PHPRunner 10.3 Manual

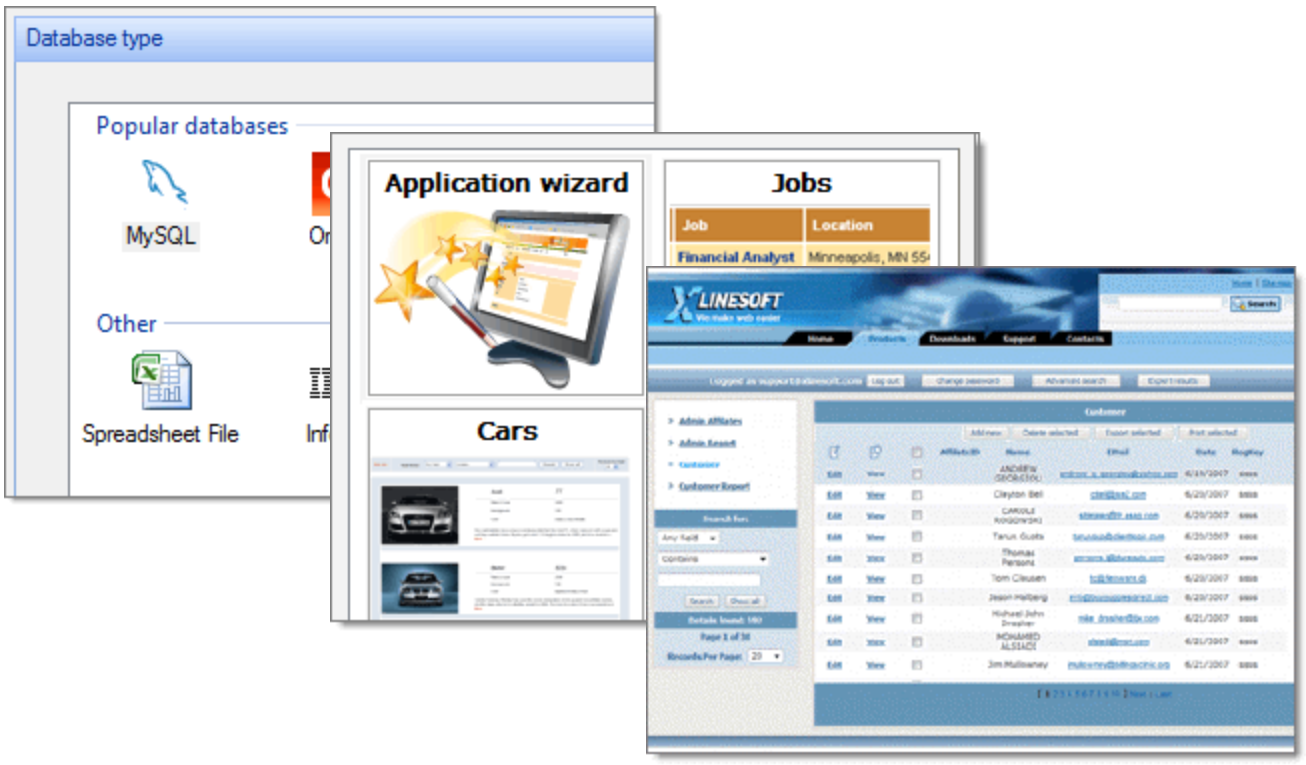


Table of Contents

1 Introduction

1.1 Welcome

Welcome and thank you for choosing PHPRunner!

PHPRunner creates a set of PHP pages to access and modify any MySQL, Oracle, MS SQL Server, PostgreSQL, or MS Access database. Using generated PHP pages users can search, edit, delete, and add data into the database.

PHPRunner is extremely easy to learn, you can get started in just 15 minutes!

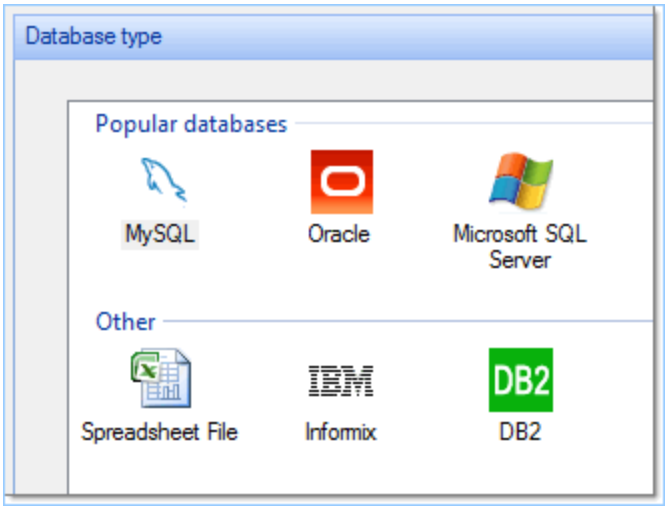
Templates

PHPRunner offers a large number of application templates - ready-made themed websites with complete graphical interface and database structure. All of the templates are easy to work with and completely customizable. The template can be used as a stand-alone website or can be integrated with other PHPRunner web applications.

Some of the templates available with PHPRunner are Cars, Classified ads, Knowledge base, Real estate, Job listings, and News.

See About templates to learn more.

Wide range of supported databases

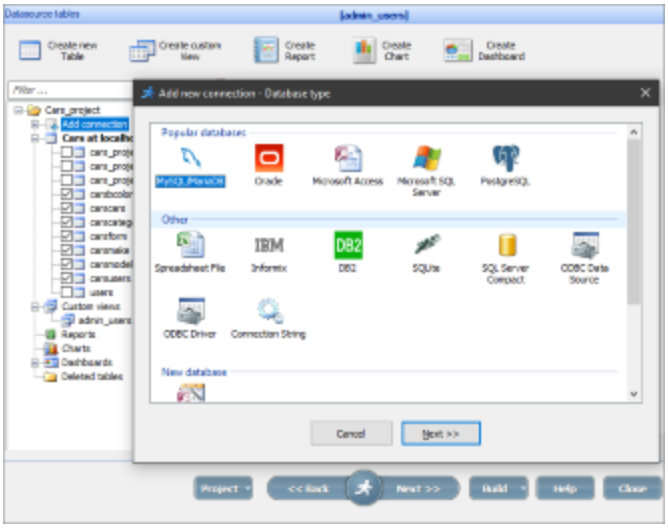


PHPRunner supports four database types, including MySQL, SQL Server, MS Access, and PostgreSQL. Even if you don't have a database, the software helps you create one.

PHPRunner lets you connect to your local database or a database located on a remote server. If you have a remote MySQL database, which does not allow a direct connection, you can connect to it using the "PHP proxy" method.

See Connecting to the database to learn more.

Multiple database connections

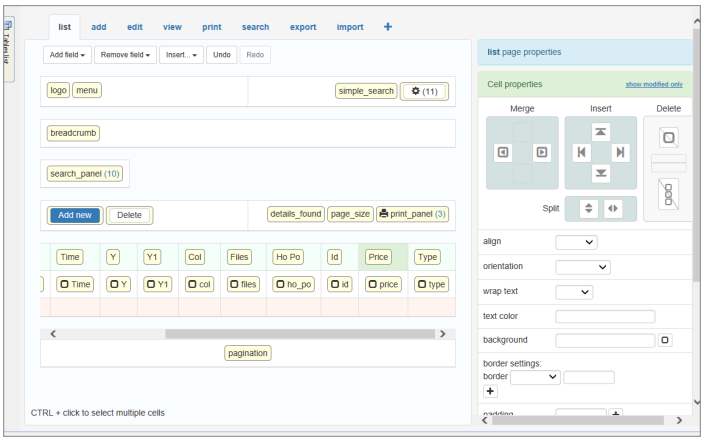


PHPRunner lets you add multiple data sources and mix several database types in a single PHPRunner project. You can have a master table in MySQL and a details table in MS Access. The same applies to lookup tables.

The multiple database connections feature is a part of the Enterprise Edition of PHPRunner.

See Datasource tables: Multiple connections to learn more.

Page Designer



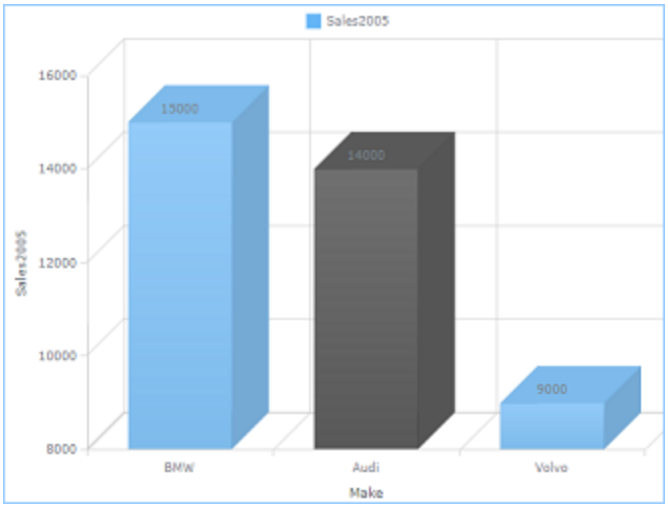
WYSIWYG **Page Designer** allows you to customize the look and feel of your application by using drag-n-drop and configuring the properties of pages, cells, and elements. PHPRunner presents you with a proposed layout for each page. However, the layout and all of the elements on the pages can be easily modified.

Page Designer allows you to drag and drop, copy and paste the objects around the page. For all of the fields and labels on the page, you can change the font, size, color, style, indentation, and alignment.

Page Designer also allows you to apply custom CSS to elements.

See About Page Designer to learn more.

Reports and Charts



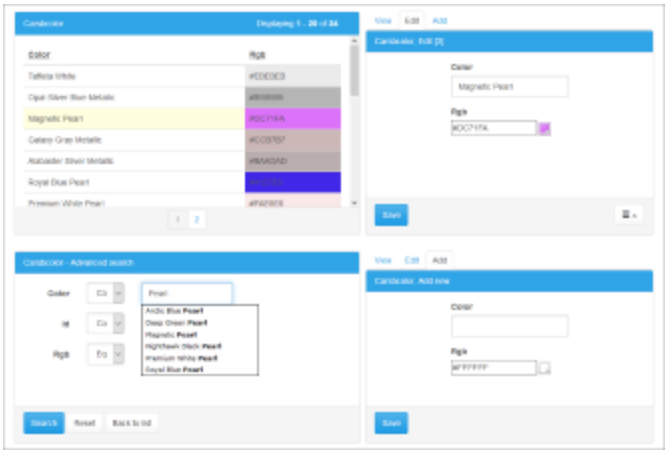
PHPRunner lets you build sophisticated, highly customizable interactive charts and reports to complement your website. You can choose from multiple chart and report types.

You also get a **Web Charts and Reports Builder** as a part of the PHPRunner Enterprise Edition, which lets you build the charts and reports online. Just like in the software, the **Web Charts and Reports Builder** offers a large selection of charts and reports that take just minutes to configure.

You can reuse the same security settings you have established in the program to decide which data sources you would like to expose to the users, and what permissions those users would have.

See Online report/chart builder to learn more.

Dashboards

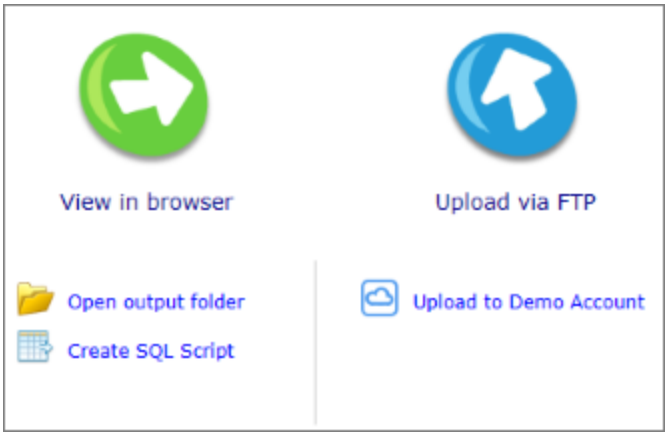


With PHPRunner you can create various types of easy to read and powerful dashboards. They allow you to display multiple related or unrelated objects on the same page like grids, single record views, master and details together, search pages, maps, or even code snippets.

You can change the dashboard layout and customize its appearance.

See Creating dashboards to learn more.

Application Preview

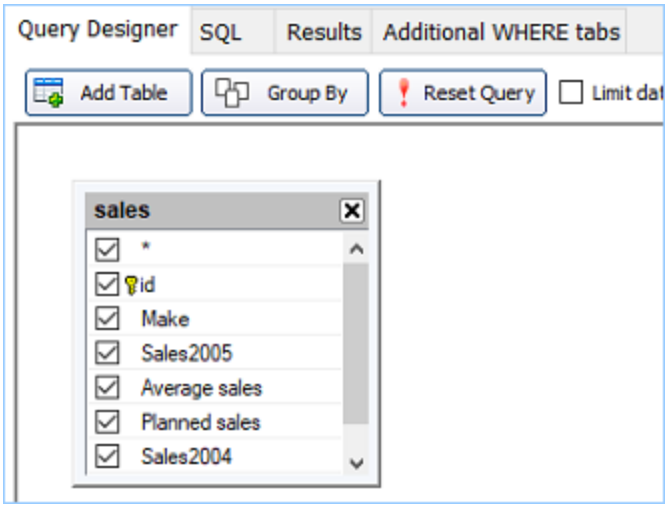


After you have built your web application, PHPRunner provides you with multiple options on how to preview it.

Preview your application in the browser locally and then upload the files to the Demo Account or to the remote web server using a built-in FTP client.

See After you are done to learn more.

SQL Editor



PHPRunner automatically creates an SQL query that can be visually modified with the Query Designer or edited as a text.

The SQL editor also allows you to preview the results of your SQL statement, create joins with drag-n-drop, and specify the criteria (where, order by, group by, etc.).

See About SQL query to learn more.

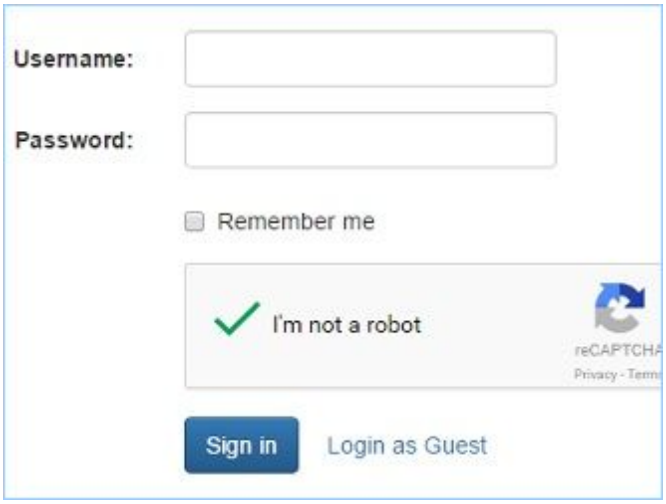
Security

PHPRunner allows you to password-protect the access to your web application. You can either hardcode the username and password, store the login combinations in the database, or - with the Enterprise edition - use the Active Directory authentication. You can add the Login with Facebook or Google option to your site and allow guest access.

PHPRunner also allows you to set up user groups permissions where you can restrict access for certain groups of users to tables, views, pages, and site functionality.

See Security screen to learn more.

User Login Settings

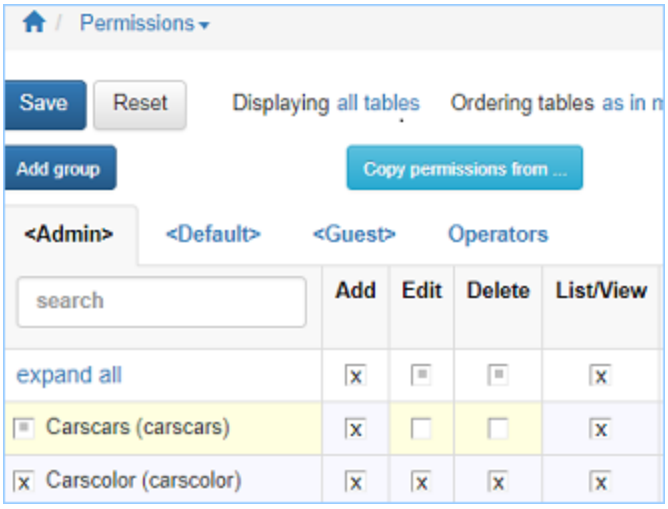


PHPRunner lets you create a new user Registration page as well as secure your pages from automated abuse using CAPTCHA protection, which easily determines whether the user is a bot or a human.

You can also restrict users from entering weak passwords and allow them to request password reminders.

See Registration and passwords to learn more.

Dynamic Permissions

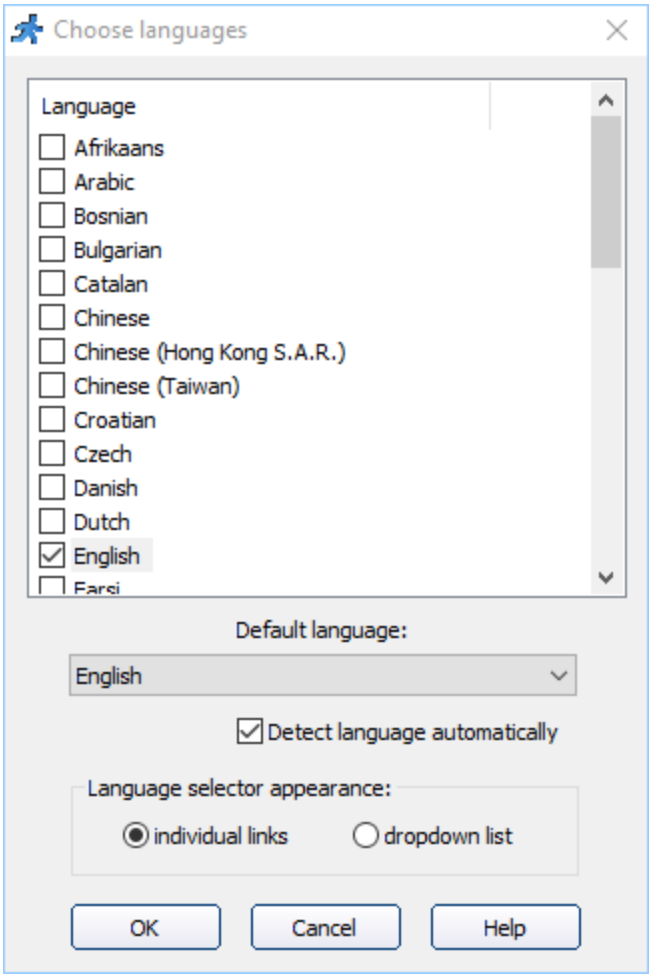


With dynamic permissions, PHPRunner lets you create and modify the permissions and assign users to certain groups within the generated web application. So that when you need to modify the permissions, create a new group, or assign users to groups, you don't have to rebuild the project.

Dynamic permissions are especially helpful for larger corporations where the application security administrators are not the actual users of PHPRunner.

See Dynamic permissions to learn more.

Multilanguage support

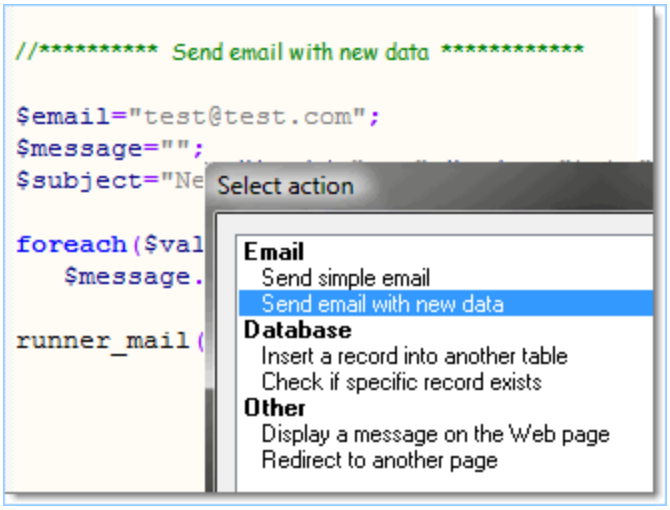


PHPRunner supports more than 30 languages giving your users an ability to choose the language while logging in. You can also add the translations for table names and fields.

The list of supported languages includes: Afrikaans, Arabic, Bosnian, Catalan, Chinese, Chinese (Hong Kong S.A.R.), Chinese (Taiwan), Croatian, Danish, Dutch (Belgian), Dutch (Standard), English, French, German, Greek, Hebrew, Hungarian, Indonesian, Italian, Japanese, Malaysian, Norwegian (Bokmal), Polish, Portuguese (Brazil), Portuguese (Standard), Romanian, Slovak, Spanish, Swedish, Thai, Turkish, Urdu.

See Miscellaneous settings: Language to learn more.

Events

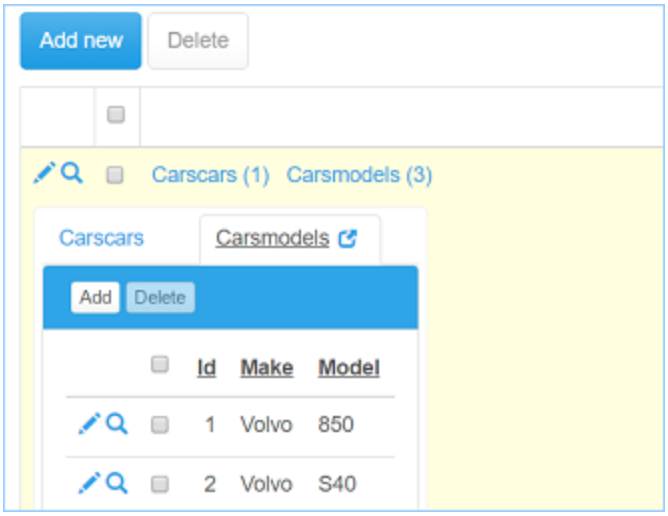


With PHPRunner you can expand the functionality of your application by adding your code to events. Events are actions that are performed automatically when certain conditions are met.

Events allow you to send an email with a new data, save data in another table, check record uniqueness, show info related to the current record, etc. You can either select one of the predefined actions, copy-paste some of the sample events, or write the code from scratch.

See Event editor to learn more.

Master-Details Relationships

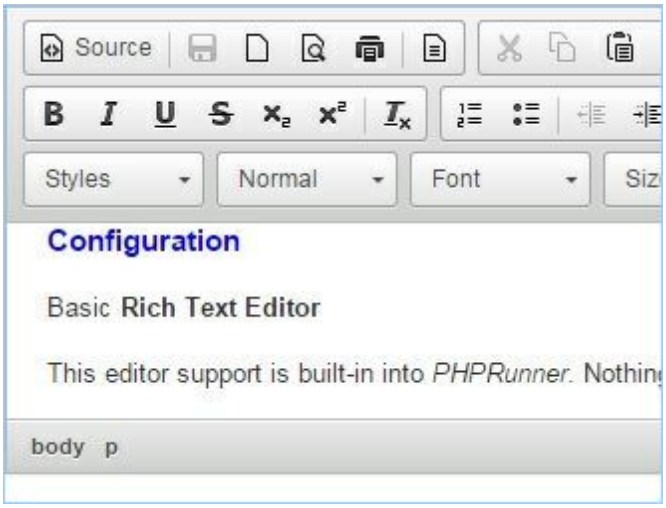


With PHPRunner all of your data sources and the relationships between them, including master-details, are visually displayed, making them very intuitive to understand and manage. You can link two or more data sets using drag-n-drop. Once you have established the relationships, you can navigate through master records and quickly jump over to the details of those records in the application.

Some basic scenarios of master-details relationships are customers and their orders data, patients and their medical records, or students and information about their courses.

See Master-details relationship between tables to learn more.

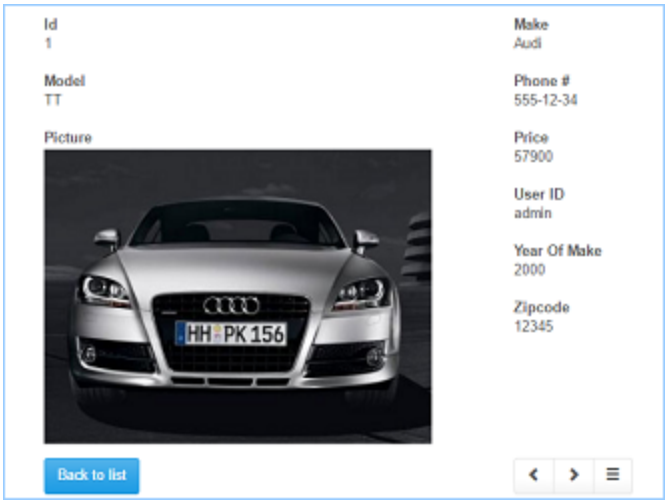
Rich Text Editor Controls



PHPRunner supports three third-party rich text editors to make sure you have a measure of control over content formatting. The included editors are Basic Rich Text Editor, CKEditor, and InnovaStudio Editor. They vary in features, versatility, and footprints.

See Rich Text Editor plugins to learn more.

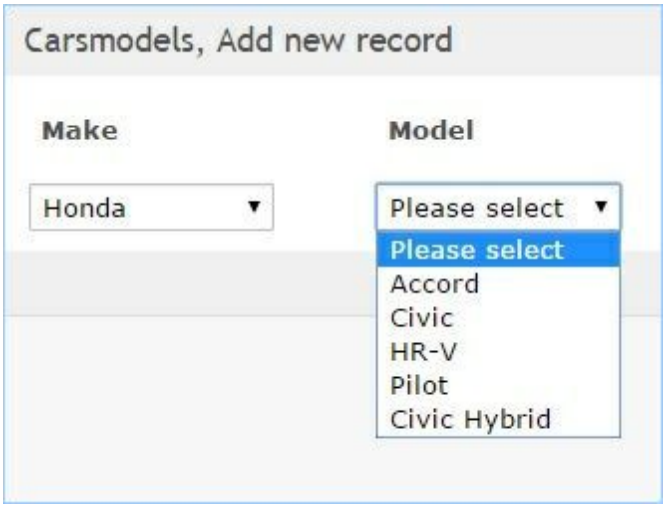
Images and Documents



PHPRunner lets you upload documents and images of any type to the database or directory on the web server. You can also create image thumbnails on the fly, resize images on upload, or display them in galleries.

See Edit as: Image to learn more.

Dependent Dropdown Boxes

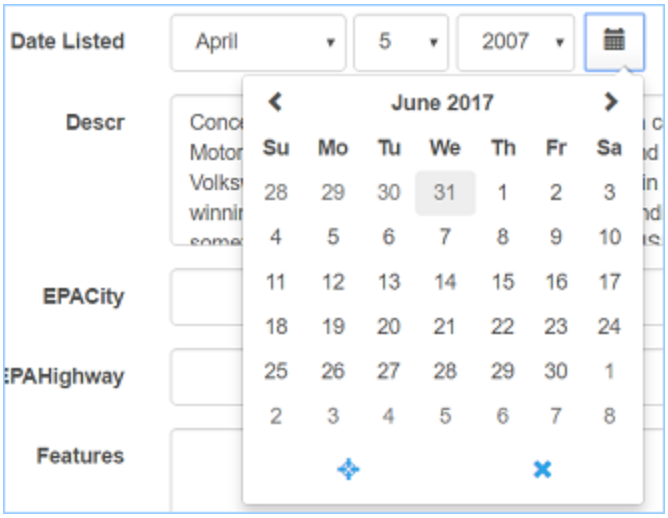


With PHPRunner, you can create and use linked dropdown boxes, where values shown in the second dropdown box depend on the value you've chosen in the first one.

You can link together as many dropdown boxes as you need in a linear chain or have multiple dropdown boxes link to the same master dropdown control.

See Lookup wizard: Dependent values list to learn more.

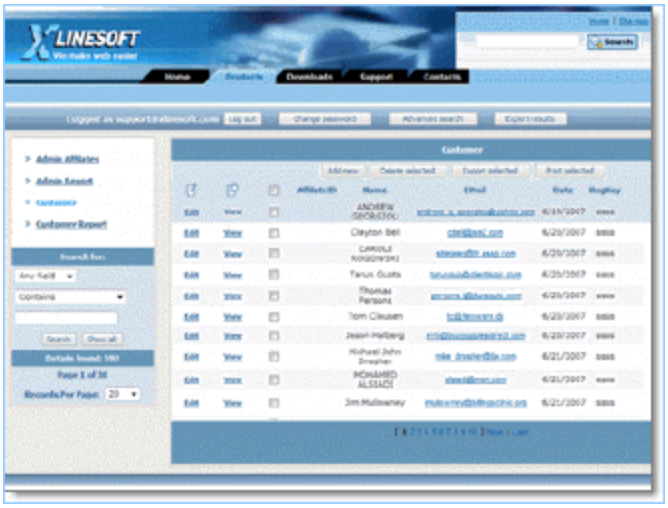
Edit Controls



PHPRunner offers a large variety of edit controls, which allow you to customize the appearance of the fields on Edit and Add pages. The field formats you can choose from include text field, date, time, checkbox, radio button, file/image, lookup wizard, and others.

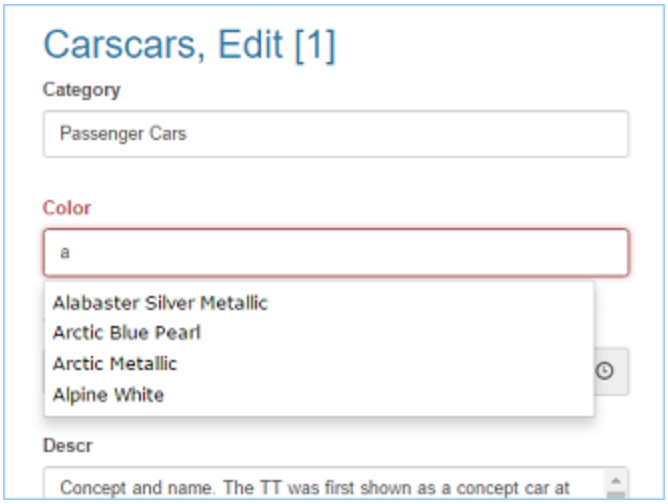
See "Edit as" settings to learn more.

Integration With Existing Website



PHPRunner lets you seamlessly integrate the web applications you build into your existing website. You can closely match the look and feel of all of your pages.

Ajax-based functionality

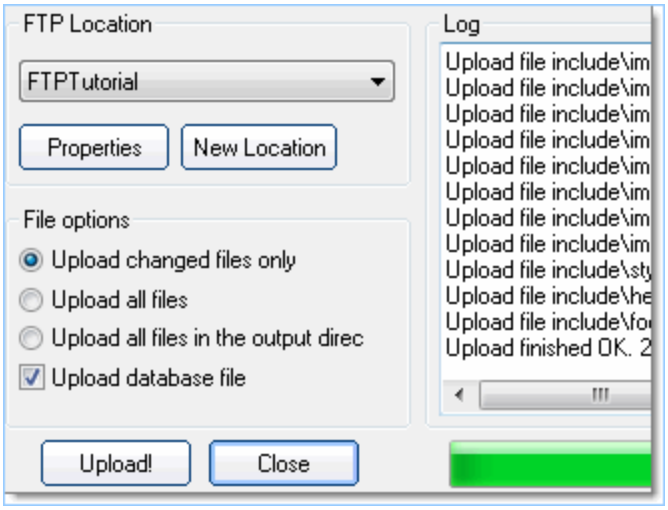


PHPRunner comes with built-in AJAX-based functionality making your websites much more user-friendly. You can search for information more efficiently than ever with a Google-like auto-suggest feature.

The pages with AJAX driven dependent dropdown boxes also load much faster. With AJAX, you can preview the content by simply hovering over the links.

See AJAX-based functionality to learn more.

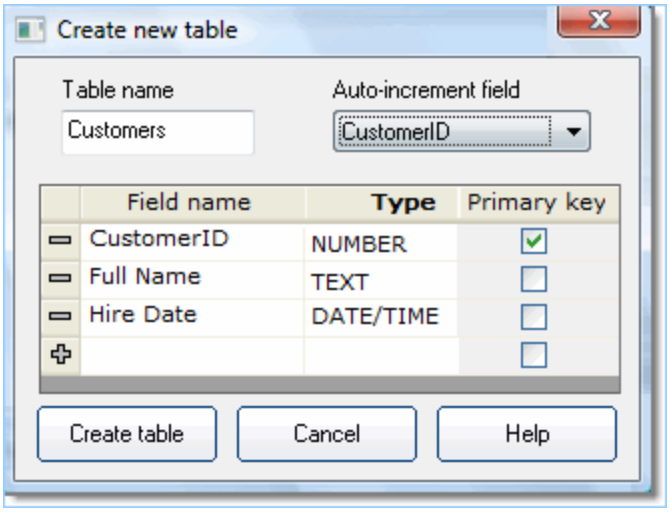
FTP Upload



PHPRunner lets you upload the entire application to your web server via a built-in or a third-party FTP. You can set the upload properties based on your needs.

See FTP upload to learn more.

Create/Modify Database Tables

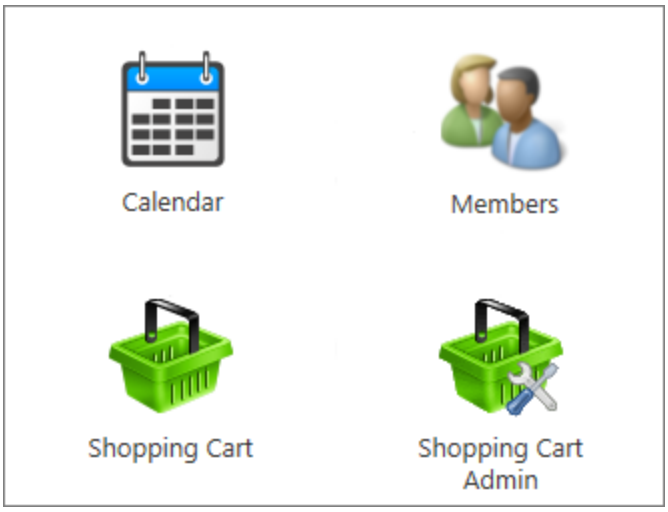


PHPRunner lets you create and modify the tables right in the software. You can access the table properties where you can set the field names, types, sizes, and select the primary key field.

If you don't have a database at all, PHPRunner helps you create one too.

See Datasource tables to learn more.

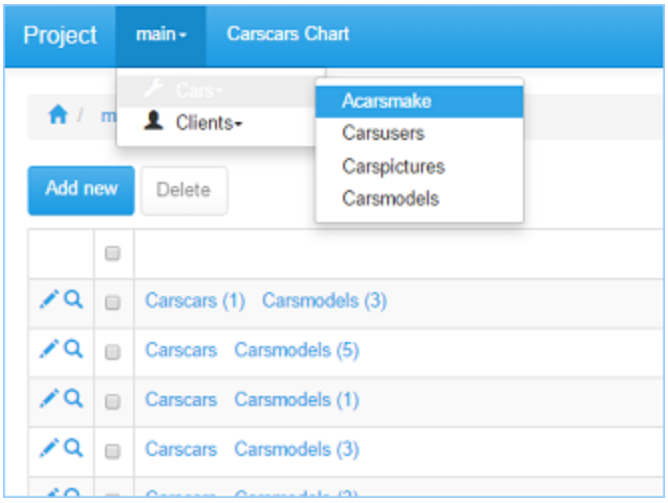
Additional Templates



To complement the list of our built-in application templates, we offer three more templates as the Templates Pack that can significantly enhance your web application. The **Templates Pack** includes the Shopping Cart template, Calendar template, and Members template.

The **Templates Pack** is available for purchase for all existing PHPRunner users and is guaranteed to integrate with other PHPRunner projects.

Cascade Menu Builder

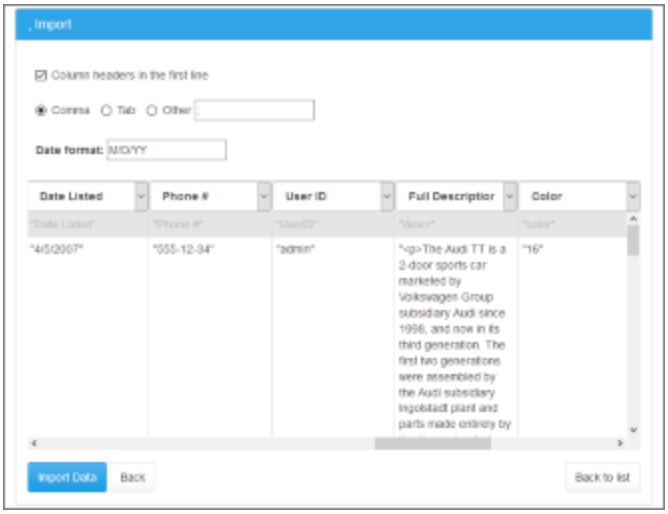


The Cascade Menu Builder lets you organize your tables into a multi-level menu for quicker navigation. This feature is particularly useful to users with a large number of tables.

You can have as many groups and subgroups as you need. Depending on your page layout, the cascade menu appears as a horizontal menu, vertical menu, or a tree-like vertical menu.

See Menu builder to learn more.

Import



PHPRunner allows you to import CSV files, Excel 2007 (.xlsx) and Excel 97-2003 (.xls) files. You can copy and paste import data instead of uploading the file that can be extremely useful on mobile devices.

Column mapping is also supported.

See Export/Import pages to learn more.

See also:

- Quick start guide
- System requirements
- Editions comparison
- Free trial and registration
- Licensing details

1.2 System requirements

Your system should be equipped with the following to run PHPRunner:

Operating system

- Windows 10, Windows 8, Windows 7, Windows 2008, Windows Vista.

Browser

- Internet Explorer 9 or better.

Web server requirements

- Internet Information Services (6.0 or later) or Apache. See How to install local server to learn more about installing a local web server.
- PHP 5.3-5.6, PHP 7.

Note: PHPRunner comes with a built-in web server, which is convenient for local testing.

Supported databases

- MySQL;
- Maria DB;
- Microsoft SQL Server;
- PostgreSQL;
- Oracle;
- Microsoft Access;
- DB2;
- Informix;
- SQLite;
- Any ODBC-enabled database.

Required extensions in php.ini

Extension	Required for
php_zip	Excel export/import
php_gd2	Images manipulation, resizing, thumbnails
php_xml	Charts, Excel export/import
php_curl	SMS functionality (requires PHP 5.6 or higher), Facebook login
php_mysqli	MySQL databases support
php_mbstring	Support of non-English languages
php_com_dotnet	SQL Server/MS Access support
php_ldap	Active Directory/LDAP
php_openssl	Encryption, HTTPS
php_oci8, php_oci8_11.g	Oracle databases support
php_pgsql	Postgre databases support
php_sqlite3	SQLite databases support
php_imap	EmailReader template

We also suggest setting write permissions on the following folders:

- templates_c;
- files/images upload folder, usually "files", depends on the settings of the PHPRunner project.

See also:

- How to install a local web server (XAMPP)
- Working with MS Access databases
- Working with Oracle databases
- Open Database Connectivity (ODBC)
- Connecting to a remote MySQL database

1.3 Free trial and registration

PHPRunner is a "Try before you buy" software. This means that we have made the software available to you for a free evaluation. You are entitled to evaluate the software for up to 21 days without obligation to pay. After 21 days, if you decide to keep the software, you must register your copy with us.

Demo version (non-registered) of PHPRunner is a "full-featured" release. This means that the same capabilities available in the registered software are present in the non-registered software. This allows you to try out all the features in PHPRunner to confirm that they work to your satisfaction.

Registration entitles you free technical support for 90 days and one year of free upgrades. Registration may also entitle you to discounts on new software releases from XLineSoft. We will also send you information bulletins by email to let you know about what's happening with other XLineSoft products.

Finally, by registering the software, you provide us with the resources and incentive to support the software with updates and develop additional quality software products in the future.

How can I register?

See also:

- Editions comparison
- Licensing details

1.4 Editions comparison

This table shows the difference between PHPRunner Trial, Standard and Enterprise Editions.

Features	Trial	Standard	Enterprise
Number of database objects per project (objects are tables, charts, reports, custom views)	20	Unlimited	Unlimited
Number of builds per project	200	Unlimited	Unlimited
"Unregistered" banner at the top of all pages	✓		
"Evaluation version" message on the charts	✓		
Web report/chart builder			✓
Active Directory support			✓
Data encryption			✓
Multiple database connections			✓

See also:

- Order PHPRunner online
- Licensing details
- Free trial and registration

1.5 Licensing details

License

By receiving and/or using PHPRunner, you accept the following Evaluation and Registered User Agreement. This agreement is a binding legal agreement between XLineSoft and the purchasers, users or evaluators of XlineSoft's software and products. If you do not intend to honor this agreement, remove all installed XLineSoft products from your computer now.

Evaluation (Unregistered) and Registered User Agreement

- You may evaluate this program for a maximum of twenty-one calendar days, after which you must register the program with XLineSoft or remove the software from your computer.
- You may allow other users to evaluate copies of the unregistered software. All evaluation users are subject to the terms of this agreement.
- The evaluator/user/buyer/owner is not allowed to attempt to reverse engineer, disassemble, or decompile any XLineSoft products.
- XLineSoft name and any logo or graphics file that represents our software may not be used in any way to promote products developed with our software. All parts of XLineSoft products are copyright protected. No program, code, part, image, video clip, audio sample, text or computer-generated sequence of images may be copied or used in any way by the user except as intended within the bounds of the single user program.
- The evaluator/user/buyer/owner of XLineSoft will indemnify, hold harmless, and defend XLineSoft against lawsuits, claims, costs associated with defense or accusations that result from the use of XLineSoft products.
- XLineSoft is not responsible for any damages whatsoever, including loss of information, interruption of business, personal injury and/or any damage or consequential damage without limitation, incurred before, during or after the use of our products. Our entire liability, without exception, is limited to the customers' reimbursement of the purchase price of the software (maximum being the suggested retail price as listed by XLineSoft) in exchange for the return of the product, all copies, registration papers and manuals, and all materials that constitute a transfer of ownership from the customer back to XLineSoft.
- Each registered copy of the PHPRunner may be used in only one single location by one user. Use of the software means that you have loaded the program and run it or have installed the program onto a computer. If you install the software onto a multi-user platform or network, each and every individual user of the software must be registered separately.
- You may make one copy of the registered software for backup purposes, providing you only have one copy installed on one computer being used by one person. If any person other than yourself uses XLineSoft software registered in your name, regardless of whether it is at the same time or different times, then this agreement is being violated!

- The sale of and/or distribution of registered copies of this software is strictly forbidden. It is a violation of this agreement to loan, rent, lease, borrow, or transfer the use of registered copies of XLineSoft products.

See also:

- Order PHPRunner online
- Editions comparison
- Free trial and registration

1.6 Quick start guide

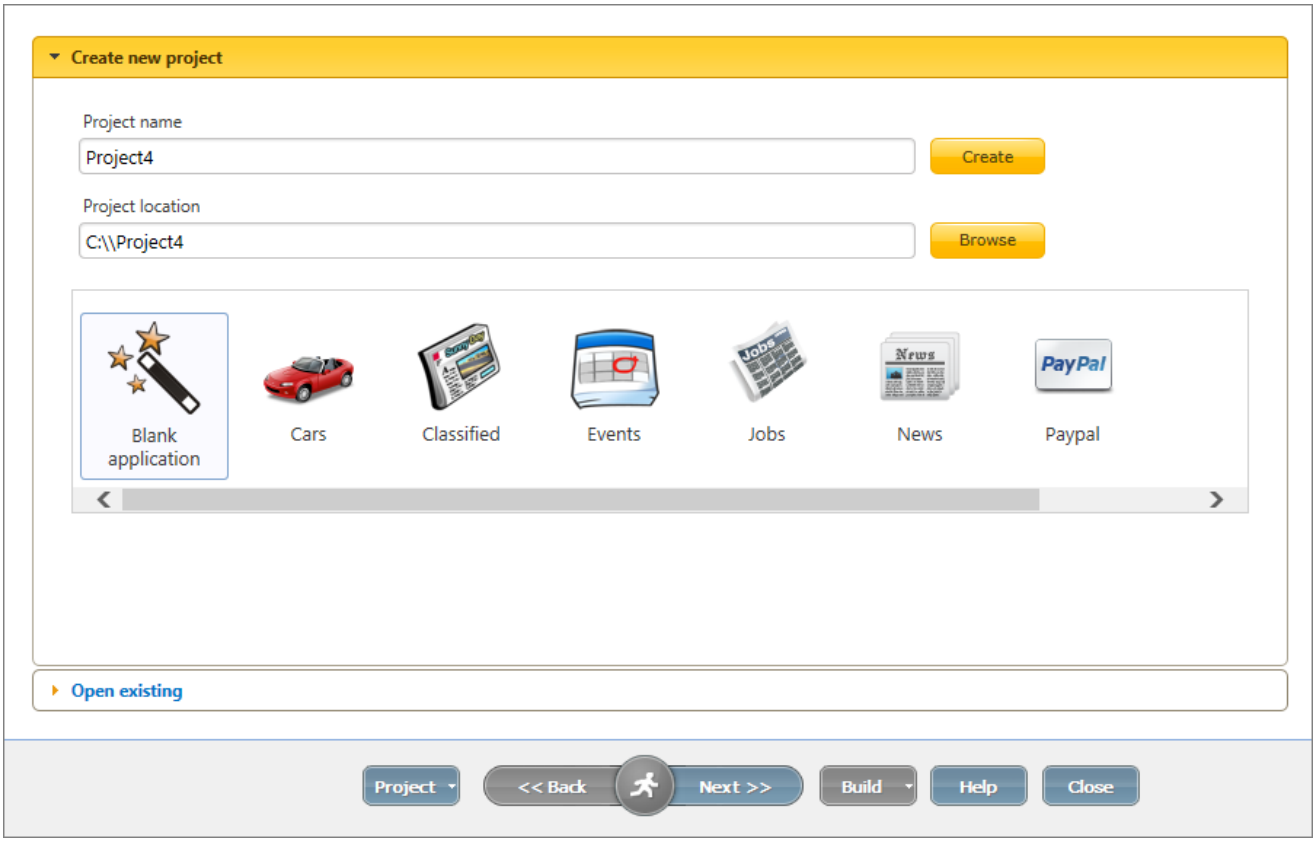
Quick jump

The aim of this step-by-step guide is to help you get an idea about PHPRunner and its main features. After reading it, you will be able to create your project promptly and smoothly. You can always open the help file by pressing the **Help** button on the navigation bar.

