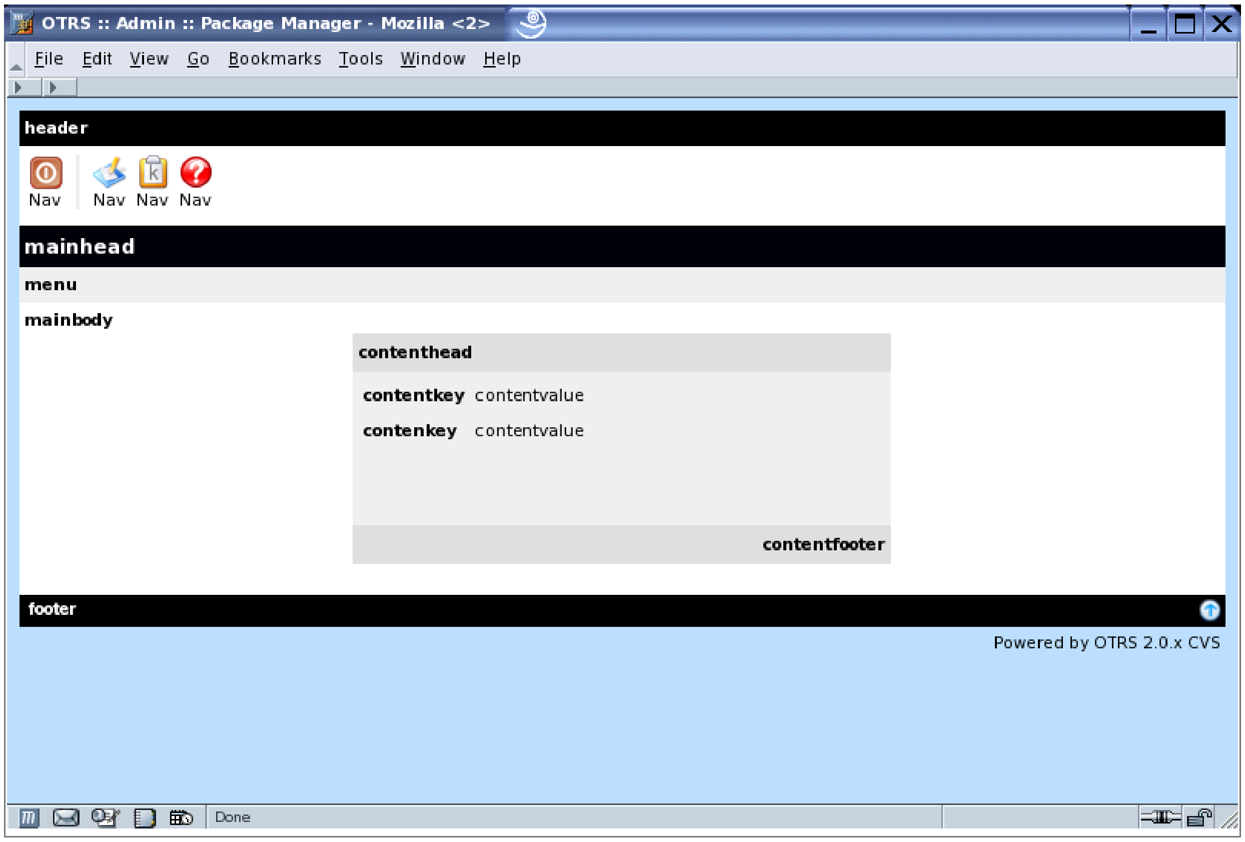
mainbody

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

footer

Powered by OTRS 2.0.x CVS

Delete


OTRS :: Admin :: Package Manager - Mozilla <2>
File Edit View Go Bookmarks Tools Window Help
header
Nav
Nav
Nav
Nav
mainhead
menu
mainbody
contenthead
contentkey
contentvalue
contenkey
contentvalue
contentfooter
footer
Powered by OTRS 2.0.x CVS
Done

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)

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"
 cols="1">
<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.

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`

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) 2001-2006 Martin Edenhofer <martin+code@otrs.org>
# --
# $Id: language-translations.xml,v 1.10 2006/04/02 23:25:09 martin Exp
 $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/gpl.txt.
# --
package Kernel::Language::de;

use strict;

use vars qw($VERSION);
$VERSION = '$Revision: 1.10 $';
$VERSION =~ s/^\$.*:\W(.*)\W.+?$/$1/;

# --
sub Data {
    my $Self = shift;
    my %Param = @_;
    # $$START$$

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

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) 2001-2006 Martin Edenhofer <martin+code@otrs.org>
# --
# $Id: language-translations.xml,v 1.10 2006/04/02 23:25:09 martin Exp
 $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/gpl.txt.
# --

package Kernel::Language::de_AgentCalendar;

use strict;
```

```
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';

