Two-spot Manual

Two-spot Manual

Table of Contents

List of Examples

Chapter 1. Introduction

Welcome

This is Two-spot, an open-source administrative package designed to keep track of customers and to store all pertinent information about them.

License

This manual is licenced under the GFDL.

The TwoSpot software system as a whole is licensed under the GPL version 3.

Features

These are some of Spot's primary features:

- an easy-to-customize system that allows you to add your own specific data
- flexible system for defining customer relationships, both between your customers and between you and your customer
- Automated order and invoice handling
- all record changes are logged and can be reviewed on a per record basis
- heartbeat and timer system that runs cleanup in the background.

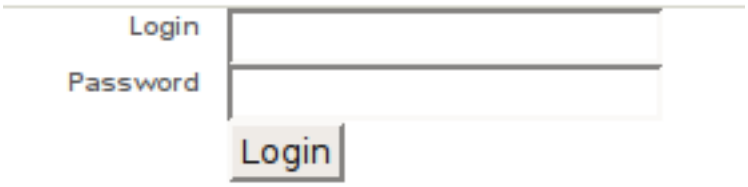
System requirements

TwoSpot has been tested mostly on Ubuntu 8.10 and Fedora 8. It should run on any LAMP compatible system. In theory most will run on a Windows WAMP stack, but there are binary files used that only come standard on a Linux system.

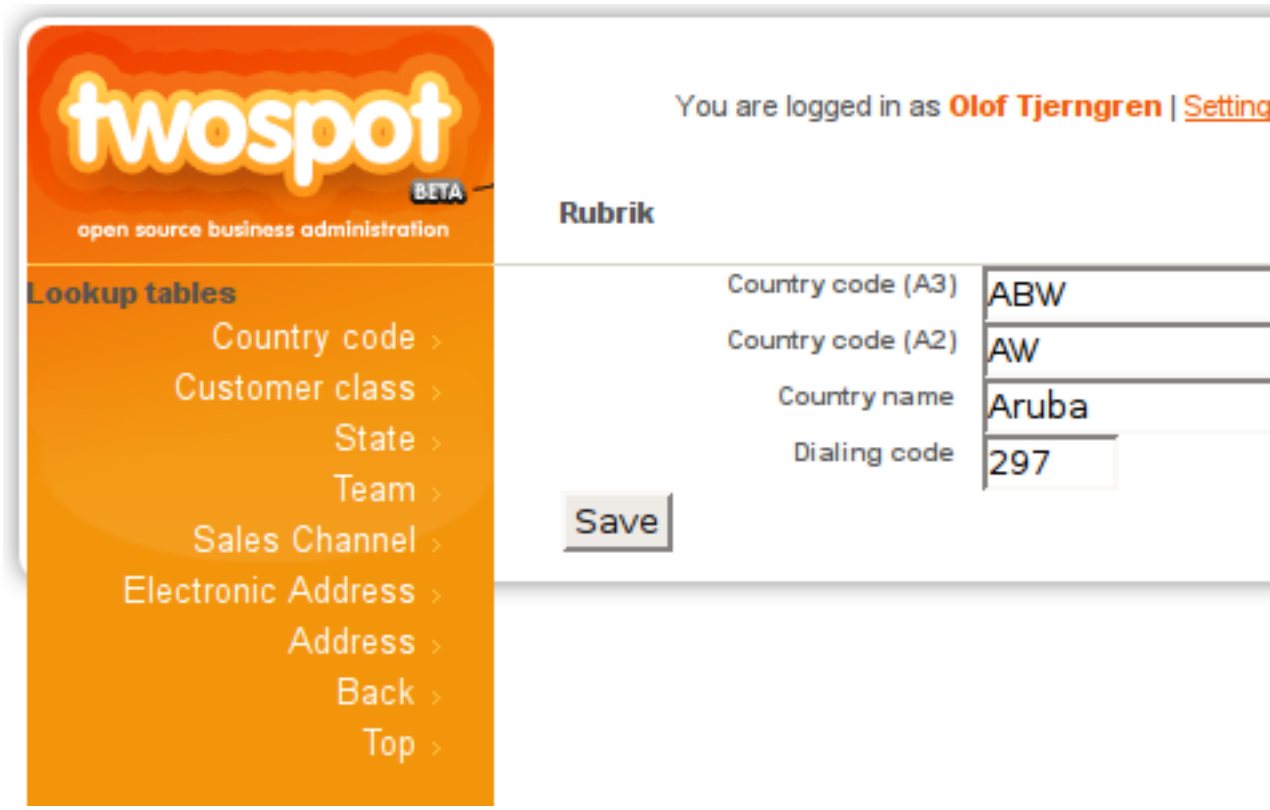
Chapter 2. General usage

This chapter describes general navigation of the system, and how the forms work.

The first contact is the login page, very simple.



After login you have a three areas available to you. on the left is the menu, on the top right you have settings, login and a search form. The middle of the screen is the workarea where all the data will be shown and where it can be modified.



The menu is fluid and will scroll to the left when a new menu is needed. Links shown in forms can also update the menu, in the above example the country code form doesn't have it's own menu, so the existing menu remains visible. Learning to navigate the menu is half the knowledge needed to use two-spot. On the top right is the login information a setting and logout link and a search form. The search will find most everything and should be the most used search function. It's also possible to use the menu to go to the section that shows what you want to search for (customers for example) and you can filter the list shown, the search function is usually quicker though.

Settings let you change some settings for your own user and how data is displayed for it.

The middle of the screen is the workarea where you'll do actual work, in the example above it's editing a country code definition. Note there's only Save an an option, if you don't want to make any changes, simply leave the form and click on something else.

Simple forms will look like the one above, you keep the menu on the side. Records that require more work will have their own menu with functions specific for that record. It could be special views or list of related records to make editing them easier. Most will also have three items shared between every form. Audit trail, Connected and Notes.

Audit Trail

This will initially show changes made to the record with the most recent change on top. Each update has a type (the first one is an insert, all the rest are updates), a date, a person who made the change and what fields where changed. It's possible to see changes to related records as well, click the link on the top of the list to show also those changes. In this case the record updated will also be visible. For Customers this allows you to see changes to email addresses and the team working on this customer for example.

Connected

Using connected will update the menu and give a list of records related to this one. This includes configuration records, if you have access to that you go from a customer to customer class and from there to see all other customer of the same class for example. If for whatever reason it's hard to find what you're looking for, you can use this to search for something related and then use connected to go to the record you wanted. The menu will say the name of the record and how many related records there are, click the menu to get a list of every record connected with links to open them up.

Notes

Notes are all stored together for easier searching, and to make it easy to add to new types of records. You can write anything here and also have discussions. Notes show discussions threaded to make them easier to follow.

Field types

Most fields, like in the country code example above consist of entering text straight up. Although there are other kinds of entry and edit fields available. Before each of these is always the name of the field you're entering.

Text fields

Text fields comes in two varieties, a single line input with a limit on the lenght and a full text entry where you can enter new lines and format text. The single line entry is always limited to the length that can be stored in the database, if you can't type more it's for a reason. Full text fields doesn't have a limit in the field, but there is still a limit in the database, although it's large enough that you'd need to enter the text of several books before reaching the limit.

Radio buttons

These allow one value to be selected and are used when only one selection is valid.

Checkboxes

Checkboxes are allow selecting several values at the same time but work and look similar to radio buttons otherwise.

Dropdowns

These have the same effects as radio buttons, you can select one value, but there are more values available and the content can be more dynamic.

Date and time fields

Date fields looks like normal text fields, but show below them what the date you enter is interpreted as. This is because it's free form. You can enter dates as "2009-01-31", "01/31/09", or "jan 31 2009" to get the same date stored. It's also possible to use relative dates. Note that relative dates are based on the previous value in the field when editing or sometimes a default value when adding new information. Entering "today" will show todays date if there is no default or previous value, but the previous entered day or default if it exists. Entering a weekday like "monday" will give you the next monday related the existing date, "+4 days" will show four days into the future and so on. See the calendar chapter for more example of usage.

Inline edit forms

Some forms are shown with only a summary text covering different parts of the data, they're surrounded by a green border. It's possible to click on the area to open a form that replaces the text and edit the data directly. If possible they have two action buttons, one to delete the information and another to save the updated data. When the data is saved, the form goes away and is replaced with the text description again.

Linked related information

This widget is a list of linked information, clicking the Edit bar will open a list of all available items that can be linked to the original record. Simply drag them to the left side to have them linked to the item, or back to the right to un-link them. The information is saved when the full record is saved.

Chapter 3. Customers

The following describes how customers, persons, companies, and organizations are added and navigated.

Creating and maintaining customer records are the most basic of Spot's functions. Almost all information is linked to customers, so having up to date customer information is essential.

Finding and creating new customer records

One option is to use the search function, the second to use the menu, from the top level menu, click customers and then companies or persons depending on what you want to find. You will get a list of all companies or all persons available in the system. Use the filter function and enter the name or any other field visible in the list to quicker find what you are looking for.

If you don't find what you're looking for and it's a new customer, click Add row or add customer to create a new record.

It's also possible to use the search box in the top right corner to find customers. The search will find several items in the database, but it will first look for customers before going on to other things.

Editing a customer record

When you open a customer the first screen will show the name of the customer followed by a tree of the relationships below that customer and next to that some information on the customer itself. If the customer belongs to other entities there will also be a list of those above it. When you find a person you can use that to go up to the company the person works for as an example. The tree will show those below, so in combination you can move both up and down in the list. When you select an employee below the customer the right will be replaced with that new record.

The menu on the side with Send email, Audit Trail, Connected and Notes applies to the company or person that you have opened, always the top level in the tree view and named on the top of the screen.

You can edit any entity in the treeview or move up to a higher level, but the menu always refer to the anchor entity currently viewed. If, for example, the Audit Trail of a lower level entity is of interest, simply click "[Edit]" on that entity to open that as the top level.

Customer information view

On the right side of the customer relationship tree is the edit forms, click on the section to change to get an editing form to update the values there. To add new data it's still required to click on Edit Entity and get full access.

Related customer information

Below the direct customer info is some related information, electronic addresses like email and phone followed by physical addresses and the team of people who work with that customer.

Roles and relationships

Customer can be either persons or companies. They are connected through roles that shows relationships between them. The system has to be configured with the types of roles needed.

For example, let's say we have employee -> company as a role. That will allow us to link persons to companies as employees. We could of course also have member of board -> company as a separate role. Let's also assume we have a role branch office -> head office that links two companies.

Note that with roles you can connect both ways, it's possible for a company to have several employees as expected, but it's also possible for a person to be connected to several companies. For example you can have a person that's a member of the board on 10 different companies and employed at yet another one.