> **Note**: To get more familiar with PHPRunner, watch the video tutorials available at http://www.xlinesoft.com/phprunner/php-database.htm

Start working with the projects

PHPRunner provides an option either to create an application from scratch or to choose one of the application templates. Choose **Blank application** if you want to build the project from scratch. Otherwise, you can select one of the predefined templates. Templates are pre-made sample projects and they also create all the required database tables for you.



See also:

- Working with the projects
- Templates

Navigation

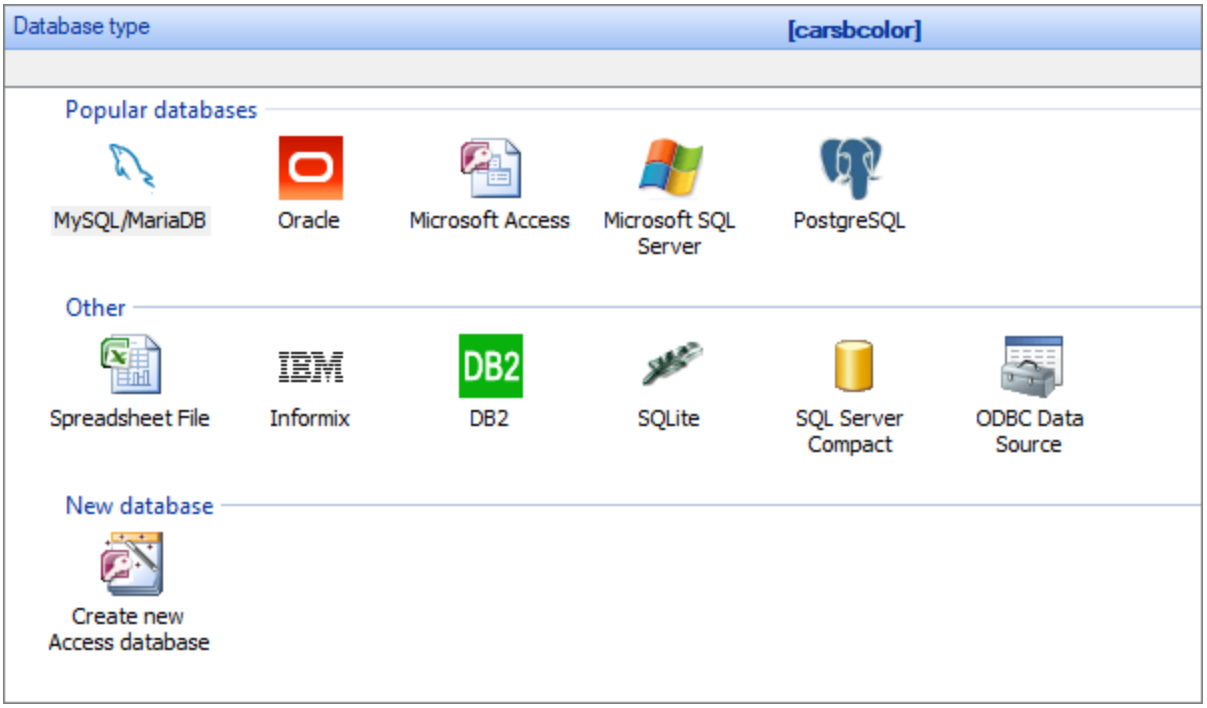
You can navigate PHPRunner by using the bottom panel and clicking **Back** or **Next** to move between the screens.

See also:

- Navigation Bar

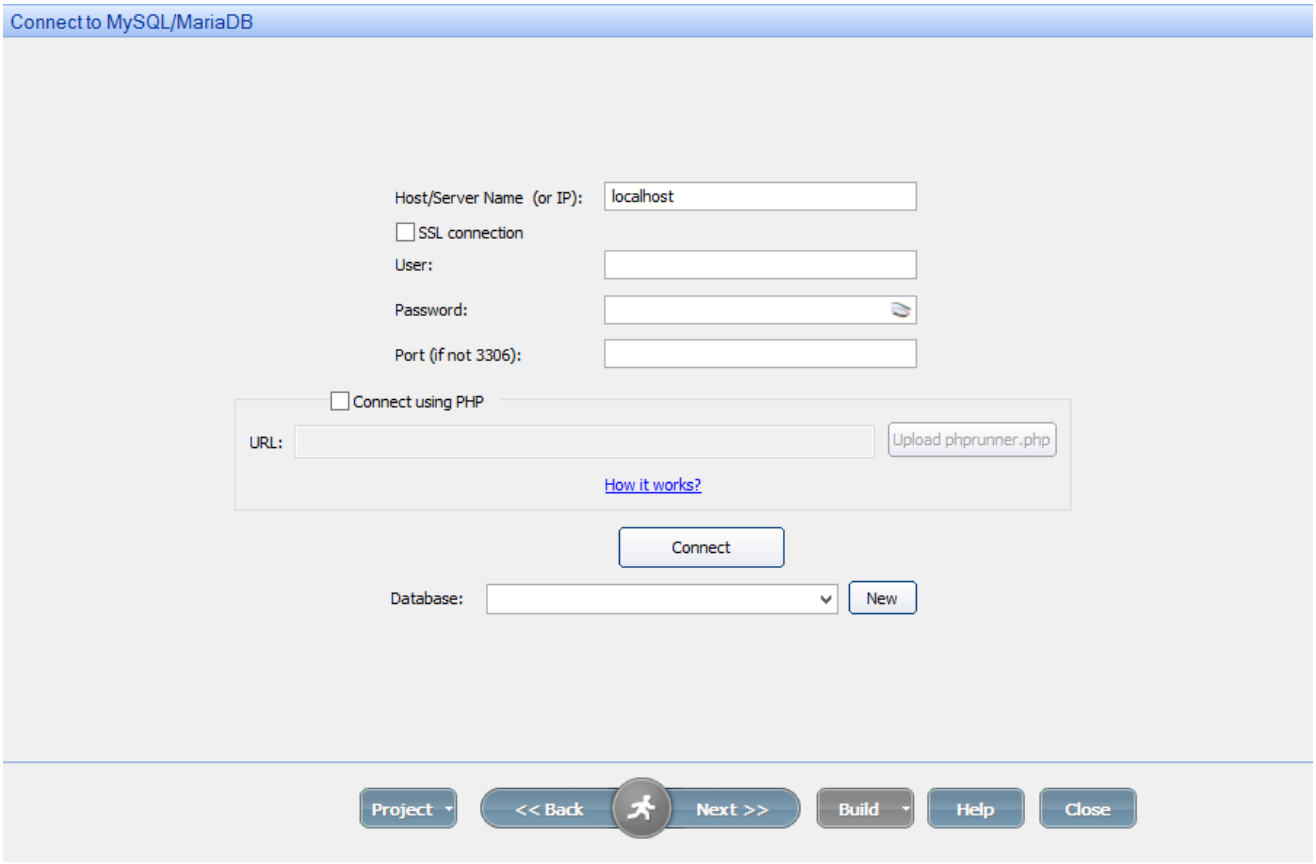
Working with databases

PHPRunner supports all popular databases: MySQL, MariaDB, Microsoft SQL Server, Oracle, MS Access, DB2, PostgreSQL, and more. Select the type of the database you are going to use.



> **Note**: If you do not have a database server or web server, see How to install local server..

Then you should provide the database connection credentials. You have an option to select an existing database or to create a new one.



If you are using MySQL, you can also Connect to a remote MySQL database via PHP.

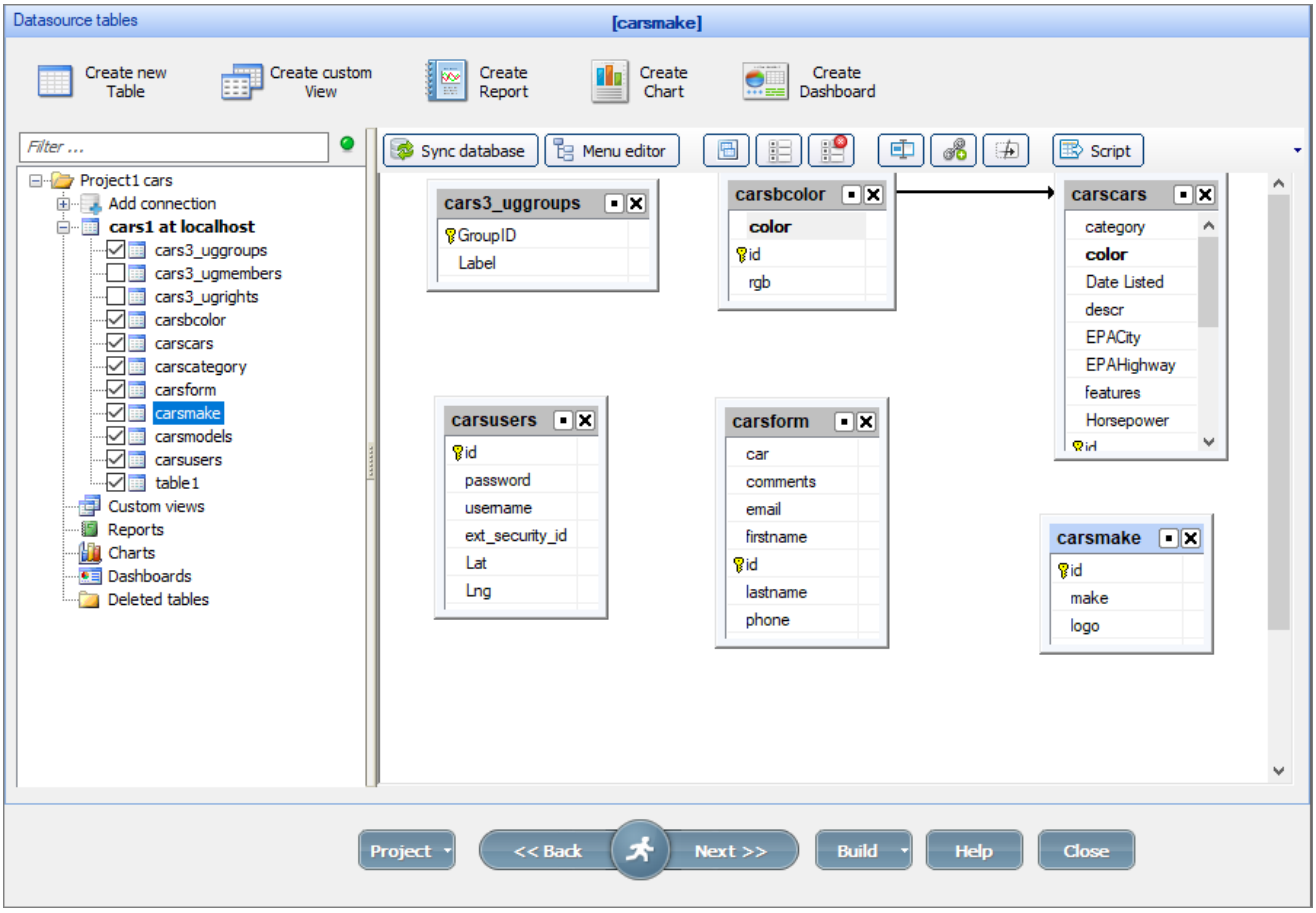
See also:

- Connecting to the database

Working with Tables

In PHPRunner, you can modify the tables, fields and the database structure, as well as create new tables, queries, and relations between them.

You can also create dashboards, charts, and reports on the **Datasource tables** screen to present your project data in a user-friendly way.

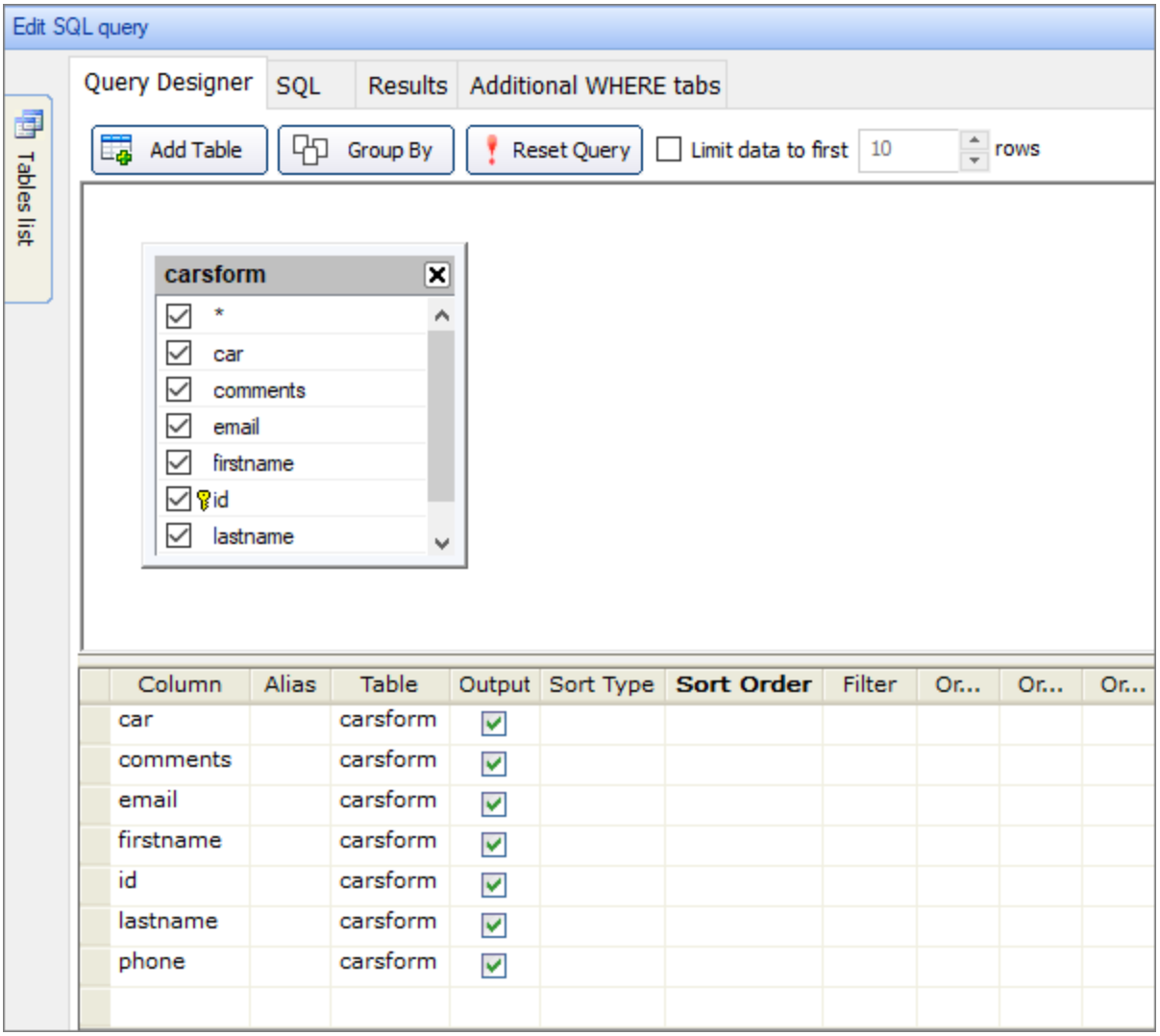


See also:

- Datasource tables screen
- Master-details relationship between tables
- Dashboards
- Reports
- Charts

Queries

This screen provides options for modifying the database and the tables using SQL queries. You can generate SQL queries via Query Designer, or write the code on your own. Using the **Query Designer** can help you sort and filter your data, create aliases to the fields, and control the data output on the pages. You also can preview the Results and create Additional Where clauses.

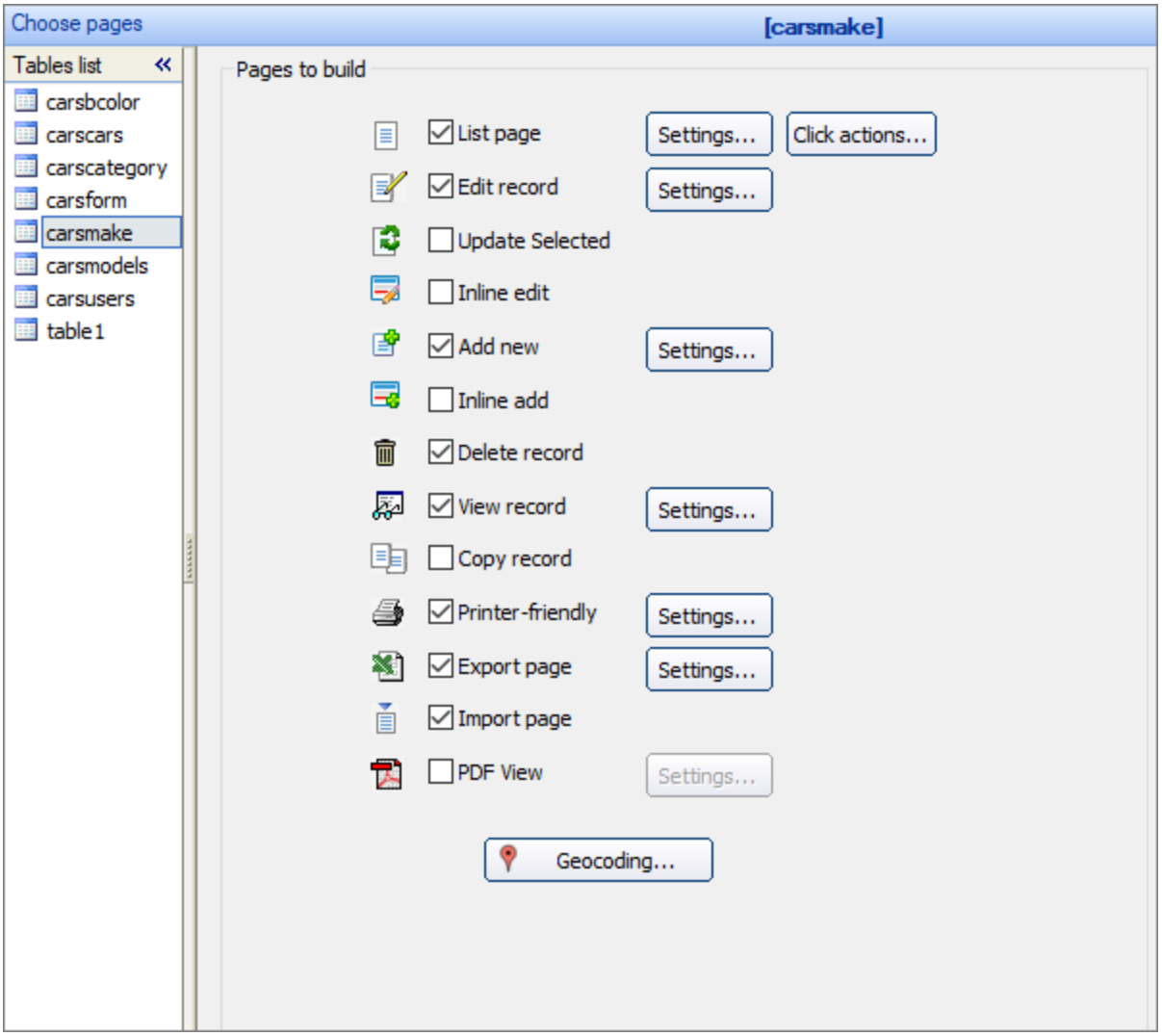


See also:

- About SQL query
- SQL

Pages

On the next screen - **Pages**, you can choose what pages to build for each table. You can also set additional options, for example, show **Add**/**Edit**/**View** pages in a popup or freeze the header of the grid on the **List** page.



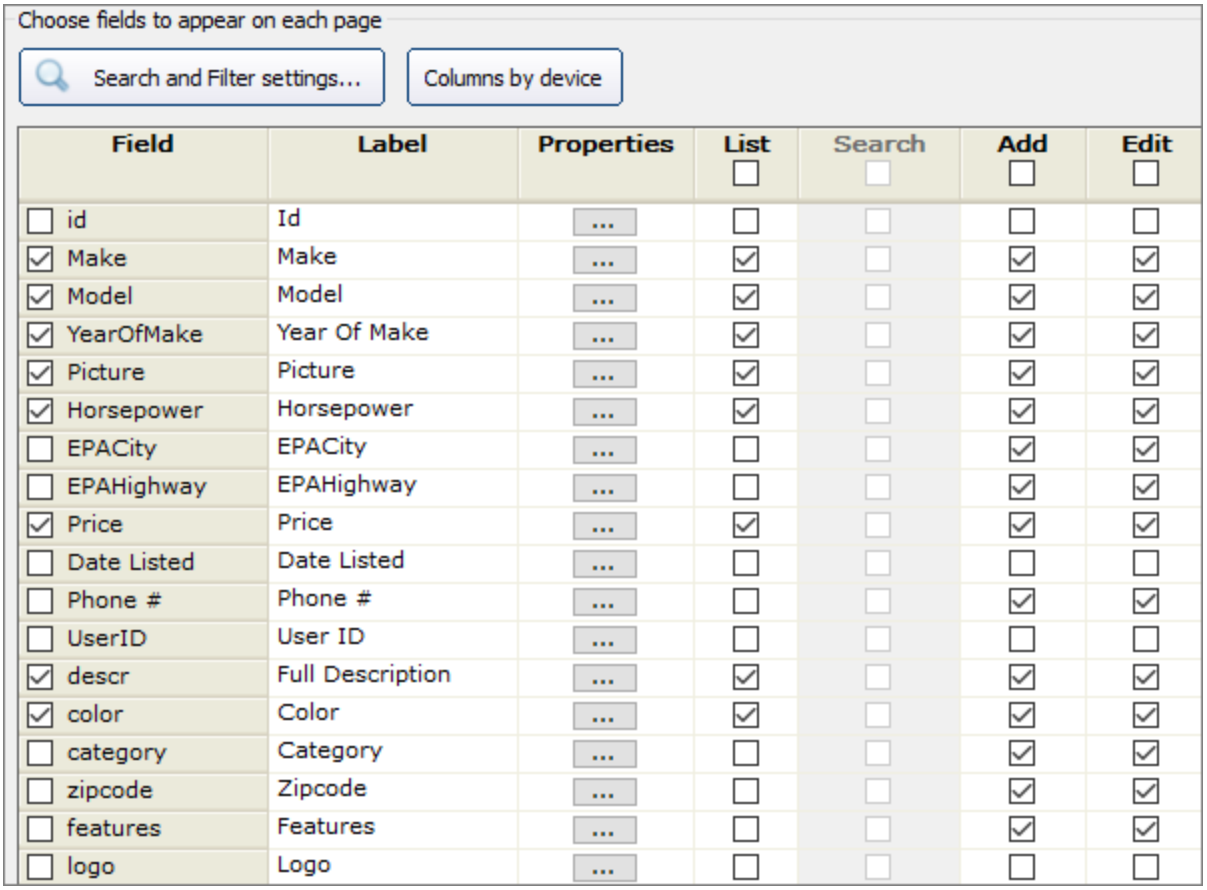
Note: Make sure to select a proper key column. A key column is used to identify table rows, so, the key values must be unique for each record.

See also:

- Choose pages screen
- Key columns

Fields

On this screen, you may choose the fields to appear on each page. Do this by selecting the respective checkboxes. You can also configure **Search and Filter** settings and set up different columns for different device types.



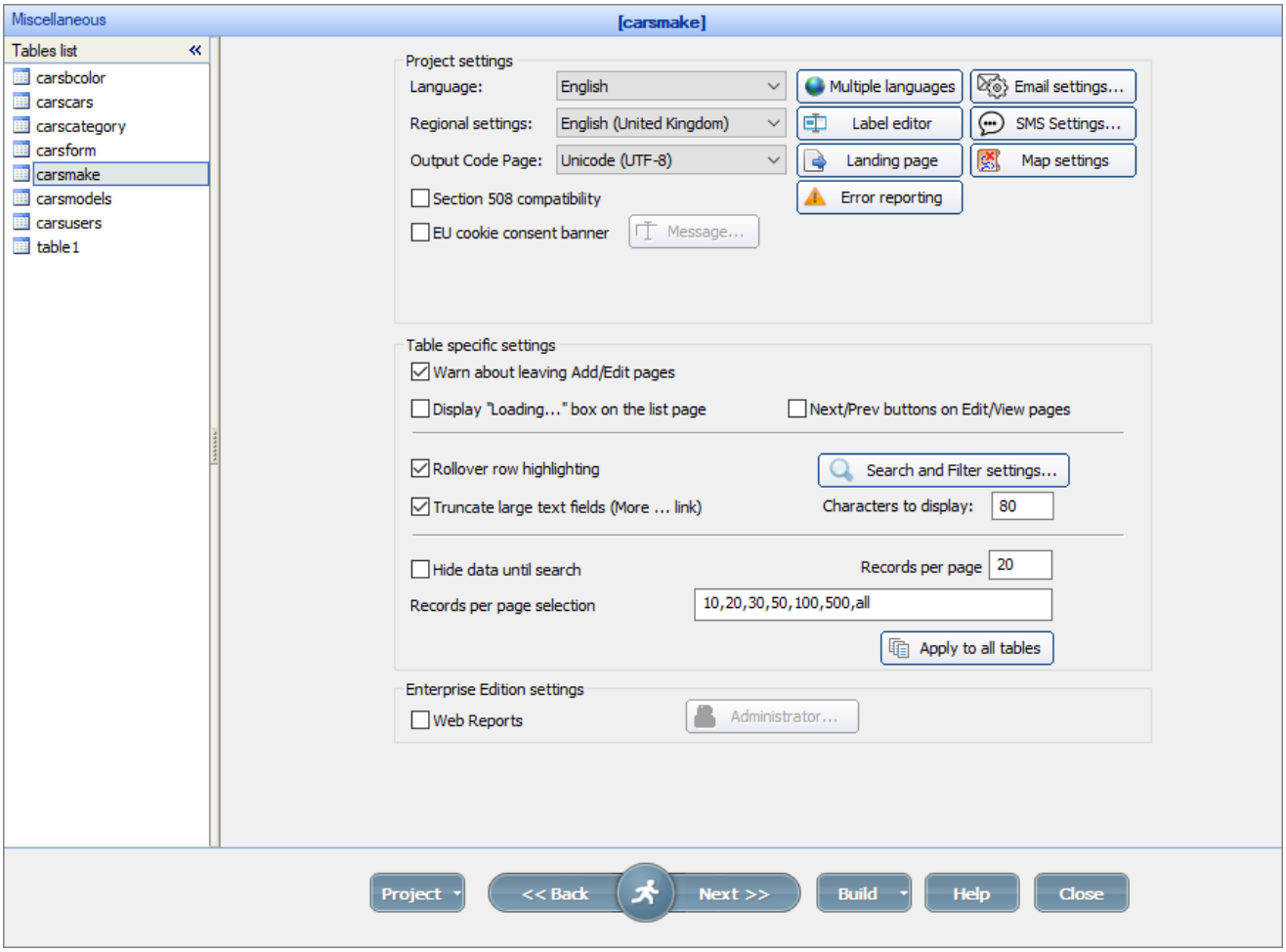
Field	Label	Properties	List	Search	Add	Edit
id	Id	...	☐	☐	☐	☐
Make	Make	...	☑	☐	☑	☑
Model	Model	...	☑	☐	☑	☑
YearOfMake	Year Of Make	...	☑	☐	☑	☑
Picture	Picture	...	☑	☐	☑	☑
Horsepower	Horsepower	...	☑	☐	☑	☑
EPACity	EPACity	...	☐	☐	☑	☑
EPAHighway	EPAHighway	...	☐	☐	☑	☑
Price	Price	...	☑	☐	☑	☑
Date Listed	Date Listed	...	☐	☐	☐	☐
Phone #	Phone #	...	☐	☐	☑	☑
UserID	User ID	...	☐	☐	☐	☐
descr	Full Description	...	☑	☐	☑	☑
color	Color	...	☑	☐	☑	☑
category	Category	...	☐	☐	☑	☑
zipcode	Zipcode	...	☐	☐	☑	☑
features	Features	...	☐	☐	☑	☑
logo	Logo	...	☐	☐	☐	☐

See also:

- Choose fields screen

Miscellaneous settings

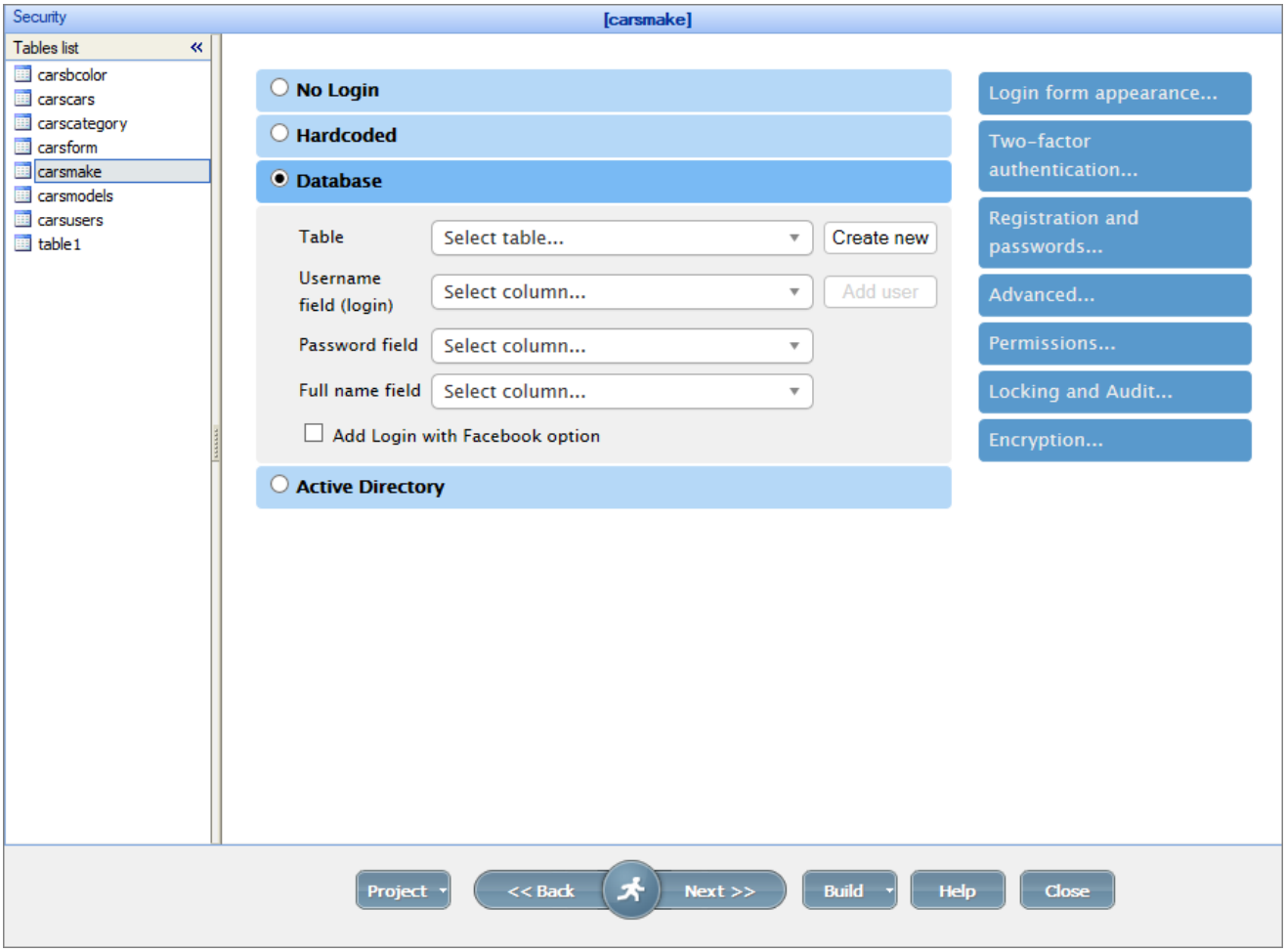
Proceed to the Miscellaneous settings screen to configure the projects settings, such as application language, map settings, notifications, SMS and email settings, as well as the Table specific settings, such as number of records per page, Search and Filter settings, and add **Next**/**Prev** buttons on **Edit**/**View** pages.



Security configuration

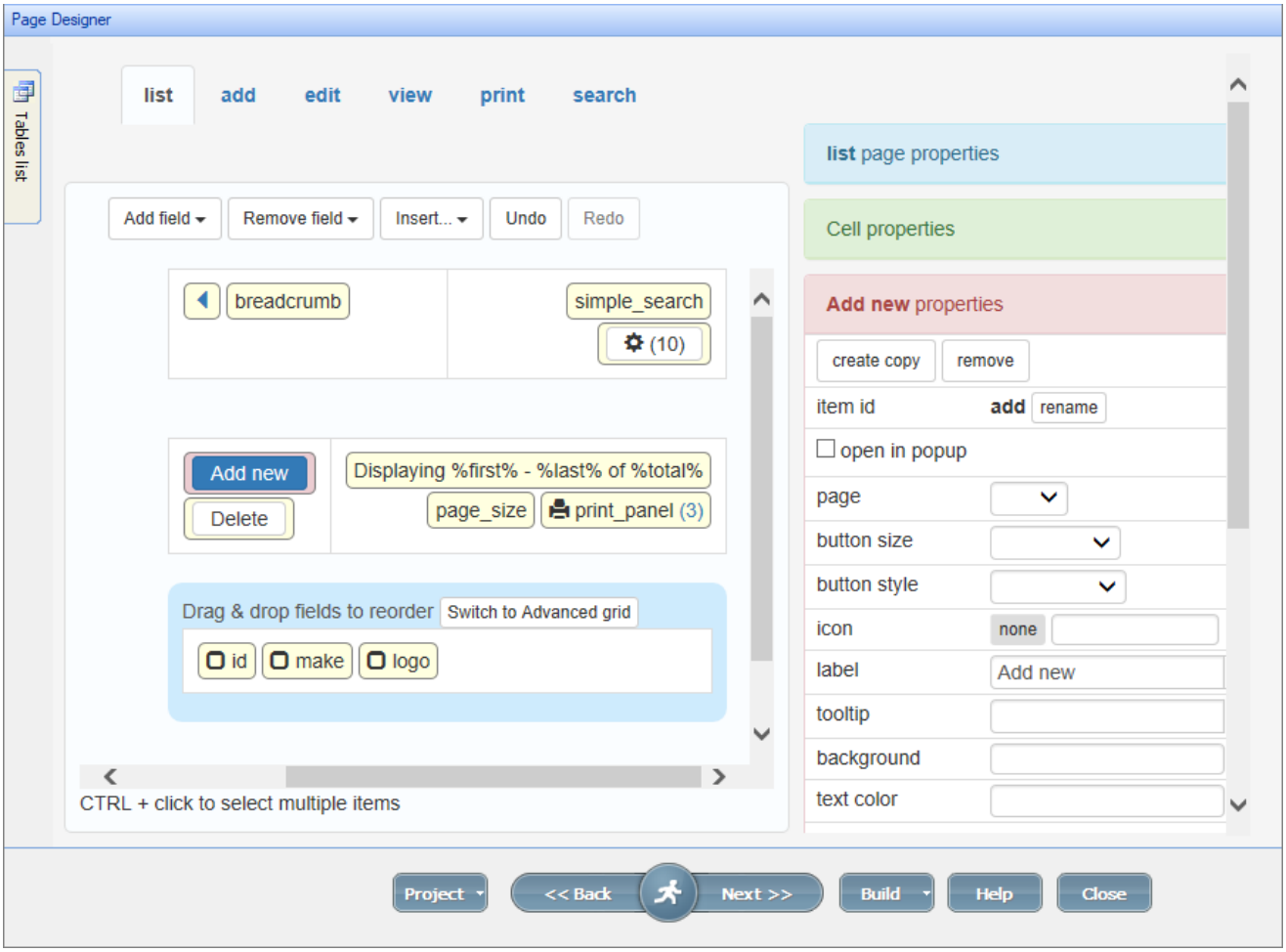
Proceed to the Security screen to configure the user registration and authentication system, as well as to restrict access to the database. You can set up a hardcoded login/password combination, or you can store it in the Database, or even use Active Directory.

You can also regulate access to the specific tables and pages via User Group Permissions.

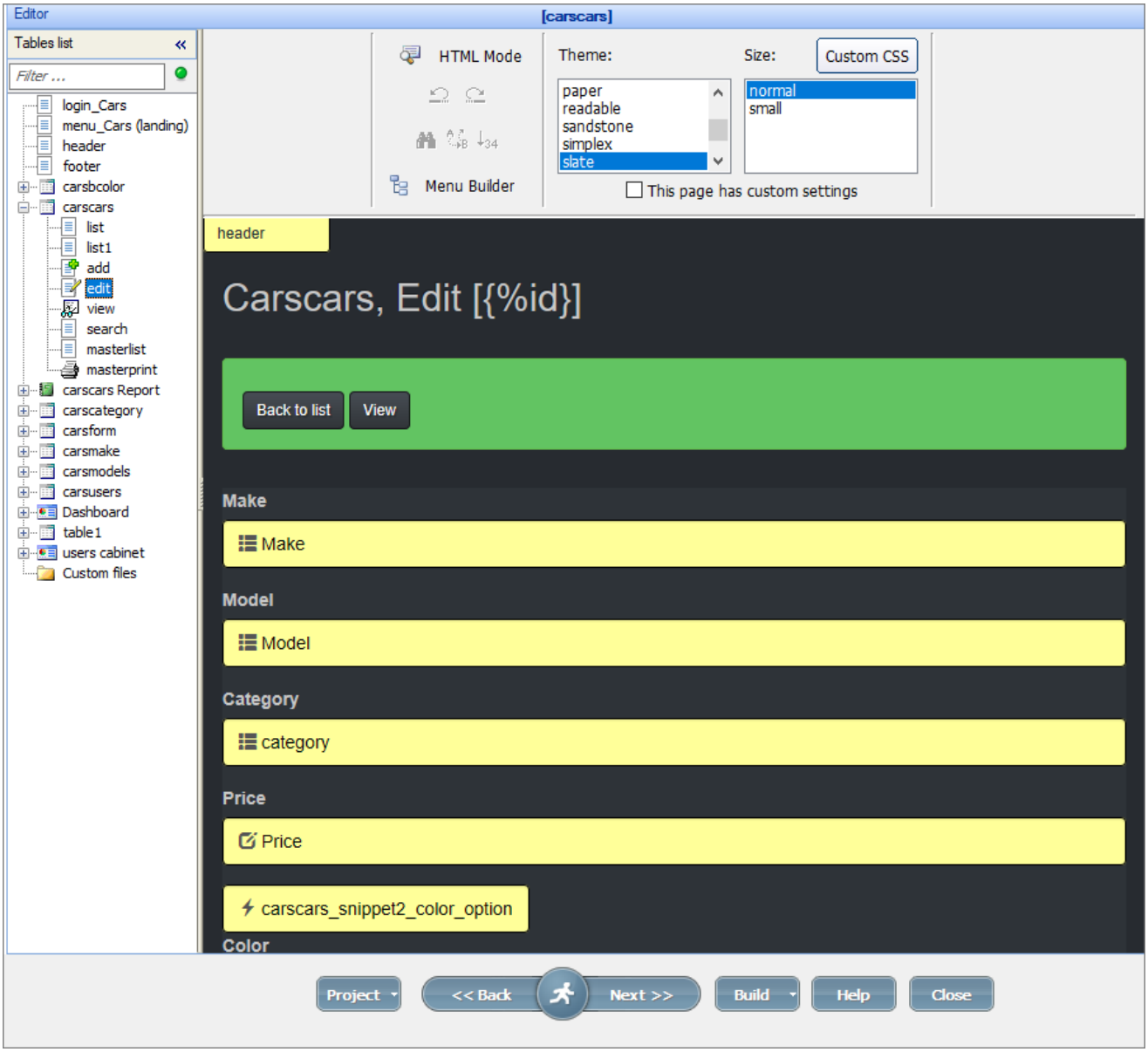


Visual appearance customization

The next step is the visual appearance customization. On the Page Designer screen you can customize and position the elements, and set the layout and grid type of the pages.



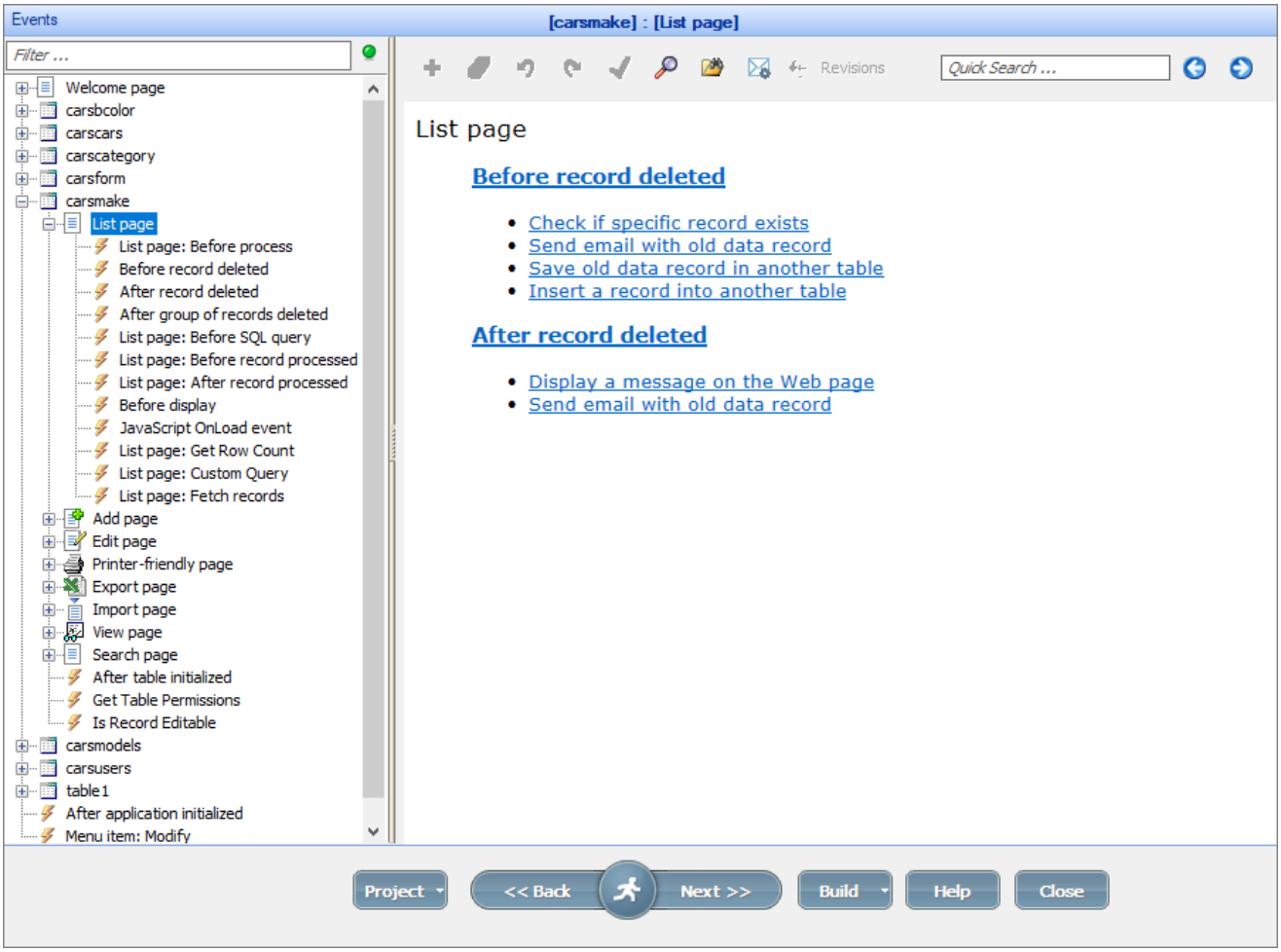
The Editor screen displays the previews of the pages. You can select a theme, set the size of the elements, add custom CSS code, edit the menu, and view the page as HTML.