}
1;
```

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) 2001-2006 Martin Edenhofer <martin+code@otrs.org>
# --
# $Id: language-translations.xml,v 1.10 2006/04/02 23:25:09 martin Exp
 $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/gpl.txt.
# --
package Kernel::Language::xx_Custom;

use strict;

use vars qw($VERSION);
$VERSION = '$Revision: 1.10 $';
$VERSION =~ s/^\$.*:\W(.*)\W.+?$/$1/;

# --
sub Data {
    my $Self = shift;
    my %Param = @_;

    # $$START$$

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

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

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 [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.

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

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)

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.

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.

NavBar Settings

A NavBar item should look like the following example:

NavBarPoint	**Prio**	**AccessKey**	**Image**
Overview	100	o	overview.png

NavBarPoint	Prio	AccessKey	Image
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
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

NavBarPoint	Prio	AccessKey	Image
ITSMConfigItem (OTRS::ITSM)	3200	-	itsm_configitem.png
ITSMLocation (OTRS::ITSM)	3300	-	itsm_location.png
ContentManager	7050	-	contentmanager.png
Calendar	8000	c	calendar.png
FileManager	8100	f	filemanager.png
WebMail	8200	w	webmail.png
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

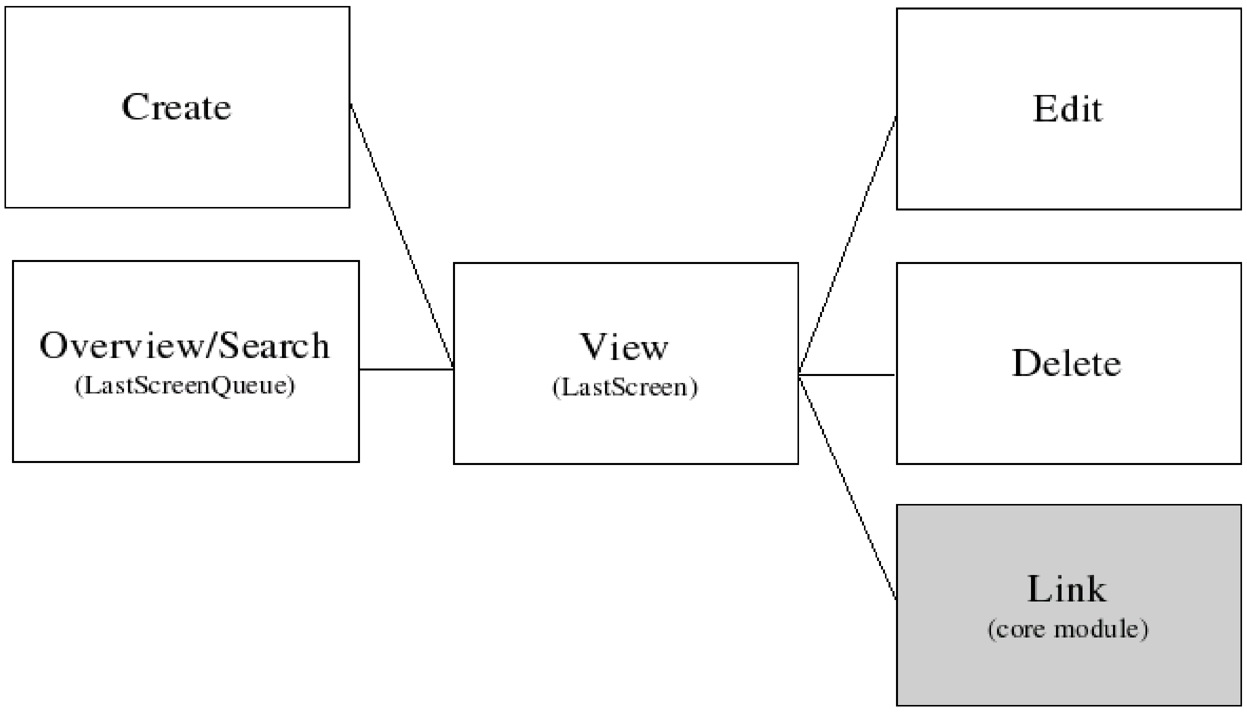
Table 3: Menu Applications -default application modules

Note

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

Screen flow

An object module should have the following process flow:


Create
Overview/Search
(LastScreenQueue)
View
(LastScreen)
Edit
Delete
Link
(core module)

Chapter 12. Development Environment

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

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 2.3 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 -r otrs
```

```
shell> cvs -z3 -d :pserver:anonymous@cvs.otrs.org:/home/cvs co -r
 rel-2_3 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.

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/
```

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.