Now we can enter the head office (Astronaut Service) as a new company with some employees. Once that is done, we click on connect entity and pick to add a new branch office (Astronaut Service London) from the list there. Once the details are entered the main screen will update with the branch office and there will be a link back to the head office.

We now have a structure that looks like this:

Astronaut Service
  Employee Jan Jansson
  Employee Lars Larsson
  Branch office Astronaut Service London
    Employee Nils Nilsson
    Employee Sven Svensson

There is a special kind of role usually named "Duplicate". Say we notice a duplicate record of Astronaut Service with another employee called Hans Hansson. We fix this in two steps, the first is to tell the system that there is a duplicate. This is done by connecting the duplicate company to the main company. We find the main company and click on the connect entity link, choose duplicate account on the role drop down and search for Astronaut.

This will give a list of both the other Astronaut service companies, one will already be connected to the main account though the branch office role so we click on the connect link for the other to create the role between them.

The new structure looks like this:

Astronaut Service
  Employee Jan Jansson
  Employee Lars Larsson
  Branch office Astronaut Service London
    Employee Nils Nilsson
    Employee Sven Svensson
  Duplicate account Astronaut Service
    Employee Hans Hansson

This new structure will stay in place until the actual merge is done. The link is used to tell the merge routine what to join, so it's important to have the main account roles setup correctly. The main account will be the one that remains after the merge, all the information on the duplicate accounts will be moved over to the main account and once it's just the name that remains the duplicate record will be deleted. This is a harmless operation as long as it's a true duplicate. Merging two different companies into one is possible, but it's very difficult to separate them out again. Other than that the operation is harmless.

After the merge is completed the structure will look like this:

Astronaut Service
  Employee Jan Jansson
  Employee Lars Larsson
  Employee Hans Hansson
  Branch office Astronaut Service London
    Employee Nils Nilsson
    Employee Sven Svensson

If you want a description of how the actual merge is done, read the Administration chapter of this manual.

Chapter 4. Orders and Invoices

A complete list of orders can be viewed on the same level as all customers. On each customer there's also a list with the orders from that customer and an menu item to add new orders.

Orders are the start of the workflow for handling quotes, sales orders, invoices, credit invoices, collection and delivery. The order record itself is always the status document, showing where in the process each customer is for one or several orders.

The workflow start with adding and direct sale order or a quote. Quotes are not implemented yet, currently it's simply a step before order.

The next stage is to invoice, To do this change the status of the order to Invoice and an invoice will be generated automatically. On the legal entity is a field to pick what template will be used for the invoice, and the template itself can have several translations that are picked by rules defined when the templates are created.

The generated invoice pdf file is attached to the customer, the order and the record of the invoice and also visible in the document view.

Adding a new order

From the customer it's possible to add new orders, they can start as Quotes or directly as a sale order. The arrival date defaults to today, but can be modified if needed, it's a pure date, the actual time is not recorded.

Currency is a list of configured currencies. Paymethod can be Invoice or credit card, since credit card handling isn't available Invoice is currently the only choice. Our ref defaults to whoever is logged in and adding the order. PO# is the customer Purchase Order number and is optional.

Contact person, delivery address and invoice address is all selected from the customer data. Sales channel and payment terms are defined in the system for possible values.

Each row contains one product and the quantity the customer ordered of that item. It's also possible to set a discount on each row. Click on "Add order row" to open up another row to enter.

Editing an order

The order editing form has the standard layout, looking similar to adding new orders. The menu has a few extra options however. "Order Dates" will show all dates on the order and allow those to be edited. The Status field will only list states that it's possible to enter from the current state, so from Quote both Order and Pending are possible, but from Pending it's not possible to Invoice, the state has to change to Order first and from there go to Invoice.

Comment is intended for internal notes, while Invoice note is intended to be printed on the invoice and visible to customer. However, this depends on the template used and that can be created to contain anything so thi is only convention.

Invoices will list all invoicing and crediting done for this order. Documents will show all documents connected to the order. Invoice PDF files will automatically be connected to the order when generated.

Orders are moved through a workflow by changing the status of the order. Any automatic handling will take place when the status changes. From status Order to status Invoice will generate invoice data for example, and switching from Invoice to Paid will also update dates and other fields.

Invoicing and crediting.

It's possible to invoice any order with the status Order. To create the invoice, simply change the status to Invoice, The Invoice record will be created automatically from the order record, and the order record will be locked from further editing.

If there was a problem with the order and the customer needs to have something altered before paying it, it's possible to change the status from Invoice to Credit Invoice and make changes. After fixing the problem switch the status back to Invoice to generate another invoice with the new information.

In the above case, looking at the invoicing history of the order will show three invoices. A normal inital debit invoice, a credit invoice matching the initial debit and finally a new debit invoice that has the updated values. The order itself will match the newest invoice.

After the invoice is paid, change the status to Paid.

Chapter 5. Employees / Users

Users are equivalent to employees, the concept will allow you to login if you have access granted, but will also allow you to be linked in as an employee to other records and transactions.

Adding a user

Click on Employees on the top menu, then "Add row" in the list. Fill in the known information and save. This is all that's required to have an employee added. If you need the employee to be a user you will also need to set a password and give access. Access is granted by adding the user to a group, the user will inherit the access granted to the group. See the Configuration chapter on how to set up groups and access rules.

Chapter 6. Calendar

The calendar defaults to a simple montly view with all selected activites listed per day. You can enter a new activity by clicking on the + icon on each day, and open up an existing activity by clicking directly on it.

Each activity has a start date-time and an end date-time, so it's possible to enter both single day events and multi day events. When you enter a new activity through the calendar it only sets a default, you can change the dates in the entry form to get the activity on a different date, it's just a bit more typing.

> **Note**
>
> On the calendar page, every time and date is converted to your local timezone, you might see a two hour activity from a differerent country span two days for example, if you open the activity you will see that it's actually in the afternoon, it's just happens to be around midnight your local time.

Calendar setup

There is some setup needed for the calendar to work properly, you need to make at least one calendar subscription - preferbly your own so you can see your own activities.

There are five subscriptions available, each with it's own colour. They're simply named color-1 up to color-5. With each colour you can select if you want to subscribe to a user calendar (all activities for that user) or an activity type. The latter case can be useful to see when people is on vacation for example.

If an activity fits into several subsciptions, your own vacation for example, it will be shown in the colour with the lowest number will be used in the calendar, so if your own activites use color-1 you will always see your own activites in the same colour. The subscription setup is personal, so different people using the system can each have five subscriptions to different activites. Note that you cannot see private activites, even if you do subscribe to another persons activities.

Calendar views

There are three calendar views, month, week, and day. They all show the same data. Under personal Settings (next to the search box) you can pick the default view you want to have when entering the calendar. In all views you can click on the + sign to add new activites for a day and click the activity to open it up to edit or change the status. Note that all views will show the activity in your local time, after you open it up it will show the time in the timezone the activity takes place.

Monthly

This will show all activities for a full month in a traditional calendar format, you can change the first day of the week by changing your settings.

Weekly

This view will show a week of activites and also non working day (holiday) information.

Daily

The only difference apart from only showing one day from the Weekly view is that you on this can click the add activity link on each hour to add an activity for that time. This view will also show non working days.

The Activity form

This form is used to both enter and edit an activity, you start by selecting the type of activity, this is defined elsewhere and is used to filter the activities for different views. You decide when setting up the software how you want to partion your activities.

The following two fields are "Starting" and "Ending", it defaults to now as the date already there, so if you click from the calendar and have already picked a day you only need to enter the start and ending times into these if it's a shorter activity that runs a fixed time. To enter a lunch for example you can enter "12:00" and "13:00" or "12pm" and "1pm" depending on what makes sense to you.

It's also possible to use the relative dates, if you need to push an appointment from monday to thursday, enter "thu, 12pm" for example, it will be calculated based on the previous date, so even if it's a few weeks in advance the date shown will be correct. If a date cannot be parsed it will say it's unfaimilar and you'll have to update. The warning will probably show while you edit it.

The information field contains any text that describes the activity. The first few words will be shown directly in the calendar so start with a short overview if possible.

The status fields are Tentative - for not fully committed activites, private - you don't want this to show up for anyone else. Completed is set after the activity is done and Cancelled after it's been cancelled. Public is the opposite of private, it means that the activity is fine to show to people outside the company on a website for example.

The last field is the timezone this activity takes place in. If you have people in different timezones the calendar will automatically convert dates and times as needed to show all activities in their local time instead of the Timezone activity. If you have an activity in a timezone different from your local timezone it can show up as midnight in the calendar, but if you open the activity to edit it you'll see that it's actually at one in the afternoon but in a different timezone instead. To people in that other timezone it will show up with the same time in both the calendar and directly in the activity view.

Date entry examples

Date are normally shown in the form 31 January 2008, 10pm (or 22:00). You can use this format, or the iso yyyy-mm-dd, the american mm/dd/yy, or others. Use a comma to separate the date from the time if you need to enter both. If you don't enter a time midnight is assumed in most cases.

Example 6.1. Sample dates

Last january

Next week

+90 days

+3 months

-4 weeks

january 2010

next monday

The date parser is based on the javascript datejs library from datejs.com. It's flexible so if you feel like something should work, it most likely does. If it works you can keep doing it, if it fails, simply try something else.

Chapter 7. Documents

Documents can be stored on the file system or in the database, currently only the database layer is available. Documents are displayed chronologically, by selecting year and month you get a list with previews of all documents created on that day. Clicking on a document preview will let you download the file. Currently only images, movies and pdf files are supported.

Documents generated automatically like invoices for example will be added to the view and also linked to the appropriate customer/order/invoice records.

Chapter 8. Administration and upkeep

Administration - Database

This set of functions allow you to control the database structure and to perform some routines to verify and fix some problems that might occur with the data stored in the database. There are two screens used to handle the database; Database that has functions to update and verify the structure of the database and Information where the functionality is focused on checking the data stored in that structure.

Upgrade DB structure

There are two functions for this, preview upgrade and upgrade. Preview will simply show you what will be done if the database structure is updated and upgrade will actually change the structure. You can use preview to get a general idea of how much will be changed. There will of course only be changes if the software has been updated. Changes can include adding columns, removing columns or adding tables. Tables will not be deleted.

There are a couple of files in different directories, all called tables.struct. These describe the table structure to use. What these two functions do is to compare the described structure for all tables that are described in the text file with the actual tables in the database. If there are differences it will issue SQL statements that update the database to be the same.

Merge duplicate customers

This function takes care of merging customers that are marked as such. You'll have to choose what role is used to mark duplicates, usually Main account - duplicate account. Select that and get the resulting list. If there are marked duplicates you can now click on the main account and get a list showing the main account followed by all duplicate accounts. If you spot any errors in the duplicate account list, you can click the company name to edit the company and remove or change the role that links it to the main account.