Events

You are able to extend the functionality of your application - by adding your code to Events. Events are actions that are performed automatically when certain conditions are met, for instance: a record was added to the database, or the user opened the **List** page.

You can use sample event code snippets or write the code on your own.



See also:

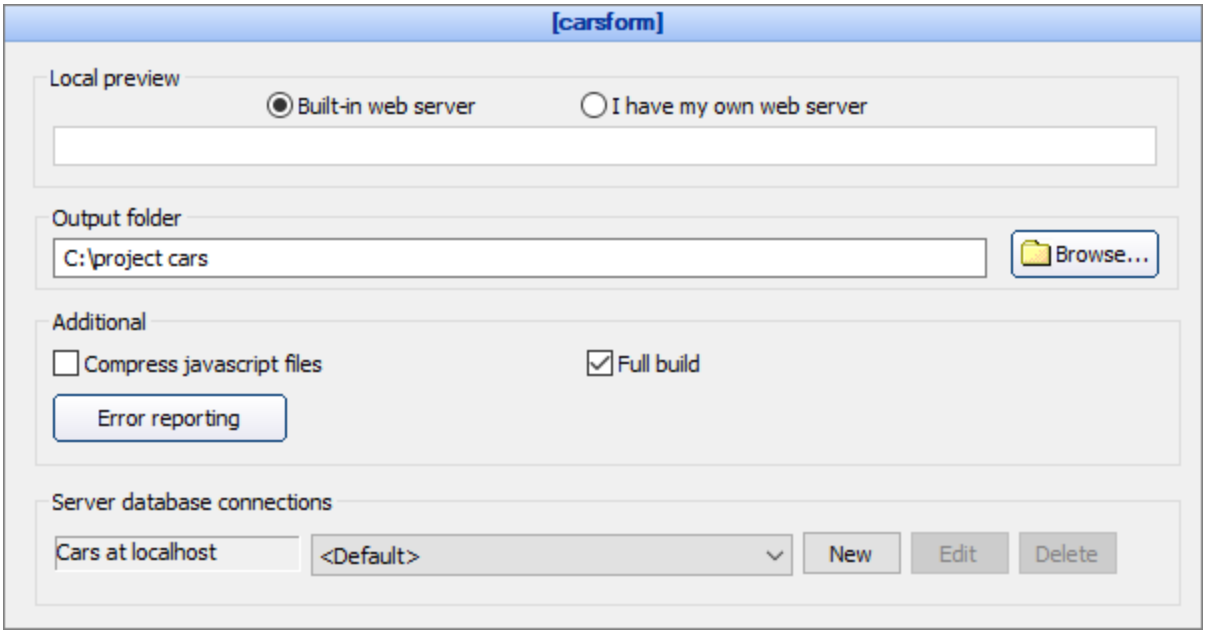
- Event editor
- Sample events

Output

Proceed to the **Output** screen to select the project output directory - a place to store your project files.

You may use the built-in Apache web-server or a custom one to preview the project locally.

You can also configure the **Server database connection**.



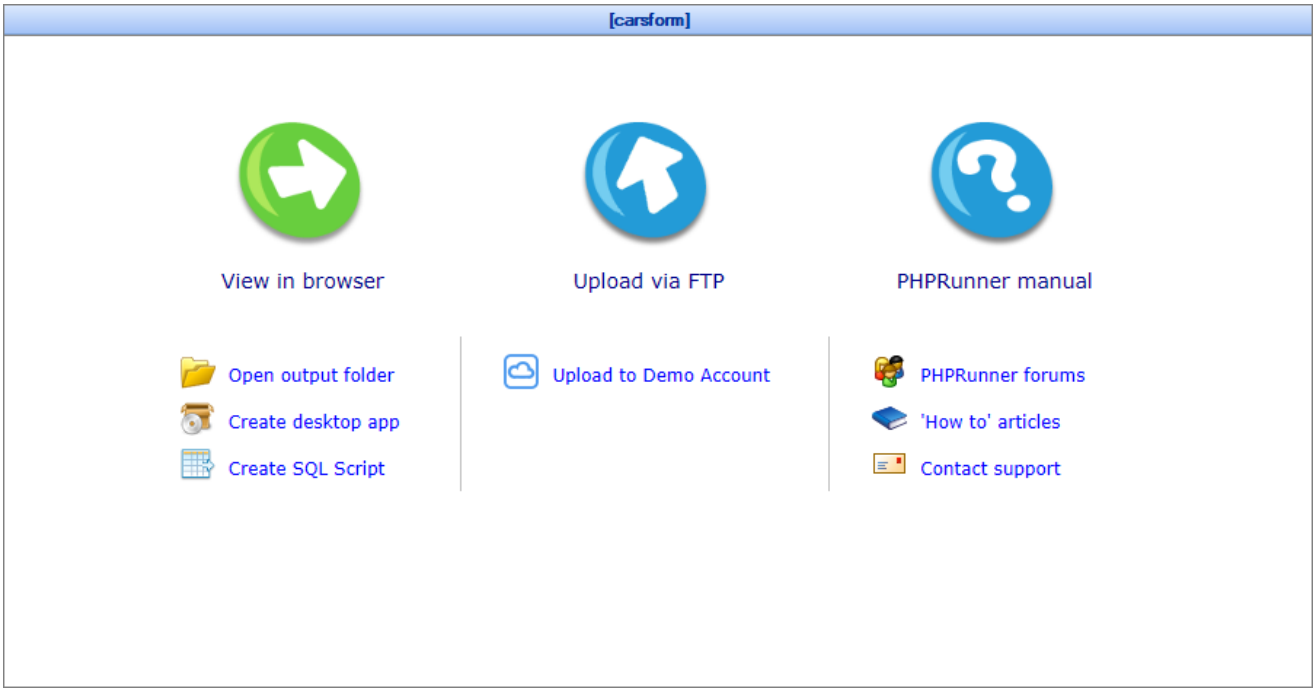
See also:

- Output directory settings
- How to install local web server (XAMPP)

Final stages

As soon as everything is completed, you are ready to build! Now you can preview the application in the browser and start testing it. Moreover, you have options to **Open output folder** of the project, Create desktop app, or **Create SQL Script**.

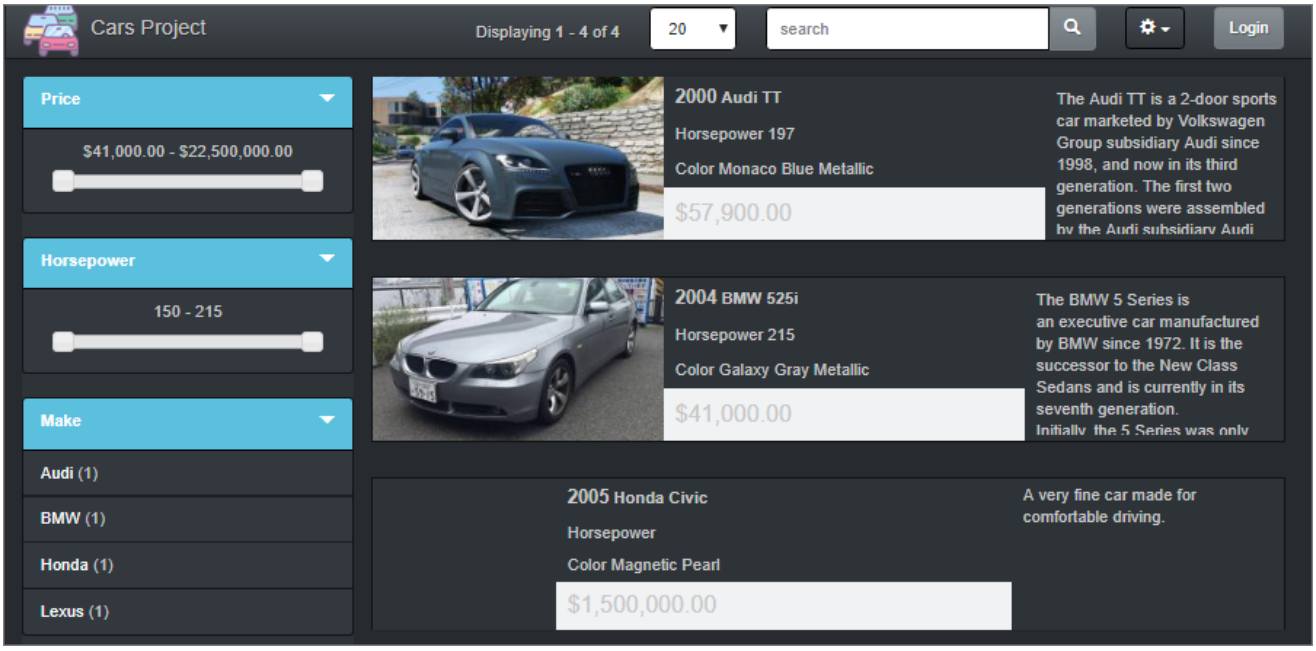
Share your application with other people - publish your project using the Demo Account provided by PHPRunner free of charge. You can also upload the files to your web server using a built-in FTP client or any third-party FTP software.



See also:

- After you are done

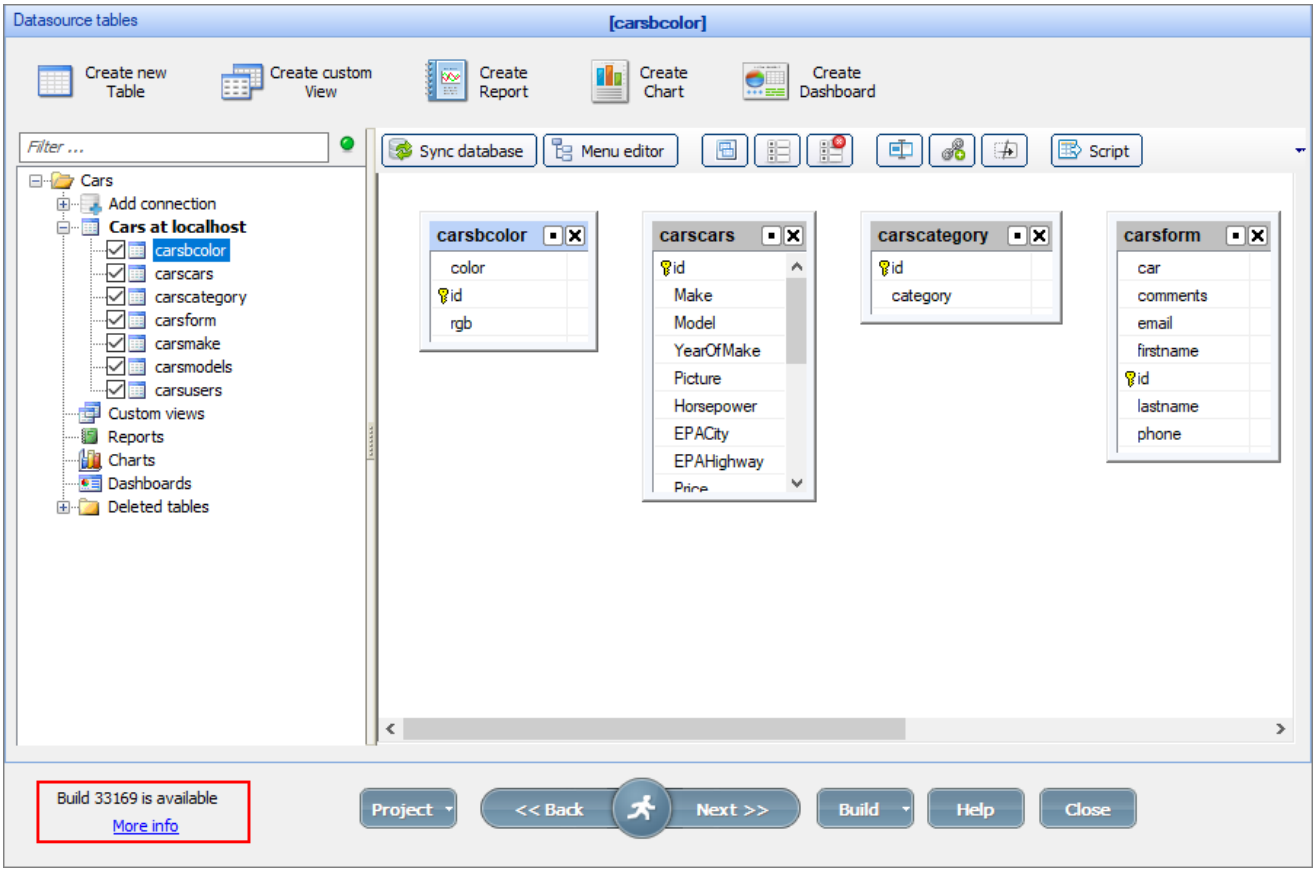
Have a look at a sample page created with PHPRunner.



2 Using PHPRunner

2.1 Updating PHPRunner

PHPRunner has a built-in new version notification. Once a new build is available, a message appears in the lower-left corner of the app.



Click the "More info" link to go to the forum post containing the build info, the latest features, and the download links.

To update PHPRunner, uninstall the version you already have, download the new one, and install it. The projects and their settings are not affected by the reinstallation.

> **Note**: You can check the list of PHPRunner releases on the forums.

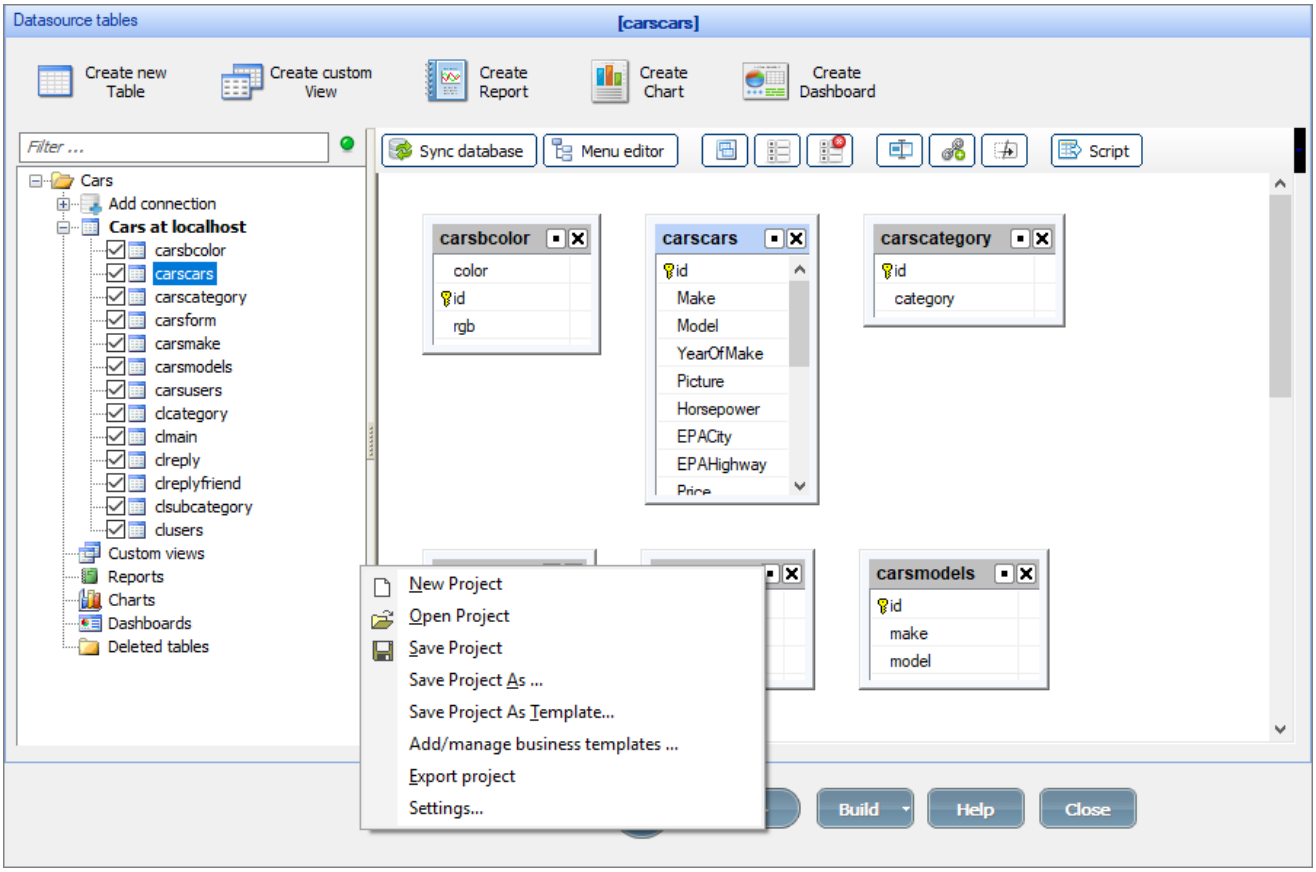
See also:

- Working with projects
- Navigation bar

2.2 Working with projects

Quick jump

Click the **Project** button and select one of the options from the dropdown list to create a new project, open or save an existing one, or configure the settings of a project.



Creating a new project

To create a new project, click the **Project** button, and select **New Project**. When you create a new project, you have two options:

- Create a new project from scratch by selecting **Blank** application.
- Create a project from the template.

For more information about built-in templates, see About templates.

Project structure

With PHPRunner, you can save all your settings in a single project file, so you don't have to go through the individual files every time you need to change something.

Each project in PHPRunner saves the files to its directory, which contains the following subdirectories:

- *visual* - contains modified visual templates.
- *tmp* - a temporary storage for the visual templates and other files. The *tmp\backup* directory stores backup copies of your project. The name of the backup file contains the date and time of its creation (i.e., *Project4.2019-11-09 10_08_00.w.zip*).
- *output* - a directory with output files. You can set the output directory manually on the Output directory screen.
- *source* - contains additional files to be included in the build.
- *styles* - contains project styles and color schemes.

The default directory for a new project is *C:\Users\<username>\Documents\PHPRunnerProjects\project_name*. The project file is saved in the project directory as *project_name.phpr*.

If you upload the project files to the web server using third-party FTP client software, you must upload the entire contents of the output directory.

When you make a backup of your project, you should include all files and subdirectories in the project directory. At a minimum, you should back up the project file itself along with all files in the visual directory.

Opening an existing project

To open an existing project, select **Open project**.

Saving a project

If you want to save the current project under a different name - for development purposes or to create a backup - select the **Save Project As** option. This action creates a new project directory and saves the necessary files into it.

Saving your project as a Template

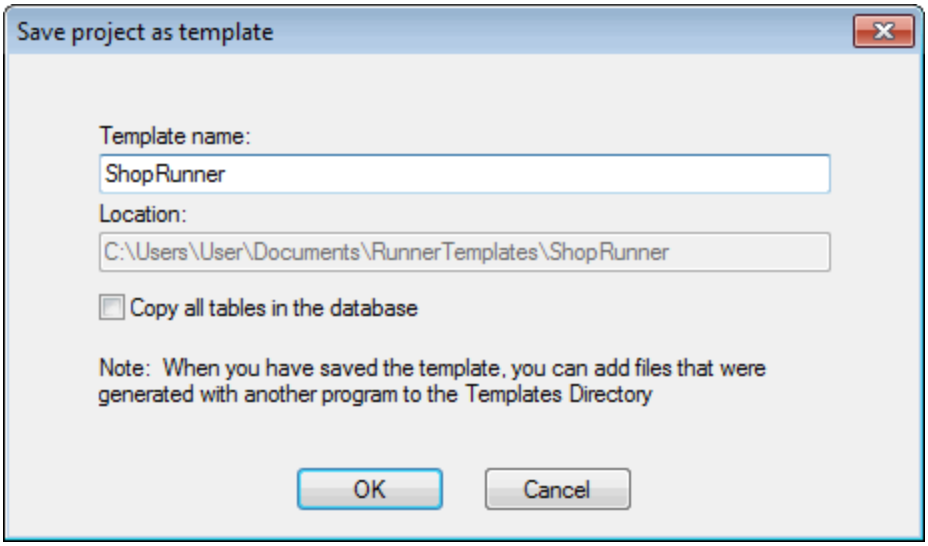
You can save your project as a template.

In this case, your database, the project file, and all the files edited with the Editor are saved.

> **Note**: the **Save Project As Template** option is available only for the MySQL, SQL Server, and MS Access projects.

When saving the project as a template, you need to type in the template name. The template is saved in the Business Templates Directory (by default in *C:\Users\<username>\Documents\PHPRunnerTemplates\template_name*).

> **Note**: after your template is saved, you can add files that are not generated by PHPRunner to the template directory.

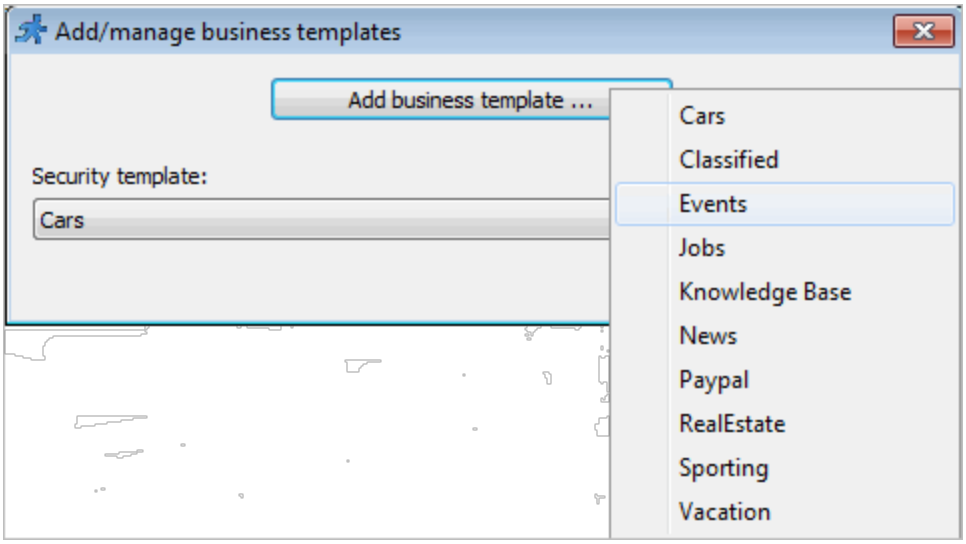


The new template becomes available on the list of templates when you create a new project.

You can also add a thumbnail image to the template to be displayed on the templates list. Place an image named *preview.gif* (JPG and PNG formats are supported as well) to the template folder. The image size should be 130x97.

Add/manage business templates

You can create a new project using two templates or add a template to an existing project. To avoid rewriting the template tables over the existing ones, all PHPRunner business template tables and files have a prefix.



If you have added several templates to the project, you can choose one to inherit the security settings from in the **Security template** dropdown box.

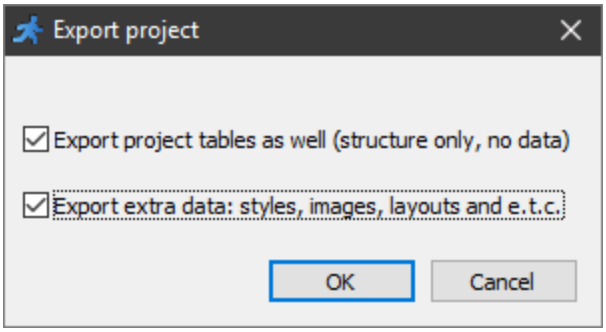
If you add more than one template to the project, it is not guaranteed to run smoothly. This is what you need to do to add two or more templates to the same project:

- Create a temporary project using the second template
- Proceed to the AfterAppInit event and copy the code there. Close the temporary project without saving.
- Open your original project and paste this code to the end of AfterAppInit event.

Export project

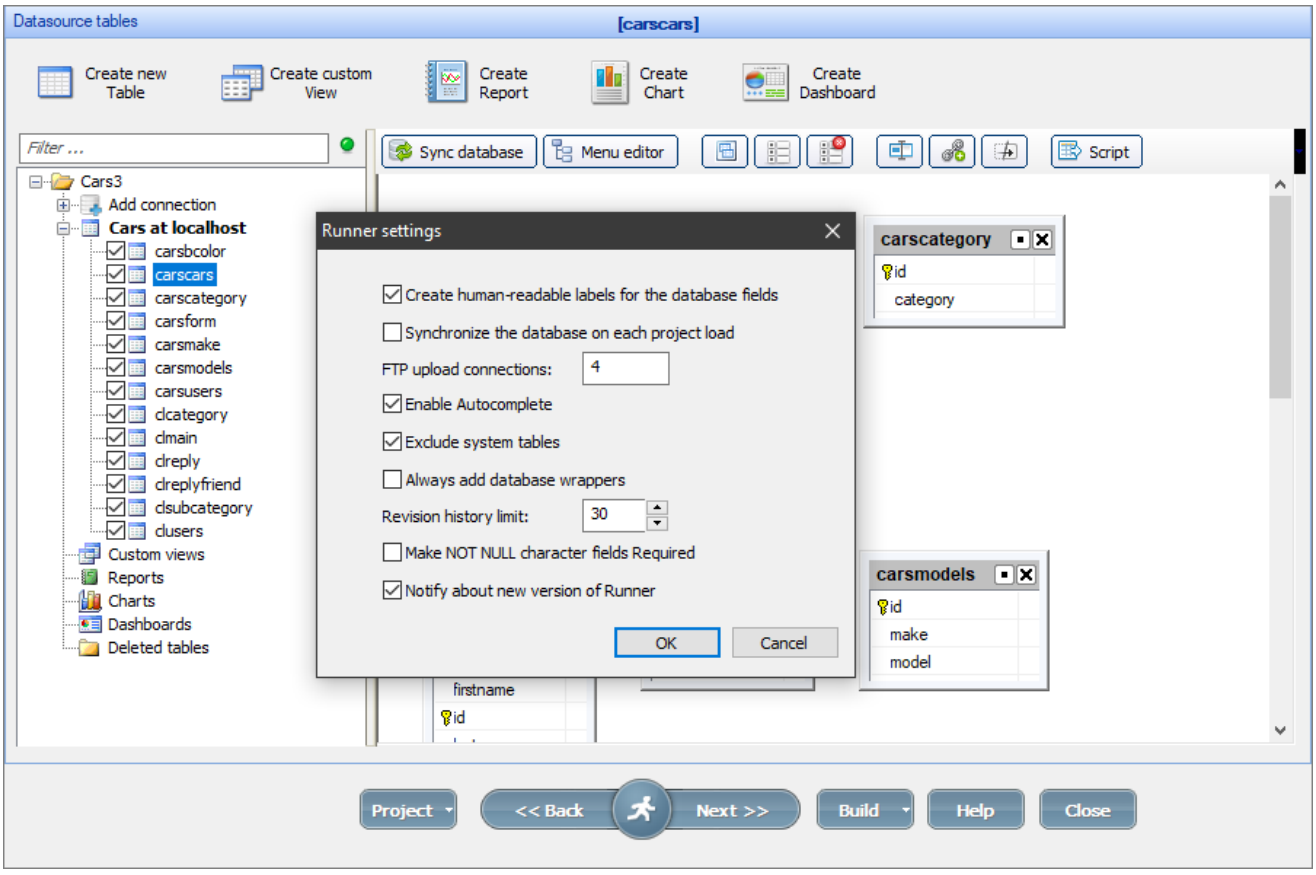
Export your project into a single *.zip* archive with the **Export project** option. This action exports the essentials like the project name, its settings, dashboards/charts/reports, and events.

You can also export the project database structure, as well as extra data like the images, styles, and layouts.



The exported project is saved to the project folder. The file name is generated using a template: *%project_name%yyyymmdd_hhmmss.zip*.

Project settings



- Use the **Create human-readable labels for the database fields** option to convert the field names into a more understandable format. For example, if this option is enabled, the field name *id* is displayed as *Id*, *last_name* as *Last Name*, *FirstName* as *First Name*, etc.
- Use the **Synchronize the database on each project load** option to enable/disable the automatic database synchronization. We recommend using this option for small or local databases. For more information about database synchronization, see Datasource tables: Synchronize database.
- To increase or decrease the upload speed of the project, change the number of **FTP upload connections**.
- The **Enable Autocomplete** option enables Intellisense that provides autocomplete popups and function calltips in the Event Editor.
- The **Exclude system tables** option excludes system tables from the list of tables (i.e., the *Users* table, *Lookup* tables, etc.).
- Use the *Always add database wrappers* option to add wrappers to all names of fields and tables. When this option is disabled, the database wrappers are added only to the field names containing spaces, and service field names.
- After you enabled the **Lock pages modified in Editor automatically** option, all the pages you modify manually become locked, so they can't be modified automatically. You can still modify the locked pages manually.
- The **Revision history limit** option defines the number of page revisions to be saved in the Editor.
- Use the **Make not NULL character fields required** option to make the character fields with the selected **Not NULL** checkbox required in the generated app.
- If you select the **Notify about new version of PHPRunner** option, a message appears in the lower-left corner when the newer version becomes available. For more information, see Updating PHPRunner.

See also:

- Connecting to the database
- Datasource tables
- About templates
- About Editor
- Event editor
- Output directory settings

2.3 Navigation bar

The **Navigation bar** provides an easy way to navigate between PHPRunner screens and manage your project.

The **Navigation bar** is always available at the bottom of the PHPRunner main window.

Button	Description
Project	Open/Save the projects. Click the button to view all options. See Working with projects to learn more.
<< Back	Jump to the previous screen.
	Quick jump to another screen.
Next >>	Jump to the next screen.
Build	Build the project. You can select between two options: **Build and proceed to 'Finished' screen** and **Build and stay on the same page**. Click the arrow to select one of the options.
Help	Open the PHPRunner manual.
Close :	Exit PHPRunner.

See also:

2.4 Templates

2.4.1 About templates

Quick jump

What is a template?

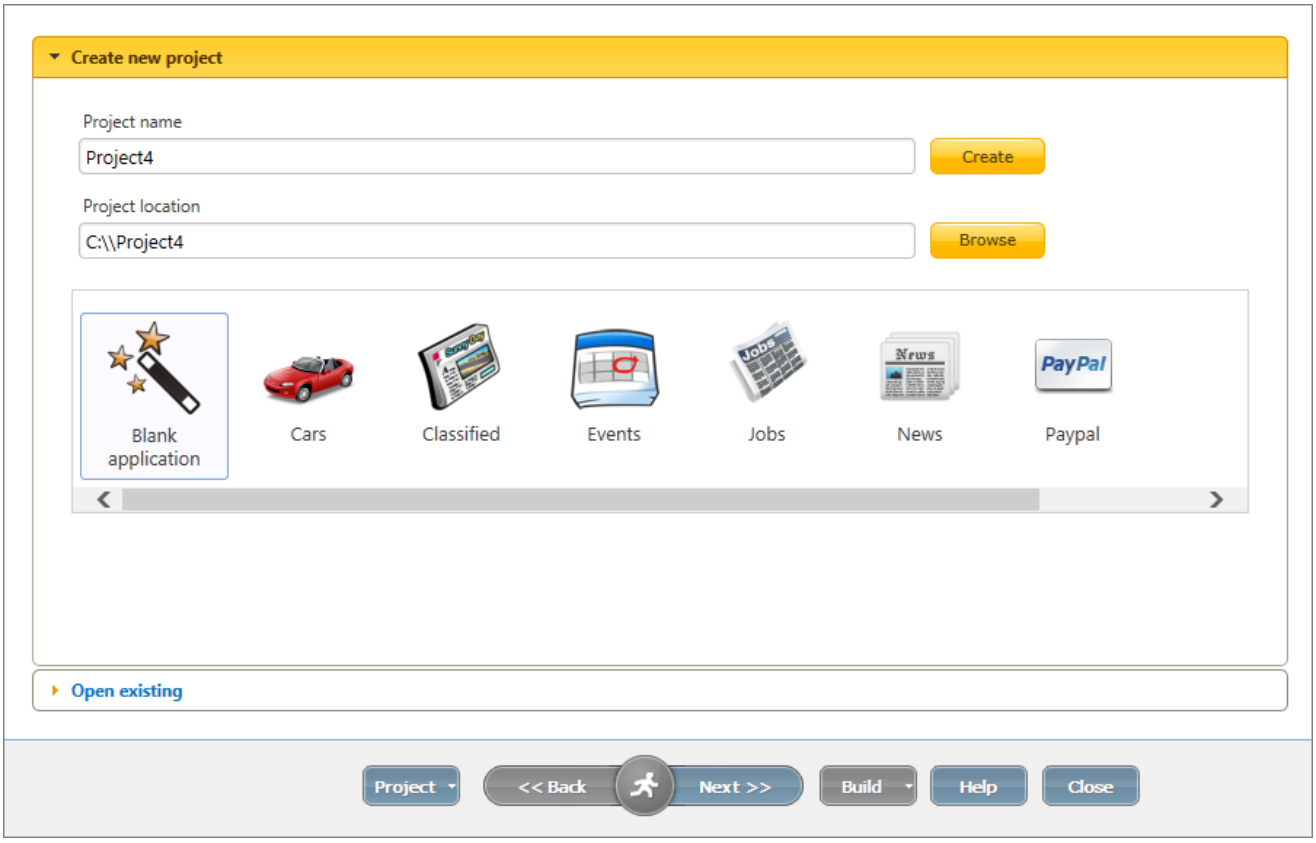
A template is a set of predefined database tables, pages, and a project file that can help you create a themed web site easily and quickly.

Templates are comfortable to work with since you don't need to plan and design the database or table structure. All you have to do is choose a template and build the project.

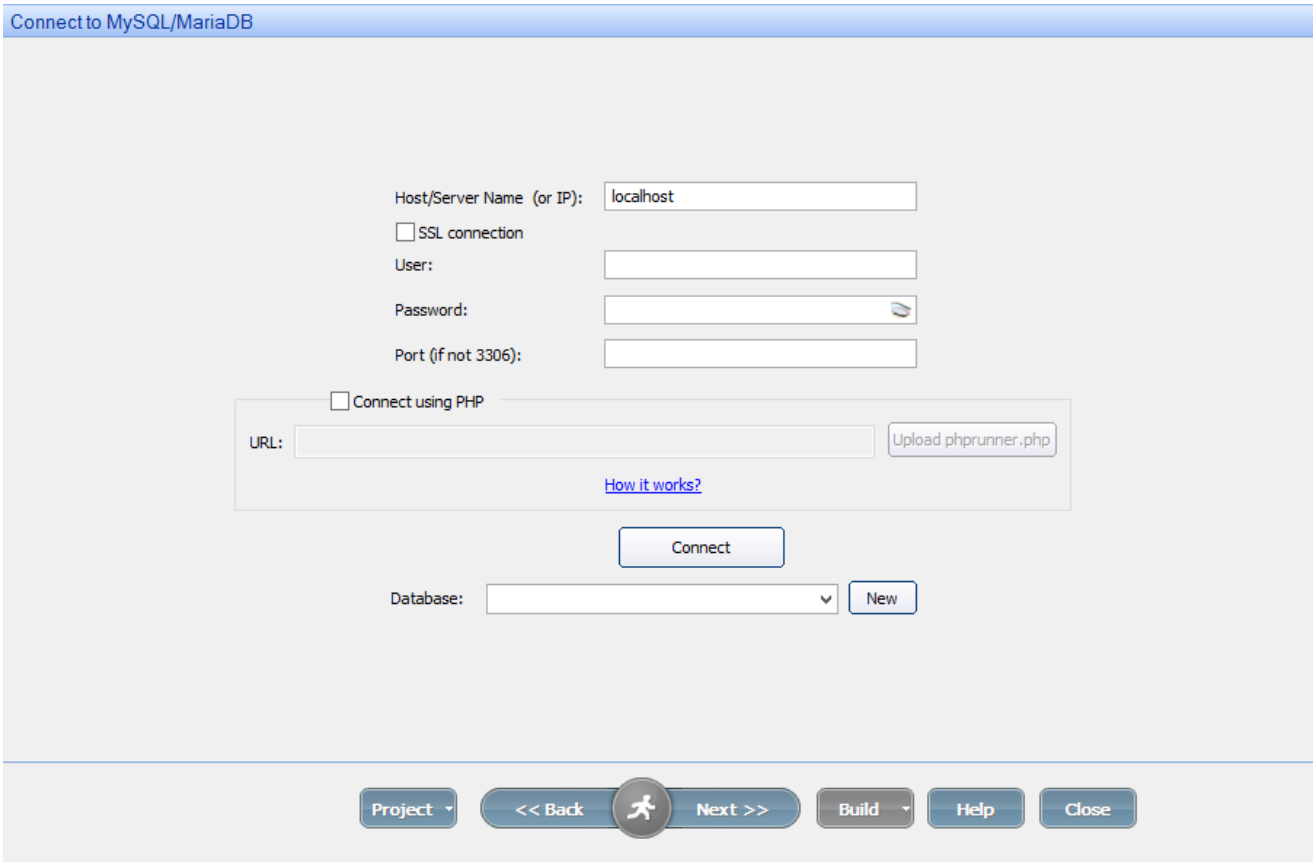
Creating a project using the template

To create a project using the template:

1. Select the template on the PHPRunner welcome page and click **Next**.



2. Connect to the database where you want the template tables to be created on the Database connection screen. PHPRunner can either create a new database named after the template or use an existing database to create the tables.



A list of templates

The following templates are available for free:

- Cars
- Classified
- Events
- Jobs
- Knowledge base
- News
- Paypal
- Real estate
- Sporting

- Vacation

These are the available paid templates:

- MassMailer
- DocManager
- EmailReader
- Shopping Cart
- Calendar
- Members
- Invoice
- Forum

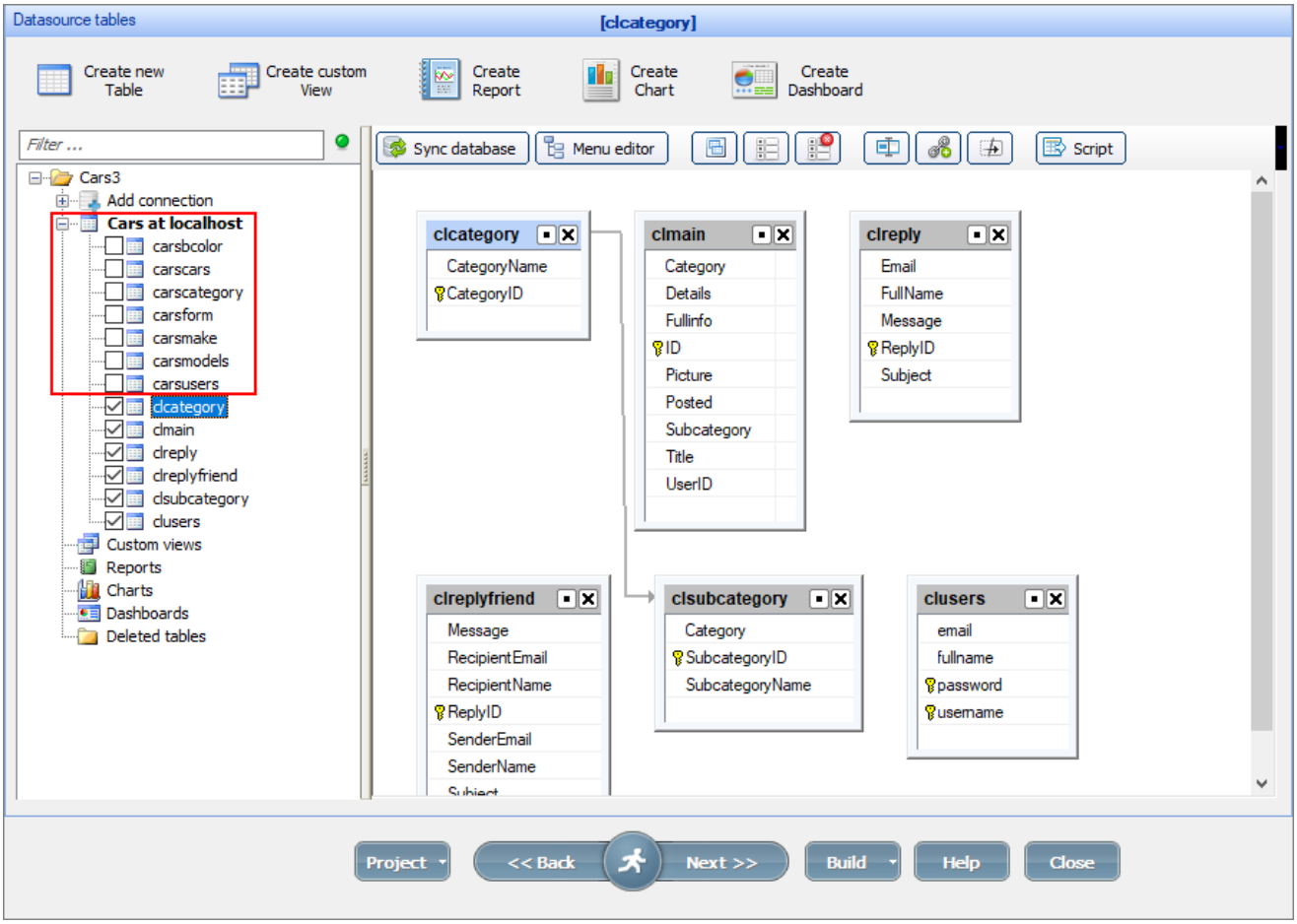
You can create a new project using two templates or add a template to an existing project. For more information, see Working with projects.

Upgrading templates

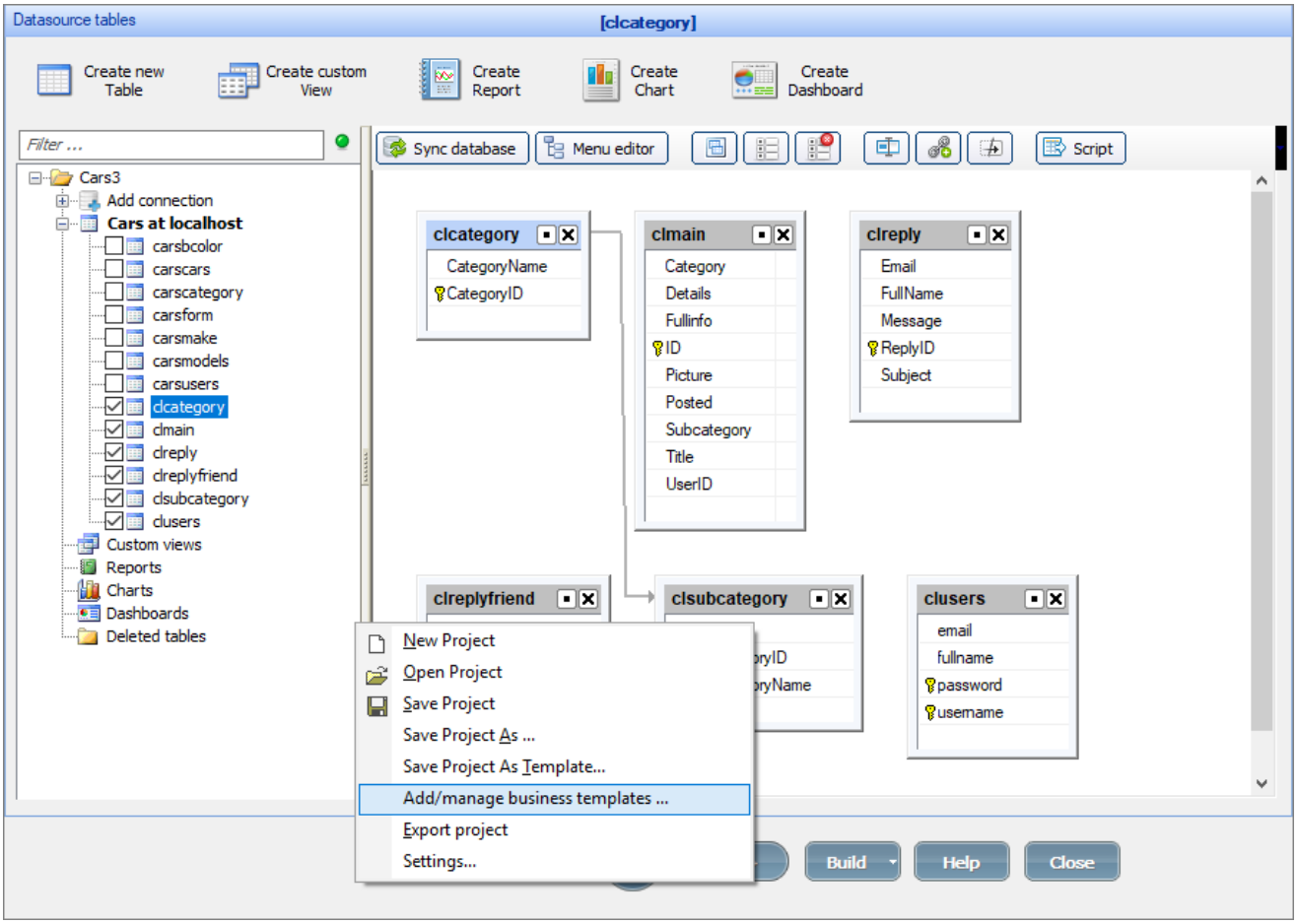
To upgrade a template, remove the template tables from the project on the Datasource tables screen, then add the template to the project again.