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/

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="iso-8859-1" ?>
<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>
```

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) 2001-2005 Martin Edenhofer <martin+code@otrs.org>
# --
# $Id: writing-otrs-application.xml,v 1.11 2006/06/07 14:10:34 tr Exp
 $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/gpl.txt.
# --

package Kernel::Modules::AgentHelloWorld;

use strict;
use Kernel::System::HelloWorld;

sub new {
    my $Type = shift;
    my %Param = @_;

    # allocate new hash for object
    my $Self = {};
    bless ($Self, $Type);
    # get common objects
    foreach (keys %Param) {
        $Self->{$_} = $Param{$_};
    }
    # check needed Opjects
    foreach (qw(ParamObject DBObject ModuleReg LogObject
 ConfigObject)) {
        $Self->{LayoutObject}->FatalError(Message => "Got no $_!") if
 (!$Self->{$_});
    }
    # create needed objects
    $Self->{HelloWorldObject} = Kernel::System::HelloWorld-
>new(%Param);

    return $Self;
}
# --
sub Run {
    my $Self        = shift;
    my %Param       = @_;
    my $Output      = '';
    my %Data = ();

    $Data{HelloWorldText} = $Self->{HelloWorldObject}-
>GetHelloWorldText();
    # build output
    $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;
```

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) 2001-2005 Martin Edenhofer <martin+code@otrs.org>
# --
# $Id: writing-otrs-application.xml,v 1.11 2006/06/07 14:10:34 tr Exp
 $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/gpl.txt.
# --

package Kernel::System::HelloWorld;

use strict;
```

```
sub new {
    my $Type  = shift;
    my %Param = @_;

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

    return $Self;
}

sub GetHelloWorldText {
    my $Self        = shift;
    my %Param       = @_;

    return 'Hello World';
}

1;
```

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) 2001-2005 Martin Edenhofer <martin+code@otrs.org>
# --
# $Id: writing-otrs-application.xml,v 1.11 2006/06/07 14:10:34 tr Exp
 $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/gpl.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.

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;

sub Data {
    my $Self = shift;

    $Self->{Translation} = { %{$Self->{Translation}},
        'Hello World'  => 'Hallo Welt',
    };
}
1;
```

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.

Package Distribution

If you want to create an OPM online repositiory, 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]',
};

[...]
```

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>
```

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.

Install

Install OPM packages.

- Web: http://localhost/otrs/index.pl?Action=AdminPackage
- CMD:

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

Uninstall

Uninstall OPM packages.

- Web: http://localhost/otrs/index.pl?Action=AdminPackage
- CMD:

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

Upgrade

Upgrade OPM packages.

- Web: http://localhost/otrs/index.pl?Action=AdminPackage
- CMD:

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

List

List all OPM packages.

- Web: http://localhost/otrs/index.pl?Action=AdminPackage
- 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.

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.

Name

The package name (required).

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

Version

The package version (required).

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

Framework

The required framework version (2.3.x means e.g. 2.3.1 or 2.3.9) (required).

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

Can also be used several times.

```
<Framework>2.3.x</Framework>
<Framework>2.2.x</Framework>
<Framework>2.1.x</Framework>
```

Vendor

The package vendor (required).

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

URL

The vendor URL (required).

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

License

The license of the package (required).

```
<License>GNU GENERAL PUBLIC LICENSE Version 2, June 1991</License>
```

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>
```

Description

The package description in different languages (required).

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

BuildHost

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

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

BuildDate

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

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

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>
```

ModuleRequired

Perl modules that must be installed beforehand (optional).

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

OS (^M)

Required OS (optional).

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

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>
```

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".

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".

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".

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".

IntroInstall

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

```
<IntroInstall Type="post" Lang="en" Title="Some Title">
Some Info formated 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).

IntroUninstall

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

```
<IntroUninstall Type="post" Lang="en" Title="Some Title">
Some Info formated 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).

IntroReinstall

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

```
<IntroReinstall Type="post" Lang="en" Title="Some Title">
Some Info formated 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).

IntroUpgrade

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

```
<IntroUpgrade Type="post" Lang="en" Title="Some Title">
Some Info formated 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).

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".

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".

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".

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).

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>2.3.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>
```

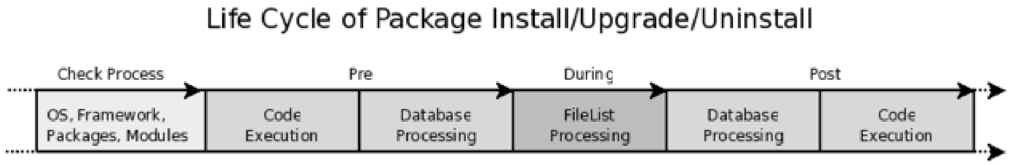
Package Build

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>
```

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.



Chapter 16. Unit Tests

OTRS provides unit tests for core modules.

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) 2001-2005 Martin Edenhofer <martin+code@otrs.org>
# --
# $Id: test-mechanism.xml,v 1.4 2006/04/02 23:25:09 martin Exp $
# --
# This software comes with ABSOLUTELY NO WARRANTY. For details, see
# the enclosed file COPYING for license information (GPL). If you
# did not receive this file, see http://www.gnu.org/licenses/gpl.txt.
# --

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()',
);

[..]
```

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>
```

True()

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

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

False()

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

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

Is()

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

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

Appendix A. Additional Ressources

OTRS.org

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

http://otrs.org/

Online API Library

The OTRS developer API documentation is available at:

http://dev.otrs.org/

Developer Mailing List

The OTRS developer mailing list is available at:

http://lists.otrs.org/

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/
```