If the list of duplicates is correct, click on the merge customer button to actually move the information from the duplicate accounts to the main account. Everything will be moved as is, there's another duplicate check done on address and phone/fax/email numbers on the company but it will have to be exactly the same to be merged automatically on the main account. Once all the records linking to the customer has been moved to the main account the duplicate account will be removed.

Chapter 9. Program overview

Spot contains several parts and subsystems. This chapter gives an overview of those from a semi technical standpoint.

The program

The program consist mostly of PHP code, with some java script to handle the interface.

PHP code

The PHP source code is divided into three parts; share - common library code. twospot - all the basic functions and the web user interface, and local - adaptations containing installation specific code.

Methods used to write adaptable/configurable code

A lot of configuration is done through the database, setup tables exist for almost everything. In a few cases the database will also include a filename and a function name. For these the general framework and engine of the function is written in code, and when the data has to be sent to another company or some calculations done, the correct file is included and the actual work done there, the database only controls what library is used. Usually the files included contains fairly simple code.

For lookup tables when the choice matters, there's usually a separate field in the data that's an enum or similar. For example in the eaddress_type table that defines what type of phone/email/fax/IM and whatever codes can be entered on a customer there's a field that says if this type of electronic address is a phone number or an email.

Notes

All records can have linked notes, they are stored in the same table to make it easy to show all notes on all records connected to a company to the company itself with only one query. Table name and key value are used to link the data to the actual record that they referee to.

Forms

The code is setup to make it easy to add new tables, this extends to making it easy to add new forms, one form for one record is the general rule. The normal presentation of data is done through the inputLayout function that will create the correct HTML input tag for the data that's supplied. There's a slew of other functions to show connected information in different ways depending on how it should be presented.

Meta data

The database structure is entered into a file called tables.struct. It's a simple XML format that describes how to generate the "create table" for new tables and "alter table" when there are changes to the existing structure. It also contains some extra information on how to use the data, like a display name for the column and search criteria to use when searching the data. When used the definition in the file is compared to the actual structure live in the DB and the appropriate commands are issued to make it the same in a non-destructive manner.

The menu system

Menus are defined in the menu.xml file, each menu has an id and a name. It can also have a file that will be loaded when the menu is picked. Then there's a list of items that link to other menus. If the menu doesn't have a list of other menus the menu itself will not be updated when it's clicked from somewhere else.

Connecting to the database

Access to the database is though the $GLOBALS["conn"] variable for selecting data. There are a few handy functions to select data from the database or display data, but nothing in the API to loop over and process data. Mostly because the mysql_query and mysql_fetch_assoc are easy enough to use on their own. The only select functions added have been to make it easier to write the code.

For insert, delete, and update the doInsert, doUpdate and delete_rec functions are used. delete_rec uses a different naming standard to make it a bit harder to write the wrong command by mistake. When these are use, the call in a bunch of classes. The first being SqlHandler that are inherited and used by SqlInserter, SqlUpdater and SqlDeleter. These are not used directly but always trough the doInsert and friends. The SqlHandler objects will in turn require and use (if they exists) two more classes, validate and postprocess. Validate is used to check the integrity of the data and to set some useful defaults. It also has the power to refuse an update if there is information missing or if for some other reason the action should not be executed. Postprocess is used as a PHP replacement for triggers and does things that would normally be done using that.

Data integrity

The SqlHandler classes described above will check for some integrity, and will also make sure that any changes to the database is logged and can be viewed later. Logging is done to the cust_log table that has the user doing the action, what type of update, the IP address, what table and the datetime the action was performed. It also has a difference as a text in the form +field: 1 -field: 2 for a change where field changed it's value from 2 to 1. Text and blob fields are only logged with the first 60 characters. The cust_log table can be accessed from each record and you can see a list of all changes to that record, it's also possible in the software to include changes in tables that link back to this table if needed. Deletes are also logged, so in theory it's possible to recreate deleted records using this. In practice since long fields are cropped it's not an option right Now. Changes to blob and text fields should be logged in a separate table with difference data to allow for that.

Java script code

The twospot.js file controls the interface. It's responsible for the menu and to call in the right new pages to show.

External software used

Spot can use a few pieces of external software, for example htmldoc is used to generate PDF files from HTML files.

The database

The database contains a lot of tables, many are configuration table and many contain actual data. Configuration data is classified into two categories, Definition and Configuration where definition is a simple lookup value, used only by the users of the system and mostly ignored by the system itself. Configuration on the other hand contains values that is used to control the logic of the program.

The central data table is cust_entity where customers are stored. Both companies and persons are stored in the same table. The table contains only names and codes that belong to the entity, with some default links for connected information. Everything else is stored in records connected to the cust_entity. The reason not to have two tables, cust_company and cust_person is to allow linking extra information directly and through a single link to the entity and be able to have the same functions regardless if it's a company, person or something else. Companies and persons are linked together in a separate table called cust_role, since the entity table has both companies and persons, the link table allows linking of persons - person, person - company, company - company and also company - person. Cust_entity has a field that contains either C for company or P for person, it can of course also be expanded to allow for other kinds of entities should the need for that be discovered.

The behaviour of the cust_role table is decided in the roletype table, here you define the roles that should be used by entering the type of entity the up link should go to, the type of link the down link should go to and also enter descriptive text for both links. For example, linking a C entity to a P entity the descriptions might be Company and Employee, or Employed at - Employee if that's preferred. Having a separate table for the roles also means that you can have several roles to a company. If owner and member of board is interesting to know, you can have one person having both those roles to a company for example.

The combination of cust_enitity and cust_role gives a very flexible system, it also gives some extra work in implementing how it should be handled, but so far the benefit of flexibility has been greater than the extra work required to implement it.

The database structure itself is fairly standard, surrogate keys are used almost everywhere mostly because of speed issues. Addresses are stored in one table linked directly to cust_entity, email, fax and phone numbers are stored in the same table as a companion to the address table and is called eaddress. This is also a shortcut that works well for the user in an interface but sometime gives problems when you want to select an email. Usually the problem is that a customer can have more than one email, more than the fact that it's mixed up with phone number. The type of electronic address are defined in a separate table where you also tell the system what names are phone and email. This allows for future expansions in adding VOIP access numbers and other future ways of communication that can be described in a single line of text.

The rest of the tables connected to cust_entity follow the same model, usually there's a type associated, if the data is processed automatically at some point, there's usually one or several type tables that are linked to get the description and if there are more tables in turn connected the graph will usually show a star shape with cust_entity in the middle. There's also a fair amount of inter linking between table when that's needed.

Chapter 10. Adaption

The system is built to be easily extended by writing new code. There are some skills required for the easy part to hold true. You have to know HTML, PHP, and MySQL.

The recommended way to do this for location adaptations is to use the local subdirectory. Any modification done there will survive an upgrade of the system with no or minor modifications.

This section has hints and tips on how to do things as well as what functions and classes are available to use. Some are optional, some have to be used for the system to work properly.

General layout of source

There are four main directories used. 2lib/, share/, twospot/ and local/. 2lib and share contains mostly library functions. twospot is the main application and local/ can contain local adaptations of the programs behaviour. r/ contains ajax response code.

Program flow

When you first load, an html file is loaded that fetches all the javascript and then tells the server what interface to use and asks for the first page or a login page if the user isn't logged in yet. This html file will stay in memory and only the parts that changes will be fetched from the server and inserted.

Modifying forms

This is the simplest kind of modification. It includes changing the layout to a form, adding more fields to a form and similar. Forms are normally stored in forms/ and the file name is the name of the table followed by the .frm extension.

The simplest case, to change the layout, is a simple case of copying the form you want to the local directory and then modified. It's handy to keep a copy of the original when you copied it so that changes in the mainline can be incorporated if needed.

So copy the file and make the changes. If you want to add more information to a form, you need to add it to the database first. This is done by modifying the tables.struct file. See the general layout of the file in the root folder of the application and simply add the required fields in a file using the same format in the local/ folder.

After this the field can also be added in the form and will be available to users.

Adding new forms

Forms and tables are linked through the name. The name of the database table is the same as the name of the form xml file.

Start by adding the table. Now create a form file under local with the same name as the table name. If you need to manually link it from some other form, copy that file from the form/ directory as when modifying a form and use one of the record linking functions to it. The foreign key field in your new table to the existing table will be automatically used and filled in when the data is edited.

Writing a Wizards

Wizard can reside in the local directory, simply call them something ending in XML and follow the wizard syntax. You can do limited processing directly, but for anything useful you probably need a php file with some functions that can be executed as needed. To insert a wizard into the right file, link it from the menu.xml file in local.

Form file syntax

To start, here's a sample of a form file.

```
<form id="cust_entity">
  <field id="cust_type" flags="hidden"/>
  <field id="cust_id" flags="RO"/>
  <field id="cust_class_no"/>
  <when field="cust_type" equals="C">
    <field id="vat_id"/>
    <field id="name" label="Name"/>
    <field id="def_payment_term"/>
  </when>
  <when field="cust_type" equals="P">
    <field id="title" label="Title"/>
    <field id="name2" label="Firstname"/>
    <field id="name" label="Lastname"/>
  </when>
</form>
```

The <form> tag encloses everything, and the id attribute must match the name of the file and the table. To display a field, the shortest version is to simply write <field id="fieldname"/>. The label will be extracted from the table definition file, or left empty if a label isn't defined. It's also possible to override the label using label="new label". Two flags exist, hidden used to enter a hidden field, and RO to simply show the content without allowing it to be edited.

The <when> tag withattributes field and equals gives a primitive option to tailor forms to the content of the row being edited. It works less well when adding new records, so if used it's handy to create a wizard for adding new records to the table. Everything between the start and end tag of <when> will only be shown if the term is true.

The last tag is <list> and here's an example of that:

```
  <list id="actual_days">
    <table id="nwd_date" using="nwd_no" link="edit"/>
    <display table="nwd_date" field="nwd_date" label="Date"/>
    <display virtual="select date_format(nwd_date,'%W in %M')" field="weekday"
  </list>
```

The id parameter, actual_days in the example, has to be unique but is currently unused apart from that condition. Two tags can be inside, <table> with and id telling what table to use, a using= that tells what field to link via in the query used to extract it and a link that tells the display what operations are allowed. link="edit,delete,add" will allow records in the table identified by id to be edited, deleted and added. <display> gives the columns to use and it has the table and field to show as well as a label. It's possible to forgo the table and field and instead use virtual that will run a subquery inside to get a single value returned in that column. Flags can be no_filter - meaning that you can't filter the list based on that column.