> **Note**: to avoid overwriting the existing tables with the template ones, all template tables and files have a unique prefix. For example, the tables of the *'Cars'* project template have the *'cars'* prefix.

1. Remove the tables of that template from the project by deselecting them on the left panel.



2. Add the template back to the project by clicking the **Project** button and selecting **Add/manage business templates**.

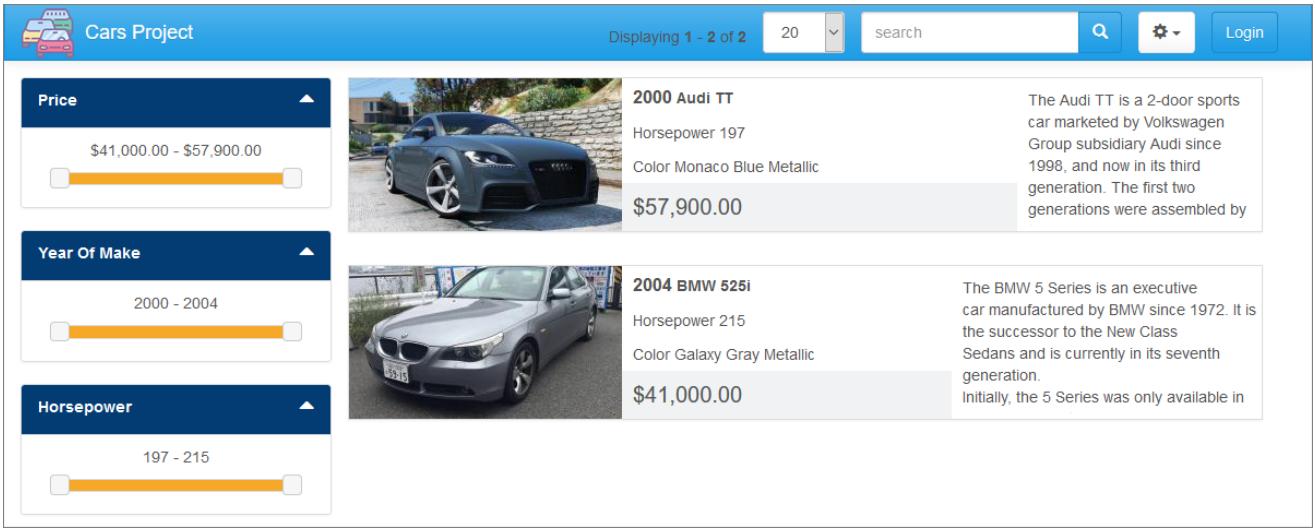


See also:

- Working with projects
- Connecting to the database
- Datasource tables

2.4.2 Cars

The **Cars template** is designed to build an application with new/used car listings.



The administrator (*admin*/*admin*) has full access to all tables. Guest users can search/view car listings and send a quote request to a dealer/car owner.

This template uses the following tables:

- *carscars* - the main table that holds all cars listings;
- *carsmake* - a lookup table with carmakers;
- *carsmodels* - a lookup table with car models;
- *carsusers* - the login table;
- *carsform* - a table to store the quote requests;
- *carsbcolors* - a lookup table with car body colors.

Check this live demo for additional information.

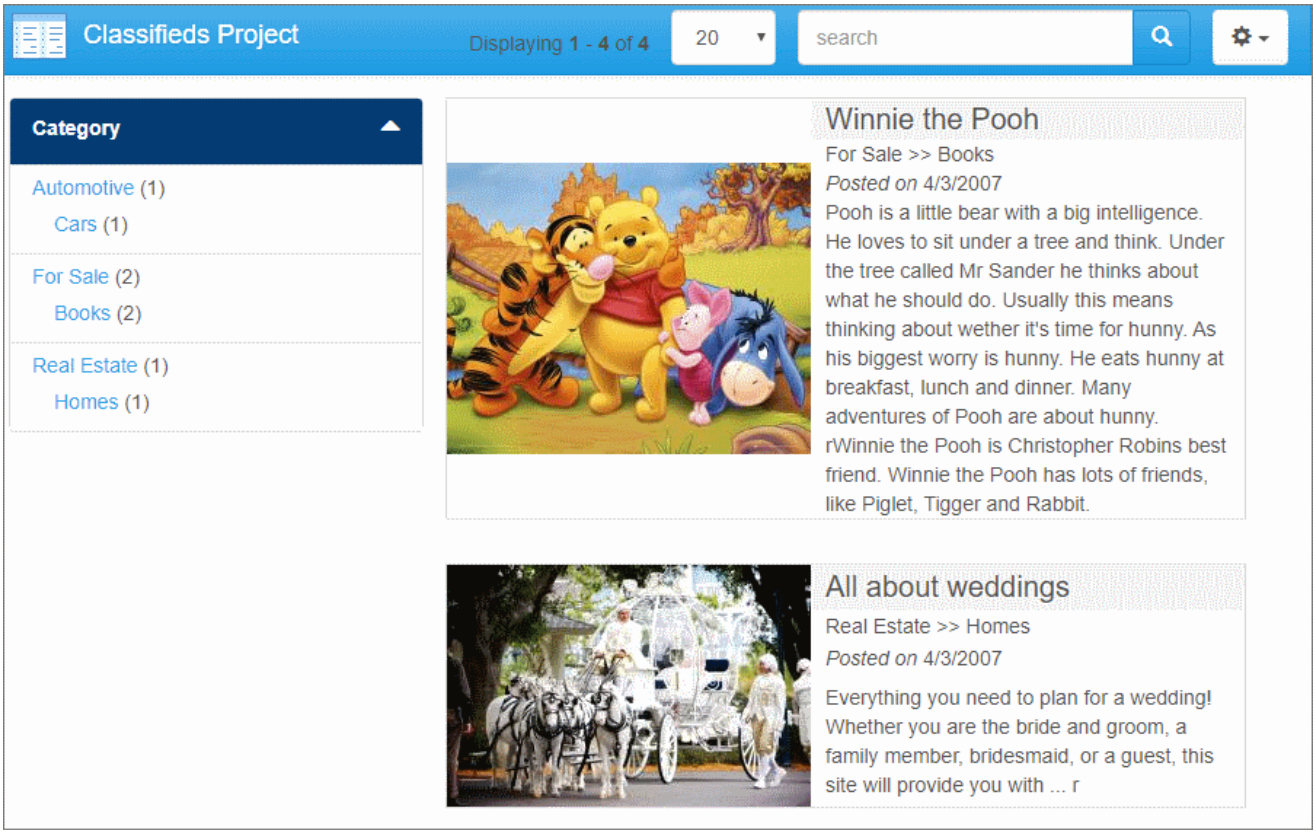
See also:

- About templates
- Classified ads
- Events
- Jobs

- Knowledge base
- News
- Paypal
- Real estate
- Sporting
- Vacation

2.4.3 Classified ads

The **Classified ads template** is designed to build a classified ads website.



The administrator (*admin/admin*) has full access to all tables. Registered users can add and edit their ads. Guest users can search/view ads, contact the ad author, and share the ads.

Check this live demo for additional information.

This template uses the following tables:

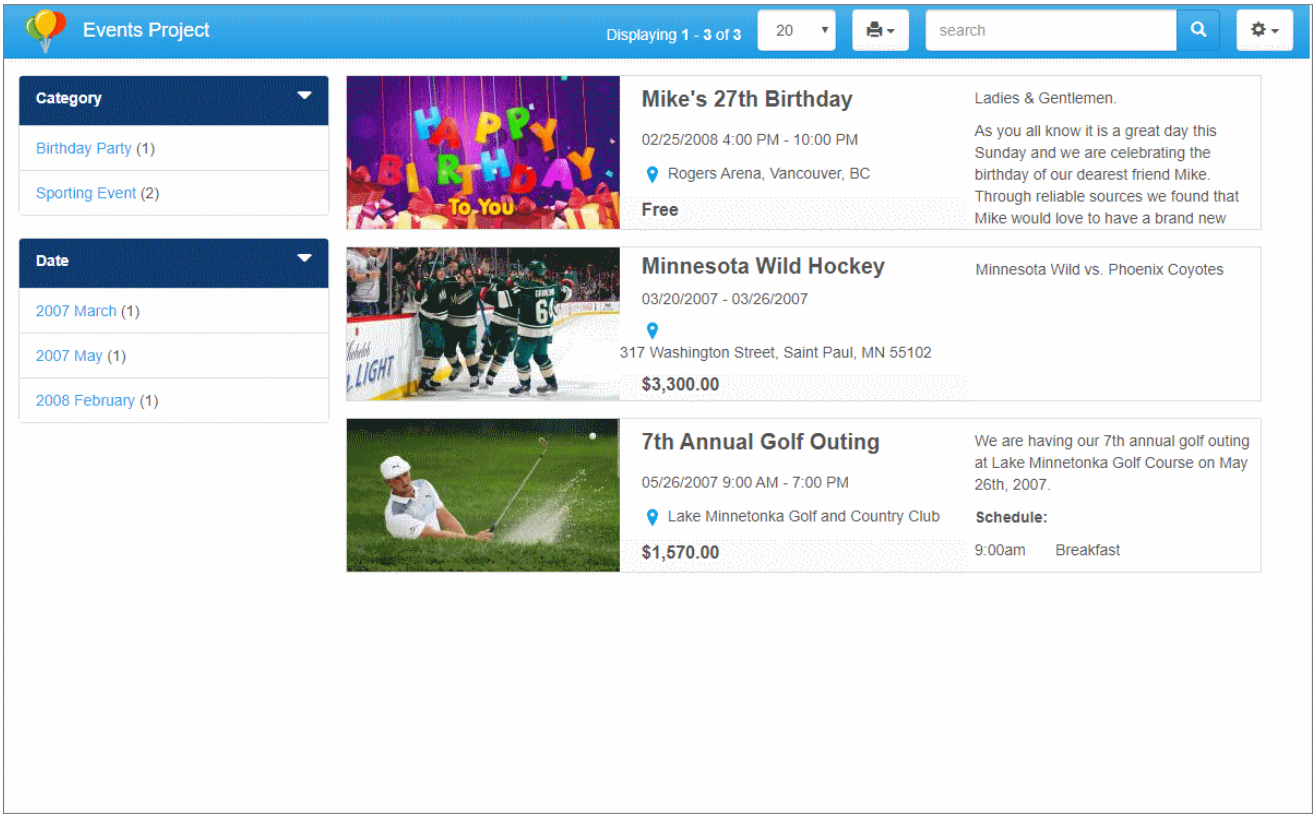
- *clmain* - the main table that holds all ad listings;
- *clcategory* - a lookup table with a list of categories;
- *clsubcategory* - a lookup table with a list of subcategories;
- *clusers* - the login table;
- *clreply* - a table to store "contact the author" requests;
- *clreplyfriend* - a table to store "tell a friend" emails.

See also:

- About templates
- Cars
- Events
- Jobs
- Knowledge base
- News
- Paypal
- Real estate
- Sporting
- Vacation

2.4.4 Events

The **Events template** is designed to build an application with event listings.



The administrator (*admin/admin*) has full access to all tables. Guest users can search/view events and share them.

Check this live demo for additional information.

This template uses the following tables:

- *evevents* - the main table that holds the events;
- *evcategories* - a lookup table with a list of categories;
- *evusers* - the login table;
- *evtellafriend* - a table to store "tell a friend" emails.

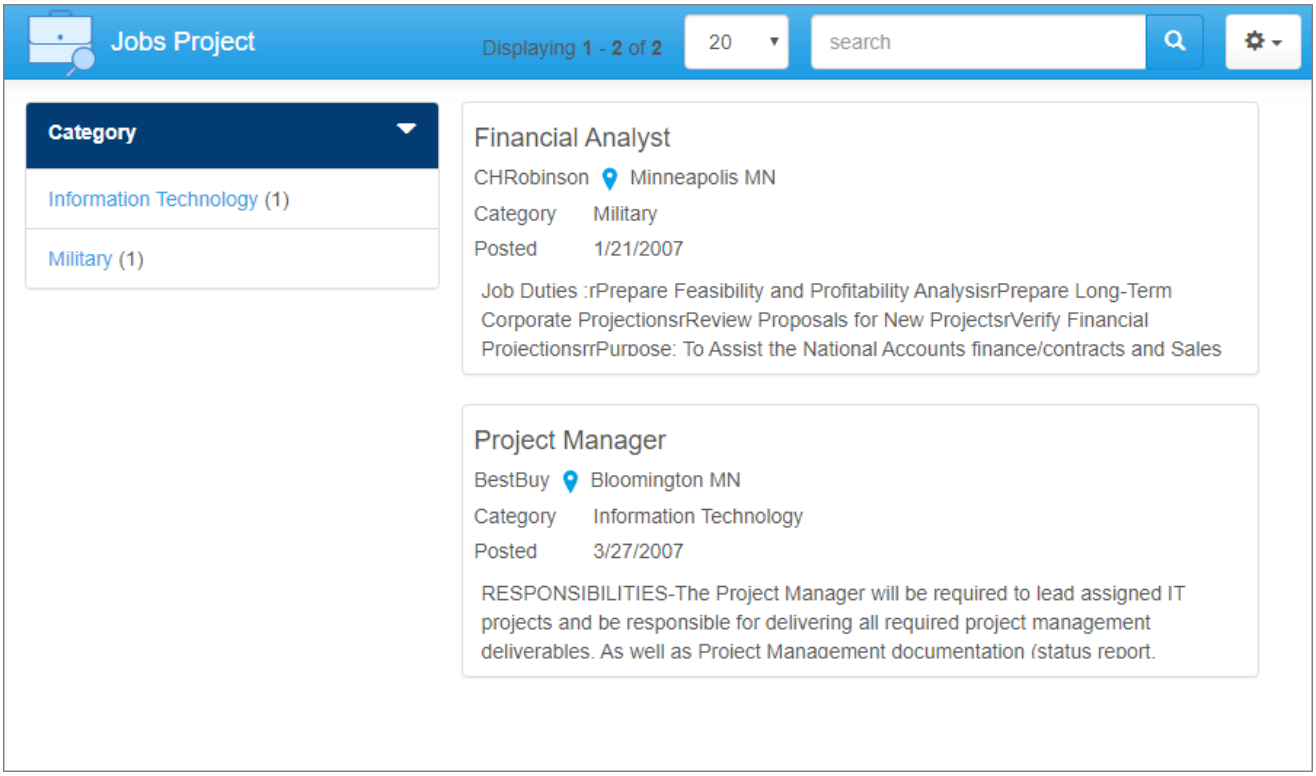
See also:

- About templates

- Cars
- Classified ads
- Jobs
- Knowledge base
- News
- Paypal
- Real estate
- Sporting
- Vacation

2.4.5 Jobs

The **Jobs template** is designed to build a job listings website.



The administrator (*admin/admin*) has full access to all tables. Guest users can search/view jobs and contact the company to learn about a specific open position.

Check this live demo for additional information.

This template uses the following tables:

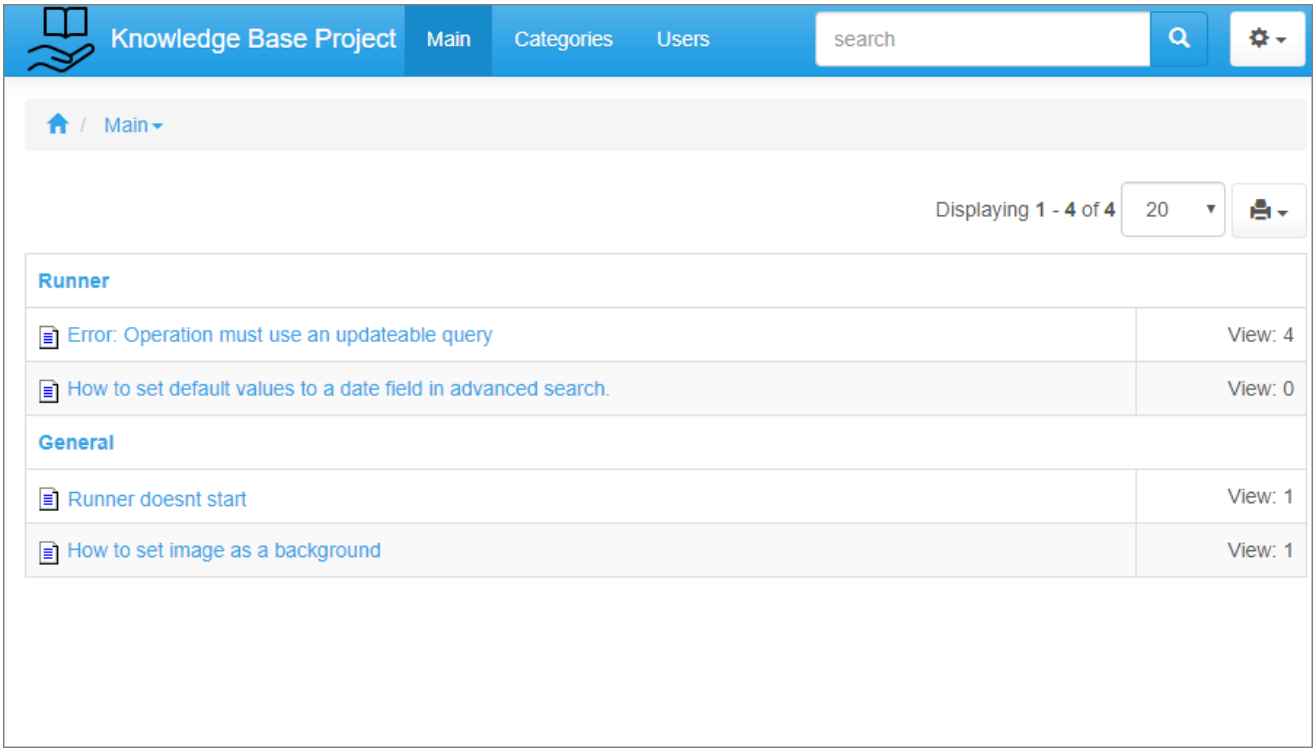
- *jobsjobs* - the main table that holds all jobs listings;
- *jobsjobtype* - a lookup table with a list of job types (categories);
- *jobsusers* - the login table;
- *jobsstate* - a lookup table with the list of US states;

See also:

- About templates
- Cars
- Classified ads
- Events
- Knowledge base
- News
- Paypal
- Real estate
- Sporting
- Vacation

2.4.6 Knowledge base

The **Knowledge base template** is designed to build a knowledge base website.



The administrator (*admin/admin*) has full access to all tables. Registered users can add comments to the knowledge base articles and edit their comments. Guest users can search/view knowledge base articles.

Check this live demo for additional information.

This template uses the following tables:

- *kbarticles* - the main table that holds knowledge base articles;
- *kbcategories* - a table for the knowledge base categories;
- *kbusers* - the login table;
- *kbcoments* - a table for the article comments.

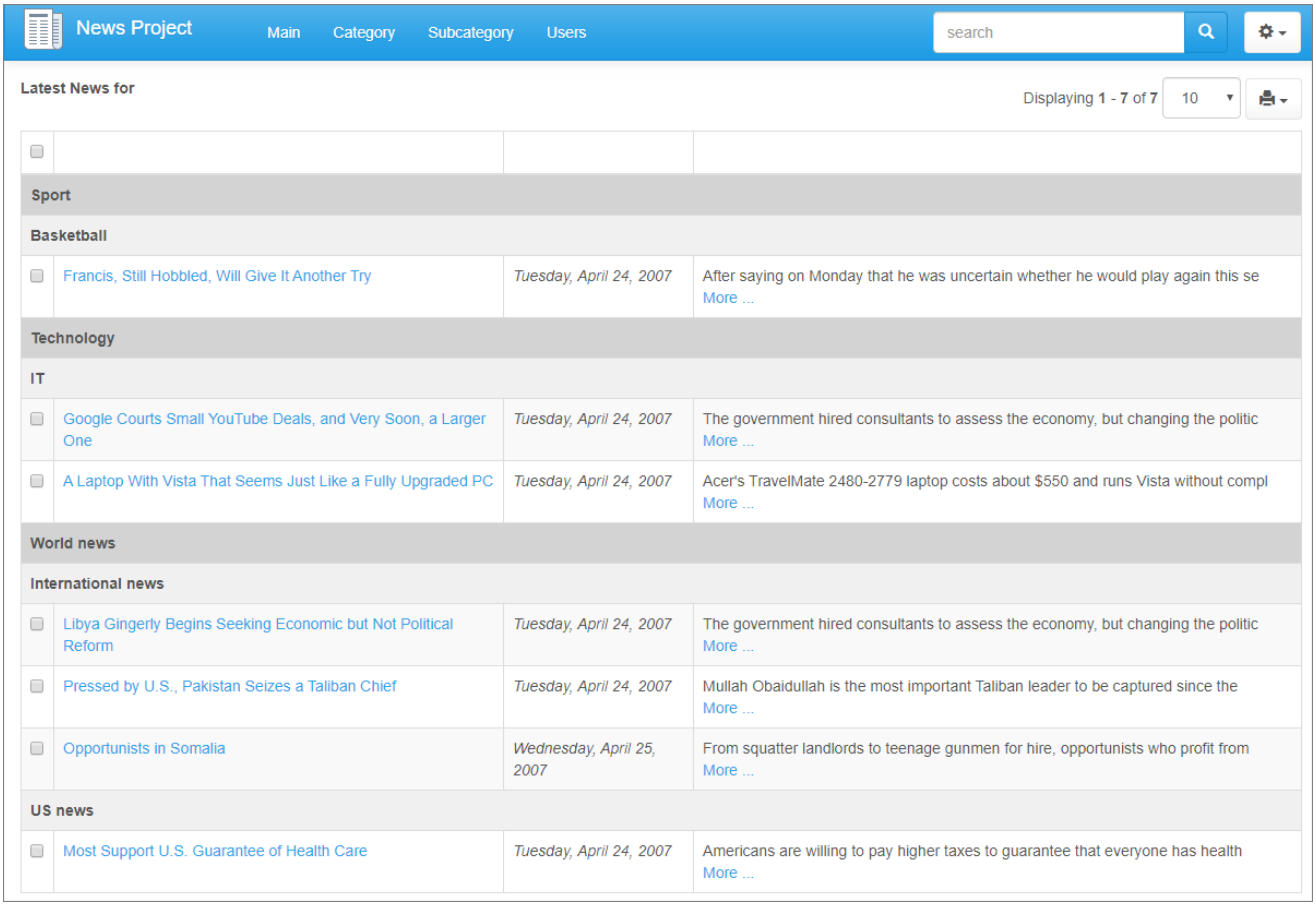
See also:

- About templates
- Cars

- Classified ads
- Events
- Jobs
- News
- Paypal
- Real estate
- Sporting
- Vacation

2.4.7 News

The **News template** is designed to build a news website.



The administrator (*admin/admin*) has full access to all tables. Guest users can search/view the news and share them.

Check this live demo for additional information.

This template uses the following tables:

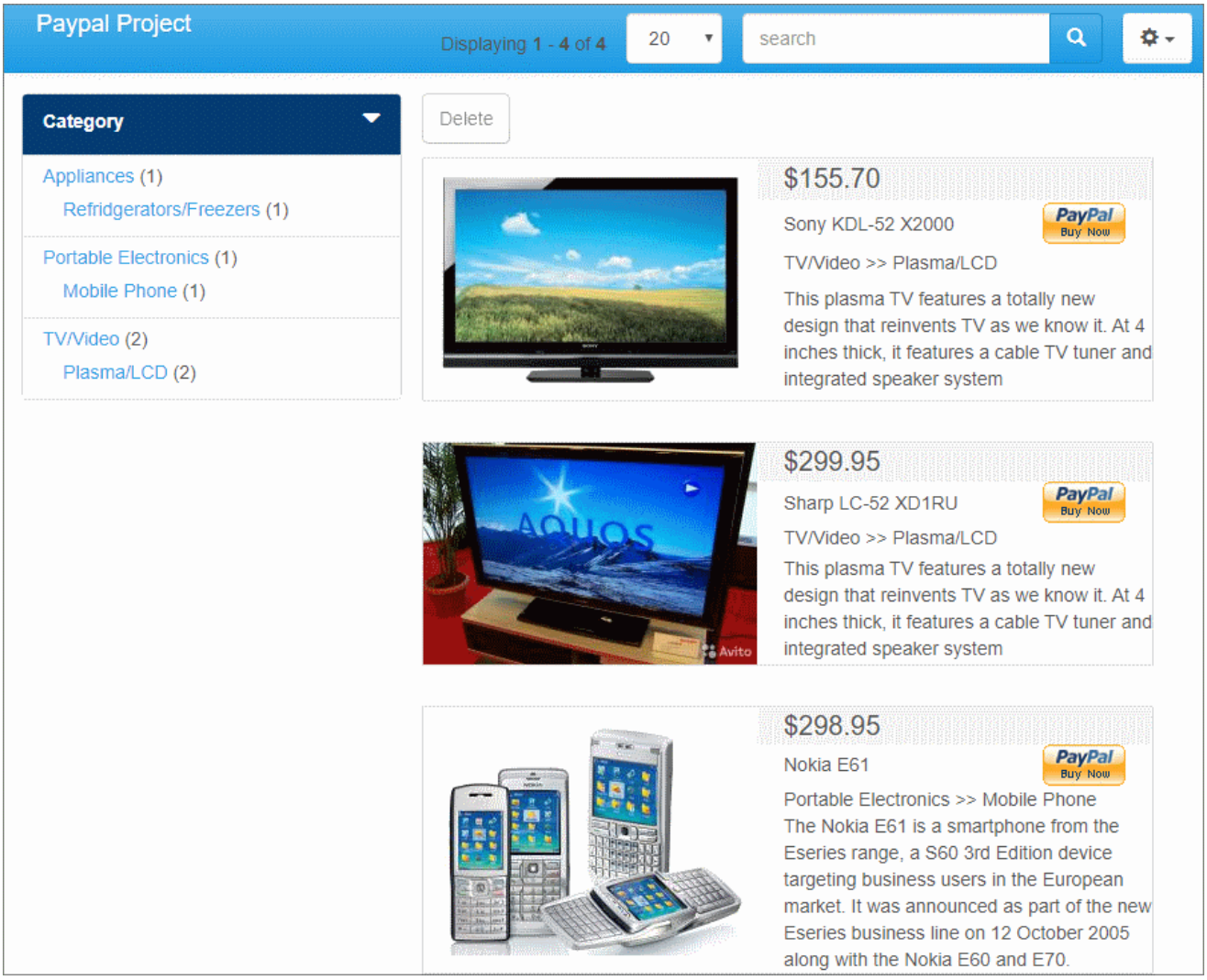
- *newsmain* - the main table that holds news articles;
- *newscategory* - a table for news categories;
- *newssubcategory* -a table for news subcategories;
- *newsusers* - the login table;
- *newsreplyfriend* - a table to store "tell a friend" emails.

See also:

- About templates
- Cars
- Classified ads
- Events
- Knowledge base
- Jobs
- Paypal
- Real estate
- Sporting
- Vacation

2.4.8 PayPal

The **PayPal template** is designed to build a simple e-commerce website with an integrated PayPal shopping cart.



The administrator (*admin/admin*) has full access to all tables. Guest users can search/view items, add the items to the shopping cart, and checkout using PayPal.

Check this live demo for additional information.

This template uses the following tables:

- *ppmain* - the main table that holds item descriptions, price, images, etc.;
- *ppcategory* - a table for item categories;

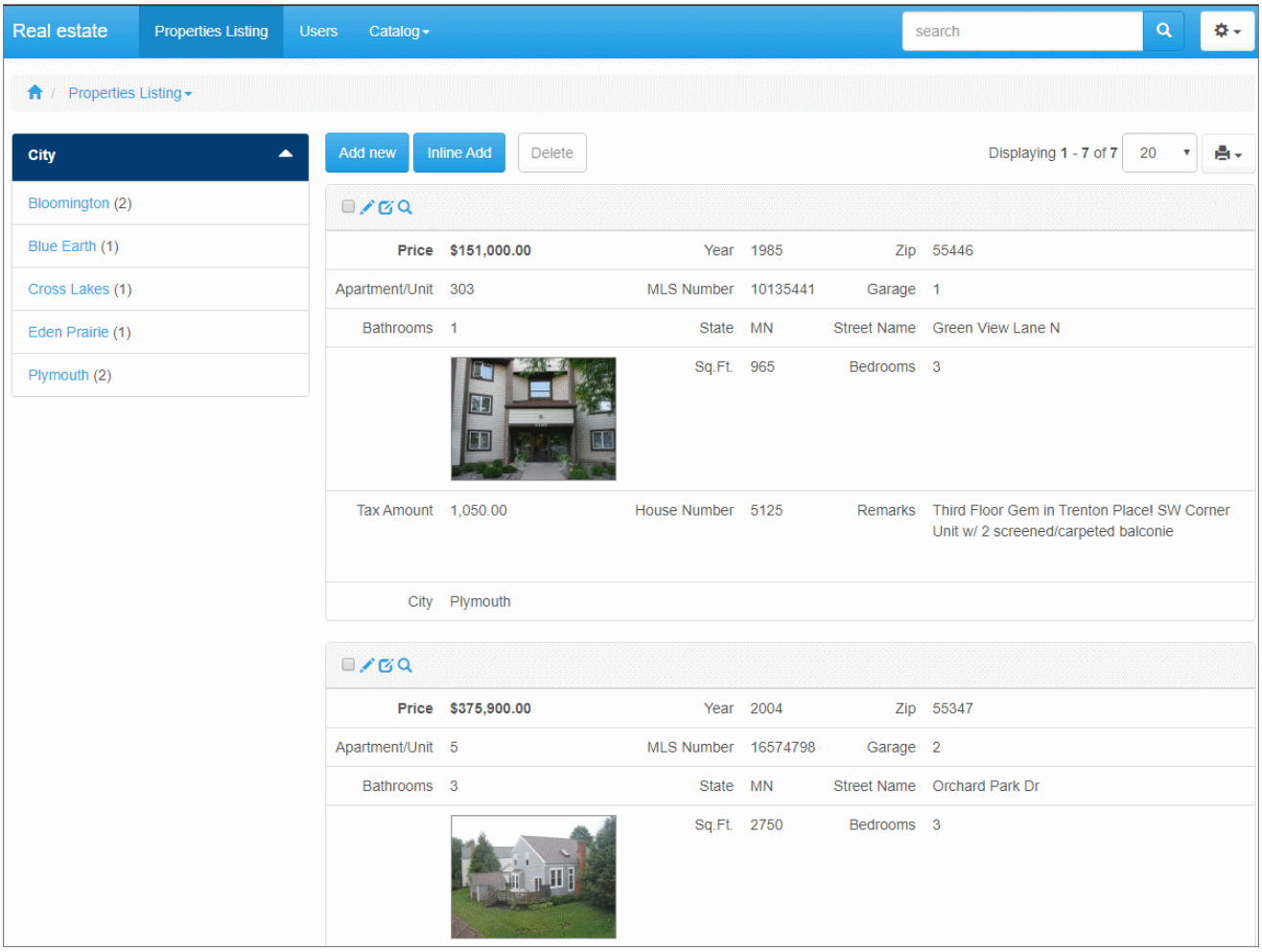
- *ppsubcategory* - a table for item subcategories;
- *ppusers* - the login table;
- *pppaypal_info* - a table for PayPal account info.

See also:

- About templates
- Cars
- Classified ads
- Events
- Knowledge base
- Jobs
- News
- Real estate
- Sporting
- Vacation

2.4.9 Real estate

The **Real estate template** is designed to build a website for real estate listings.



The administrator (*admin/admin*) has full access to all tables. Guest users can search/view property listings.

Check this live demo for additional information.

This template uses the following tables:

- *retblresults* - the main table that holds the property description, price, images, etc.;
- *reusers* - the login table;
- *retblbathrooms* - a lookup table for the number of bathrooms;
- *retblbedrooms* - a lookup table for the number of bedrooms;

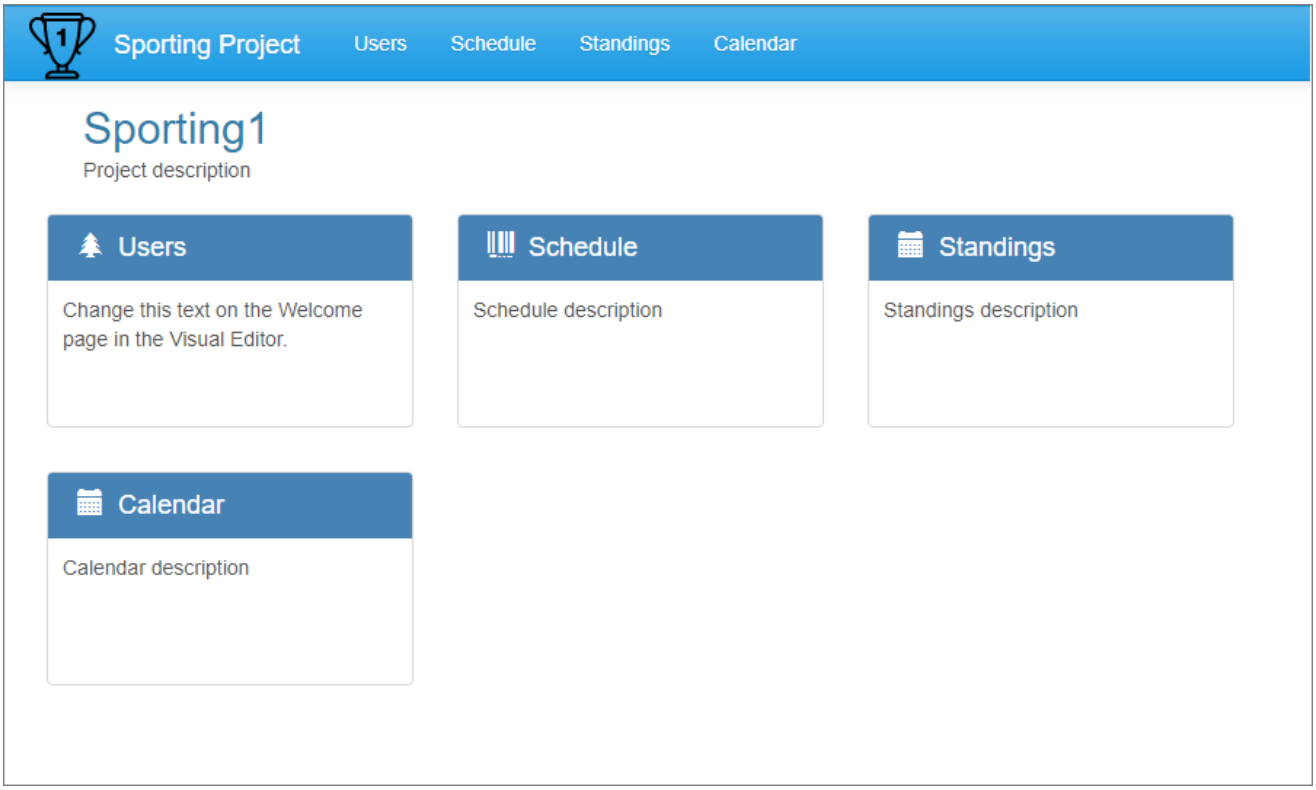
- *retblcooling* - a lookup table for the AC type;
- *retblgarage* - a lookup table for the garage type;
- *retblheating* - a lookup table for the heating type;
- *retblstyle* - a lookup table for the property style;
- *retbltype* - a lookup table for the property type.

See also:

- About templates
- Cars
- Classified ads
- Events
- Knowledge base
- Jobs
- News
- PayPal
- Sporting
- Vacation

2.4.10 Sporting

The **Sporting template** is designed to build the website with sport event listings.



The administrator (*admin/admin*) has full access to all tables. The admin can enter the schedule, standings, and game results. Guest users can search/view the team standings and schedule.

Check this live demo for additional information.

This template uses the following tables:

- *spschedule* - a table for the games schedule;
- *spstandings* - a table for team standings;
- *spusers* - the login table.

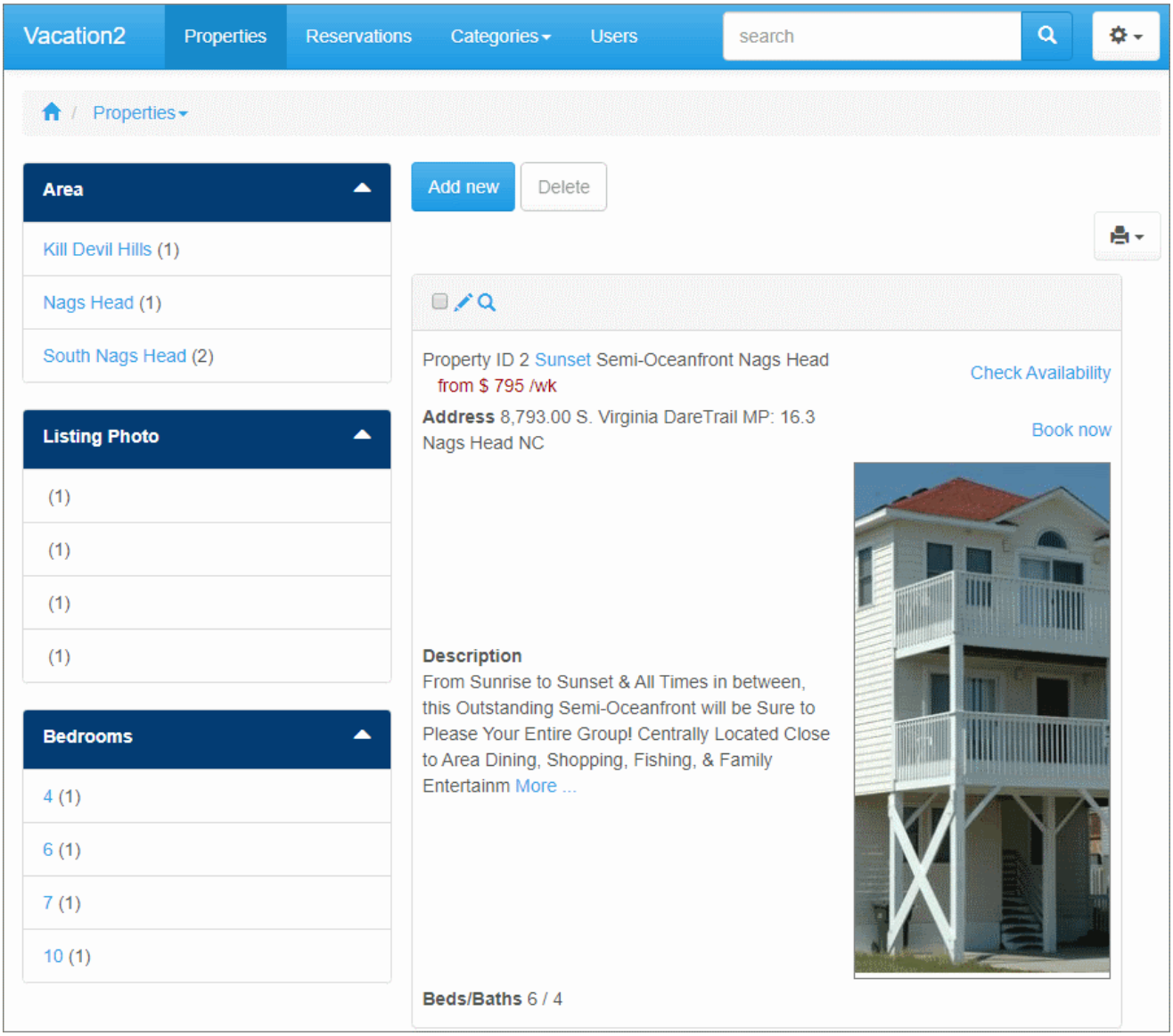
See also:

- About templates
- Cars

- Classified ads
- Events
- Knowledge base
- Jobs
- News
- PayPal
- Real estate
- Vacation

2.4.11 Vacation houses

The **Vacation template** is designed to build a website with vacation house listings.



The administrator (*admin/admin*) has full access to all tables. Admin can add/edit the properties, make/cancel reservations, etc. Guest users can search/view properties, listings, and availability.

Check this live demo for additional information.

This template uses the following tables:

- *vacproperties* - the main table that holds the vacation house listings;

- *vacreservations* - a table for house reservations;
- *vacusers* - the login table;
- *vacbedrooms* - a lookup table for the number of bedrooms;
- *vacbathrooms* - a lookup table for the number of bathrooms;
- *vacareas* - a lookup table for the area type;
- *vaclocations* - a lookup table for the location type;
- *vacpropertytype* - a lookup table for the property type;
- *vacstates* - a lookup table for the states;
- *vacstatus* - a lookup table for the reservation status.

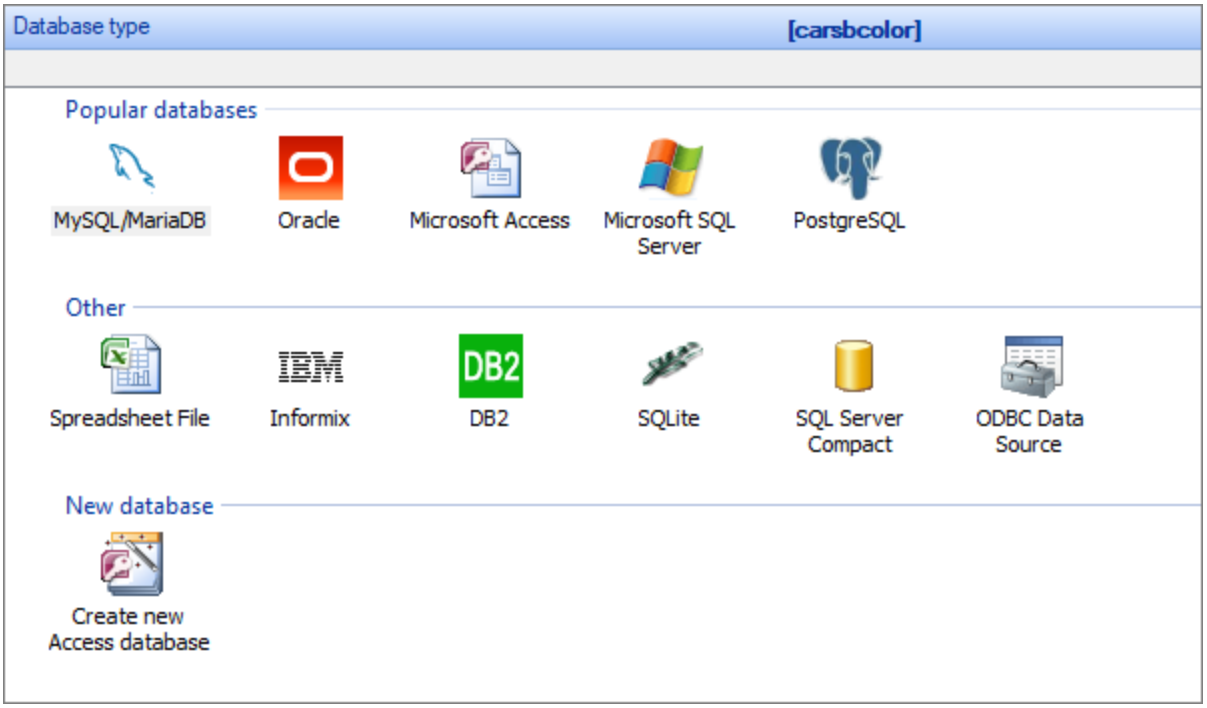
See also:

- About templates
- Cars
- Classified ads
- Events
- Knowledge base
- Jobs
- News
- PayPal
- Real estate
- Sporting

2.5 Connecting to the database

Quick jump

PHPRunner supports the following databases: MySQL, Oracle, Microsoft SQL Server, MS Access, PostgreSQL, Spreadsheet File, Informix, DB2, SQLite, SQL Server Compact, and any ODBC-enabled databases.