Wizard file syntax

The wizard xml files are similar to form files, but intended for data entry and due to that will accept more than one table at a time. There's also some process control function to allow the wizard to go to different pages depending on what you want to enter. Here's an example of a simple wizard.

```
<wizard>

  <page id="default">
    <set group="1" table="cust_entity" id="cust_type" record="cust_type"/>
```

```
    <when field="cust_type" equals="P">
      <field group="1" table="cust_entity" id="title"/>
      <field group="1" table="cust_entity" id="name2"/>
      <field group="1" table="cust_entity" id="name" label="Surname"/>
    </when>
    <when field="cust_type" equals="C">
      <field group="1" table="cust_entity" id="cust_class_no"/>
      <field group="1" table="cust_entity" id="vat_id"/>
      <field group="1" table="cust_entity" id="name"/>
      <field group="1" table="cust_entity" id="def_payment_term"/>
    </when>
    <field group="2" table="eaddress" id="eaddress_type_no"/>
    <field group="2" table="eaddress" id="eaddress_code"/>
    <field group="3" table="eaddress" id="eaddress_type_no"/>
    <field group="3" table="eaddress" id="eaddress_code"/>
    <field group="4" table="eaddress" id="eaddress_type_no"/>
    <field group="4" table="eaddress" id="eaddress_code"/>
    <field group="5" table="address" id="address_type_no"/>
    <field group="5" table="address" id="line1"/>
    <field group="5" table="address" id="line2"/>
    <field group="5" table="address" id="line3"/>
    <field group="5" table="address" id="line4"/>
    <field group="5" table="address" id="city"/>
    <field group="5" table="address" id="postcode"/>
    <field group="5" table="address" id="state_code"/>
    <field group="5" table="address" id="country_code"/>
    <button id="saveit" label="Save" action="savegroups" success="redirect_enti
  </page>

  <page id="redirect_entity">
    <text>Customer saved.</text>
    <redirect id="redirect" act="edit" table="cust_entity" group="1" key="cust_
  </page>

</wizard>
```

<wizard> is the containing tag in this case, without any attributes. field and when work the same as for forms, but field have two new attributes, group and table to indicate where to store the information. Each group will result in one record. A wizard usually consist of several pages, the first must have the id attribute set to "default".

A wizard can also have buttons, when the button is pressed the action indicated in the action attribute will be run and if it's successful the page indicated in success will be opened, if it fails the page indicated in failure will be opened instead. In the sample above, if the builtin action "savegroups" succeeds the page "redirect_entity" is activated, the <text> tag allows text to be shown directly, but in this case the redirect will mean that the user never sees the text, instead they will be redirected to the newly created record. If the save fails it will show the entry page again.

The "savegroups" action will save all groups as separate records following the order of the groups and transfer primary keys from the first to the last so that they'll be connected.

It's possible to define custom actions by adding a <action> before the pages in the file. <action id="preview" file="../twospot/preview_email.php" function="previewEmail"/> for example will let you write <output id="preview_it" action="preview"/> inside a page. This will open the "file" attribute and call the "function" attribute returning the result.

```
function previewEmail($req,$action,$pagedata) {
   return "This is a sample output action response";
}
```

An action always takes three arguments, $req is the request parameters from the browser to show the page. $action is the name of the action (to allow certain kinds of reuse), and pagedata is a structure that holds the content of the wizard data as shown in the forms. The content is empty when the wizard is a first entered, so an <output> call on the first page will not have data to work with. Using a custom action for a button will have all fields included, regardless of what page they where on.

Here's another example of a wizard, this time to show a list of records that can be edited:

```
<wizard>

  <page id="default">
    <text>Company customers</text>
    <list id="company" rows="20">
      <table id="cust_entity" filter="cust_type='C'" link="edit,add,delete"/>
      <display table="cust_entity" field="cust_id" width="5"/>
      <display table="cust_entity" field="name" width="39"/>
      <display table="cust_entity" field="country_code" width="4"/>
      <display table="cust_entity" field="vat" width="15"/>
      <display table="cust_entity" field="updated" width="12"/>
    </list>
  </page>

</wizard>
```

In this case the <table> tag also has a filter attribute to further filter the query, and we're allowing whoever is using the list to edit, delete and add new records. The "width" attribute can be used to tweak the width of each column and the "rows" attribute in the <list> tag can be used to set how many rows will be displayed at once.

Modifying behaviour

To have a bit if control over how data is saved to the database, you can use the validation or post processing classes.

To add validation or default values, create a file under local/ named validateclass_tablename.php. As with new forms, there's a template file called validateclass_template that you can use as a starting point.

The class name has to be Validator_tablename, there are four methods, checkContent is called for all types for updates and allows you to add default values and do general checks. isInsertOK, isUpdateOK, and isDeleteOk allows you to return false if you want to prevent any of those operations for any reason. $this->getOldValues() and $this->getNewValues() can be used to get the content of the record before and after it was changed by the user. $this->setValue(fieldname,new value) can be used to update the new values stored to the database. Note, it's not allowed to do any changes to the database in the validate classes. The functions here can be called several times before the record is saved and this can give duplicates in the database.

If you want to update the database, you will have to use the post processing classes. They are called postprocessclass_tablename.php and as usual there is a template file called share/postprocessclass_template.php that can be copied. Your new file has to be called local/ postprocessclass_tablename.php and the class defined in it has to be Postprocessor_tablename.

There is only one function in post processing; process. $this->getOldValues() and $this->getNewValues() are available here as well and return the same values as they do in the validation classes. The process function is called after the database is updated, if the operation was an insert and the table contained an auto_increment field the new value will be available in the array returned by the getOldValues() function. Since this is called after the insert/update/delete is done, it's to late to do anything about that, it's very handy to be able to update cached values in other tables, make summaries, or do other things that should be done to keep the database on track. Anything done here could also be done in triggers in the database.

menu.xml - adding new menu items

One of the main adaptation files is menu.xml, her you can modify the menu and also add completely new menus called in from somewhere. If you want to have a menu associated with a form, name the menu "form_tablename" where the form is called "tablename.frm" and the table it uses is called "tablename".

Four different kind of files can be added to a menu, forms, wizards, html files and php files. PHP files will get data from the menu in the $_REQUEST variable, some values are added there before it's called by the underlying system.

Forms are defined as above and should be linked by name only, the system will search through local versions of the form first and only if none is found load a default template. If there is no template available, one will be generated using available metadata. For simple records it's enough to add labels to the database structure to have the form generated automatically.

Wizards are added the same way, simple link the wizard file as file link to have it included.

PHP and HTML files can be added directly, if it's added to a menu of a form record the information to recover the record is available in the $_REQUEST variable. Clicking links and submitting forms will reload the same page again, if you wish to link outside the page use the class="download" on the link to avoid it being rewritten. Normally the text in the html and php is returned wrapped in a menu, but's also possible to override that and generate a dynamic menu from a php file, if you decide to make us of this the format is explained below.

Menu JSON format

This section describes the menu format sent from the server to the browser client.

```
{ menu:
[
{ name: 'item 1 in menu', link: 'url 1 to load' },
{ name: 'item 2 in menu', link: 'url 2 to load' },
{ name: 'item n in menu', link: 'url n to load' }
],
title: 'Title above menu' ,
page: 'filename of loaded file' ,
pagedata: 'htmlcode to insert' ,
id: 'id in menu of this choice' }
```

As can be seen it's a pretty simple format, the url in item originating from the menu.xml has a special format, but it's possible to put anything you want in there. If you want to call another section of the menu the format has to follow what is used elsewhere however.

To return a structure like this from PHP code to make a dynamic part of the menu, prefix the structure with {%menuoverride%} followed directly by a structure like the one above.

Useful functions

This contains a list of useful functions, there are of course lots more, but these are the main ones that has to be used.

The functions are listed with other functions defined in the same file.

metadata.php

This is a low level file, and all it's functions are usually wrapped in one of the files below. There are cases when it's useful to access this directly though.

When this is included, and it always is unless you're writing a top level file, you will have a class instance in $metaInformation. The class definition is shown below.

```
 MetaInformation {

  findMiddleTable($sourceTable,$targetTable);

  getPrimKeyWhere($table,$data);

  getFieldType($table,$field);

  getReferenceTable($table,$field);

  getReferencedTables($table);

  getReferesToTable($table);

  quoteValue($table,$field,$value);

  getColumnArray($table);

  getDisplayArray($table,$field);

  getExtraArray($table,$extra);

  getPrimaryKey($tablename);

}
```

findMiddleTable will find, if any, the table connecting the source and target tables.

getPrimKeyWhere will return a column=value string where column is the primary key of the table and value is taken from the $data array.

getFieldType will return the type of the field.

getReferenceTable will return the name of the table that the field references.

getReferencedTables will return an array of all tables that are referenced from this table.

getReferersToTable will return an array of all tables that are refers to this table.

quoteValue will return a string with correct quoting depend on the type of the field.

getColumnArray will return an array of names and types for the table.

getDisplayArray will return the display properties of the table.field combination.

getExtraArray will return extra information stored.

getPrimaryKey will return the name of the primary key for the table.

All these are there to help write code that will work on any data, regardless of future changes to the structure.

2lib/selecthelp.php

Here's all functions useful for getting data from the database, either to handle or to simply display to the user.

$row=selectOne($SQL) will return the first record returned by the $SQL statement.

$value=selectOneValue($SQL,$columnname) will return the content of one column in the returned array.

2lib/template.php

This contains two functions and a class to handle templates. templateRow takes two arguments, an array with values and the template text itself. templateQuery also takes two arguments, an SQL statement and the template text. templateQuery will run the query and call templateRow for each resulting row returned.

templateRow($row,$template)

This can be used for simple templates where you only want to do a simple page. See the Template system section to see what can be included in templates.

Example 10.1. templateRow example

templateRow(array("name"=>"Olof Tjerngren","email"=>"olof@two-spot.org"),"Name: %name %
Email: %email%
"); will return a string like this:

Name: Olof Tjerngren

Email: olof@two-spot.org

templateQuery($query,$template)

This is very similar to templateRow, but takes a query as the first argument, if there is only one match it will work similar to templateRow, if more than one row matches it will repeat the template for each row returned by the query. See the Template system section to see what codes can be used in a templates. Templates can be changed both by the system administrator and used directly in code.

Example 10.2. templateQuery example

templateQuery("select * from cust_entity","Primary key: %cust_no% Name: %name%
"); If there are two rows in the cust_entity tables with known content this will be returned:

Primary key: 1 Name: Test 1

Primary key: 2 Name: Test 2

share/updatehelp.php

Update and insert functions are defined here.

list($ok,$text)=doUpdate($table,$row) will execute an update statement on $table with data stored in the $row array. $ok will be true if the update went OK and $text will contain a textual description of what happed, good or bad.

list($ok,$text,$keyvalue)=doInsert($table,$row) will execute an insert statement on $table with data stored in the $row array. $ok will be true if it went OK and $text will contain a text that says what happened. $keyvalue will return the generated primary key if it was set as auto increment or 0 if it isn't.

So a simple update of a field can be this:

$row=selectOne("select * from cust_entity where cust_no=1");

$row["country_code"]="NOR";

list($ok,$text)=doUpdate("cust_entity",$row);

echo $text;

That will fetch the customer with cust_no = 1, set the country_code to NOR and save it back again. Silly example...

share/deletehelp.php

Delete is taken care of separately, it will require a special require command and is named differently.

list($ok,$message)=deleteRec($table,$row) will issue a delete statement to remove the record. If a single value is used for primary key it is sufficient to have that value in the $row array. If it's a multi column primary key all of them have to be supplied. $message can be shown to the user, it will contain a message from the delete operation.

share/formhelp.php

Here are some useful form functions that can come in handy, addRecordLink, editRecordLink and deleteRecordLink will all return a <a> link text that can be used for the obvious reason. They will all open a window that shows the data and ask for confirmation before updating, adding or deleting.

share/fieldhandlersclass.php

Every function used in forms/ is defined here. The basic function to use in forms is inputLayout($row, $tablename,$fieldname). Used in the form <?=inputLayout($row,$tablename,$fieldname)?> in a form file it will show the field with current value if editing in a useful to edit format. The partner function inputLayoutRO takes the same argument but will always show the value as read only, otherwise the inputLayout will show the appropriate form depending on if you're adding a new value, editing an existing value or confirming a delete.

share/graphlib.php

This file holds a low level class to generate a flot graph. It will create a string with html and a script tag that generate a graph based on a sql query.

```
$tst=new Flot("select if(country_code='','Unknown',country_code) country_code,
               "count(*) cnt from cust_entity where cust_type='c' group by count
$tst->setColumns("country_code","cnt");
$tst->setSize(400,200);
$tst->setBar();
echo "<h1>Customers per country</h1>".$tst->getFlotString();
```

The sample above shows how to create a bar graph with country code on the X axis and number of customers in each on the Y axis.

Other possible adaptations

Some files will check if a corresponding file exists in local and then load that overriding the class defined in standard. This allows you to fine tune some behaviour or add more actions. We'll add a list here in the manual with everything that can be overwritten in that way.

validateclass... And postprocessclass...

You can add validation and post processing in these two kinds of classes, to do so create a file called validateclass_tablename.php and/or postprocessclass_tablename.php.share/fieldhandlersShowList.phphp

Starting with validateclass, these exists to give you a chance to check the data before it's inserted into the database. This is done through a class inside the file called LocalValidator_tablename that inherits from either Validator or an existing class for that table.

You have access to two function, $this->getNewValues() and $this->getOldValues() to have something to work with. To update an new value you call the function $this->setValue(column,newvalue) If you want to send a message to whoever is working on your table you can use $this->setMessage(text) to have that text shown. This is used when you refuse an update to say why it was refused.

You can add the functions checkContent() that should return 1 if it's OK and 0 (zero) if you don't approve the action to the table. checkContent is mostly useful to modify the data according to some rules before it's later saved.

The other functions you can add, isUpdateOk(), isDeleteOk() and isInsertOk() can be used to prevent a certain action to take place. There is one very important rule when writing validation code, you must not issue and statements to the database that changes it's content. The validation code can be called without there being any intention of doing a later update, if you change the database that change will remain even if the change you checked doesn't go through. To alter the database, you should instead use the post processing.

postprocessclass_tablename.php is loaded and used after an update, delete or insert takes place. This is similar to how triggers work in the database. Once you have created the file, write the class Postprocessor_tablename extends Postprocessor in it. You can of course also inherit from an existing postprocessor class if you want to keep the default behaviour and just add to it.share/fieldhandlersShowList.phphp

The function you create here is process($type), where $type can be insert, update, or delete. In this class you also have access to the getNewValues() and getOldValues() methods to have something to work with. If the table you're working with has an auto_increment primary key that value will be set if $type is insert. After the insert you have free reign on changing the database. It's the whole purpose of this class. Keep in mind that if you doUpdate on the same table you can get an infinite recursion going on. Beware of that.

heartbeat_ and info_

In the heartbeat/ directory are a few files called info_<name>.php these should contain a class info_<name> with a function called getContent that returns a html string that will fit into the banner space in TwoSpot.

The backend server will look for files named info_ and randomly execute this to serve interesting information to the user of the system.

General idea of adaptability

The simplest way to make something adaptable is to write the general code in a base class and call is MySomethingBase, for example. Check if a local adaptation is available in a specified filename in the local directory, if it is just include that file. If it isn't, create a class MySomething extends MySomethingBase and is empty.

The base code would look like this:

```
class MySomethingStd {
  // filled with functions.... Of course.
}

if(file_exists("../local/something.php")) {
  include_once("../local/something.php");
} else {
  class MySomething extends MySomethingStd {
  }
}
```

The local something.php would of course also contain the same definition as in the else but would not be empty.

This works best with small functions in the class that has single and well defined functions, in some cases it might even be worthwhile to split a function into several sub functions to make it more flexible for adaptions. In the overriding class "MySomething" you can call the parent class at any time to get standard behaviour run. Note that you should always run the standard code, it's extremly rare that you'll need to completely override a behaviour. If you think you do, follow the code path first to make sure nothing of importance is skipped.

tables.struct

This file defines the database structure used. The format is not complicated. You add a field or to this file, go to the admin part of Spot and run upgrade DB. That will compare the structure defined in the file and issue any required changes to the database with alter table or create table statements. You can also remove fields from this file and they will be removed from the database. Removing whole tables will not have them removed from the database however.

Currently there's only one tables.struct file, but it's in the process of being rewritten so it can be split into several parts to keep adaptions separate from the core tables. When it's split into several files it will be possible to add more fields to a table defined in the core, but not to delete tables or fields from the core.

The tables are defined this way <table id="tablename"> followed by the definition of fields and some other metadata. To close </table>. Fields are entered as a single <field ... /> entry. Primary keys are defined with <primarykey field="somefield"/> and other metadata not related to the database are either <display ...> that contains info on how it should be displayed on screen,<extra .../> contains additional information on how to handle the data. If the primary key consist of serveral items, they are separated by comma.

Here's an example:

```
<table id="sometable">
  <field id="somekey_no" type="int" notnull="yes" extra="auto_increment"/>
  <field id="descript" type="varchar" len="60" notnull="yes"/>
  <field id="cust_no" type="int" notnull="yes" references="cust_entity"/>
  <field id="rawdata" type="longblob" notnull="yes"/>
  <field id="notefield" type="text" notnull="yes"/>
  <primarykey field="somekey_no"/>
  <index id="ix1" field="cust_no">
  <display id="table" name="Some Table" width="400" height="400"/>
  <extra id="descript" view="%descript%"/>
  <extra id="type" value="Data" group="Data"/>
</table>
```

Display can have id=table where the name and the width/height controls the name and the size of the window to display it, or id can be one of the fields and then gives some hints on how to display

that field. For fields it's possible on text fields to set rows="" and cols="" to set the size in characters how large the textarea should be. For all fields you can set the name="" field to give a better name for the field.

extra id="descript" view="" is there to allow a template to be entered that will be used to present the information stored in the record in lists and links. extra id="type" can have value="Data" | "Definition" | "Configuration". group can be either "Data" or "Config". This is used in some cases to limit searches for links to only include related information in the same group or set as the table you start from.

On fields, references is used to tell if a field references another table and what fields are used for that link. It comes in two forms, references="cust_entity" will assume that the fieldname that links are the same in both tables. If it's not it's possible to write references="cust_entity.cust_no" to make sure it links via the correct record. It's never wrong to use the dot notation.

The above example contains the minimum required for a table to work well for all uses, so x number of <field/>, one <primarykey/>, at least one <display/> and two <extra/>, one with descript and view and one with type and what group the table belongs to.

For the table grouping, the difference between definition and configuration is a bit gray, but in general definition is only used for lookup tables that doesn't affect the flow of the program. If it contains information actually used by the software and not just displayed it should be Configuration instead. The theory is that group Config information could be used to seed a new installation, while data shouldn't be moved.

tablename.frm

Here's an example file for the address table.

```
<form id="address">
  <field id="address_type_no"/>
  <field id="line1"/>
  <field id="line2"/>
  <field id="line3"/>
  <field id="line4"/>
  <field id="state_code"/>
  <field id="postcode"/>
  <field id="city"/>
  <field id="country_code"/>
</form>
```

As you can see it only controls what to include and the order on the simplest level, it's possible to add a label="Label" if the description defined in the table data isn't enough. It's also possible to add lists and a few other things.

<querytemplate>

The attributes for this are LABEL, TEMPLATE and SQL. Label is simply the label to show before the output and SQL is the query to run. It's possible to enter the template as an argument, but also to put it as CDATA content inside the tag. This will simply run the query and fill in the template with the data required.

<list>

List defines linked lists, it works the same at it does for wizards, with the exception that the first <table> entry should be used to link the data to the current record, USING="fieldname" can be used to tell the system what field should be used to link them together.

<when>

A when section controls when to show the content inside it. Two attributes can be used, field and equals, it looks like this: <when field="fld1" equals="ex"> ... contained layout </when>.

It can be used to limit some fields to only show when a field in the data contains a specific value.

<field>

The attributes for field are ID - the field name, LABEL - if the standard label from tables.struct should be overridden in some cases, FLAGS that can be RO for Read Only. Field will display as an editable field if the form is used to add or edit data, and as plain text if it's a readonly view. The actual display will depend on the data type used in the database, for example will ENUM fields in the database give you check boxes, SET will show a drop down, different type of dates shows a calendar lookup. If a field referes to another table, the rows in that table will show in a drop-down list.

<form>

<form> is the containing tag, the only argument is ID that has the name of the table that the form edits.

<llist>

This is similar to <list> but can only be used with a single related table. On the plus side it's editable directly by the user in the interface. While list is generic in it's usage, llist is tailored to adding linked data to head records. It can be used for adding, deleting and modifying. <llist> takes a few attributes directly that is included inside the list tag.

<llist> attributes

id - the ID of the list.

table - what directly linked table should be used.

label - what label to put in front of the list.