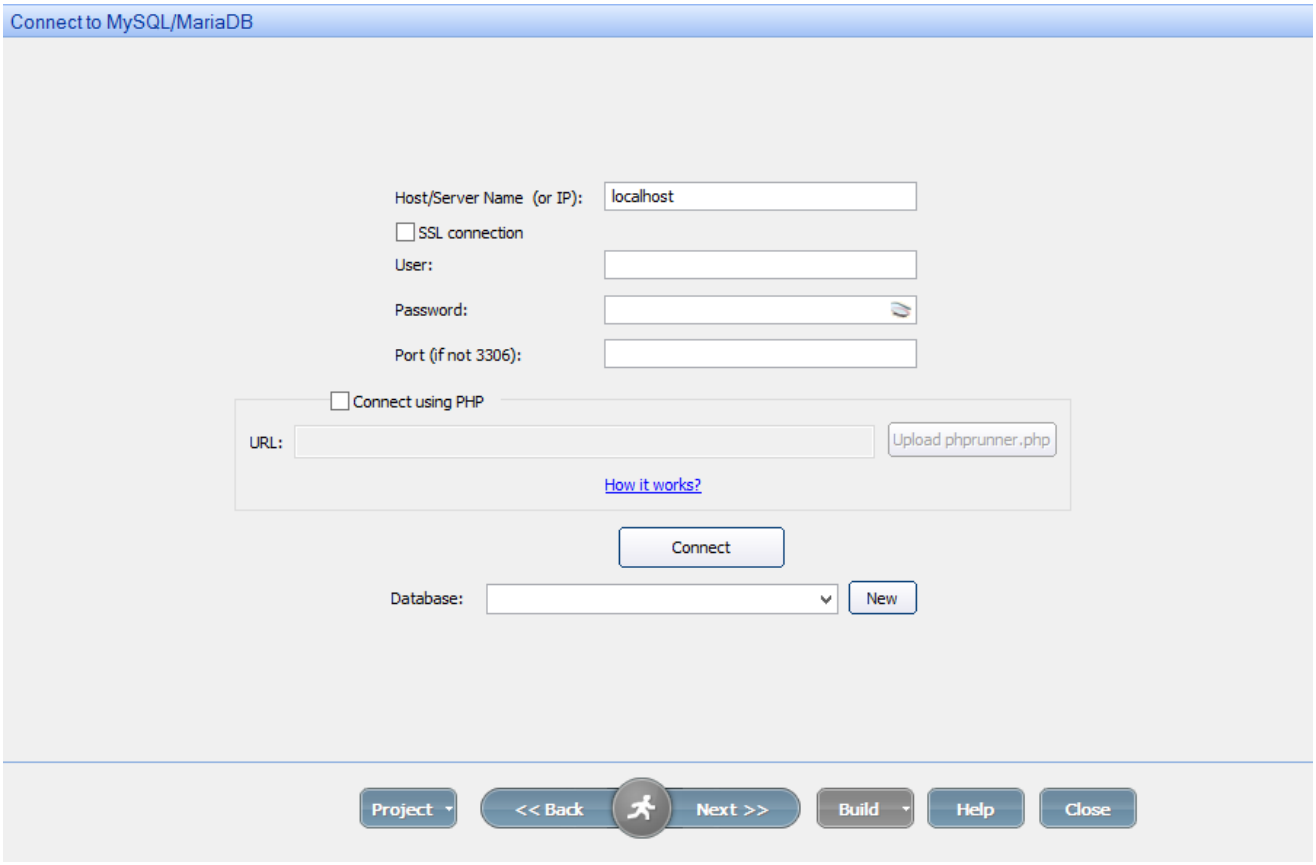
Select the database type and press **Next>>**. Depending on the selected database type, one of the database-specific dialog boxes shown below appears on the screen.

> **Note**: use **Recent connections** to connect to the previously opened databases quickly.

After successfully connecting to the database, you can select the datasource tables.

Connecting to MySQL/MariaDB

Type in the **Host/Server Name** (usually - localhost), username, password, and click **Connect**. See How to install a local web server (XAMPP). to learn more about installing a local web server.

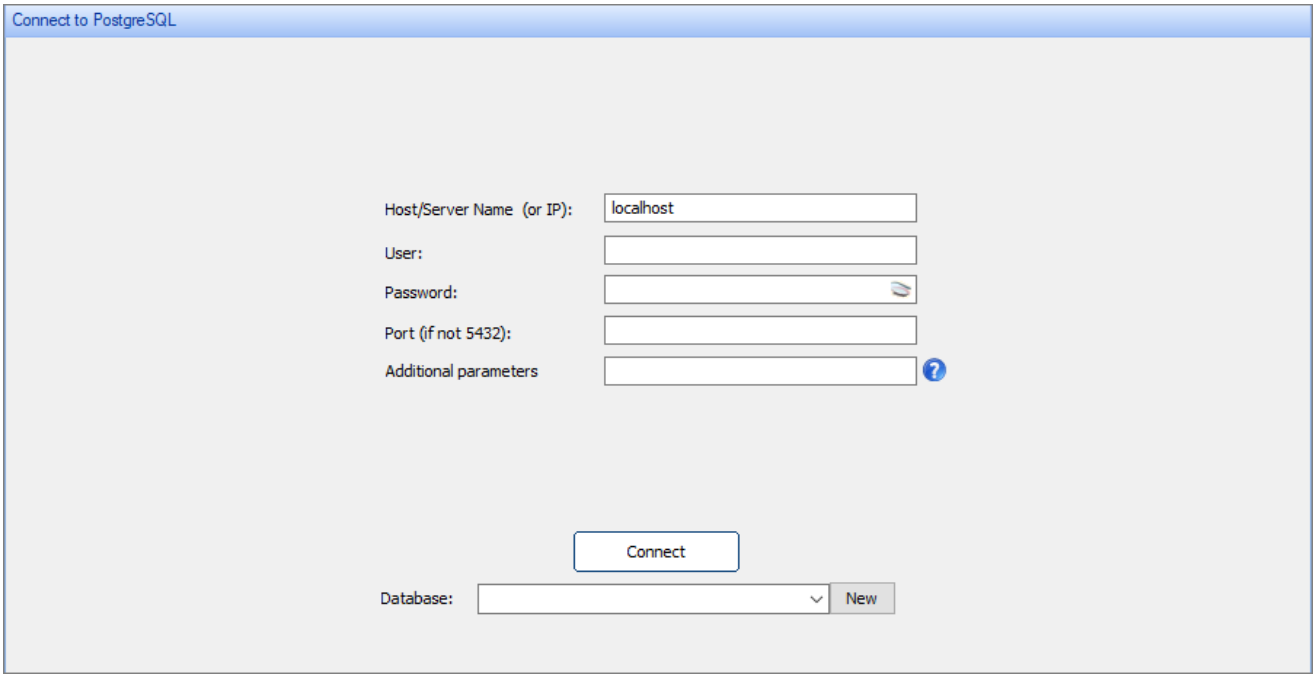


Select the database and click **Next >>**.

If your MySQL server doesn't allow remote connection, you can connect via PHP.

Connecting to PostgreSQL

Type in the *Host/Server Name* (usually - localhost), *user* name, *password*, set the additional parameters if required (use space as the delimiter, e.g., *key1=value1 key2=value2*), and click **Connect**.



Select the *Database* and click **Next >>**.

If your PostgreSQL server doesn't allow remote connection, you can connect via PHP.

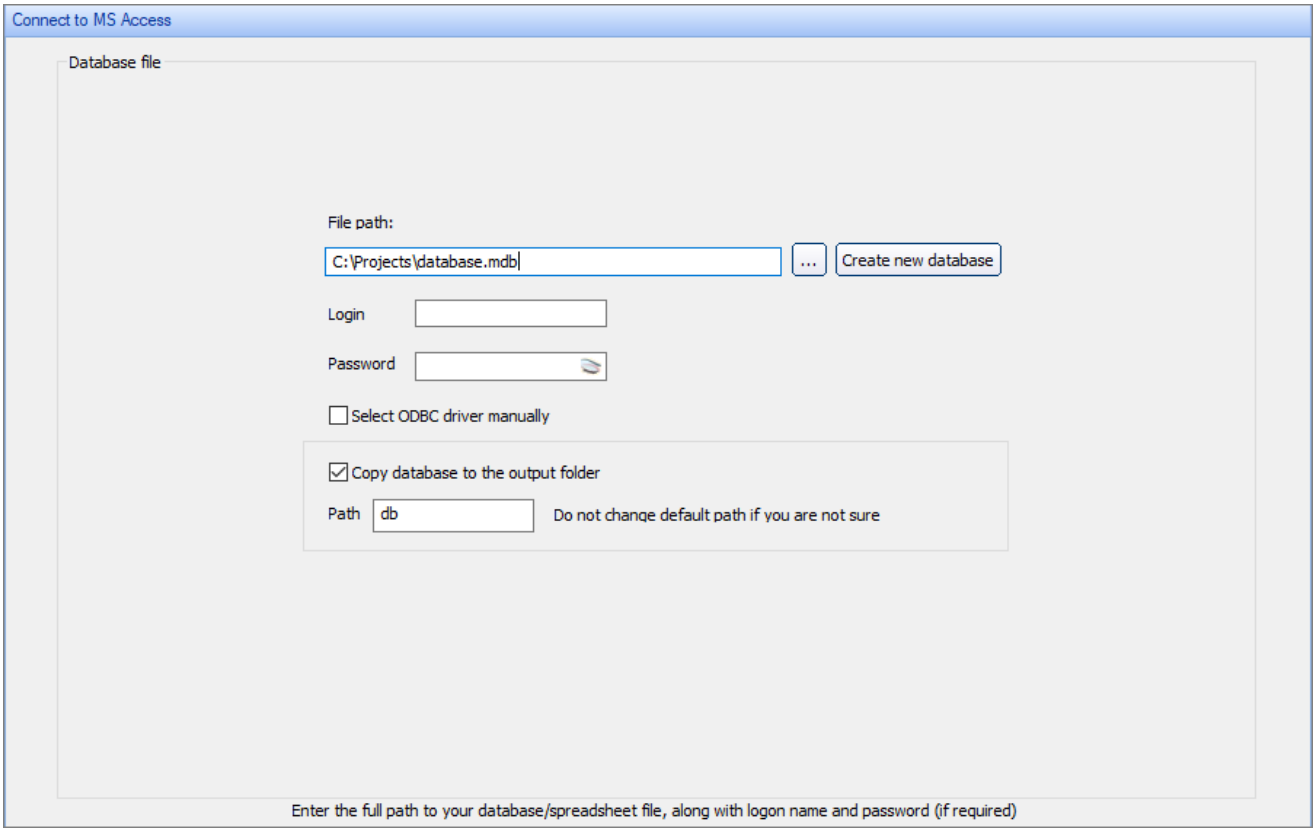
Connecting to Oracle, Microsoft SQL Server, Informix, DB2

Type in the Host/Server name, Database Name, username, password, and click Connect.

Connecting to MS Access, spreadsheet file, SQLite

Choose the **Spreadsheet File** option to select an Excel, FoxPro, DBase, Paradox, or text database file.

Define the file path to your database/spreadsheet file, enter the *login* and *password* if required. Click **Next >>**.



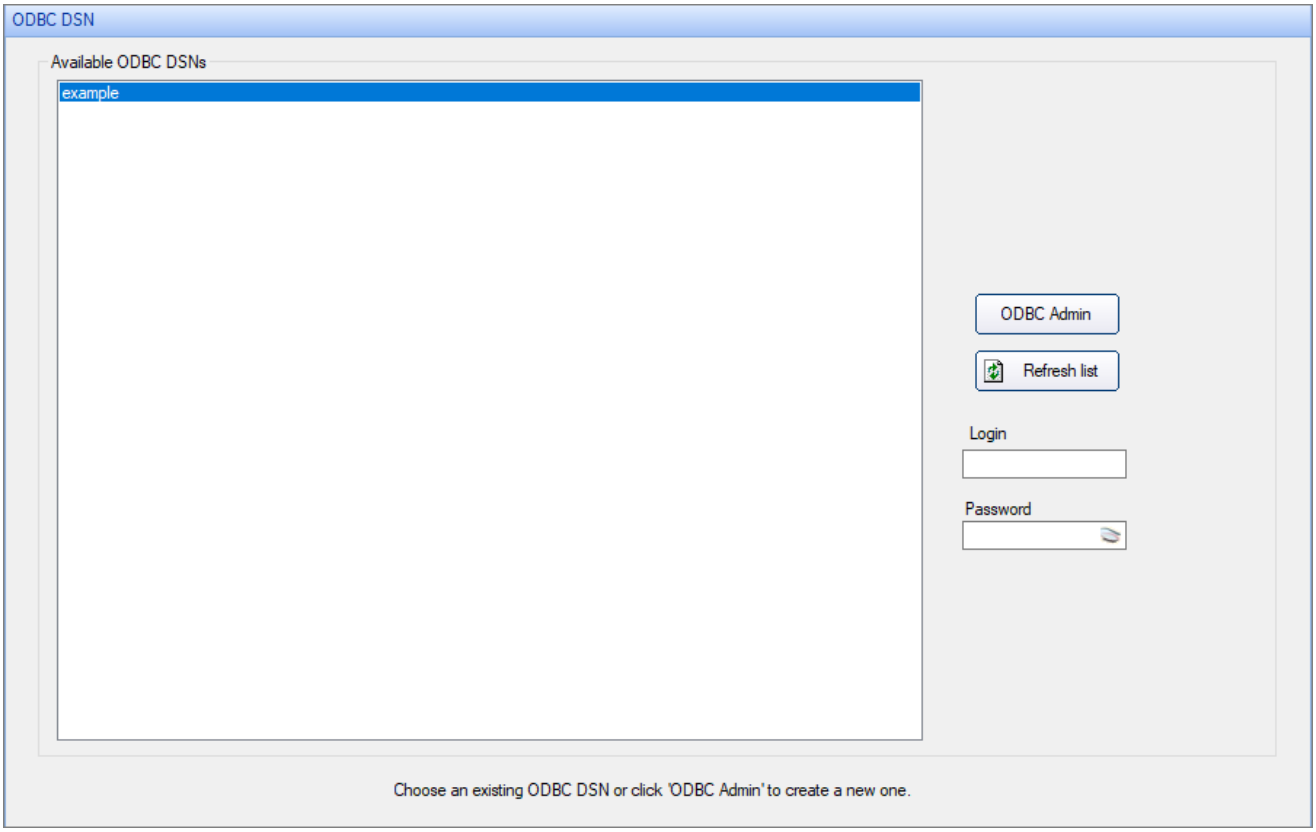
PHPRunner tries to find which ODBC driver to use to connect to the selected database file. Select the **Select ODBC driver manually** checkbox if you'd like to select the ODBC driver manually.

To learn more about connecting to MS Access, see Connecting to MS Access database.

Connecting to ODBC Data Source

If you already have an ODBC Data Source Name (DSN) associated with your database, select **ODBC DSN** from the list box. Enter the *Login* and *Password* if required. Press **Next>>** to continue.

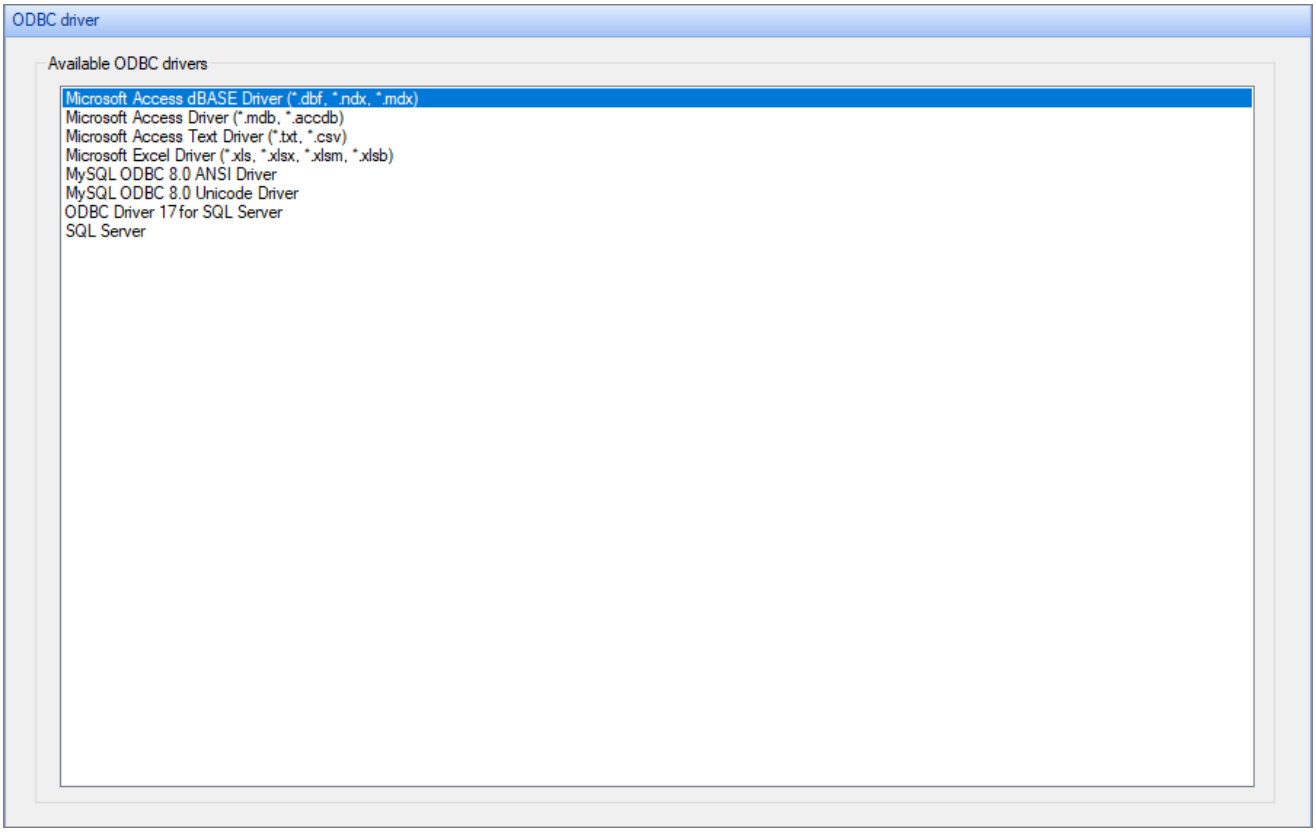
To create a new DSN, press the **ODBC Admin** button. Add a new DSN in the popup, then click **Refresh list**.



Connecting through ODBC driver dialog

Choose this option to connect directly through the ODBC driver dialog.

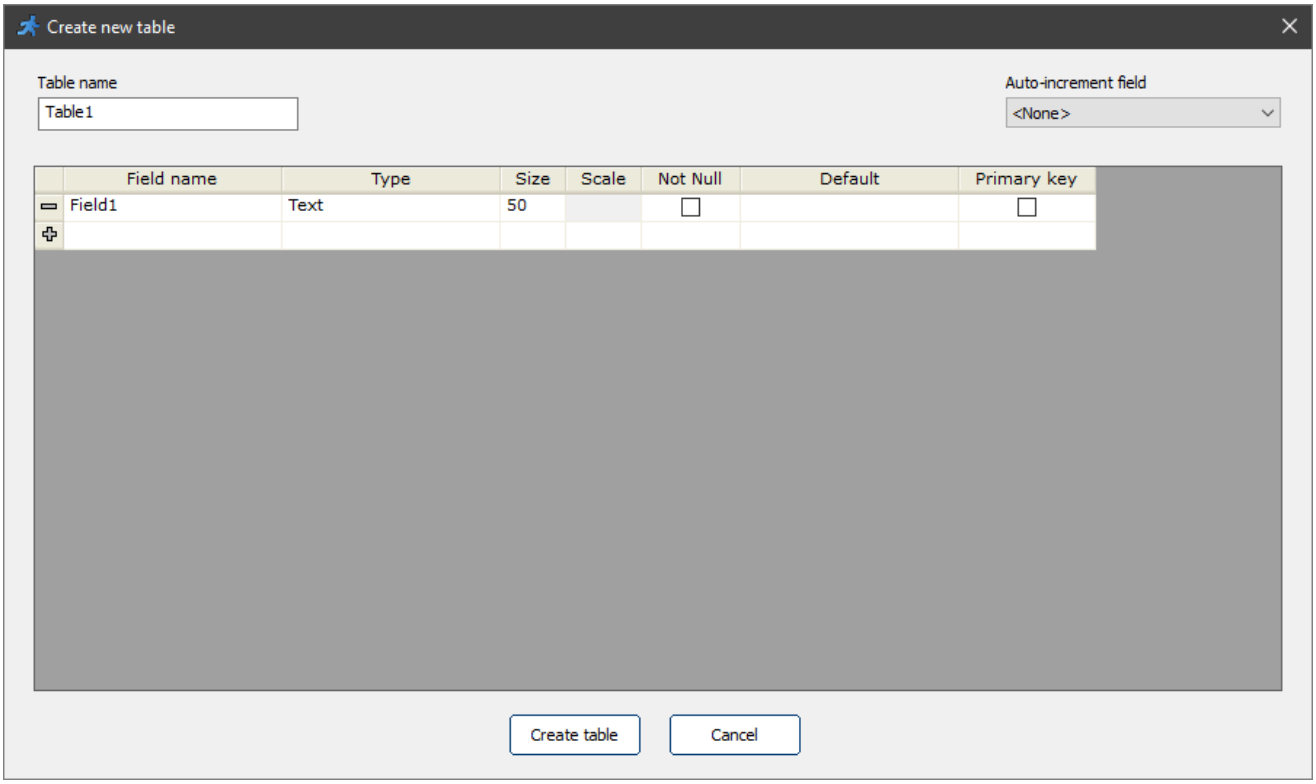
Select **ODBC Driver** from the list box, pick the necessary driver from the list of available drivers, and click **Next>>**.



Create a new MS Access database

If you do not have a database yet, choose this option and click **Next>>**. PHPRunner creates a new empty MS Access database with this option.

Use the **Create new table window** to create new tables in your database.



Type in the name, type, size, scale (applies to the DECIMAL field type in SQL Server, Oracle and MySQL only) for each field. Set the Primary key field.

Databases created with PHPRunner are saved to the project directory.

> **Note**: don't change the database settings after you've built your project and added records into the database. All data will be lost after the database settings modification.

Downloading drivers

If you use a 32-bit version of PHPRunner, you need to download the 32-bit drivers. If you use a 64-bit version of PHPRunner, download the 64-bit drivers.

32-bit drivers:

- Oracle - Download link (requires an Oracle account).
- Informix - Download link (requires an IBM account).

- DB2 - Download link.

64-bit drivers:

- Oracle - Download link (requires an Oracle account).
- Informix - Download link (requires an IBM account).
- DB2:
  - version 9.7 - Download link;
  - version 10.1 - Download link;
  - version 10.5 - Download link.

See also:

- Navigation bar
- Datasource tables
- How to install a local web server (XAMPP)
- Connecting to a remote MySQL database
- Working with MS Access databases
- Working with Oracle databases
- Open Database Connectivity (ODBC)

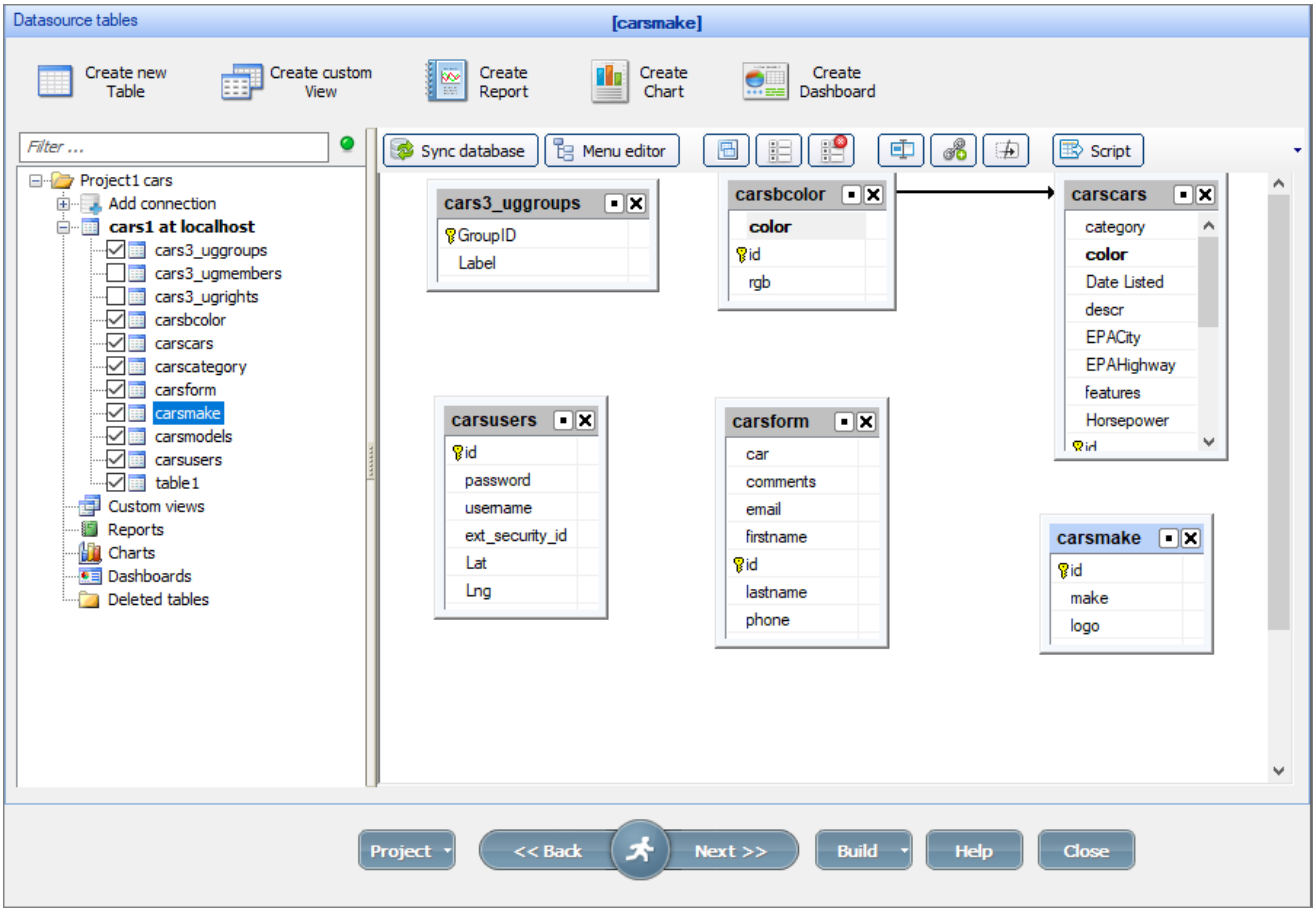
2.6 Datasource tables

Quick jump

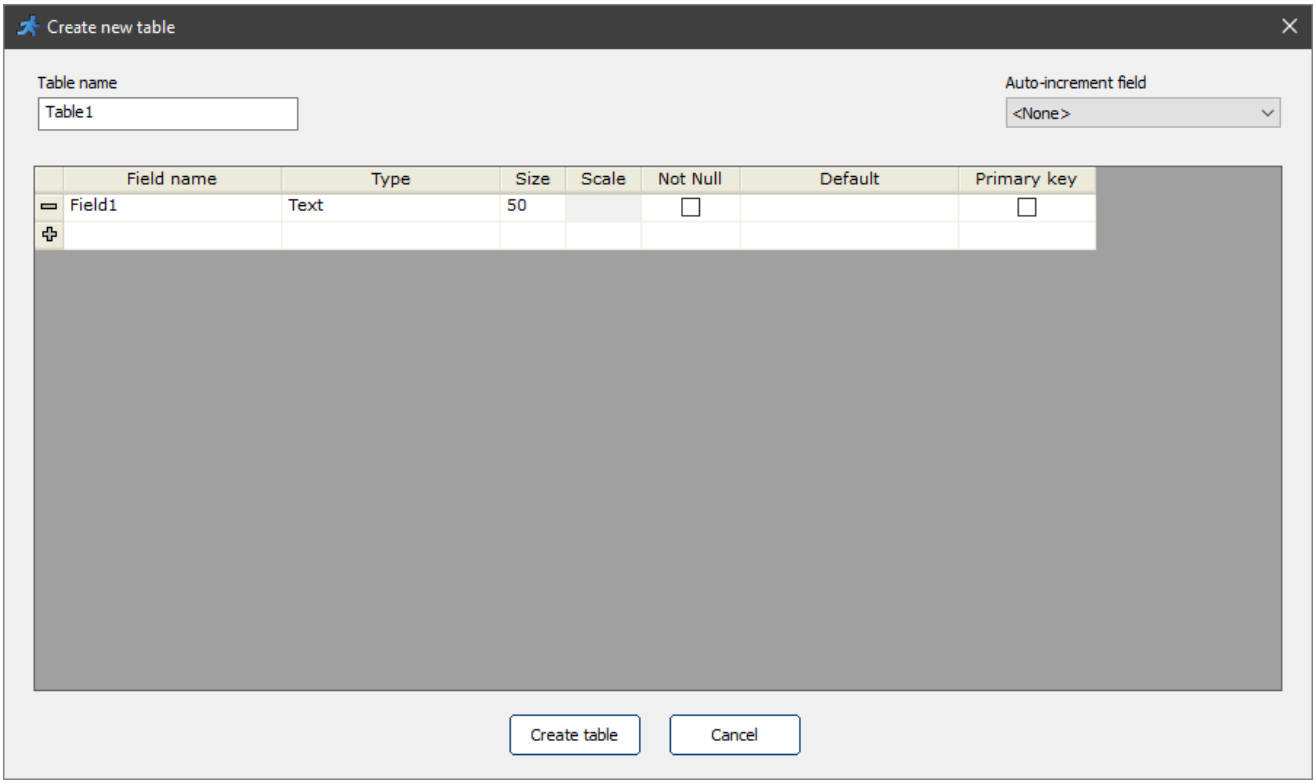
Working with tables

After you successfully connected to the database, select all datasource tables you'd like to build PHP code for. Next, highlight one of the selected tables and proceed through the other screens in PHPRunner adjusting settings for the selected table.

You can always see the name of the currently selected table in the middle of the blue info pane on top. To switch between selected tables, use the **Table list** pane on the left.

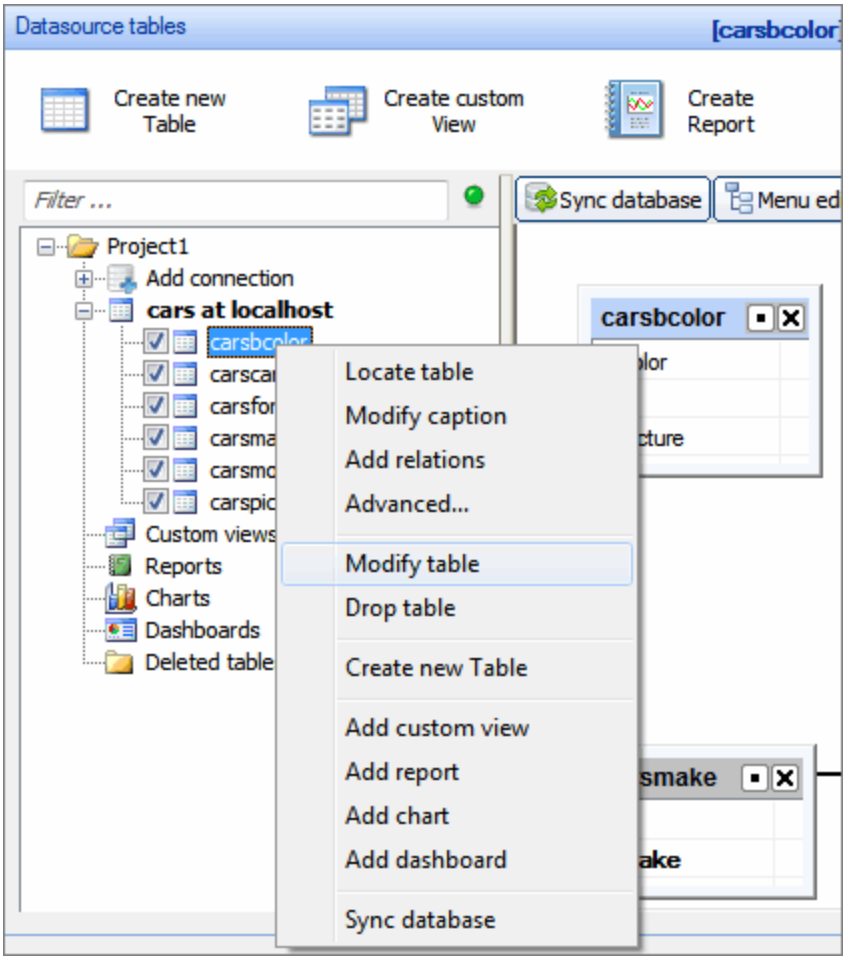


Type in the name, type, size, scale (applies to the DECIMAL field type in SQL Server, Oracle and MySQL only) for each field. Set the Primary key field. Click **Create table**.



Note: you can also add a new custom view, report, chart, or dashboard to your project.

To edit or delete the tables, right-click the table name in the list.

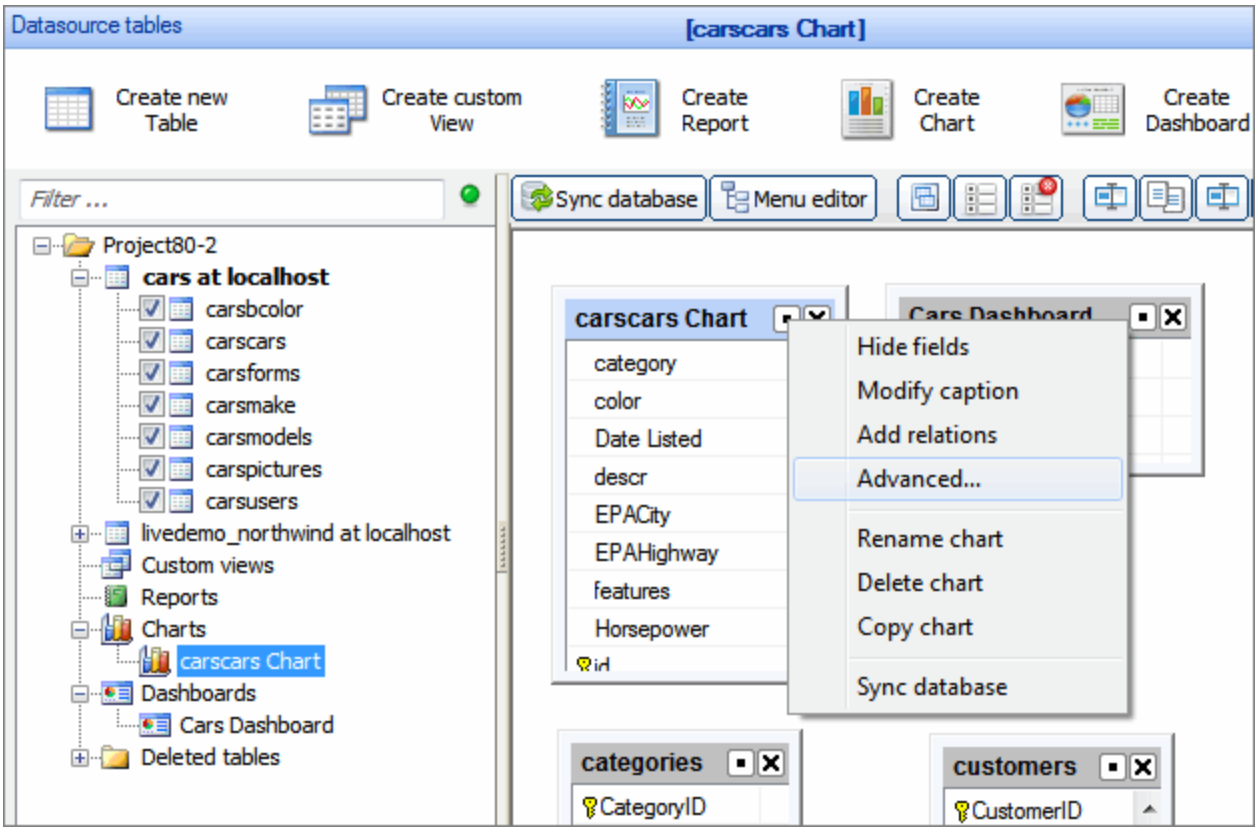


Toolbar description:

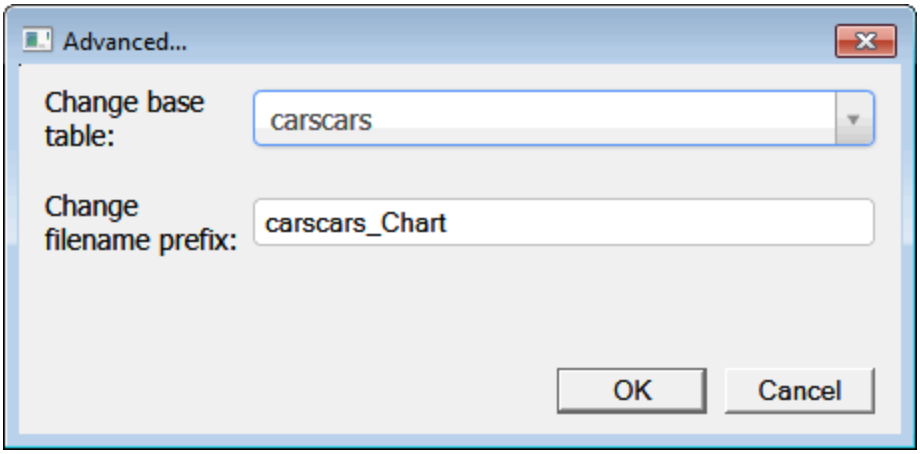
Button	Description
Sync database	Synchronizes the database.
Menu editor	Opens Menu Builder.
	Arranges tables alphabetically.
	Shows all fields in all tables.
	Hides all fields in all tables.

	Opens the Label Editor where you can change the table captions.
	Opens the **Table link** properties window where you can add new table relations.
	Shows/Hides fields for the currently selected table.
	Opens the **Create SQL script** window where you can create SQL scripts for tables/data transfer to another server.
	Searches the table names.

To open the context menu of a table/view/chart/report/dashboard, click  near its name or right-click the table.



The **Change caption** option allows editing caption and field labels. Click the **Advanced option** to change the base table for view/chart/report or filename prefix.

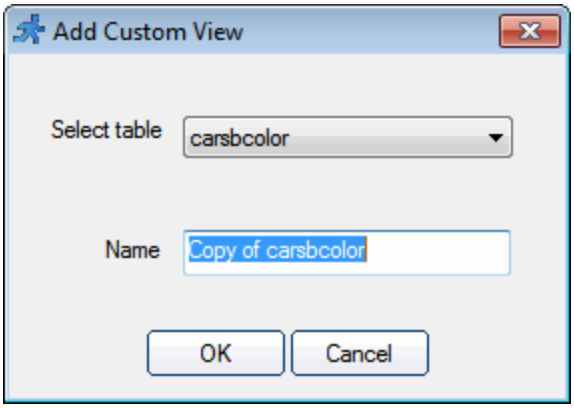


Since the view, chart, and report may include fields from several tables, the base table is the table where the data is added, edited, and deleted. Key columns can be selected for the base table on the Choose pages screen.

By default, the prefix for each generated file name is the table name. E.g., if the table name is *OrderDetails*, the file name of the **List** page becomes *OrderDetails_list.php*. If you change the prefix to *details*, the file name becomes *details_list.php*.

Create a custom View

Click **Create custom View** to create an additional view of the same table. This feature is useful if you want to present several views of the same data.



When you create a new custom view, all current settings of the table are copied into it, except for visual templates and events. You can create a copy of an existing custom view: right-click the custom view and select **Copy**.

> **Note**: custom views are not created in the database and only exist in the project.

Example:

The *Cars* table shows all cars in the database. SQL query: *select * from Cars*.

The **Active listings view** displays only the active listings. SQL Query: *select * from Cars where status='active'*.

The **Closed listings view** displays only the closed listings. SQL Query: *select * from Cars where status='closed'*.

The SQL query can be modified later on the SQL query screen in PHPRunner.

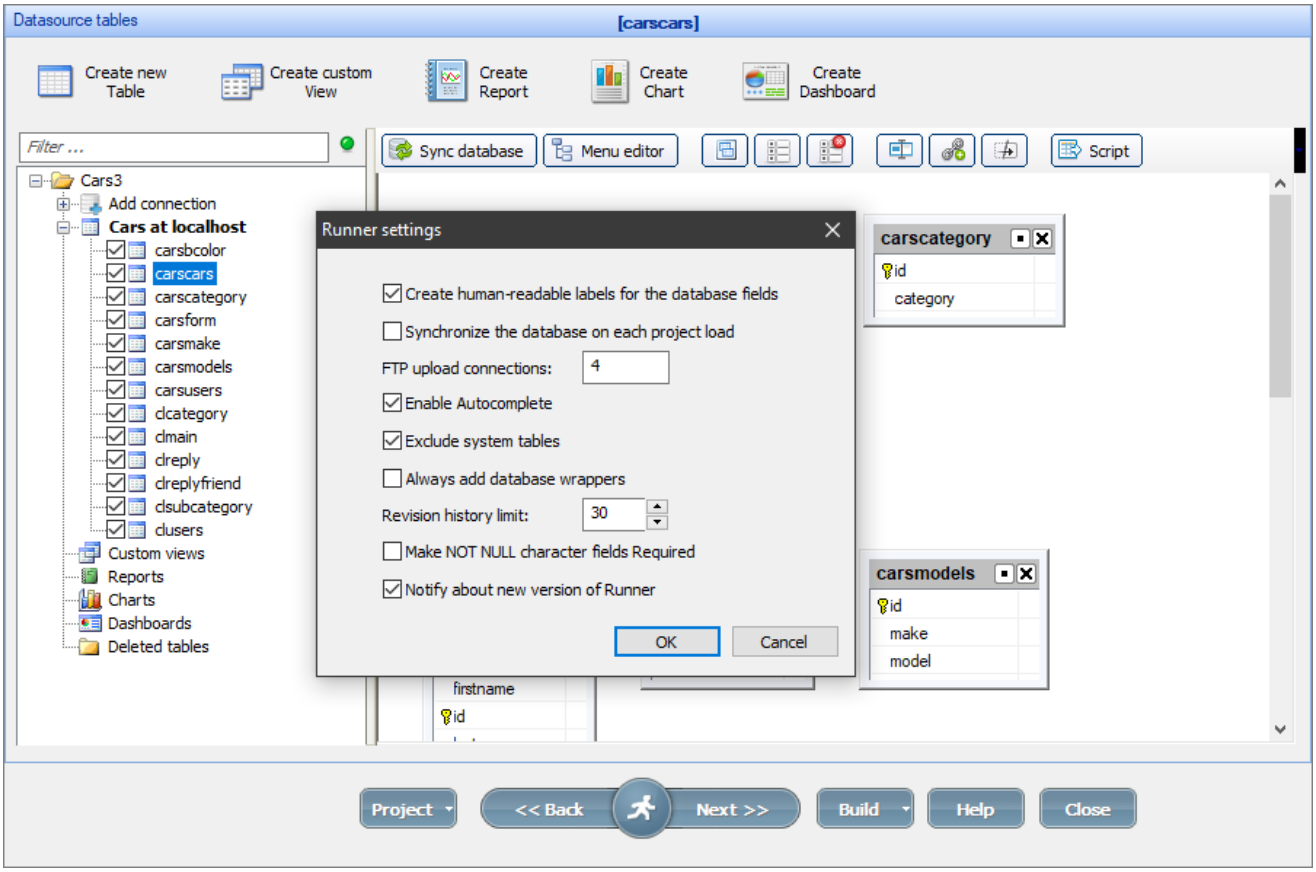
> **Note**: when you create a custom view, all table settings are copied to the **Custom view settings**.

Synchronize database

Every time you make changes on the **Datasource tables screen** (e.g., create a new table, add or edit table fields, etc.), make sure that these changes are synchronized with the database. The structures of the database and PHPRunner project should be synchronized.

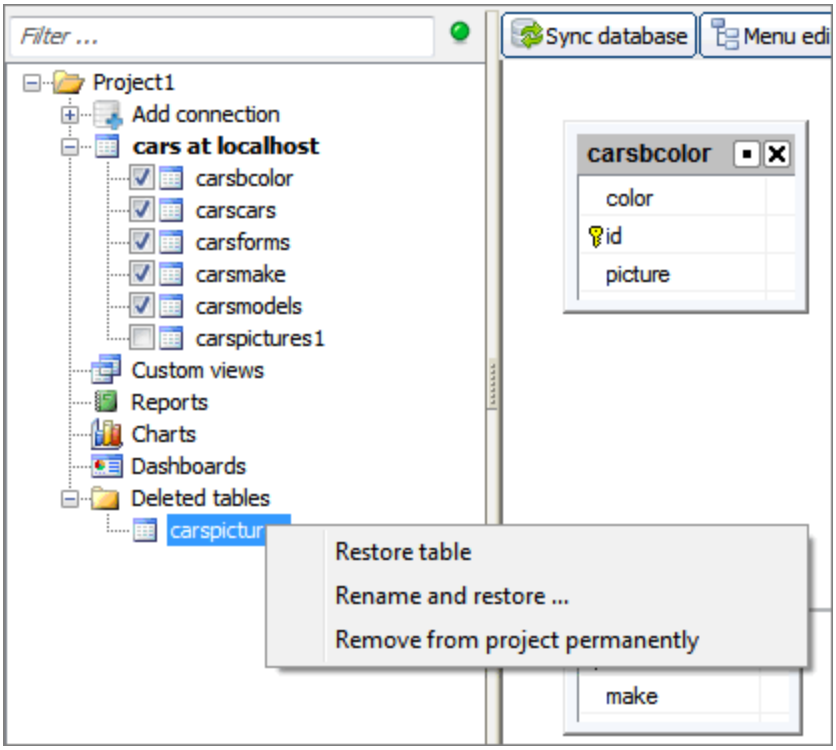
To synchronize the database manually, use the **Sync database** button or right-click the table tree or blank area near the table tree and select **Sync database**.