Inside the llist use <display> similar to <list> but with only the field and label attributes where label is optional and the header will default to the name given in the structure file.

wizard.xml

Wizard files can be used for multi step data entry and to collect data before an action is required, here's a sample.

```
<wizard>
  <action id="preview_planning" file="../review/reviewlib.php" function="mgrLis
  <action id="preview_self_review" file="../review/reviewlib.php" function="mgr
  <action id="start_self_review" file="../review/reviewlib.php" function="mgrSt
  <page id="default">
    <text>
      Dear Manager, you will launch the Self-review phase for your
      team using the below function.
    </text>

    <text>
      Below is the list of your team members who will get the Self-review e-mai
    </text>
    <output id="list_avail" action="preview_planning"/>
    <field id="send_email" type="checkbox" label="Send email"/>
    <field id="email_text" type="textarea" label="Email content"
```

```
            size="80" height="8">Dear employee,

This e-mail is the official launch of our Focal Review process this year.

Your Manager
</field>
    <button id="start" label="Start self review" success="done" failure="failed
  </page>

  <page id="done">
    <text>These people who report to you can now do their self review.</text>
    <output id="list_done" action="preview_self_review"/>
  </page>

  <page id="failed">
    <text>
      The starting of the review failed!
    </text>
  </page>
</wizard>
```

action defines an action that can be done later, with filename and function name it maps to. The function will always be called with three arguments that describes the content in the wizard when called.

The default page is the starting page, calling the preview_planning will list people in a certain stage, and simply inserting the output in that section. Pressing the button will call another action that will return success or failure that maps to two other pages that will be displayed. Neither of those pages has any further information available, so after it hits done or failed, the form is done.

<row>

This can be used to show fields for a record and allow the user to edit them. It will show the fields that are available

<row> attributes

id - the ID of the row.

table - the table the row data comes from and should be added or updated to.

keystorage - what page that keys should be stored in.

method - can be update, view or insert. Update for editable values to update, view for a read only view and insert to add new data to the field.

hidefields - List of comma separated fields that should be hidden.

requiredfields - fields that are required.

values - default values for fields, use field=value,field2=value2 syntax.

Chapter 11. Template system

The template system is used both inline in the code using the templateRow() and templateQuery() functions as well as configurable by the system administrator for generating emails and documents. The basic template system is located in share/template.php, the UserTemplate class for emails and pdf template is located in share/templateclass.php. UserTemplate uses the base templating functions.

Simple Substitution

Simple substitution can be made by placing % signs around the name you want, %name% will be replace by the content of "name" in the array sent. If you want to put a text before or after (or both) only if there is a value to be shown you can use this form %(Name:)name(
)% for example. Using that nothing will be shown if name is empty and "Name: Test 1
" will be returned if there is something to be shown. %name(
)% and %(Name:)name% also work as expected.

Advanced substitution

For more complex substitutions the %{<cmd>(arg1,arg2,...):<options>}% can be used. <cmd> is one of the commands listed below with the argument that command takes and the optional :<options> can be used to apply some text processing to the string before it's inserted into the text being transformed.

Substitution commands

Commands can be run to get more advanced substitutions, all commands take arguments. When most commands are run the arguments are run though the template engine to replace any substitutions that might be needed later. For example to show a customer record and all addresses that belong to that customer you can have a template for the customer that looks like this.

Example 11.1. Customer template example

Name %name%
 %{query("select line1,city from address where cust_no=%cust_no%"," %line1% %city%
")}%

In this example the %cust_no% in the SQL argument of the query template will be replaced with the cust_no primary key from the record being shown before the query is run. The line1 and city used in the template will come from records in the query. Only columns returned by the query is available inside the second template. If you try to add %cust_no% to the inner template an error will result.

It's possible to put any template code inside an argument that is expanded before run, but since the argument runs to the end of the string expansions inside an argument must have it's argument string characters escaped.

query(sql,template)

Execute the SQL query and expand the template for each row. SQL is expanded with the current row data but template is left alone.

editLink(tablename,pkvalue,linktext)

Generate a link to edit a record in tablename. pkvalue is the primary key for the record to edit, linktext will be shown in the link. If pkvalue is left empty it's extracted from values in the current row.

deleteLink(tablename,pkvalue,linktext)

Generate a link to delete a record in tablename. pkvalue is the primary key for the record to delete, linktext will be shown to the user. If pkvalue is left empty it will be retrieved from the current row.

addLink(tablename,basetable,linktext)

Generate a link to add a new record, if basetable is set it should be the name of a table with values in current row, the primary key will be transported to the new record so it's linked to basetable.

editField(table,field)

Insert an edit widget for table.field. If there is a value in the current row for this it will be inserted as default, otherwise it will be blank.

translate(tablename,fieldname,key,language_no)

Translate the row with the primary key "key" in "tablename"."fieldname" to language_no.

describe(tablename,pkvalue)

Generate a one line description for the current record of tablename if no pkvalue is set, or use pkvalue to get a record otherwise. The description is stored in the tables.struct file.

address(address_no,cust_no)

Display a formatted address based on the country of the address. cust_no is optional and if exists will add an "Attn: name" line after the company name of the address. The output of this is formated to html, if you need to use it as plain text add the "text" option after this command.

Options

There is currently only one option, "text" that will convert html code inside the expanded text into pure text.
 will be converted to new line and everything else will simply be removed.

Chapter 12. Expressions

Some configuration sections can use expressions to help the system pick the right action when there are several options available. The expressions themselves are pretty standard fare.

Expression can be comparisons, == equal, > greater than an so on. It can be combined with AND or OR. String, numbers or the content of fields in the current record. There's also a special function called follow that will follow a database link to a new record and updates the current record with a new one.

Variables

The current record to be examined is available as variables. Assuming your currently doing something with a cust_entity record, you can write (name2=='Steve') to see if the first name is Steve for example.

Comparisons

All comparisons will return a boolean true or false.

== - equals, name='Smith' for example, or 1==2.

!= - not equal., name!='Smith', or 2!=1.

> - Greater than.

< - Lesser than.

>= - Greater or equal to.

<= - Less or equal to.

in - name in ('Smith','Jones') or cust_id in (1,2,3,4,5,'122') for example.

Boolean

or - 1==2 or 3==3 will be true for example, only one of the two expressions have to true.

and - 1==2 and 3==3 will be false, they both have to be true for this to be true as a combination.

Functions

follow('fieldname'). This will take the value of the field and assuming the field is a link to another table will follow it, load the other record and update the current record variables to those of the new records. Follow will return true if the operation was successful and false if it wasn't. So if it was not able to follow the link or you misspelled the fieldname it will be false. For that it's recommended that it's used like this: follow('cust_no') and country_code='SWE'. This will try to follow a link to the cust_entity record and from that record see if the cust_entity country_code field is SWE. If there is a problem following the link to the entity the full expression will fail.

Math and combinations

Some math is also available, 1+2 will yield 3 for example so (1 + 2 == 3) will give true. ((1+3)*2) will give 8 and so on.

Chapter 13. Email and PDF template

Under System - Configuration is Manage templates. A template can be for email, pdf files or both. Email and PDF templates are used automatically in some cases and it's possible to enter expression rules that let's the system pick the right one. They can also be used manually to send emails and pick a template text dynamically or to generate a PDF file dynamically that can be downloaded directly.

Base template

The base of a template has a name, a required key field, a base table and a query used to collect the data used for the template text.

The required key field is used by Two-Spot to know when the template can be used, if the field is available on a record the template is viable. The query is run to collect a single row to apply to the template text. It can always rely on the required key field to be available for substitution. LInked to the base template data is a list of expressions, this can be used to simplify template writing and used to replace complex expressions that collect data under a simpler name. This is especially useful if templates should be edited by translators.

In addition there's a list of default TO addresses for the emails, a list of email templates and a list of pdf templates. The email and pdf templates contain the actual text with template replacement. It's possible to have several of each with different languages, so it's intended for translations. However, it's possible to have several variations for other purposes than translations if needed.

Email template

Each email template has a language, a character set used for the actual sending of the email. The body text of the email can use the template definition system described in the "Template system" chapter above. Then there's the standard email fields of From, Subject, CC, and BCC. These can also use template codes to be replaced based on the data, useful mostly for Subject in most cases.

The "Select this" expression and the Default flag are used to automatically pick the right template when the system needs to pick a template on it's own. If there is a default template it should have both "Select this expression" set to "1==1" and the Default flag set. The system will pick a non-default template if there is more than one match, so the default flag is there to make sure it's not selected and the 1==1 expression there to make sure it is selected. For other languages you can pick the template based on country code, or other criteria if available. You can also let the customer or sales person select the language directly and in that case a default flag should not be required at all.

PDF template

The pdf template are similar in setup to emails, but you have fewer fields to fill in, only Language and Template text is used for this. For PDF files it's entered using an html editor (tinyMCE) but since it's template code it's not really WYSIWYG - pretty close however, depending on what y ou enter it as. In some cases, like if you want to have a table list and loop over a unknown set of rows use the html source editor to add the %{query()}% around the template row data. Other things can be entered directly in the template row.

"Select this expression" and the default flag works exactly the same as it does for email templates.

Chapter 14. Installation

To install you will need a LAMP server running. You will need access to the file system and root access to the MySQL database. In theory WAMP should work, but it hasn't been tested. LAMP stands for Linux/Apache/MySQL/PHP and WAMP is the same but Windows instead. PHP will have to be configured to work with both MySQL and Apache.

Note

If you want to use all functions you cannot use safe mode in PHP. Affected functions are file uploads and PDF generation. You have to set open_ basedir so it can access the /tmp/ directory for both.

Putting the files in place

Once you have the files downloaded, copy them to the location you have chosen that matches your apache configuration. Run the fixFilePriv.sh script to set file permissions on a few files that will be updated by the install process. You can place the files in a virtual host section that specifies a new root when used from a browser, or in a directory in an already configured apache section. The system used relative paths so it can be placed in any relative location.

Create a database to use by starting a MySQL client and entering "create database twospot" or any name if twospot isn't suitable. Then create a user that has full access to that database, "grant all on twospot.* to twospot@localhost identified by 'secretpassword'". Please choose a decent password for this user.

Apache configuration

There needs to be an apache configuration section in some form, below is an example using a virtual host. The proxy setup is required for Orbited to work properly, see the chapter on Orbited for more information. Note that for the Proxy configuration to work, both the Proxy and the Proxy_html submodules must also be activated in Apache. In Debian systems a2enmod Proxy and a2enmod Proxy_html should do the trick.

Example 14.1. Virtual host example

```
<VirtualHost *:80>
   DocumentRoot /var/lib/2spot/
   ServerName two-spot
   ProxyRequests on
   ProxyPreserveHost Off
   ProxyPass /tcp http://localhost:8000/tcp
   <Proxy *>
     Order deny,allow
     Allow from all
   </Proxy>
</VirtualHost>
```

Orbited and TwoSpot backend server.