> **Note**: you can also use the option to synchronize the database automatically each time the project loads (select **Project** -> **Settings**). Use this option for small or local databases. With the remote or large databases, the automatic synchronization takes more time upon opening the project.
>
> See Working with projects to learn more

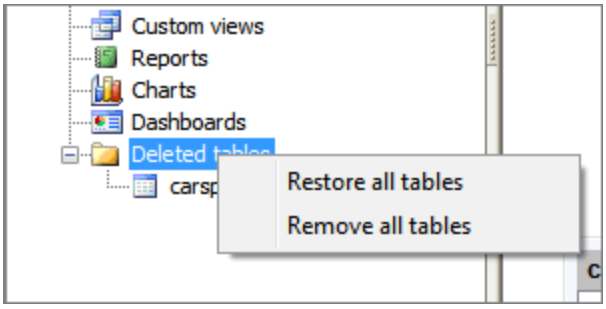


Renamed/deleted tables

The tables that were renamed outside of PHPRunner or deleted in PHPRunner are moved to the **Deleted tables** folder. You can restore, rename and restore, or remove the selected table from the project permanently.



Right-click the **Deleted tables** folder to get two options: **Remove all tables** and **Restore all tables**.



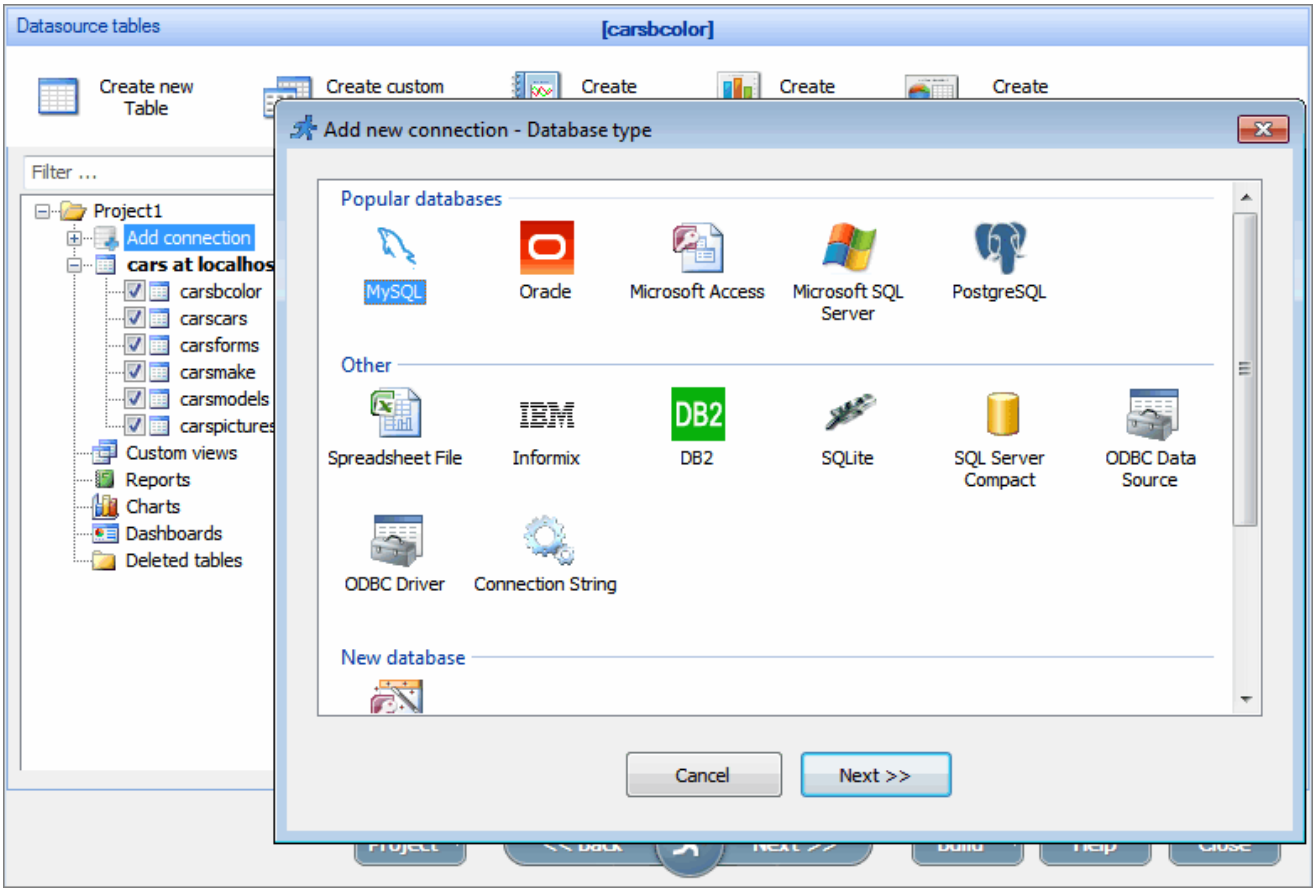
Multiple database connections

You can add multiple data sources and mix several database types like MS Access, SQL Server, and MySQL in a single PHPRunner project. You can have a master table in MySQL and a details table in MS Access. The same applies to lookup tables.

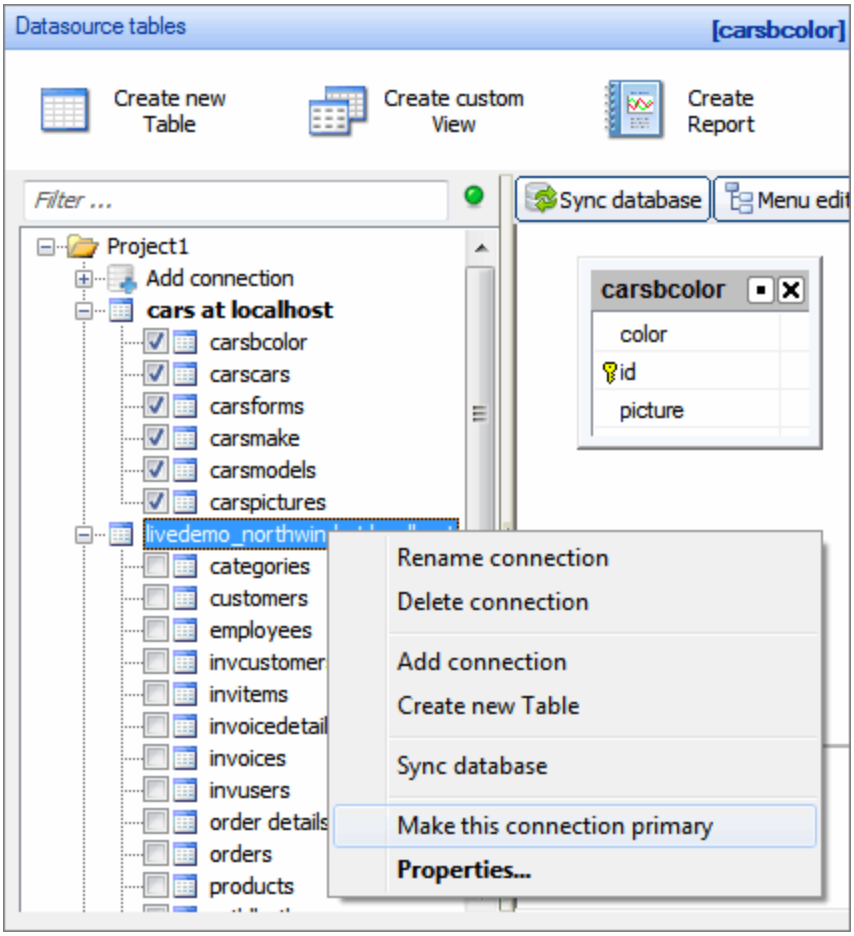
Note: the **Multiple database connections** feature is available only in the Enterprise Edition of PHPRunner. See Editions Comparison to learn more.

To add a new database connection:

- click **Add connection**;
- select the database type and connect to the database.



The first database connection is considered "primary". However you can make any other connection primary by right-clicking the database name and selecting the corresponding option.



The following features work only with the primary database connection:

- Data Access Layer (deprecated; use Database API - it works with multiple database connections);
- Add template to project - the template is added only to the primary database;
- Upload to demo account - the tables from the primary database are uploaded to the demo account.

All other features work with all database connections, including the **Save project as template** option.

We do not advise to add tables with the same names that belong to different databases to avoid conflicts.

See also:

- Create a report
- Create a chart
- Create a dashboard
- Master-details relationship between tables
- Connecting to the database
- SQL query screen
- Choose pages screen

2.7 Master-details relationship between tables

Quick jump

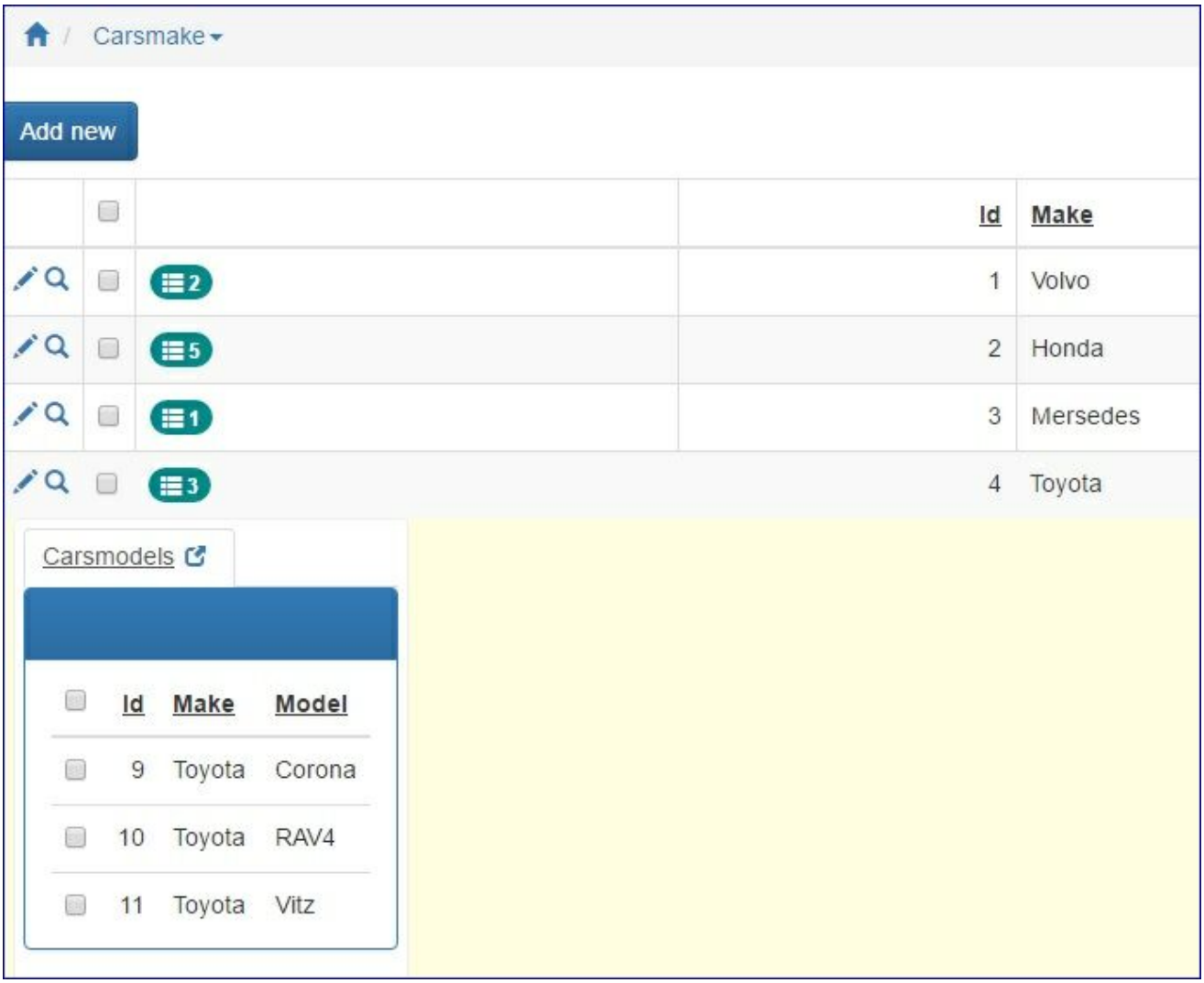
What is a Master-details relationship?

A one-to-many relationship, often referred to as a "master-details" or "parent-child" relationship, is the most usual relationship between two tables in a database. Common examples of this relationship include customer/purchase data, patient/medical-record data, and student/course-result data.

For example, each customer (Master table *Customers*) is associated with at least one order record (Details table *Orders*). Valued customers have several order records and often a user needs to view one table in connection with the other.

Master-details relationships are commonly used in applications. In PHPRunner, you can join or link several tables that have at least one common field. You can **Add**/**Edit** the records of the linked tables on the same page in the generated app.

Here is an example of a Master-details relationship between the *Carsmake* and *Carsmodels* tables, where the details table shows the *models* for the selected *make*.



Note: one master table can have multiple details tables. You can display data from multiple details tables on the same page.

For example, you can display the *customers*, their *orders*, *order details*, and edit all three tables at the same time without leaving the page.

Setting up the Master-details relationship

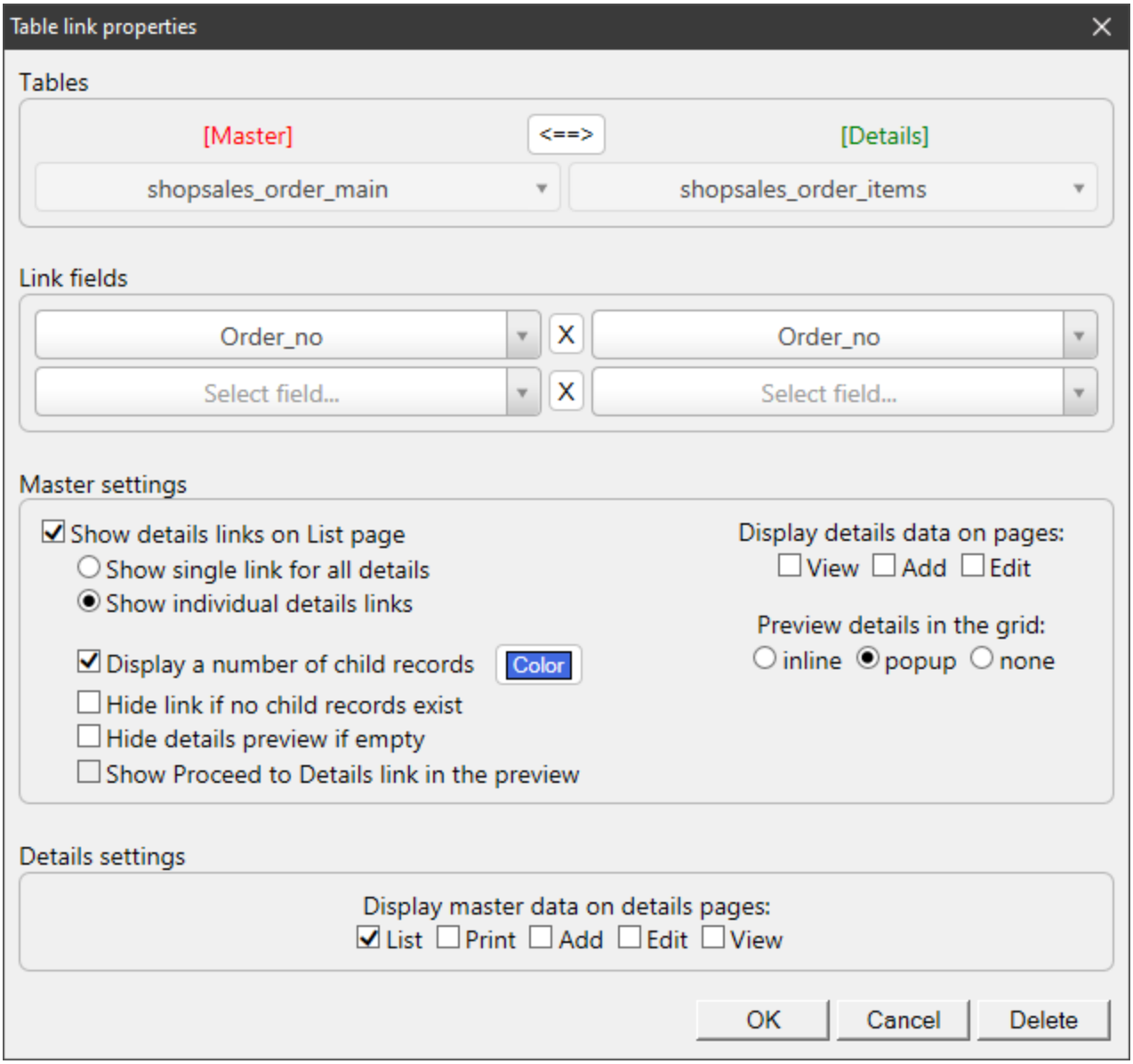
To create a master-details relationship in PHPRunner:

1. Go to the Datasource tables screen and enable the tables you wish to link in the left panel.

2. Drag a field from one table to another.

> **Note**: when dragging a field, other fields of the same type become highlighted in **bold**.

The **Table link properties** dialog opens. The options are described below.



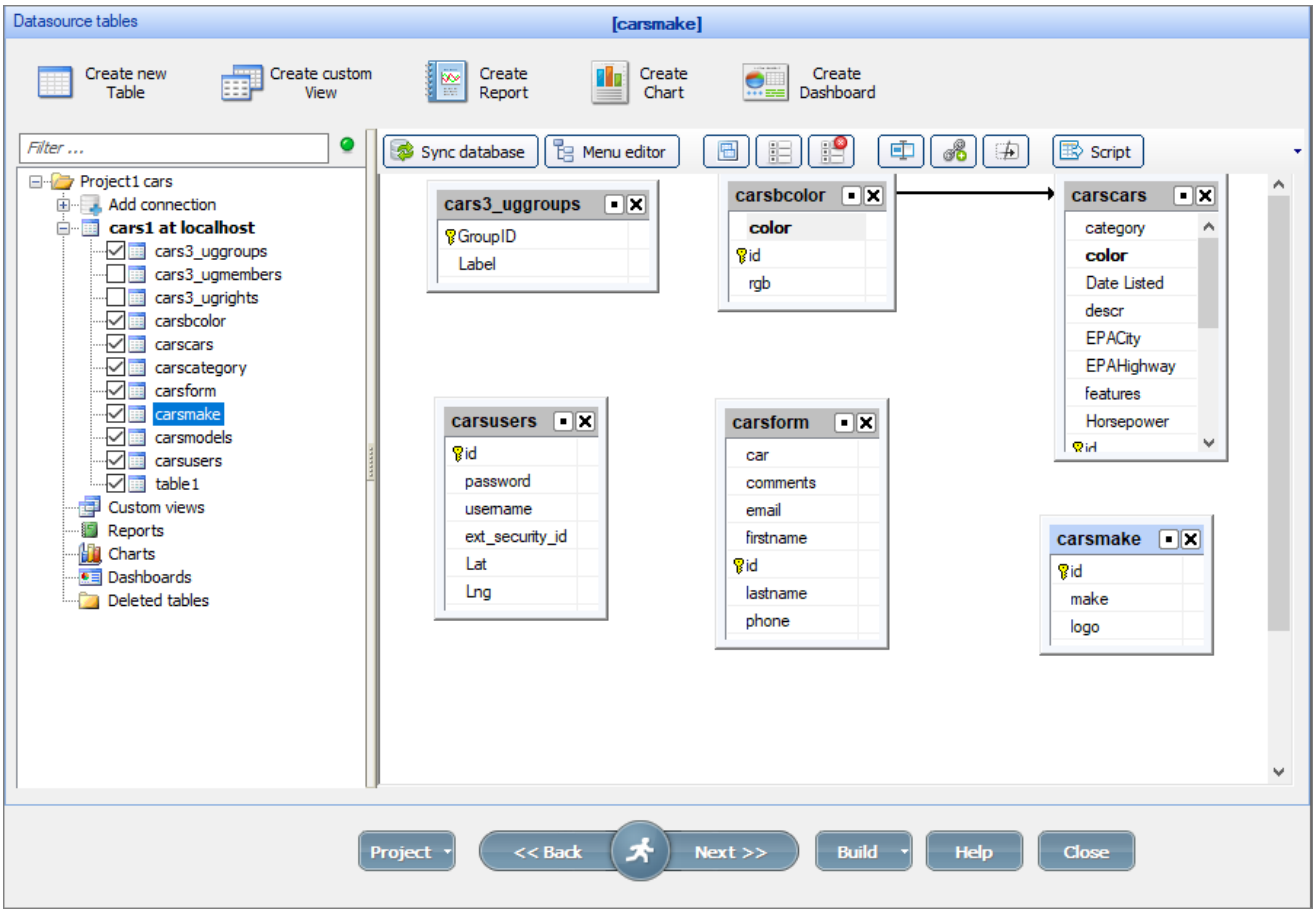
3. Choose which table is the Master one and select the **link** fields for both tables.

> **Note**: make sure that the **link** fields are of the same type.

4. Set up the Master and Details settings. If you'd like to display the master table data on the details page, select the corresponding checkboxes. Click OK.

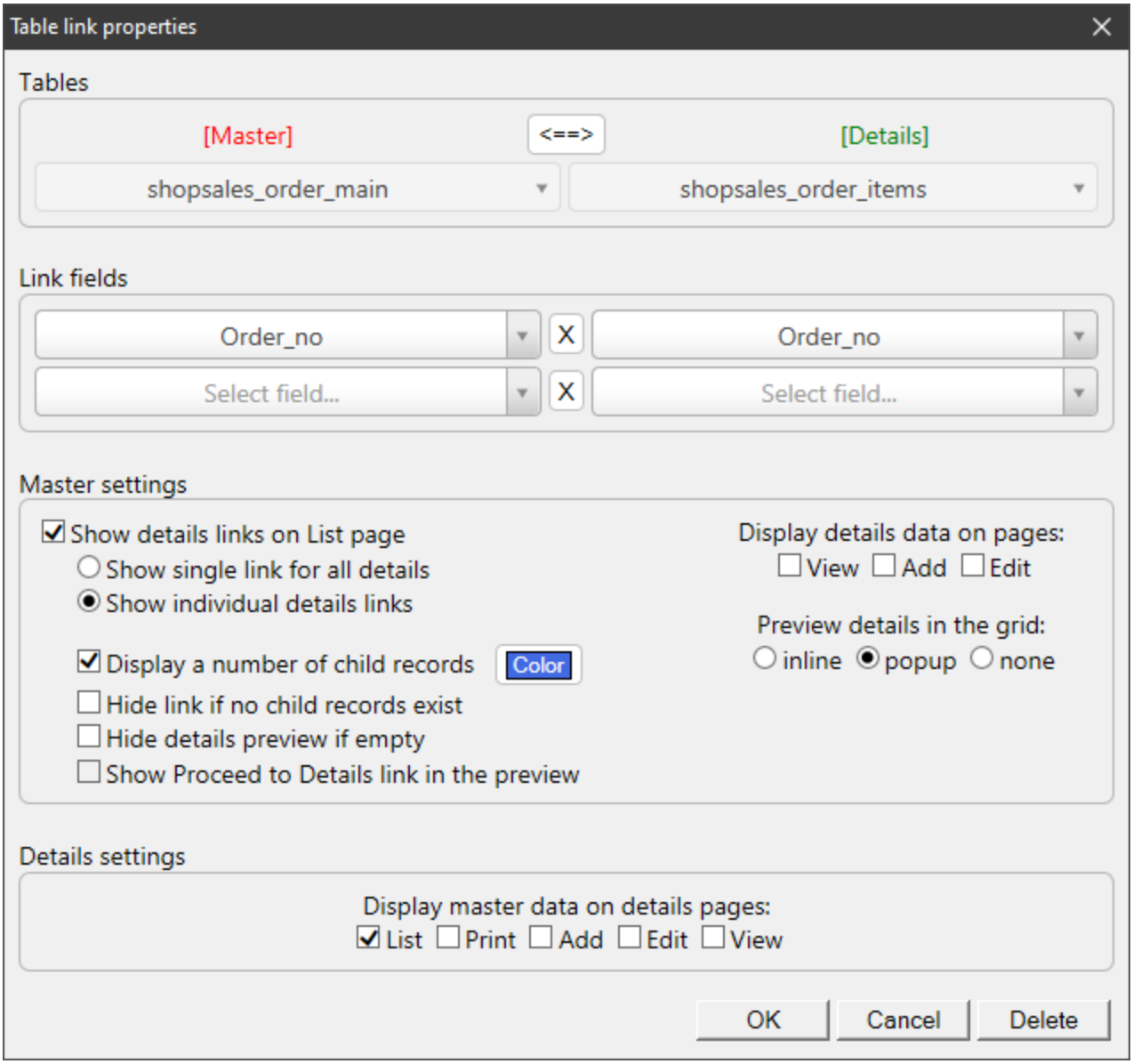
Note: starting with PHPRunner version 10.3, you can also **Hide details preview if empty**.

5. The master-details relationship is created and displayed as an arrow between the tables.



Examples of the Table Link Properties options

Here is how the Table Link Properties window looks like:

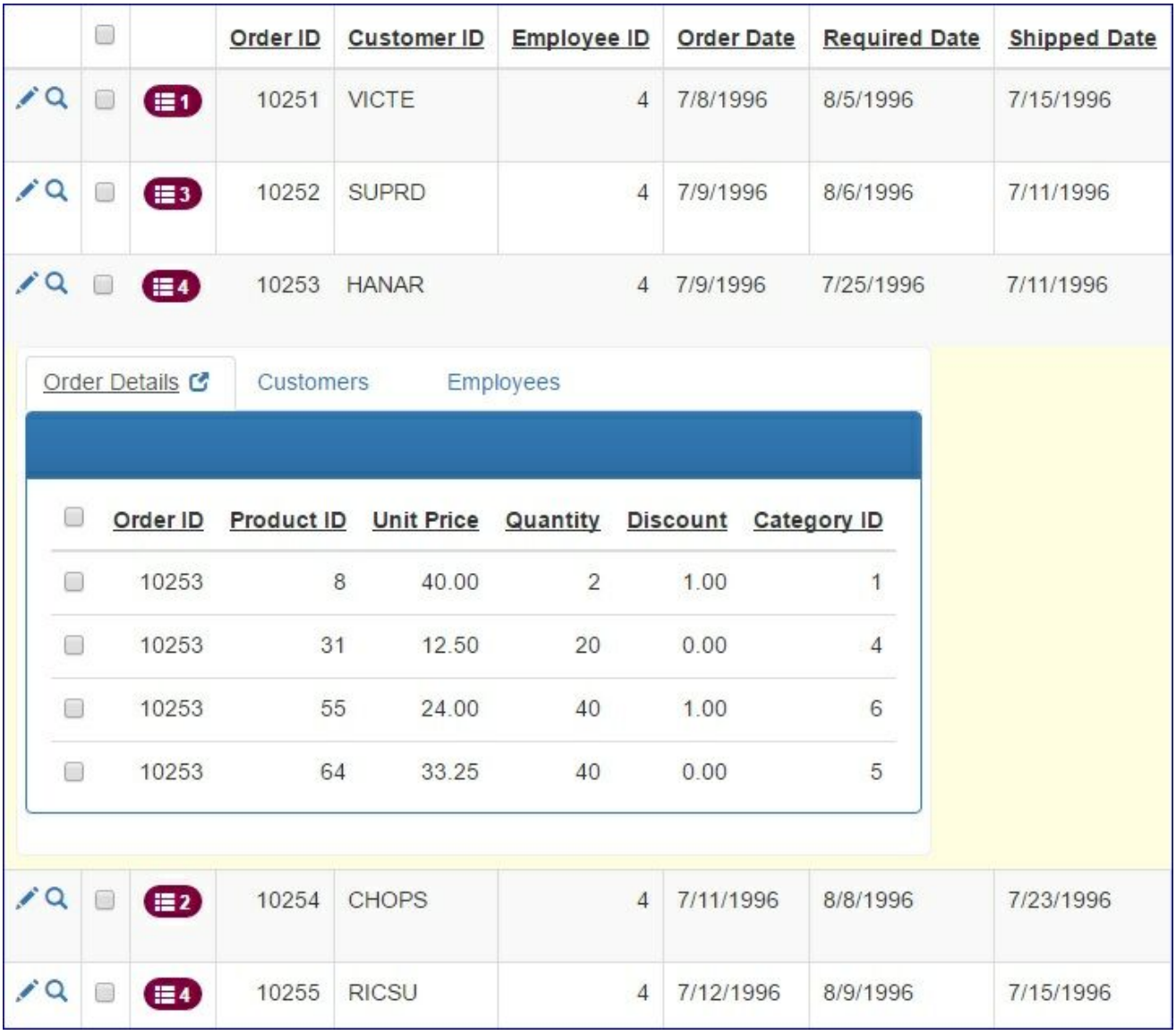


Here are some of the examples for the Table Link Properties options:

Show single link for all details:

A link for details is displayed as 

icon. Click it to view the details tables. Use this option if the master table has multiple details tables.



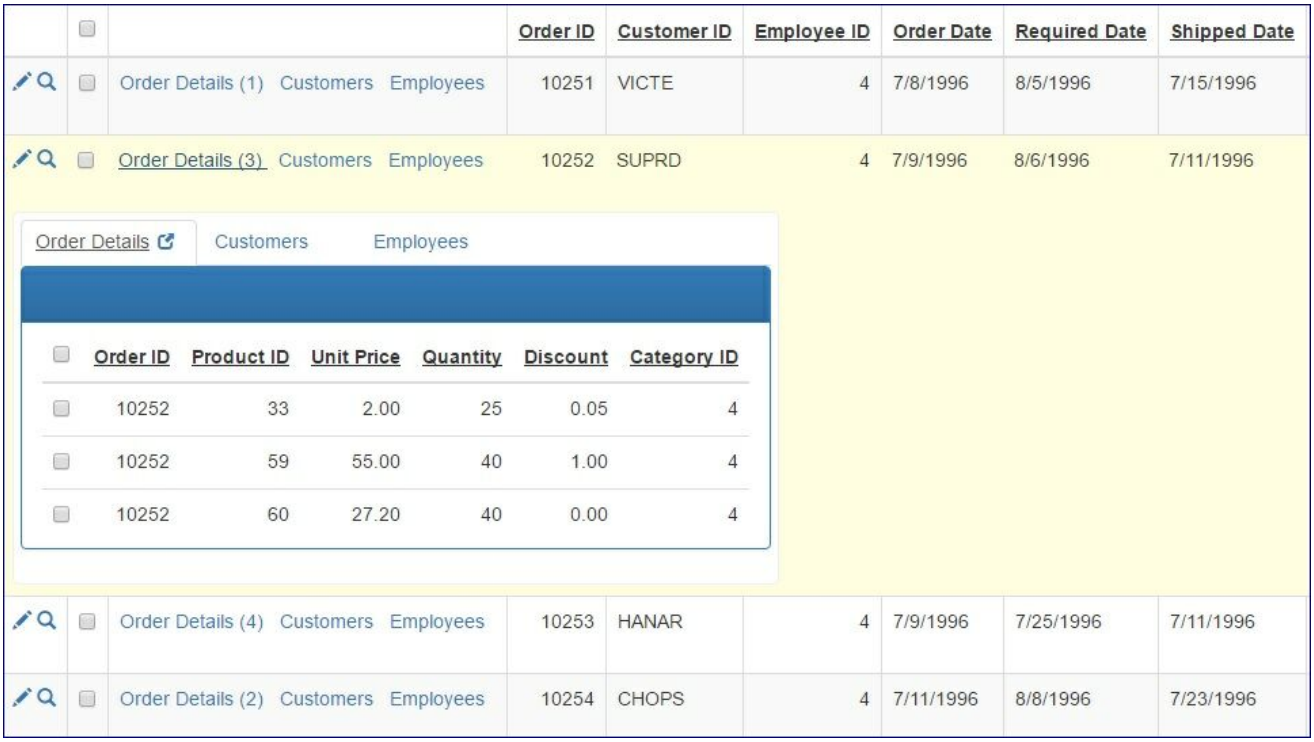
	Order ID	Customer ID	Employee ID	Order Date	Required Date	Shipped Date
1	10251	VICTE	4	7/8/1996	8/5/1996	7/15/1996
3	10252	SUPRD	4	7/9/1996	8/6/1996	7/11/1996
4	10253	HANAR	4	7/9/1996	7/25/1996	7/11/1996

Order Details | Customers | Employees

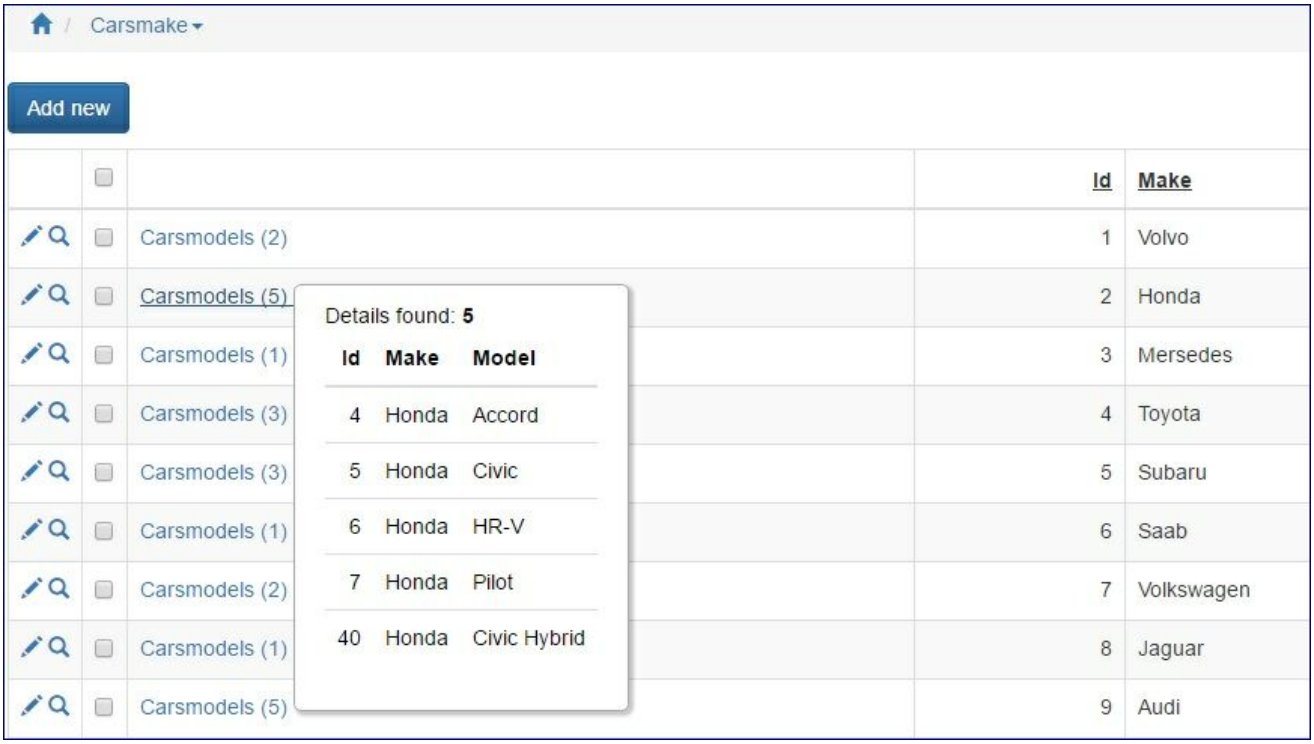
Order ID	Product ID	Unit Price	Quantity	Discount	Category ID
10253	8	40.00	2	1.00	1
10253	31	12.50	20	0.00	4
10253	55	24.00	40	1.00	6
10253	64	33.25	40	0.00	5

2	10254	CHOPS	4	7/11/1996	8/8/1996	7/23/1996
4	10255	RICSU	4	7/12/1996	8/9/1996	7/15/1996

Show individual details links:



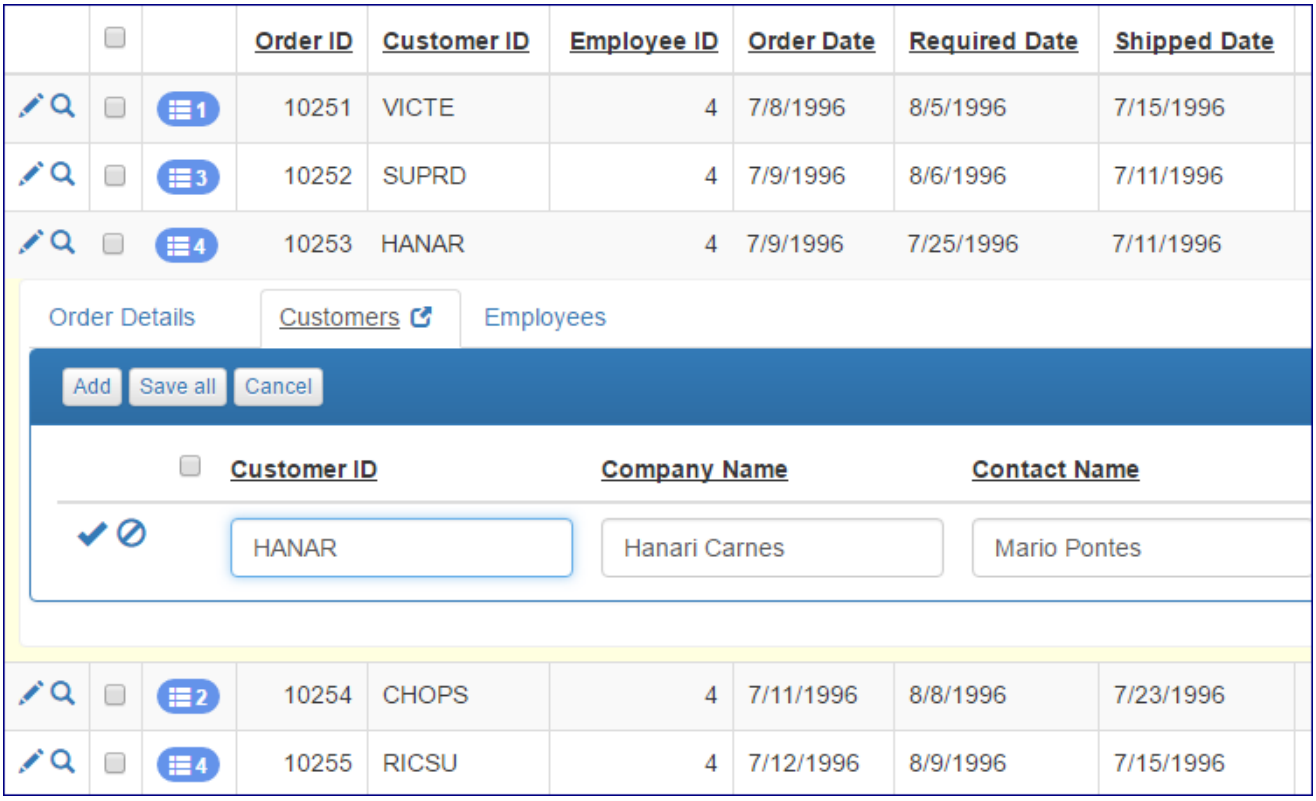
Preview details records in popup:



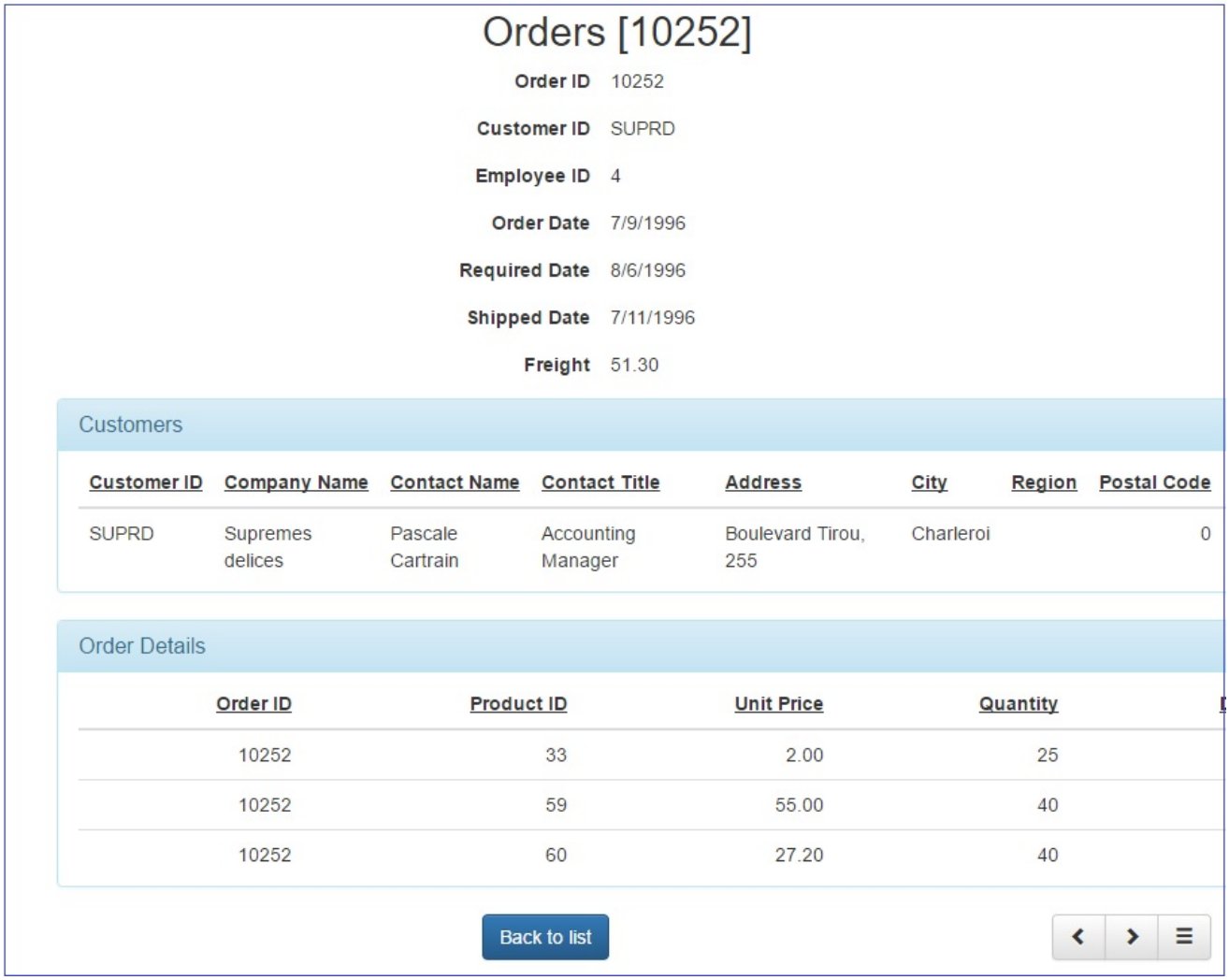
Note: If you select to display details records in a popup, you can define the popup appearance in the **Editor** -> **Details preview** screen for the details table.

Preview details records inline:

If you select to display the details records inline, you can **add**/**edit**/**delete** the details in an inline mode on the master table page.