Note

This is not required for simple usage, but to take full advantage of the system it is recommended.

Orbited is used for heartbeats, banners and messages that needs to be sent without interaction directly from the user. If not included with the operating system it can be downloaded from http://orbited.org. The default configuration should be sufficient, but note that TwoSpot uses port 8000 for normal communication and required the Stomp protocol to be available on port 61613, so the following setup has to exists.

[listen]
http://:8000
stomp://:61613

[access]
* -> localhost:8000
* -> localhost:61613

The stomp:// line will activate the built-in Morbidq module that handles the Stomp protocol. It should also be possible to run rabbitmq as a message server instead of morbidq however.

Once the configuration is done, start orbited by running it as a service according to the instructions on orbited.org.

To have any use of Orbited, a TwoSpot server also has to be run, There should be a script for that and better instructions, but for now simply go to the heartbeat directory and write "php backend.php" to start it up.

Running the installation script

Open a browser at the address configured in Apache, for example http://localhost/app/ This will start the setup code that will handle most of the initial configuration. You will be prompted for a login and password and where the database is located as well as an administrator login to create for the system. When run the setup script will look for missing software on the server that is used, some can be ignored if you don't intend to use every feature. Once done you click the link to go to where the application is located and login with the administrator login entered earlier. Now the real work begins with the setup, read the chapter on Configuration to see how TwoSpot can be configured to your needs.

Configuration files

There are two configuration files used that control how the database is accessed and used, and how you can login to the system.

local/conndb.inc.php

This file has the database configuration stored. A sample can look like this.

```
$mysql_server="localhost";
$mysql_user="exampleuser";
$mysql_password="examplepass";
$mysql_database="customerdb";

$server_id=1;
$server_min_no=1;
$server_max_no=9999999;
```

The mysql_ variables are set so it can connect to the database. How much the user and password is used depends on the login configuration, it can range from accessing the user records and nothing more to having access to the full database. The server settings are used for inserts in the database, all generated keys will be between the min and the max values. The current value is stored in the aseqno table to keep track of what's being inserted. The id must correspond to the MySQL database server

ID. This is to allow several servers to be running in distant locations with replication between them to keep it all in sync.

local/settingsGlobal.php

This file has some base configuration to do with how users are authenticated.

```
//$loginSystem="dbAuth";
$loginSystem="internal";
//$loginSystem="fixedAuth";
$module_setup=array("auth"=>
                    array("ldap"=>array(0=>array("description"=>"Local LDAP",
                                                 "hostspec"=>"192.168.0.1",
                                                 "basedn"=>"dc=home,dc=example,
                                                 "uid"=>"uid"),
                                        1=>array("description"=>"Enterprise LDA
                                                 "hostspec"=>"ldap.example.com"
                                                 "basedn"=>"dc=example,dc=com",
                                                 "uid"=>"uid")),
                          "imap"=>array("description"=>"Local Imap",
                                        "folder"=>"INBOX",
                                        "hostspec"=>"192.168.0.1",
                                        "port"=>143,
                                        "protocol"=>"imap/notls/novalidate-cert
                          "spotsql"=>array("description"=>"Local DB",
                                        "phptype"=>"mysql",
                                        "encryption"=>"md5-hex",
                                        "table"=>"user",
                                        "password_field"=>"md5pwd",
                                        "username_field"=>"username",
                                        "database"=>$mysql_database,
                                        "username"=>$mysql_user,
                                        "password"=>$mysql_password)
                          )
                    );
```

There are three methods to connect to the database. "dbAuth" uses the login/password to connect directly to the database. That way the access can be set for each user to exactly what is needed. "internal" uses the login/password supplied in the conndb.inc.php file to get a connection to the database. "fixedAuth" connects using the login of the user, but a password stored in the user table. "internal" and "fixedAuth" is useful if you're using a secondary system to validate users.

The module_setup variable holds the different methods of authentication that will be used in sequence. If the type holds an array, the same module will be used several times with different configuration.

Setting up the database

The database is generated by the installation script, to update the database login to the application and starting at the Top menu level go to "System", then "Database" and select "Preview Upgrade" to see if there are any changes to the physical structure of the database and "Upgrade" to actually run the upgrade.

Checking the installation

You should now login as the administrator account and check that login works and that the configuration looks OK. If you didn't choose any of the predefined installation options above, read the next chapter to see how to configure Spot.

Troubleshooting

This chapter will contain common pitfalls as they occur.

Chapter 15. Configuration

This chapter is fairly long, since it contains all information needed to configure two-spot. There are many items that will need to be configured, of course depending on what you intend to use Spot for. For a smooth running operation though, it will take a bit of work to set it up properly.

General configuration

Configuration is done through the System application. Here you can do general maintenance covering almost every aspect on how the system works.

Employees and access

Everything for the employee setup is under Configuration and Employee Setup. You need to use most of these, but start with the groups you want to have and after that the access control you want to put in place for those rules. It's convenient to start with an Employee group and below that put more specialized groups. When you enter you start with adding the Employee group, and then the next one under that, for example Administration, setting the parent group to Employee. You can have as many levels of groups as you want and it's used for access so it's worth spending some time to think about it.

Then go to Access control list and add a rule, begin with adding a rule for the top level Employee group and set the Access to Allow, pick anything that you want everyone in the system to be able to use. Employees and Employees logged in are relatively safe in most environments for an example of what everyone can see.

Then save and add another rule, picking one of your real groups. Let's say we pick Administration and give those access to see Customers and all related items under that. Keep going for all rules.

Now add the employees, set the appropriate group you want. If you set the group to Administration the added employee will now have access to both the currently logged in persons as well as customer data. The Administration group inherits the access rights from the Employee group.

Having a well defined group structure makes it easier to handle access well, it's possible to change the structure of the groups, for example adding another layer between Administration and Employee without changing the access given, moving Administration to another place by changing it's parent might give the people in the Administration group more or less access.

Note

Access is granted from the top and down of the group tree, an Allow to see the currently logged in users in Employee can be reverted by having a Deny on a lower level group, and it can also be give back again at an even lower level.

It's possible for an employee to belong to more than one group, but if so take care not to get conflicting access rules. If there are two conflicting rules at the same level for two different groups either rule could be used, there's no way to know what rule will take effect.

Calendar information

Here you can set calendar configuration, specifically right now NWDs or non working days.

You start by adding some general information. The name is required, then an optional description of the holiday. You can also add a link to more information.

Type determines how the date is set for each year. It can be one of the following.

- Day

A weekday, basically for entering Saturday and Sunday or equivalent. For the definition enter the english name of the weeday.

- Month-Day

  A fixed calendar date every year. The definition is the month number with two digits, a dash to separate and a the day number. For January 5, enter 05-01 for example.

- Manual

  A manual holiday is different each year, it will not be set automatically and you'll have to set it manually each year. There is no definition for it.

- No-Weekday-Month

  The X weekday in a month. The second Monday in May would be entered as 2-Monday-05 for example.

- Weekday-Month-Day

  The first weekday after the set date every year. For Swedish Midsummer day it's the Saturday on or after the 19 of June for example, the definition is thus Saturday-06-19.

- LastWeekday-Month

  This is for holidays occuring on the last weekday of a month. The last monday in May is entered with a defintion of Monday-05 for example.

- EasterOffset

  EasterOffset is defined as number of days before or after Easter. With a definition of 0 it means Easter and with a definition of 1 it's Easter Monday.

- OrthodoxEasterOffset

  This works exactly the same as EasterOffice, but uses the Orthodox Easter calculation instead.

- Seasontype-Offset

  This is for holidays occuring on or near an equinox or solstice. For the definition enter Spring, Summer, Fall or Winter for the type and number of days for the offset.

You can set a flag to not show this holiday in the calendar. This is useful for the weekend Saturday and Sunday for example.

Hours not working should be 1 for a full day and .5 for half a day.

The country list is what countries have this holiday on the same day.

The last item is a list of dates this holiday occurs on. It's for reference, but also to add manual days or be able to modify the dates if the automatic assignment went wrong for a specific year. If you change an automatic date you should check the "Fixed" flag so that it's not switched back when updating the holidays.

Once you have your NWDs defined you can add them to countries, the "Days off per country" will show a list of all countries with a count of number of holidays defined for that country and the number of employees in that country. You can use it to check that you have all holidays defined for each country where there are employees. If you open the country you'll get a list of holidays that apply to that country. You can also list all assigned dates by clicking on list coutry dates where you can tweak them if needed.

"Update NWD this year and next" is there to generate the automatic holidays for this and next year. It should be run once per year to have next year defined as well.

Customer and role configuration

You define what customer types to have. Standard ones include Company, Person, and Organisation. This is done by editing customer types.

After those are set, you need to define how the types you've set should relate to each other. This is done by entering customer role types. You define a role for both ends, entering a description and an down description. Paired with those are what customer type that description belongs to.

Common role connections are Employee - Company, Owner - Company, Head office - Branch office, and Child - Parent.

Legal Entity

Here you define your own company or companies if you work at a bigger company that consists of several legal entities. The Legal entities are used in several other places to configure different behaviour depend on the legal entity that should be used.

Note

TwoSpot is built to be used as a single system looking similar to everyone using it. Rules make sure orders for example end up on the right legal entity so it's possible to get reports separate for the entities even though you don't normally have to manually enter the legal entity.

On the legal entity you also define the sales tax rules that should apply.

Sales tax rules

Sales tax is set per legal entity, so first open the legal entity that should be updated. Each legal entity has a list of tax rules that apply. Edit or add a new rule and select the Country and possibly state that it should cover. Not select state simply means that the rule applies to any state.

Next add the tax rates that should be used, it's entered in a combination of rate, product group and product. Product group and product are both optional and if left out indicates that the rate should apply for all groups and individual products. If there isn't a rule that matches the sale, no sales tax will be applied.

For a company in Sweden the most common tax rate is 25% so if no products are sold at any other rates a simple 25 followed by no group or product is all that has to be entered. If any products use a lower rate of 12% or 6% another rule can be entered with the product group that it applies to. The system will use the more specific rules in preference to less specific rules so a rule that matches a product will be used ahead of the broader rule that covers every product sold.

For sales inside EU each country sold to should be added as 25% (or a lower rate) with the VatIDExempt flag set so that a valid VAT ID can be used. For countries outside the EU companies in Sweden should not add sales tax, so nothing needs to be entered for those since the default is to not add sales tax.

Products

Products have three levels, Product groups, Products and Pricepoints. Starting with adding the groups that the products should belong to, other configuration is simple if you have well defined groups, but even with the two extremes of a single group that holds all products or having one group per product it's possible to configure the system as desired, it's simply the two cases where the configuration and maintanence will take most time.

The second step is to add pricepoints. Pricepoints are simply a name and a collection of prices with dates and currencies. The price within the current date range will automatically be the active one, but

there should only be one currency with a price active at a time. The dates are there to allow price changes to be entered ahead of time and be activated on the right date automatically.

Once the pricepoints are entered the products can be added. Products belong to a product group. They have a name that identifies and is printed in pricelists and on invoices. There is a status that can be nothing or a combination of Online and Active. They also have a pricepoint as defined earlier. Finally there's an activation date and a deactivation date. It's possible to set the deactivation date ahead of time instead of marking the product as deactivated on that date.

There is only one list of products and prices in Two-Spot. There will be other ways to modify the price based on location, customer, timed campaigns, discount codes or legal entity.

Heartbeats

Heartbeat configuration can be found under System - Configuration - Heartbeats.

Heartsbeats are batch functions that can be run every hour, day, week, month, or year. The record simply describes how often it should run and what file and function should be run. When run the heartbeat system will track the last result recieved from the function, and each function also has the option to cancel a heartbeat if a serious error has occured. It tracks how many times it was run, when it was last run and what time will run next. The next run time is usually the same minute every hour, or hour every day and so on, but it can drift if it takes longer than a second to run the heartbeat. This can be used for just about anything.

Heartbeats are run from the backend process so that has to be setup and be active for heartbeats to function.

There are some heartbeats included with the TwoSpot distribution, but as the layout suggests it's primarily intended to be used for specific needs of each installation.

The included heartbeats cover currency updates, timers, some caching, checks, and clean up. Not all are enabled by default.

- Remove inactive users

  This will run every hour and cleanup after users who do not logout but turn of the browser instead.

- Timers refresh

  This runs timers, checks for new timers to be started and old timers to be stopped or moved to the next stage. Timers are excellent to handle reminders for example.

- Update cached country code

  This short function will simply check if the cached country code on a company entry needs to be updated due to address changes.

- Update preferred contact

  This will check for missing preferred main / invoice physical address, missing phone/email preference and missing currency preference on customer and fill it in. It will use recent orders and take values from there if a preference hasn't been added manually before. Values can of course be changed manually after the automatic setting as well.

- Update daily currency rate

  This should be run once per day and will fetch the current currency rates, adapt it for the selected base currency and insert. It uses the ECB values for this and will only fill in the currencies defined as used, the rest will be ignored. It will also fill in (if missing) the historical table with the values for that day, but in that case will fill in all available currencies.

- Update historical currency rate

This should be run once a month at the most, it will fetch the rates for the last 90 days and fill in the historical currency table with those values converted to match the selected base currency of the system. This is a safeguard to make sure no rates where missed. It will only fill in values that are missing.

Timers

Timers have a trigger condition, a set of steps that can either send emails or update a single field in a record, and a stop condition. Each step can send out an email or update a single field in the record of the table it covers. It was made for sending out reminders, but can also be used for other automatic processes where pauses are suitable between steps.

Timers are complex to enter, but simpler and has better tool support than writing it from scratch as a modification to the code. It is possible to add other actions in additions to email and simple updates in code however.

When a timer is added the base information is entered as well a list of all the required steps. once that is done and saved, open the timer and tweak parameters, For emails the only extra configuration options is the template used and an optional query to get a file attachment to include. The email template itself has all info needed for the email. Update steps simply have a field name and the value to set it to.

From each timer record it's possible to get a preview of all steps to run, and also too see all active timers in the system.

Note

The preview will be empty if the next run of timers doesn't have anything hits on the trigger.

The preview and active timer gives pretty good visibility into the running system, and the active list shows all timers as inline edit fields, so the timers can be modified or stopped if required.

To enter new timers a good understanding of the database layout is required, both for the tables used to initiate the timers and the timer records themselves. The configuration above referes to timers, but is actually stored in timer_type and timer_step. When run the actual instance of each timer is stored in the timer table, pointing to the configuration tables.

Misc other

There are a many minor settings that can be done, usually limited to simple definitions, like what kinds of phone type and emails you want to store on a customer or person.

Customer class

This is a way to classify customer, all you need to enter is a name for each class, example of customer classes can be Customer, Supplier, Reseller, Partner, Suspect, or Prospect. All depending on your needs and workflow.

Customer type

This is more basic than customer class, there are some predefined types like C - Company, P - Person, O - Organisation, A - Animal. PIck the ones you want to use. In the normal installation Company and Person are activated.

Eaddress types

Here you define what kind of electronic address you want to store on customers. Email and phone numbers are usually helpful to know. So is Fax and mobile phone number. You can also enter web

addresses, IM addresses of different types or even split up web addresses to you have several kinds of addressThe basic table es available. You will have to define what code you use for emails so the system will know what value to use to send emails.

Addess types

This is similar to eaddress but revoles around physical addresses instead, so invoice address, shipping address, main address, Department address or old address if you don't want to delete the previous address from the system when the customer moves.

Countries

You can define the countries you want to be able to access here. There is a ready made list of all countries that is inserted when you install the system. If you want to keep the dropdown limited you can delete the ones you're not interested in. The values entered is the country code, the three letter long standard ISO code for countries. The short_code, is the two letter ISO country code (also used as TLD names for countries). The country name itself is stored as the description and a phone code.

User roles.

This is to define relationships between users in the system. It's not used by the system for anything, it's just to record for others how people work together. You define both parts of the relationship, so if you have for example "reports to" the other column should be "get's reports from". You can have works with, shares office with, has mentor, or anything else that makes sense in your environment.

Departments

Here you can enter a list of the departments that exist in your company. Note that this is not linked to legal entity. The user you enter is the one responsible for this department.

Timezones

This is a simple list of timezones and what country that timezone applies to. There is a ready made list that is imported at start.

Currency

This list is used to defined what currencies should be used in the system. If automatic currency rate updates are not available it is possible to also enter the rates here. Automatic updates are recommended if more than one currency is needed, currency rates should be entered as the reverse rate for calculations to work and that can be confusing and lead to errors.

With automatic updates a full historical record is also kept, and it's possible from some sources to get such a historical record automaticaly as well. With automatic updates the base currency will first have to be selected on the configuration screen. In the currency rate table, the default currency will have the rate of 1. All currency values stored are collected with both a value and a currency, so convertions to the base currency is never stored permanently. That makes it possible to change the base currency at a later date, as long as the historical currency table can also be updated. If values for that is taken from an external source the simples is to simply delete the historical table and fetch it again.

Translations

The translation system can be used to handle translating parts of the database content to get localized text to use for email templates or parts of the website.

Translate fields

Here you enter the table and fields you want to translate. You can also enter a term to filter on in case you don't want to translate the entire table but only a few rows. If you edit a line here you can also click on "Translate texts" on the side to change all translations to different languages for the specified table and field. This is less useful than doing the same thing per language where you get all translated fields in a specific languages instead.

Translate languages

In the list you enter the languages you want translations to. If you edit a language and click on "Translate texts" you can change all translations. The orange box holds the original text and the white box below it the translated text. Click on the translation text to update it. Starting out the original language will be used for every translation. Once you're done editing, click update to save the new text.

Refresh translations

This will simply take the fields and languages and make sure there is translation data available for it. After making changes to the fields or the languages you should run this to get the translation data set up. The original text will be copied into a translation field and will have to be updated manually by someone who knows the language.

Appendix A. MySQL Backup and Replication

Database backups

Semi related to replication is backups, here's a simple script that will copy the content of the database.

```
#!/bin/sh
mysqlhotcopy customerdb -u root -p *password* --record_log_pos=customerdb.mysql
tar cfjh /data/backup/current/backup.tar.bz /data0/backup/current/customer
mv /data0/backup/current/backup.tar.bz `date +"/data0/backup/custback_%Y_%m_%d.
```

The above script works on Linux and does a complete backup of the entire database. It's recommended to have at least daily backups of the data. Keeping a copy of every day is also very convenient in case there is a corruption that has been unnoticed for some time, having older files to choose from when trying to find a clean copy can be invaluable.

Setting up replication

Twospot can be used locally in multiple locations sharing the same data, to do this you set up MySQL circular replication and enter different key sequences for each server. The data will travel round the replication chain and stop when it returns to the server where it was first entered.

Servers are numbered both in the MySQL setup and in the TwoSpot setup, they don't have to match, but it's convenient to have the same number for both to avoid confusion. Each server must have it's own sequence interval defined in the TwoSpot layer, server 1 starts at 1 and ends at 999999, server 2 starts at 1000000 and ends at 1999999 for example. This is set in the local/conndb.inc.php that also defines how to connect to the database. The three variables server_id, server_min_no and server_max_no will make sure that any auto increment value will be within those intervals. If a server runs out of digits within the sequence the update that fails will be stored in an update log and the user is notified that the update failed. If this happens, the right thing to do is to assign a new previously unused interval for the server.

To get the clone started it's handy to use the backup script above, note the --record_log_pos directive in the mysqlhotcopy statement. The row inserted by that can be used start a new slave or repair a broken slave.

When using the replication scheme described above it's possible to have every server in operation at the same time to spread the amount of work being done amongst the servers and scale the system to many more concurrent users. It's also possible to have a simple one way replication set up and only use the spare machine as a hot backup in case the master goes down due to hardware failure or similar. In either case, it's convenient to run the backups on the least busy server.

Repairing or starting a new replication node

For starting new nodes or resetting nodes that broke for whatever reason, follow these steps. You'll need a backup file as created by the script above.

Note that to reset a clone, you need to use the following - this is mostly useful for a two node setup, if you have more it's a tad complicated in comparison.

1. Copy to the new machine (Node2) the most recent backup from the existing machine (Node1).

2. Stop the MySQL server on Node2, unpack the files it into the database directory, start the server again.

3. On MySQL @ Node1: grant replication slave on *.* to repl@Node2 identified by 'mypassword';

4. On both servers, make sure you have the right my.cnf settings, server-id and log-bin must be set.

5. Double check local/conndb.inc.php to make sure you have the same server_id as server-id, and that you have different sequence intervals. If the sequences isn't setup correctly there will be replication failures and much sadness.

6. On MySQL @ Node2: change master to master_host = 'Node1', master_user='repl', master_password='mypassword', master_log_file='<take from mysql_replication_log_pos table, column log_file>', master_log_pos='<take from mysql_replication_log_pos table column log_pos>';

7. On MySQL @ Node2: start slave;

8. show slave status; and show master status; will give you an idea of what's going on. If Node1 has been active for awhile after the database backup was extracted the Seconds_behind_master value on Node2 will work it's way down to zero as the replication catches up. NULL in that fields means replication stopped and the time to catch up is unknown.

Appendix B. GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/ .

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (C) YEAR YOUR NAME.
>
> Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.