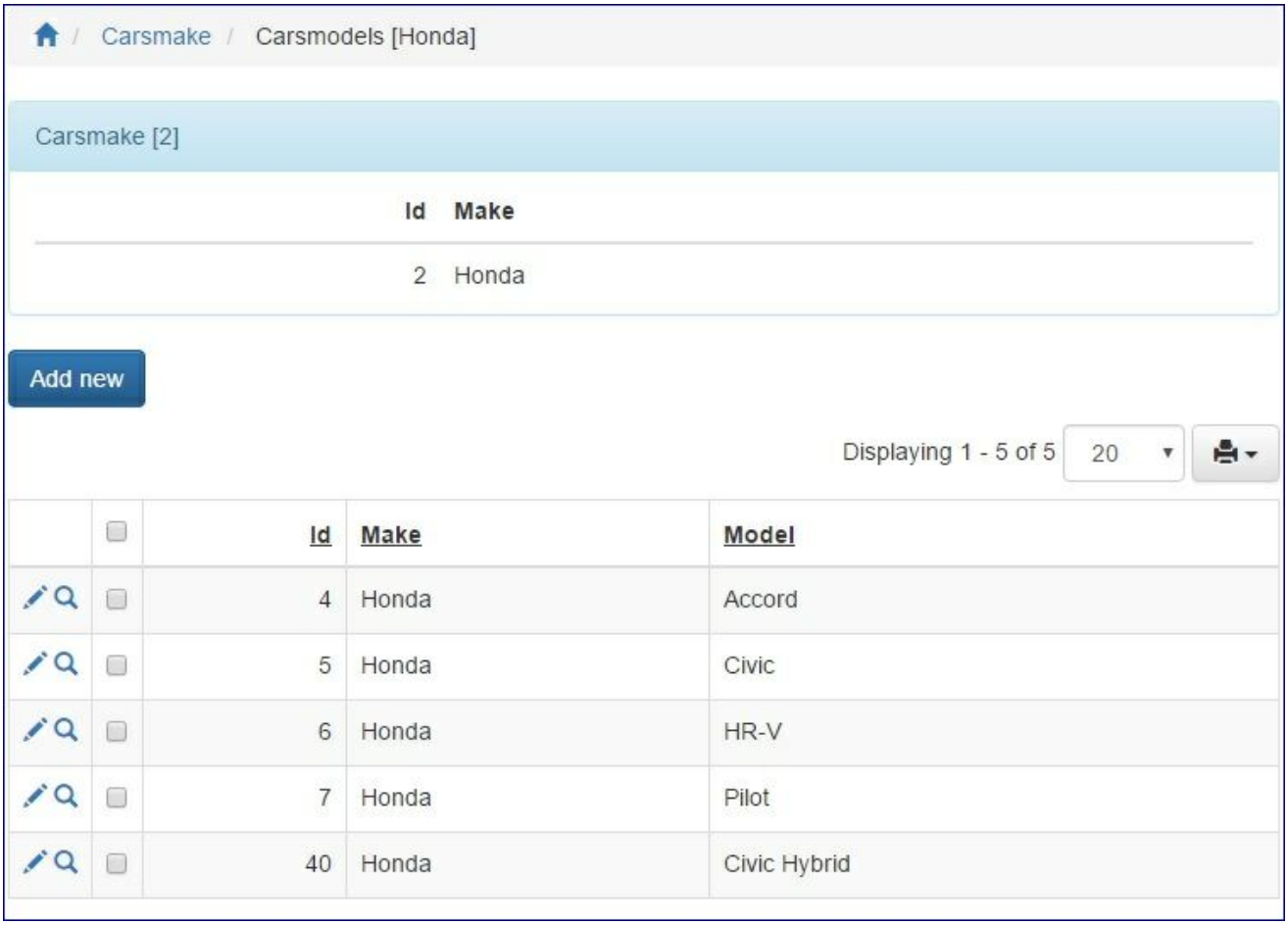
Display details records on the View/Add/Edit page:



Hide details preview if empty:

This option allows you to hide the details preview on the master table **View**/**Add**/**Edit** page if the details table does not contain any records for the current master table record.

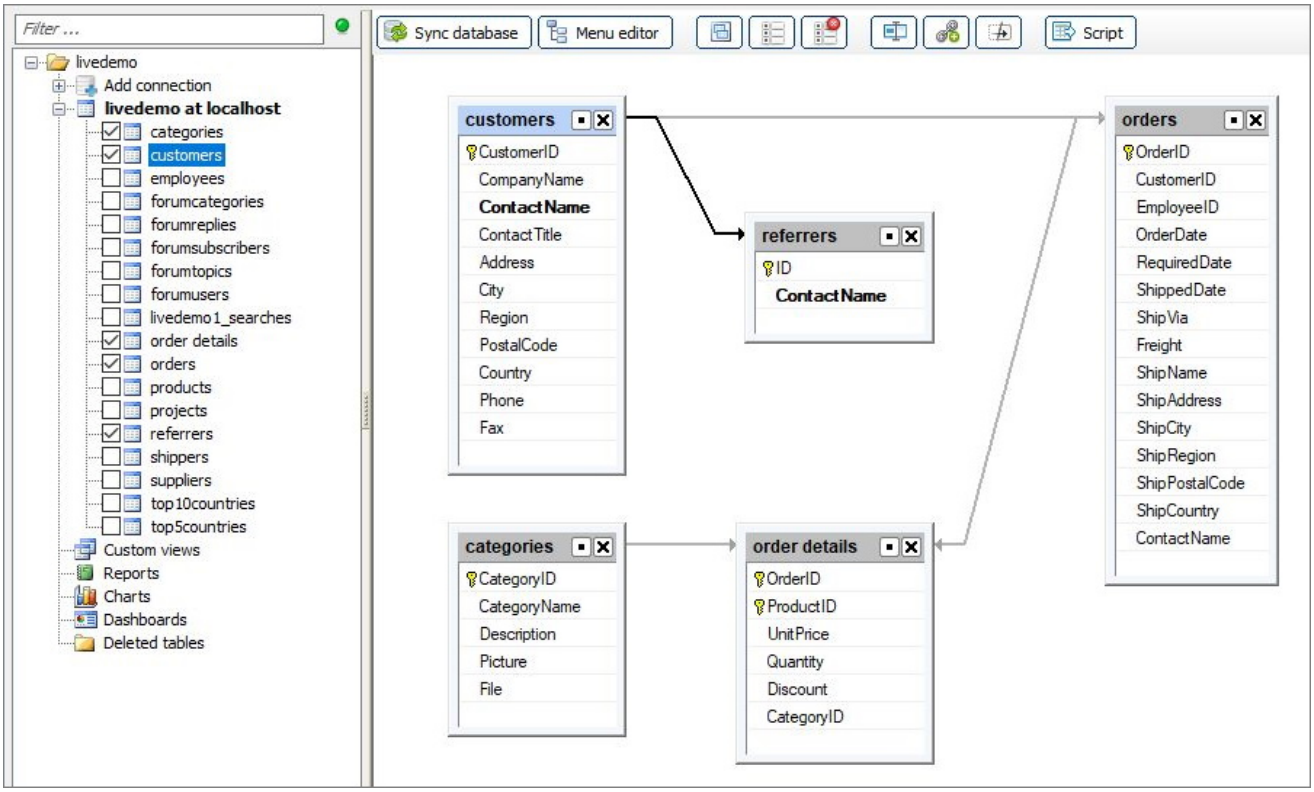
Display master table data on the details page:



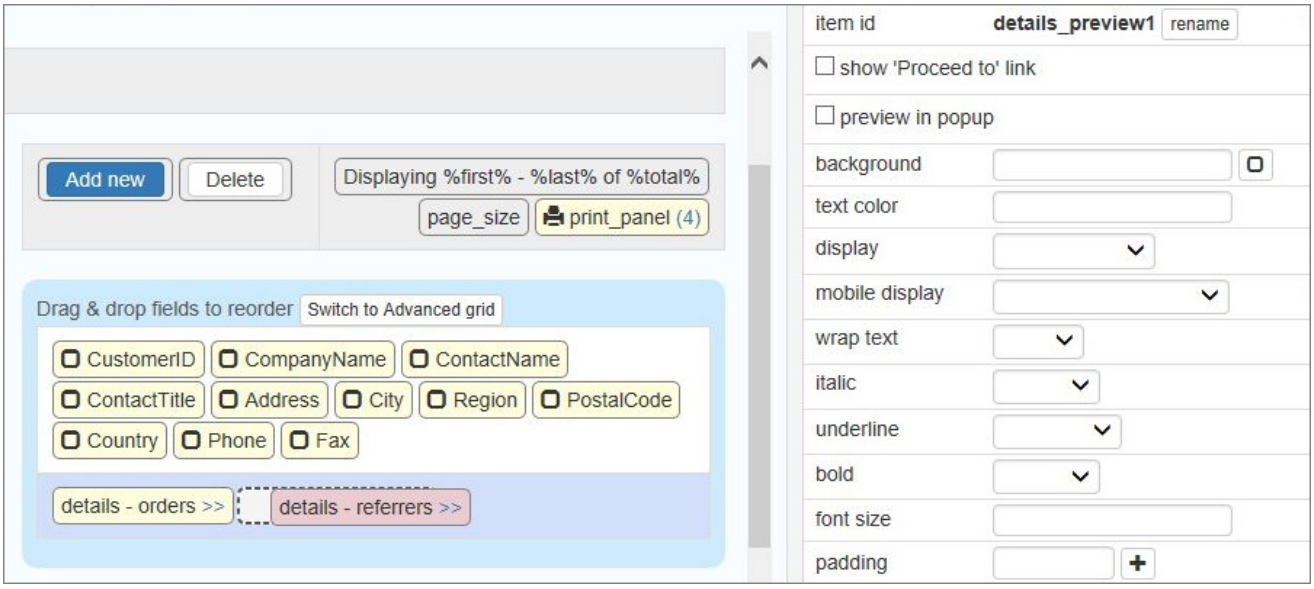
Changing the order of details tables

If a master table has two or more details tables, you can re-order the details tables as well as choose the orientation (vertical or horizontal). You can do that in Page Designer.

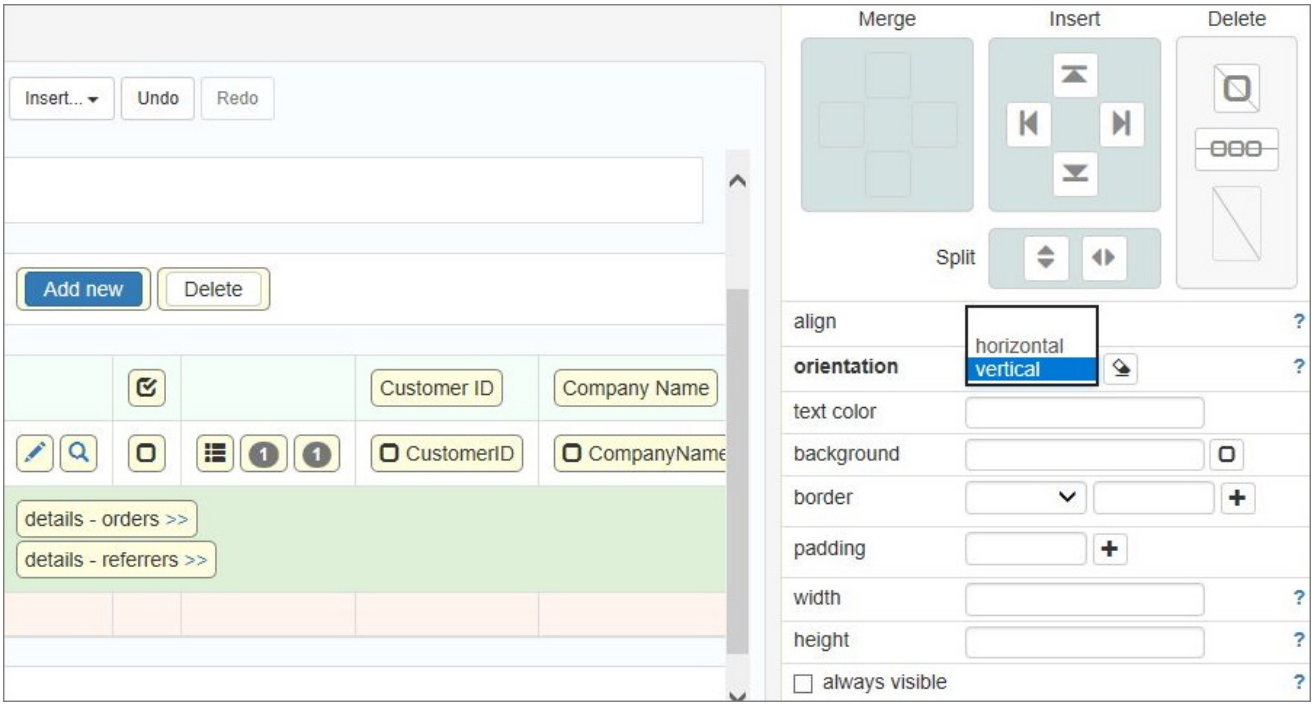
Here is an example of a "*customers*" table that has two details tables: "*referrers*" and "*orders*".



Proceed to the **Page Designer** to change the order of the details tables using drag-n-drop:

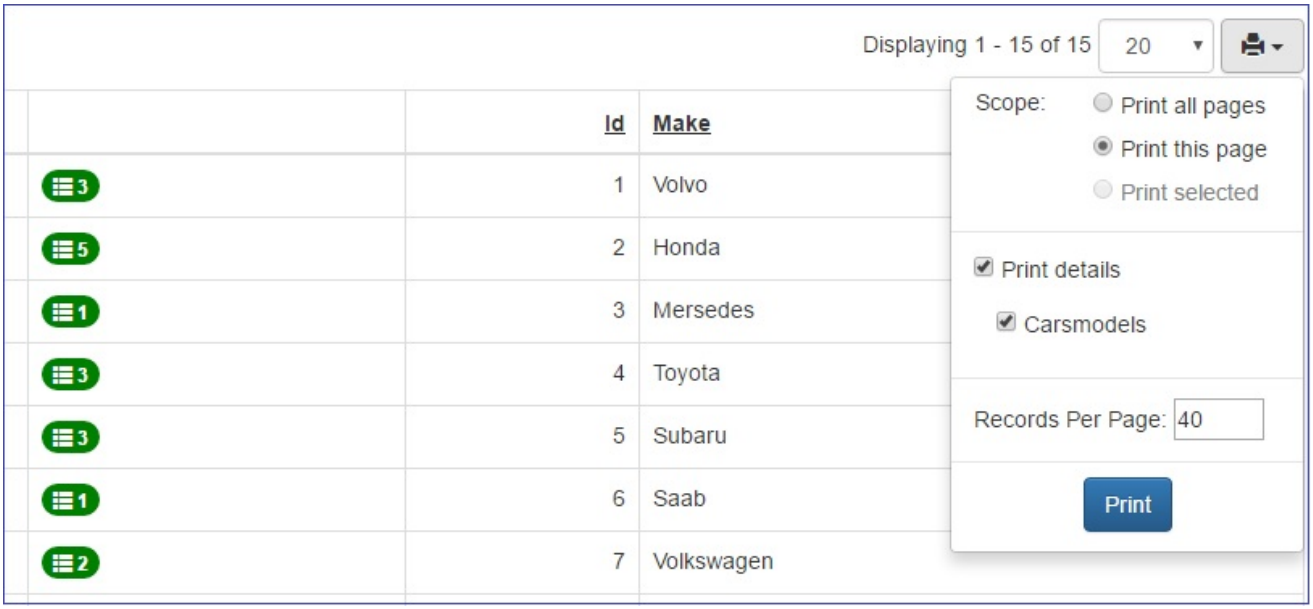


To change the details tables orientation, switch to Advanced grid first. Then click on the details tables cell to open the Cell properties and choose the orientation:

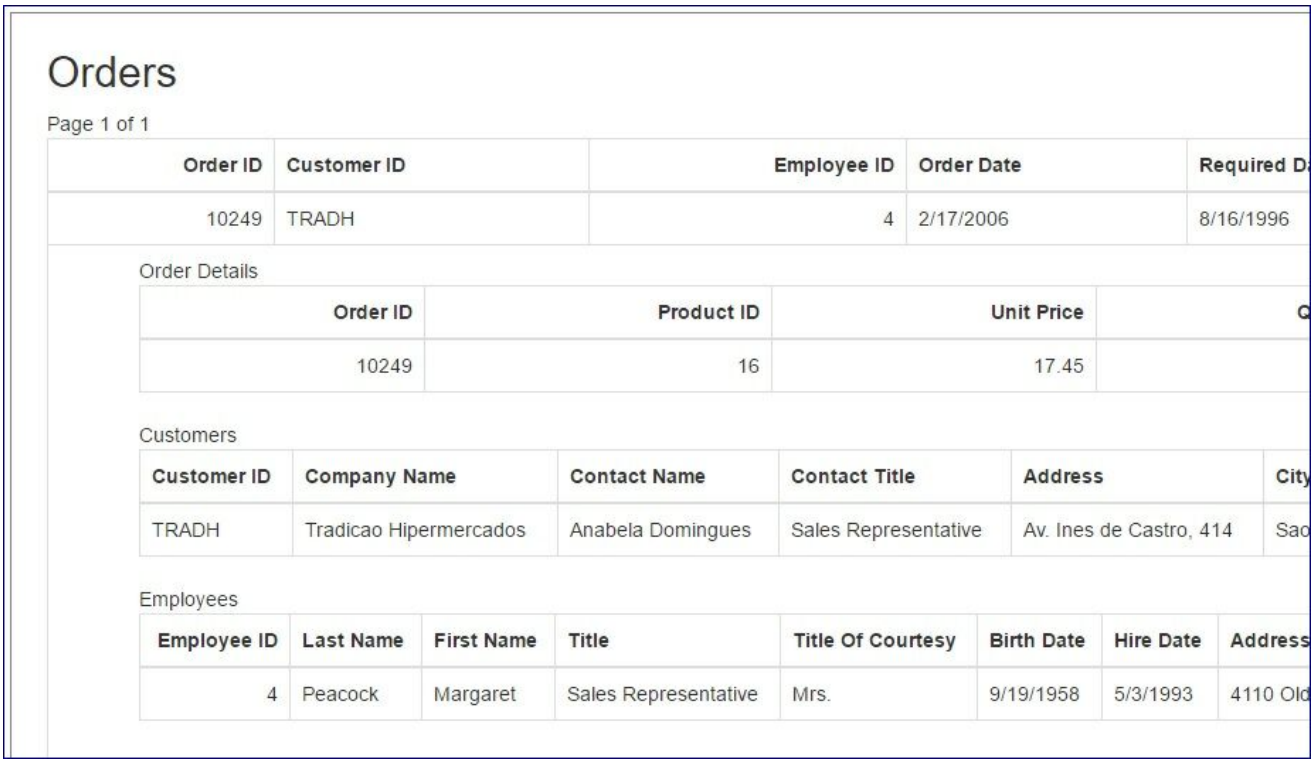


Printing the master table data with the details

You can print the master table data on the **List** page with the data from the details tables. Click the **Print** icon and select the corresponding checkboxes to do so.



Here is an example of a page to be printed:

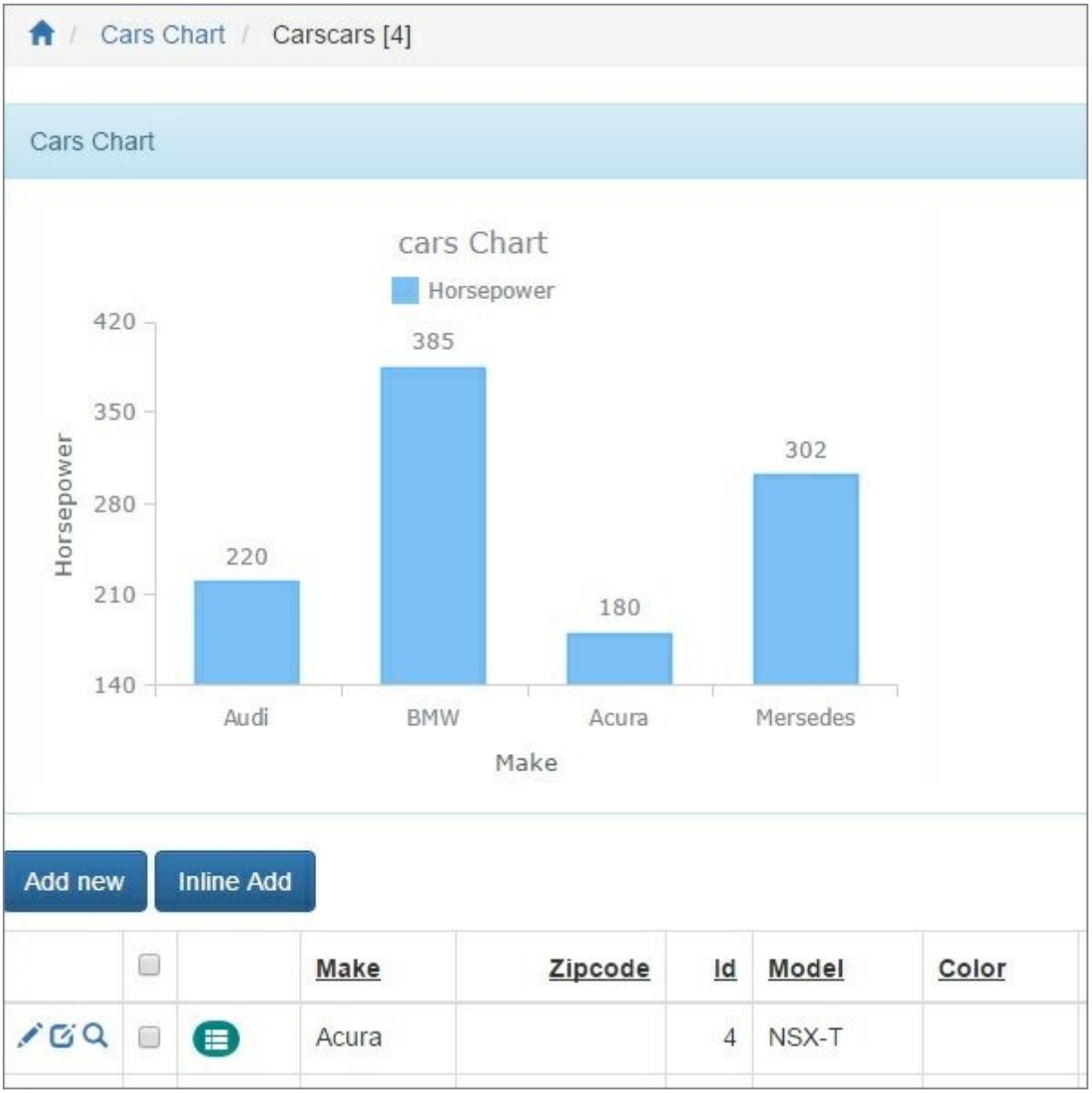


Charts and reports as master and details tables

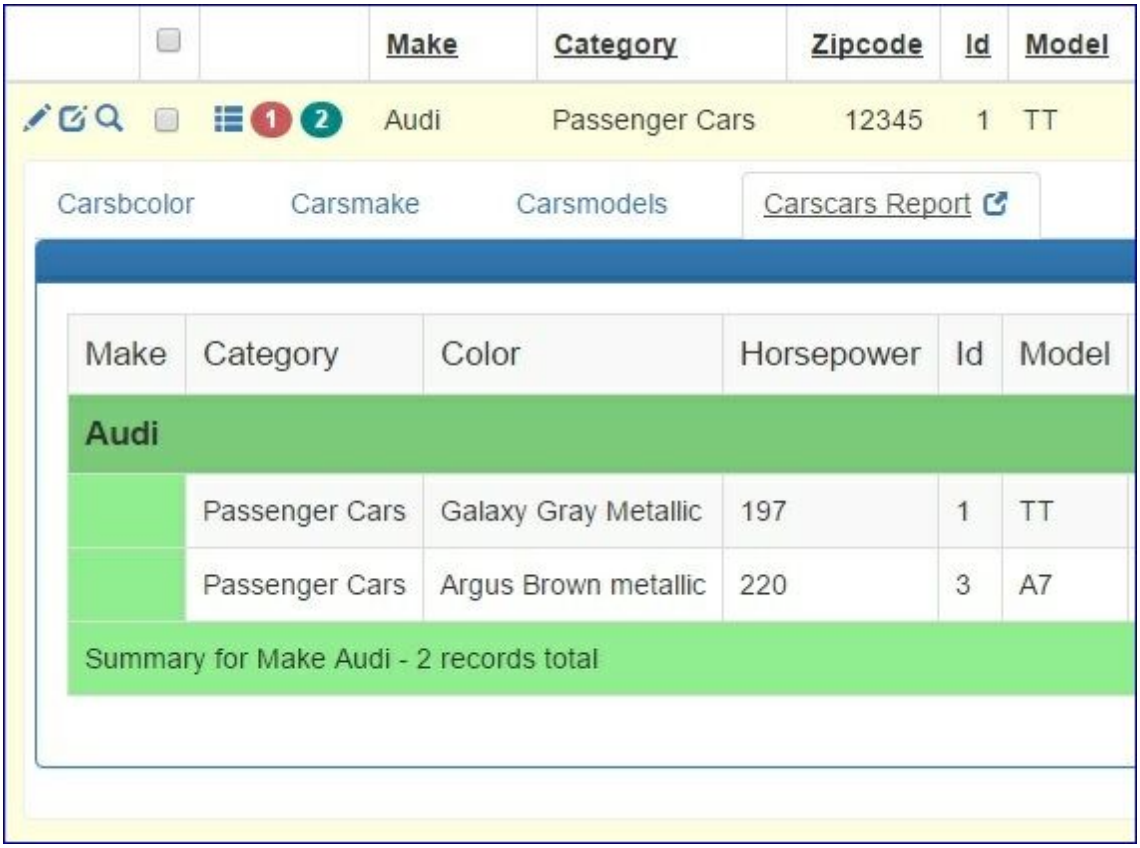
You can use charts and reports with master or details tables.

A chart as a details table:

A chart as a master table:



A report as a details table:



See also:

- Datasource tables
- Charts
- Reports
- Page Designer
- SQL Query

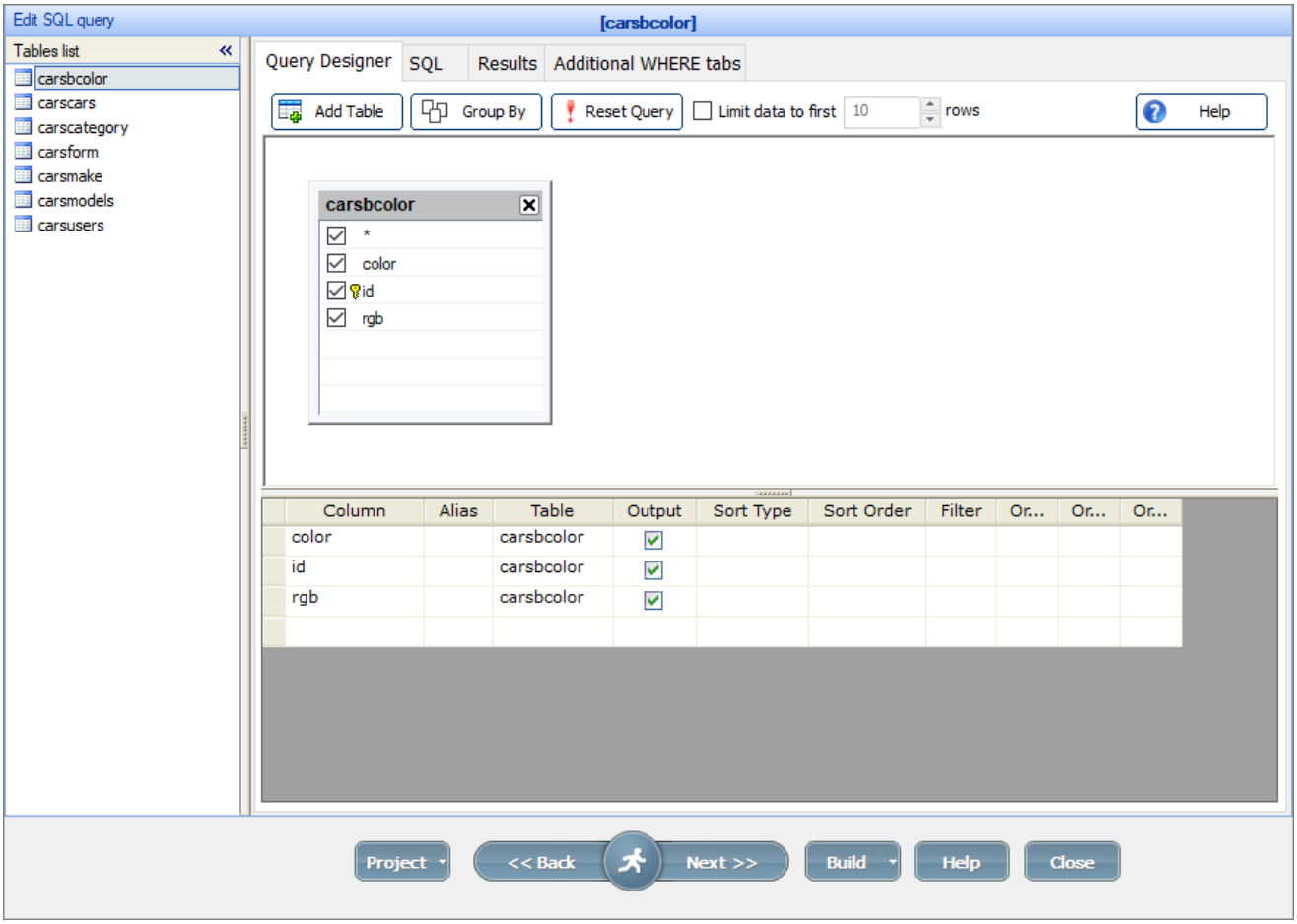
2.8 SQL query screen

2.8.1 About SQL query

About SQL query

The **Edit SQL query** screen allows you to modify the SQL queries that PHPRunner has built for you automatically.

> **Note**: the automatically created SQL queries work in most cases, so you don't need to adjust them every time.



This page includes:

- graphical panes that display your SQL statement (Query Designer tab);
- a text pane that shows the text of your SQL statement (SQL tab);
- the results of your query as a table with values (Results tab);
- additional WHERE clauses for the tabs on the **List** page (Additional WHERE tabs).

Design the SQL query in the **Query Designer** or **SQL** tab, then click **Results** to view the resulting table.

> **Note**: the **Query Designer** synchronizes the views, so they remain relevant to the changes made on the **SQL** tab.

Features

- a visual interface to design the queries;
- an automatic SQL Statement generation;
- creating joins with drag-n-drop;
- a grid pane to specify the criteria (ORDER BY, GROUP BY, WHERE, etc.);
- SQL parsing: enter the SQL statement to fill in the grid and diagram.

See also:

- Connecting to the database
- Datasource tables
- Master-details relationship
- Choose pages screen
- Using JOIN SQL queries

2.8.2 Query Designer tab

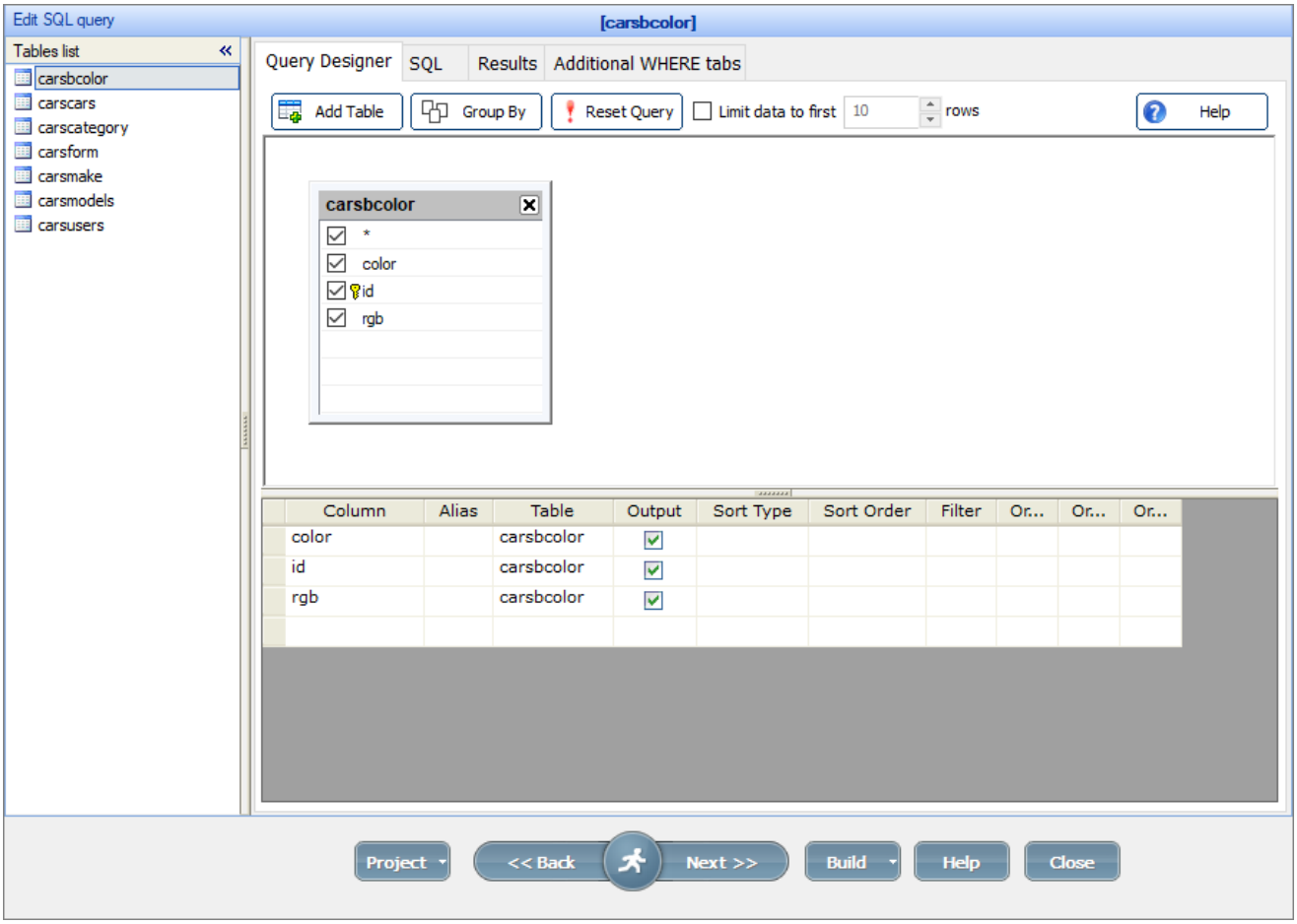
Quick jump

What is not supported

About Query Designer

The **Query Designer** allows you to use a simple graphical interface to construct SQL queries. The fields selected in the *Output* column are added to the SELECT clause.

> **Note**: if you modify the default SQL query, make sure that the key column(s) are included in the field list. This is required to provide the edit/delete functionality.
>
> If a table participates in a Master-Details relationship, make sure the link fields (primary/foreign key) stay on the list of fields.



To switch between the tables, use the **Tables list** panel on the left.

Limit data to first "N" rows

This feature allows you to limit the number of records to be displayed on the **List**, **Print**, **Export**, **Details** pages. It can be useful if you need to speed up the loading of the webpage; or when your chart has too many items in it, and you need to show only the most significant ones. It also works with the **List** pages added to the Dashboard.

The search and filters are applied first, and then the results are limited to first "N" rows.

This option works with all PHPRunner project items except with the reports that have group fields selected.

What is supported

Aliases

When you connect to databases like DB2, Oracle or PostgreSQL, and your SQL query contains aliases, we recommend to enclose them in double-quotes. Here is an example:

```
select FieldName as "FieldAlias"
from TableName
```

If the field was assigned an alias in the SQL query, then the values array gets the alias instead of the field name from the database. E.g., if you have an SQL query *SELECT salesrep_id AS Inv_Salesrep ...*, you should use *values("Inv_Salesrep")*.

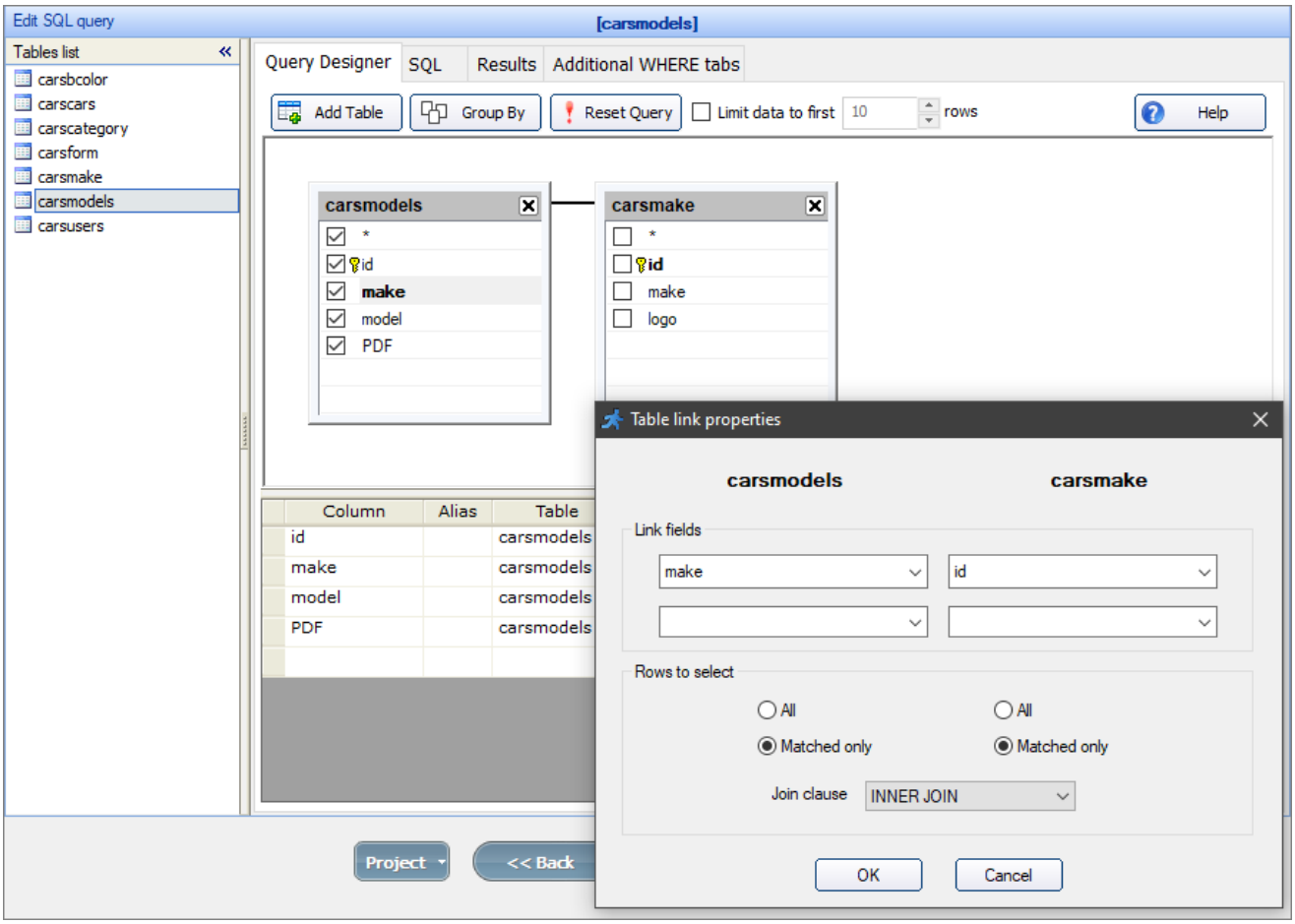
Note: we do not recommend using aliases to give a field another name. If you have long or complicated field names, you can assign a label to the field on the Choose fields page or in the Label editor instead of using aliases.

Inner joins, outer joins

To add a join, click the **Add Table** button, select the table, and then drag-n-drop the fields from the first table to the joined table.

Note: when dragging a field, other fields of the same type are highlighted in **bold**.

To set up the join clause type, double-click on the line between the tables, select the link fields in the **Table link properties** dialog for both tables, and choose the join type.



Note: to learn more about join types, see Using JOIN SQL queries.

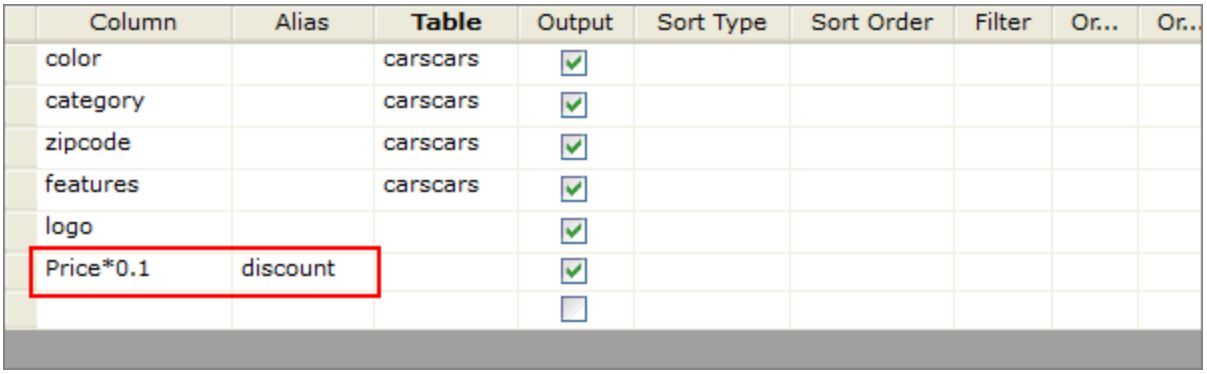
SQL query:

```
SELECT
carsmodels.id,
carsmodels.model,
carsmodels.make
FROM carsmodels
INNER JOIN carsmake ON carsmodels.make = carsmake.id
```

Note: it's recommended to use aliases for fields from joined tables to avoid confusion when two fields from different tables have the same name.

Calculated fields

To add calculated fields, use the empty cells below the field names:



Column	Alias	Table	Output	Sort Type	Sort Order	Filter	Or...	Or...
color		carscars	☑					
category		carscars	☑					
zipcode		carscars	☑					
features		carscars	☑					
logo			☑					
Price*0.1	discount		☑					
			☐					

SQL query:

```
SELECT
category,
color,
Date Listed,
descr,
EPACity,
EPAHighway,
features,
UserID,
YearOfMake,
zipcode,
Price*0.1 AS Discount
FROM carscars
```

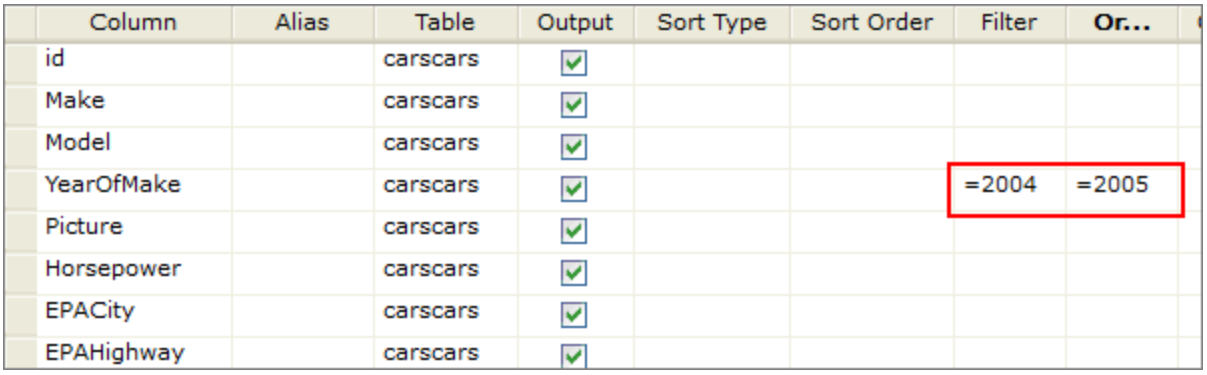
In the example above, the alias *Discount* is assigned to the calculated field *Price*0.1*.

Note: if the field was assigned an alias in the SQL query, then the values array gets the alias instead of the field name from the database. So you should use *values["Discount"]* instead of *values["Price*0.1"]* in your events.

For more information about events, see Events.

WHERE clause

You can add a WHERE clause in the *Filter* column. If you need to add two or more conditions, use the *Or...* columns.



Column	Alias	Table	Output	Sort Type	Sort Order	Filter	Or...
id		carscars	☑				
Make		carscars	☑				
Model		carscars	☑				
YearOfMake		carscars	☑			=2004	=2005
Picture		carscars	☑				
Horsepower		carscars	☑				
EPACity		carscars	☑				
EPAHighway		carscars	☑				

SQL query:

```
SELECT *
FROM carscars
WHERE YearOfMake =2004
```

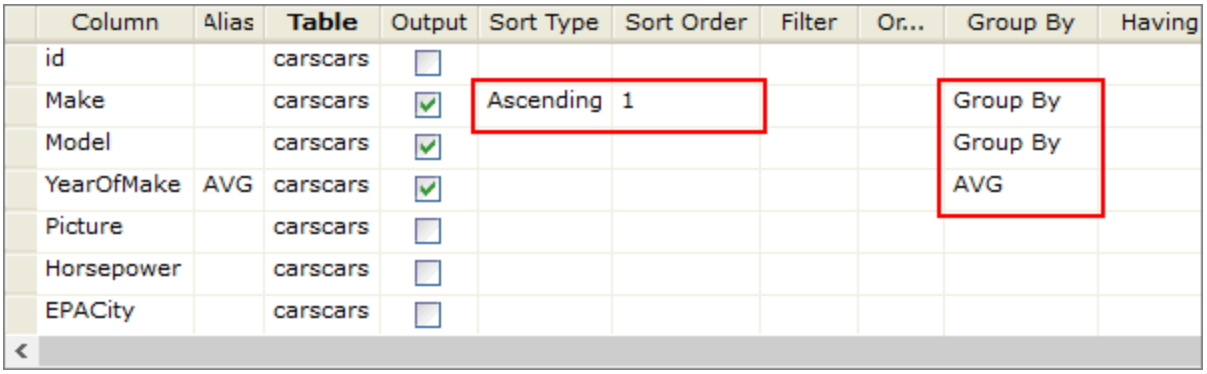
For more complicated queries, wrap the condition with parentheses:

```
SELECT *
FROM carscars
WHERE ( YearOfMake =2004 OR YearOfMake =2005 )
```

ORDER BY/GROUP BY clause

If you'd like to specify the default sorting order on the **List** page (ascending or descending), select the **Sort Type** in the corresponding column for the necessary fields.

To add a GROUP BY clause, click the **Group By** button and select one of grouping function in the *Group By* column.



Column	Alias	Table	Output	Sort Type	Sort Order	Filter	Or...	Group By	Having
id		carscars	☐						
Make		carscars	☑	Ascending	1			Group By	
Model		carscars	☑					Group By	
YearOfMake	AVG	carscars	☑					AVG	
Picture		carscars	☐						
Horsepower		carscars	☐						
EPACity		carscars	☐						

SQL query:

```
SELECT
Make,
Model,
AVG (YearOfMake)
FROM carscars
GROUP BY Make, Model
ORDER BY Make
```

What is not supported

- Stored procedure calls;
- Update/Delete/Insert/Create queries;
- Unions;
- DISTINCT keyword.

> **Note**: If your query doesn't work for some reason, create a view/query in your database and use this query as a datasource in PHPRunner.

Here is how you can create a query in MS Access (other database types provide similar options):

1. Run MS Access and create a new query. Switch to **SQL** view.

2. Insert your SQL query there.

3. Save this query as *qryNewQuery*.

4. Run PHPRunner and use *qryNewQuery* as a datasource table.

See also:

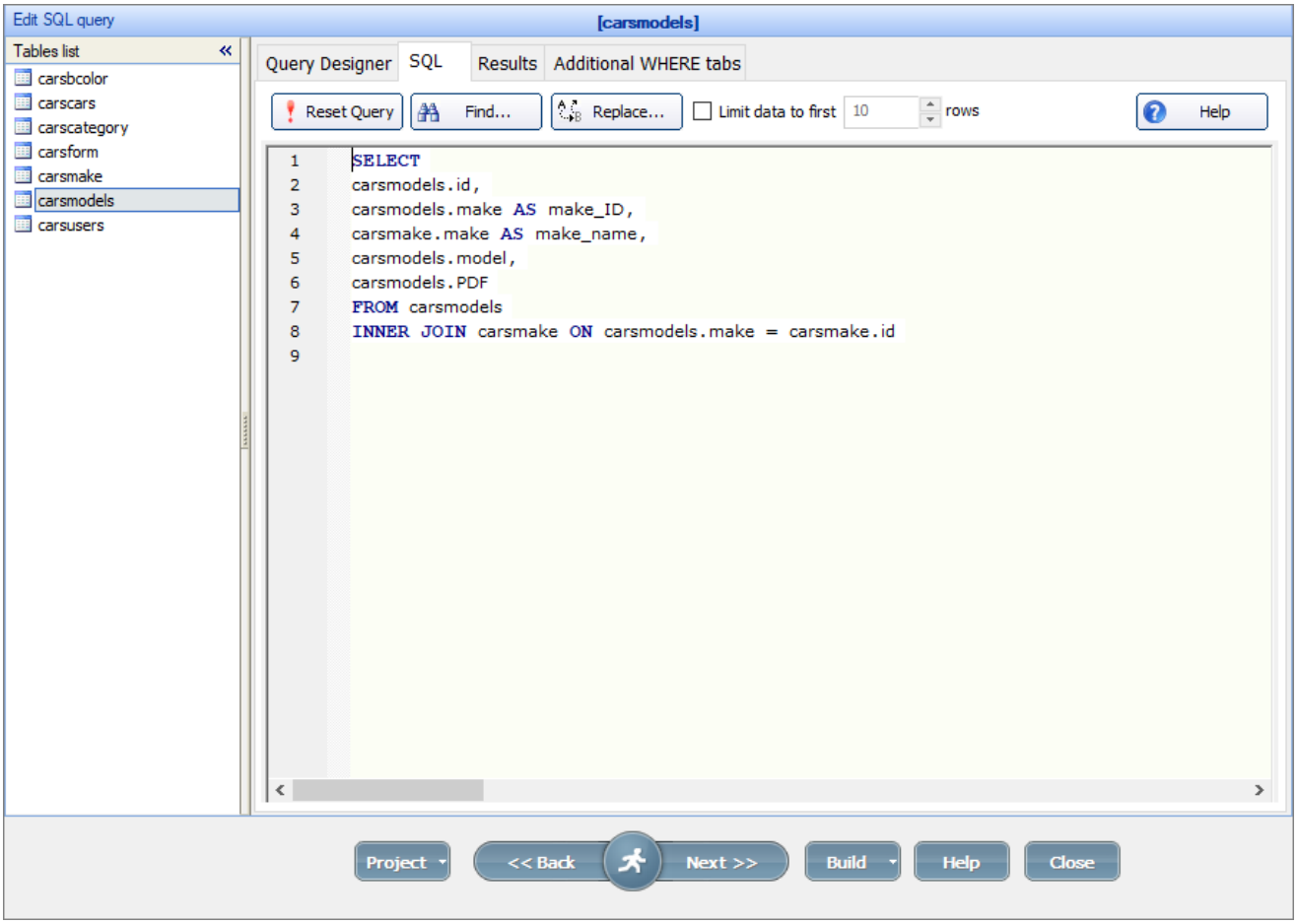
- About SQL query
- SQL tab
- Results tab
- Additional WHERE tabs
- Using JOIN SQL queries

2.8.3 SQL tab

Quick jump

About the SQL tab

The **SQL** tab allows you to modify the SQL queries manually. The changes in this tab automatically appear in the **Query Designer** tab.



To switch between the tables, use the **Tables list** panel on the left.

Limit data to first "N" rows

This feature allows you to limit the number of records to be displayed on the **List**, **Print**, **Export**, **Details** pages. It can be useful if you need to speed up the loading of the webpage; or when your chart has too many items in it, and you need to show only the most significant ones. It also works with the **List** pages added to the Dashboard.

The search and filters are applied first, and then the results are limited to first "N" rows.

This option works with all PHPRunner project items except with the reports that have group fields selected.

What is supported

Aliases

When you connect to databases like DB2, Oracle or PostgreSQL, and your SQL query contains aliases, we recommend to enclose them in double-quotes. Here is an example:

```
select FieldName as "FieldAlias"
from TableName
```

If the field was assigned an alias in the SQL query, then the values array gets the alias instead of the field name from the database. E.g., if you have an SQL query *SELECT salesrep_id AS Inv_Salesrep ...*, you should use *values("Inv_Salesrep")*.

Note: we do not recommend using aliases to give a field another name. If you have long or complicated field names, you can assign a label to the field on the Choose fields page or in the Label editor instead of using aliases.

Inner joins, outer joins

Note: to learn more about join types, see Using JOIN SQL queries.

SQL query:

```
SELECT
carsmodels.id,
carsmodels.model,
carsmodels.make
FROM carsmodels
INNER JOIN carsmake ON carsmodels.make = carsmake.id
```

Note: it's recommended to use aliases for fields from joined tables to avoid confusion when two fields from different tables have the same name.

Calculated fields

SQL query:

```
SELECT
category,
color,
Date Listed,
descr,
EPACity,
EPAHighway,
features,
UserID,
YearOfMake,
zipcode,
Price*0.1 AS Discount
FROM carscars
```

In the example above, the alias *Discount* is assigned to the calculated field *Price*0.1*.

Note: if the field was assigned an alias in the SQL query, then the values array gets the alias instead of the field name from the database. So you should use *values["Discount"]* instead of *values["Price*0.1"]* in your events.

For more information about events, see Events.

WHERE clause

SQL query:

```
SELECT *
FROM carscars
WHERE YearOfMake =2004
```

For more complicated queries, wrap the condition with parentheses:

```
SELECT *
FROM carscars
WHERE ( YearOfMake =2004 OR YearOfMake =2005 )
```

ORDER BY/GROUP BY clause

SQL query:

```
SELECT
Make,
Model,
AVG (YearOfMake)
FROM carscars
GROUP BY Make, Model
ORDER BY Make
```

What is not supported

- Stored procedure calls;
- Update/Delete/Insert/Create queries;
- Unions;
- DISTINCT keyword.

> **Note**: If your query doesn't work for some reason, create a view/query in your database and use this query as a datasource in PHPRunner.

Here is how you can create a query in MS Access (other database types provide similar options):

1. Run MS Access and create a new query. Switch to **SQL** view.

2. Insert your SQL query there.

3. Save this query as *qryNewQuery*.

4. Run PHPRunner and use *qryNewQuery* as a datasource table.

See also:

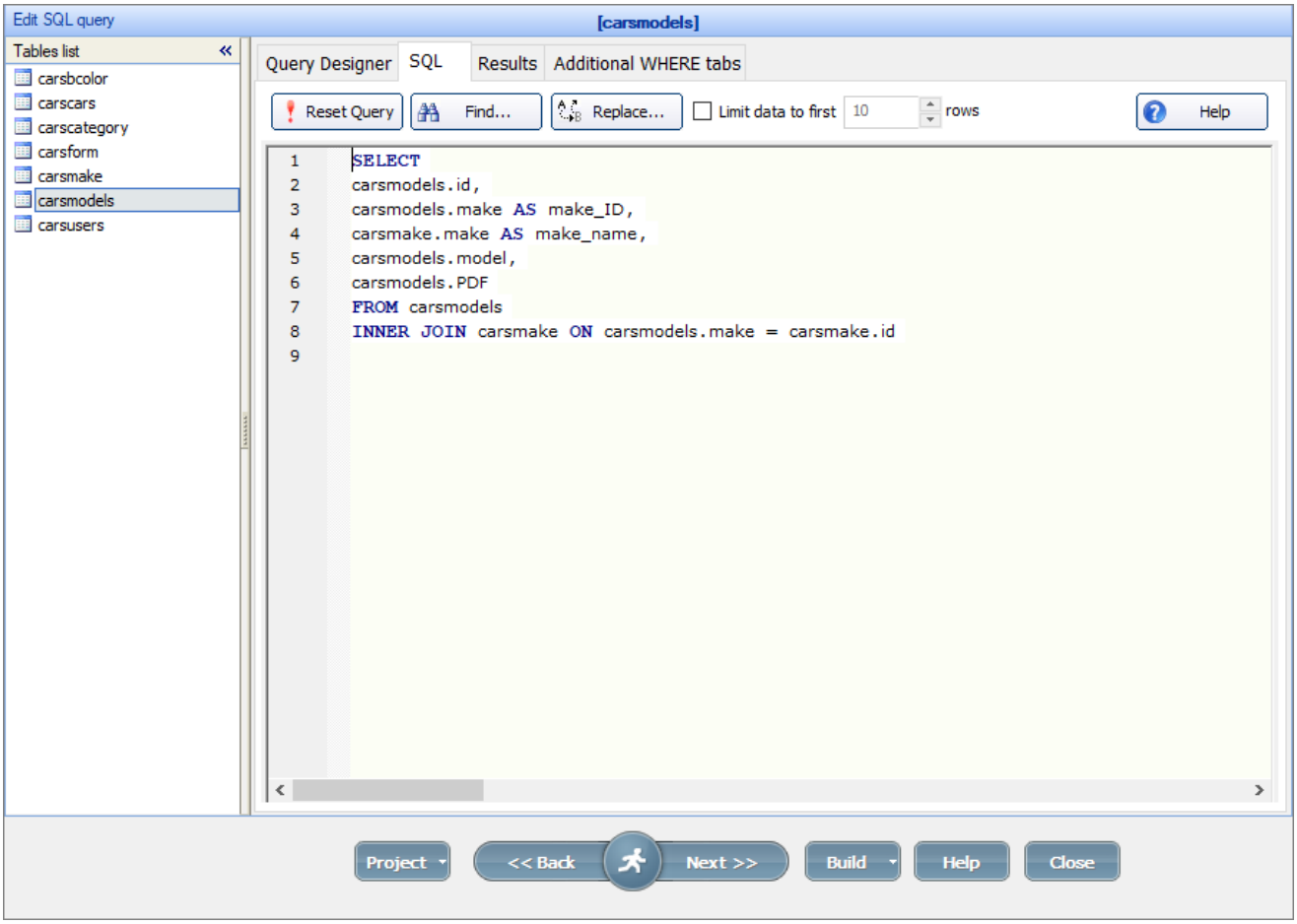
- About SQL query
- Query Designer tab

- Results tab
- Additional WHERE tabs
- Using JOIN SQL queries

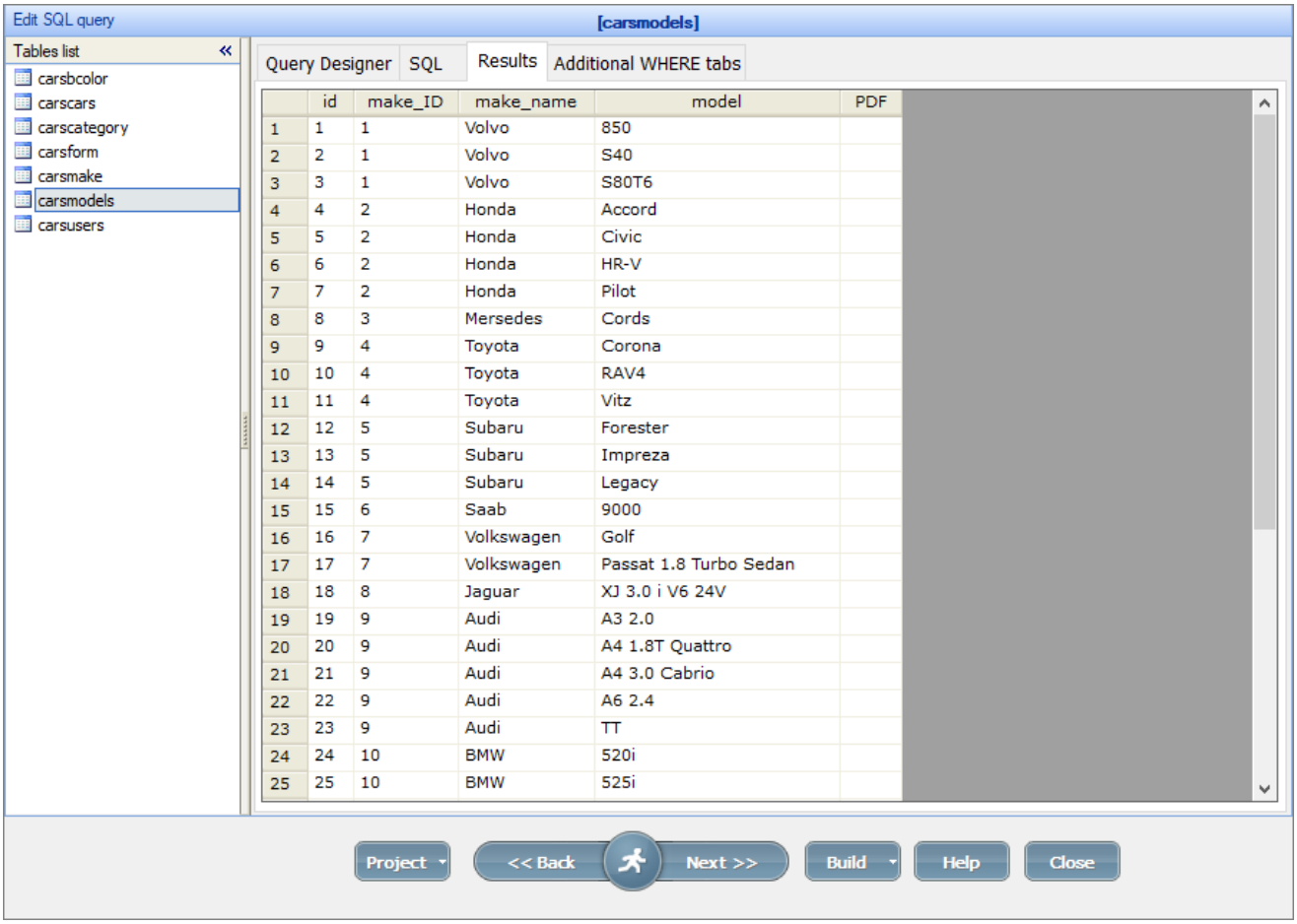
2.8.4 Results tab

The **Results** tab lets you see the results of the SQL query created with the Query Designer or SQL tab.

If you, for example, take the *carsmodels* table and modify the SQL query to get the following:



Clicking on the **Results** tab shows the results of the query above:



You may use the **Tables list** panel on the left to switch between tables.

> **Note**: the **Results** tab shows up to 200 records per table.

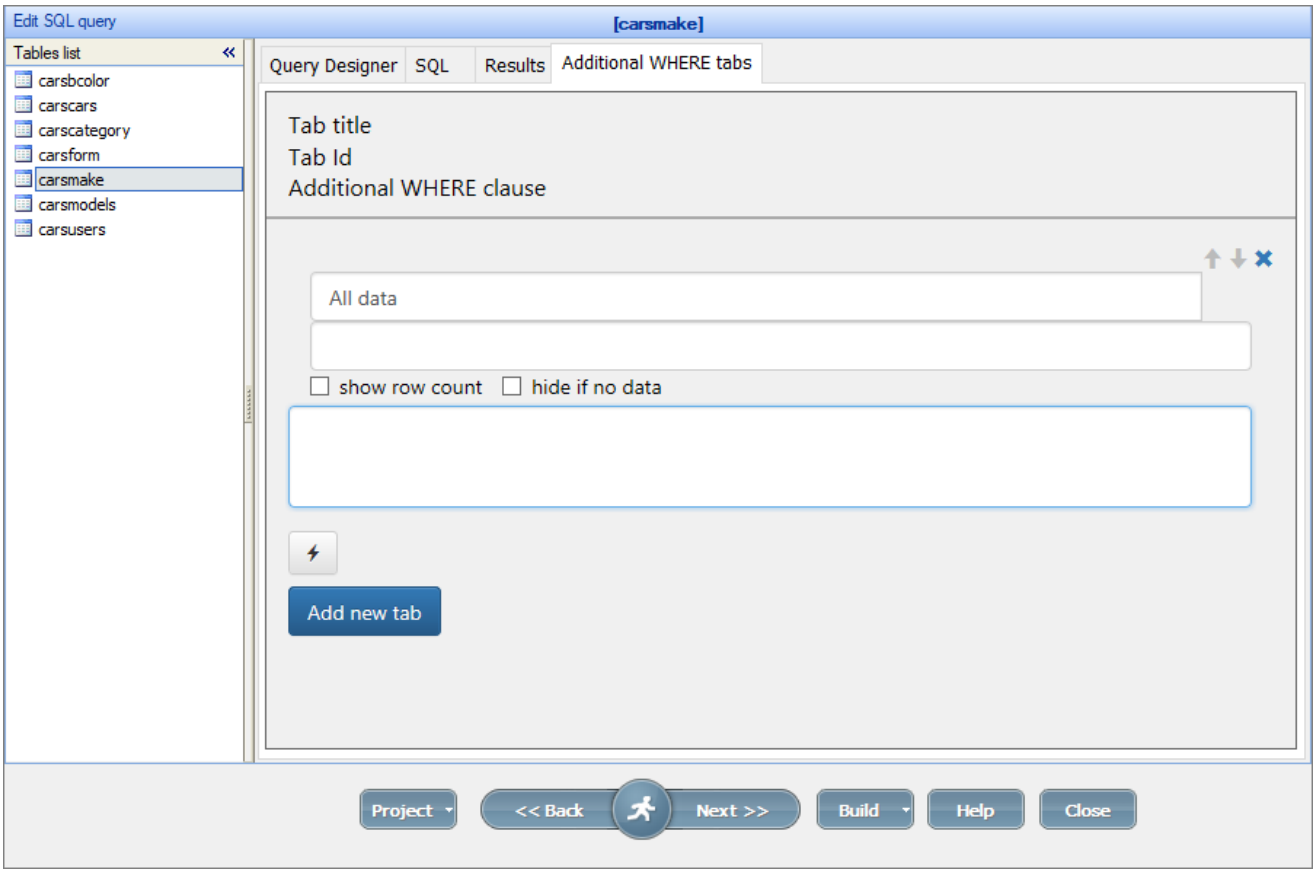
See also:

- About SQL query
- Query Designer tab
- SQL tab
- Additional WHERE tabs

2.8.5 Additional WHERE tabs

Tabs and additional WHERE clauses

This screen lets you create additional tabs that show the records limited by WHERE clauses.



The interface consists of several fields, checkboxes and buttons:

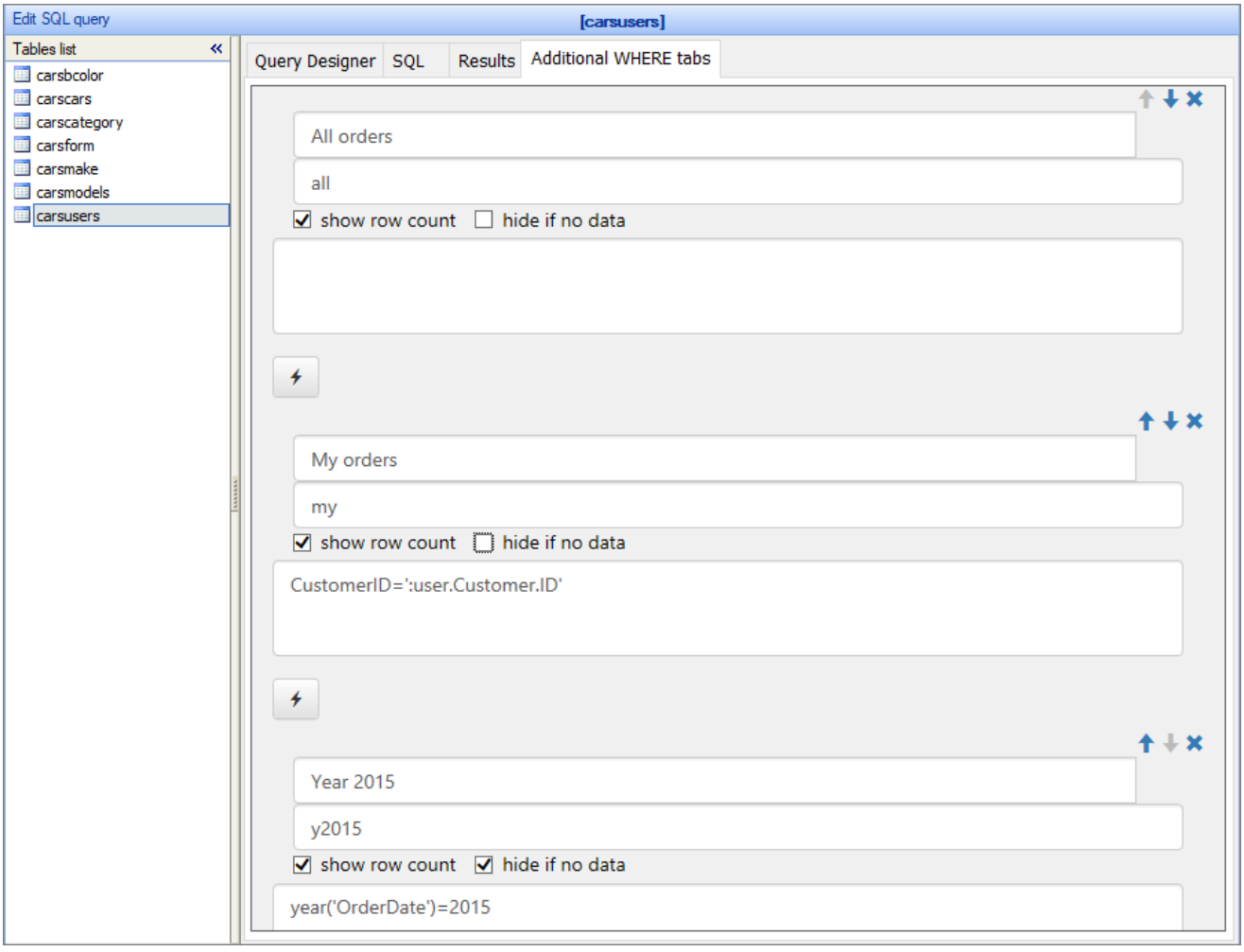
- The first field is the **Label** of the tab;
- The second field is the **ID** of the tab;
- The text box is where you add the WHERE clauses;
- The **show row count** checkbox shows the number of rows in the tab next to the label;
- The **hide if no data** checkbox hides the tab if there is no data for the resulting query;
- The **button with a lightning icon** shows the results of the WHERE clause in the Results tab;

- Click the **Add new tab** button to add a new tab;
- Use the **Up**/**Down**/**Remove** buttons in the upper-right corner of the tab panel to reorder or remove the tabs.

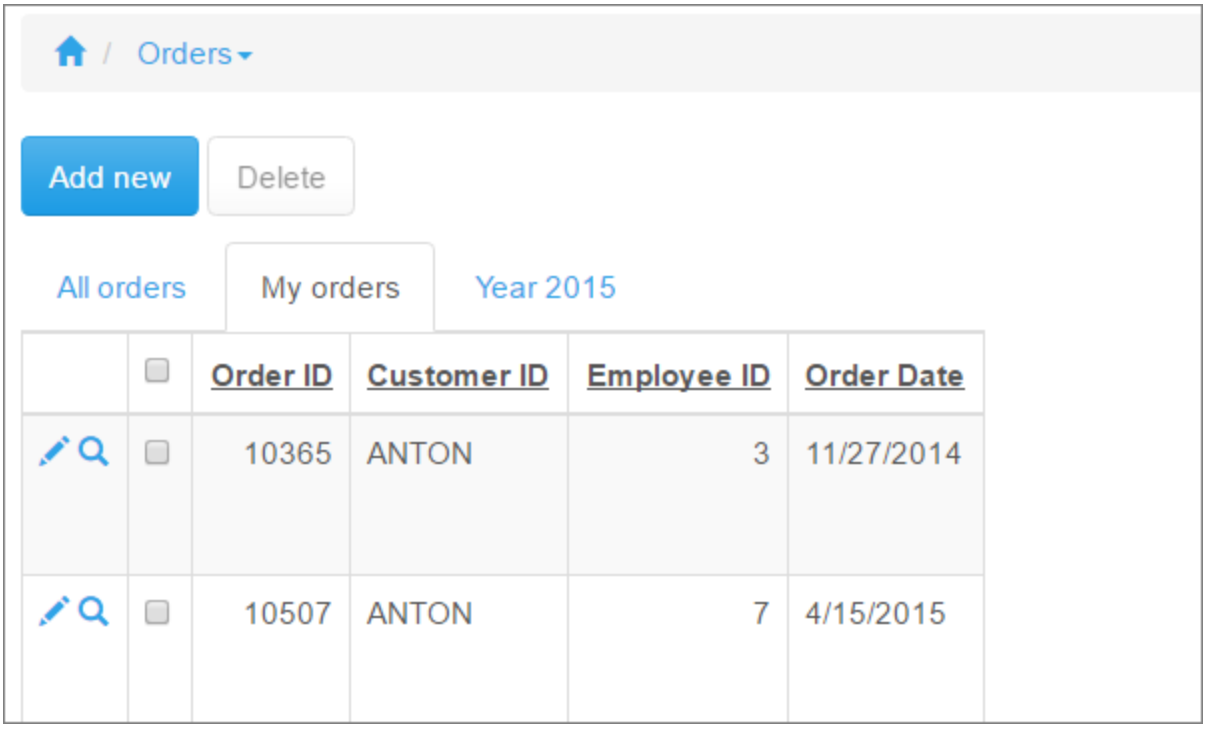
> **Note**: you can also use the SQL variables in the WHERE clause.

Example

Here is an example of two additional tabs called *My orders* and *Year 2015* for the *Orders* table:



Here is how it looks like in the generated app:



Sample WHERE expressions

- "phone like '%555%'"
- "city like 'M%'"
- "salary>50000"

Using functions to create and manage WHERE tabs

PHPRunner lets you create and manage the WHERE tabs using the following PHP functions:

- create a tab dynamically with the addTab() function;
- delete the tab with the deleteTab() function;
- change the title of the tab with the setTabTitle() function;
- change the WHERE clause of the tab with the setTabWhere() function.

See also:

- About SQL query

- Query Designer tab
- SQL tab
- Additional WHERE tabs
- SQL variables in the WHERE tabs

2.9 Charts

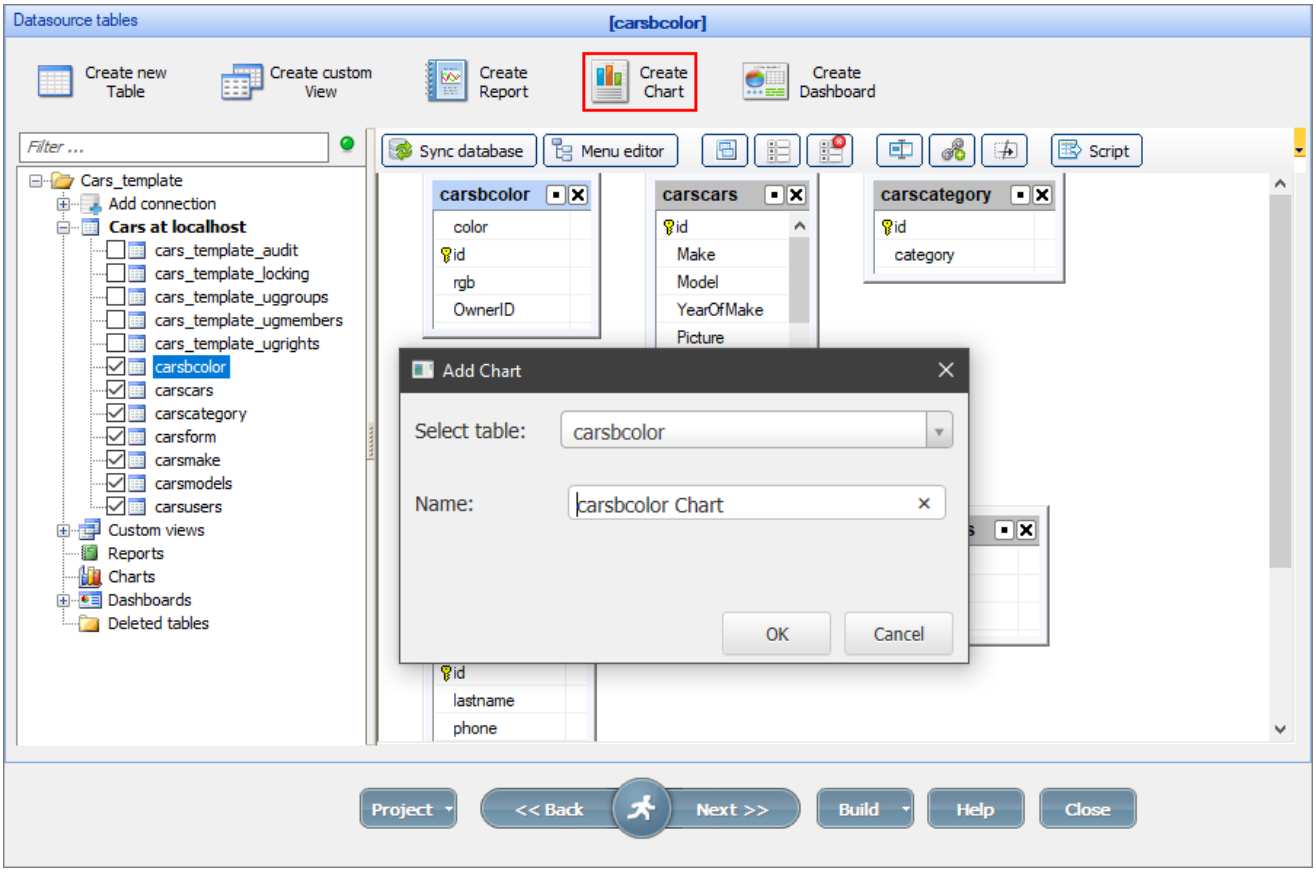
2.9.1 Creating charts

Creating a chart

To create a chart:

1. Proceed to the Datasource tables screen and click **Create Chart**.

2. Select the table and set the chart name. Click **OK**.

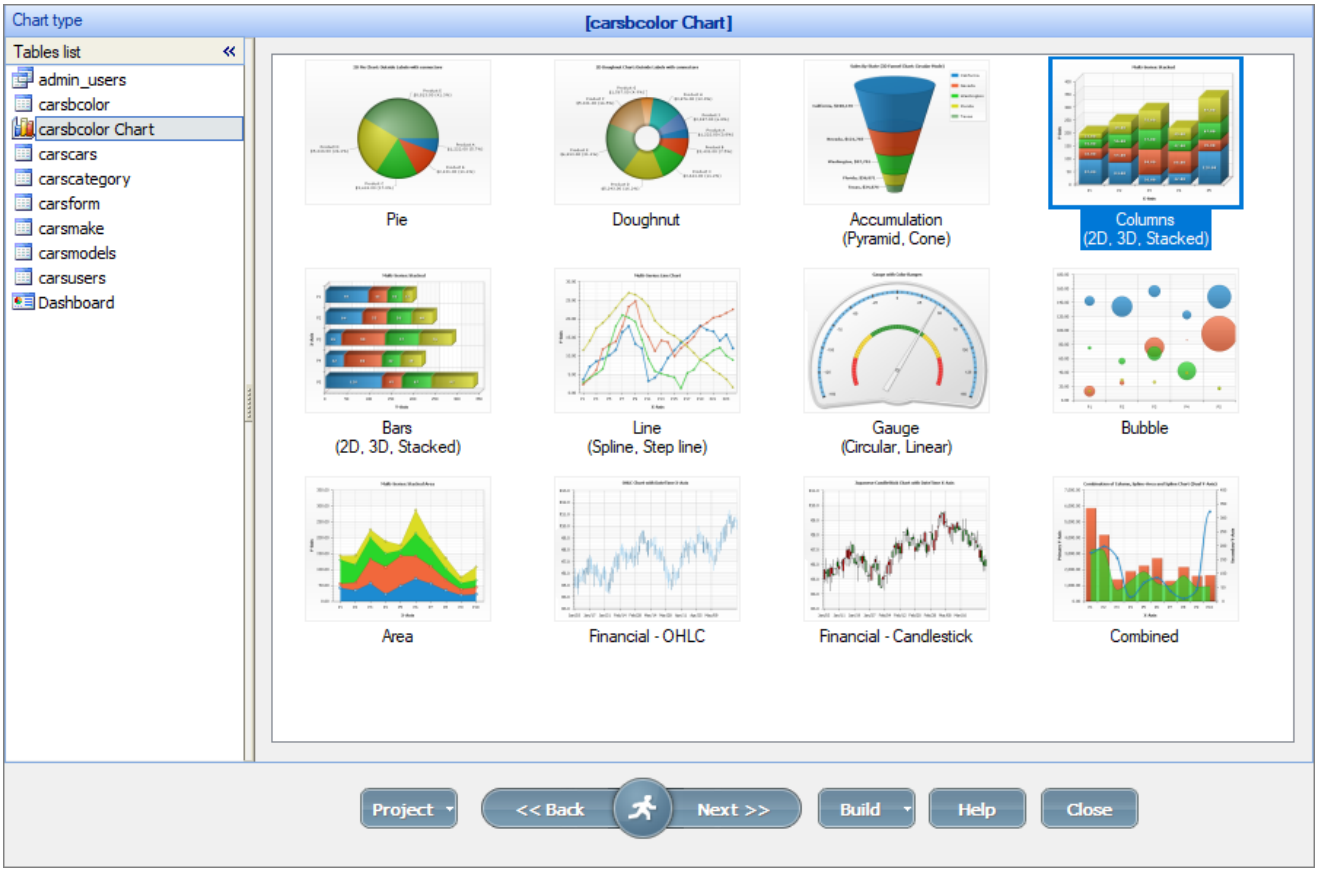


Note: you can create a copy of an existing chart (right-click the chart and select **Copy**).

Modifying the chart

On the next several screens (use the **Next** button to navigate) you can:

- make the changes to the SQL query. More info about editing SQL queries;
- select the type of chart to build. More info about chart types;

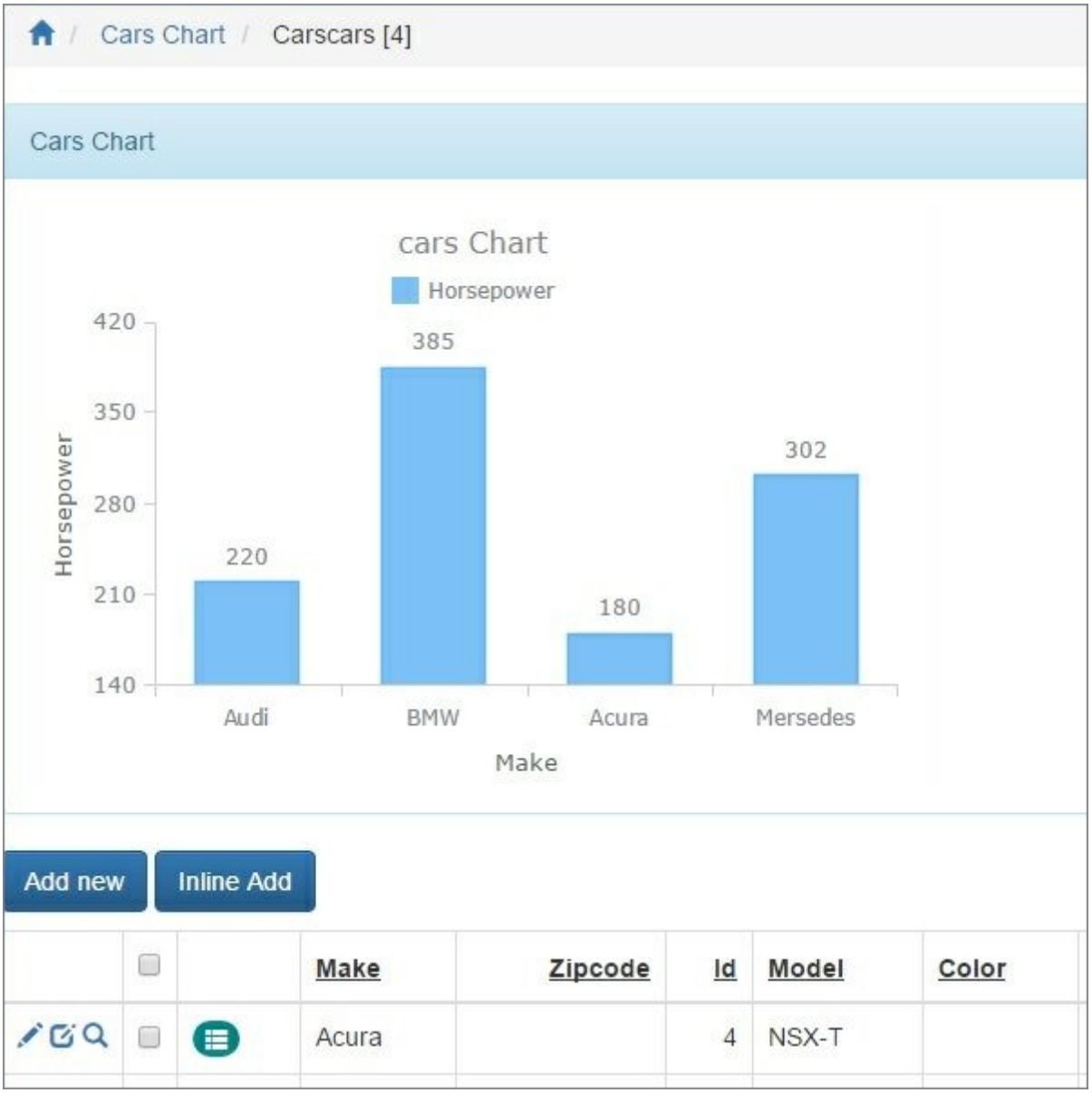


- choose the Data Series fields (more info about setting chart parameters);
- modify the chart appearance options.

> **Note**: you can use charts with both master and details tables. For more information, see Master-details relationship between tables.

A chart as a details table:

A chart as a master table:



To further customize the chart appearance like colors, fonts, or the chart title, use the ChartModify event.

See also:

- A list of chart types
- Chart parameters
- Chart appearance

- Using SQL to shape chart data
- Datasource tables
- Master-details relationship
- Creating and configuring reports
- Creating dashboards

2.9.2 Chart types

2.9.2.1 A list of chart types

A list of chart types

Chart type	Single-series	Multi-series	Horizontal	Vertical	3D
Accumulation	Yes	No	Yes	No	No
Area	Yes	Yes	Yes	No	No
Bubble	Yes	Yes	Yes	No	Yes
Column/Bar	Yes	Yes	Column	Bar	Yes
Combined	Yes	Yes	Yes	No	No
Financial OHLC/Candlestick	Yes	Yes	-	-	No
Gauge	Yes	No	-	-	No
Line	Yes	Yes	Yes	No	No
Pie/Doughnut	Yes	Yes	-	-	No

See also:

- Creating charts
- Chart parameters
- Chart appearance
- Using SQL to shape chart data

2.9.2.2 Pie/Doughnut charts

Description

A **Pie** chart is a circular chart that is divided into sectors, illustrating percentages. A **Doughnut** chart is identical to the **Pie** chart, except for having an empty center.

You can create single-series and multi-series **Pie**/**Doughnut** charts by choosing one or several Data series fields.

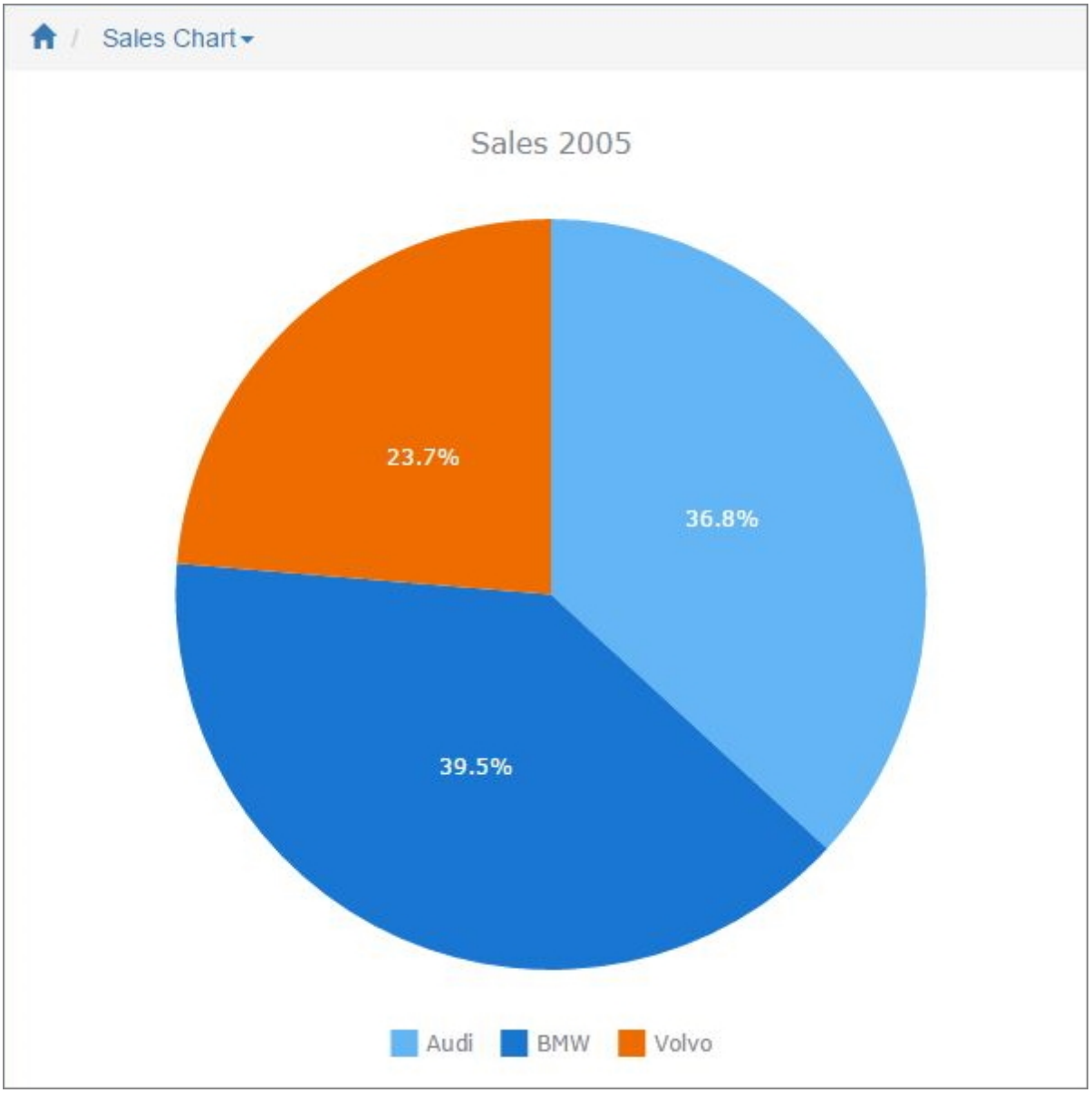
Examples

Sample data table:

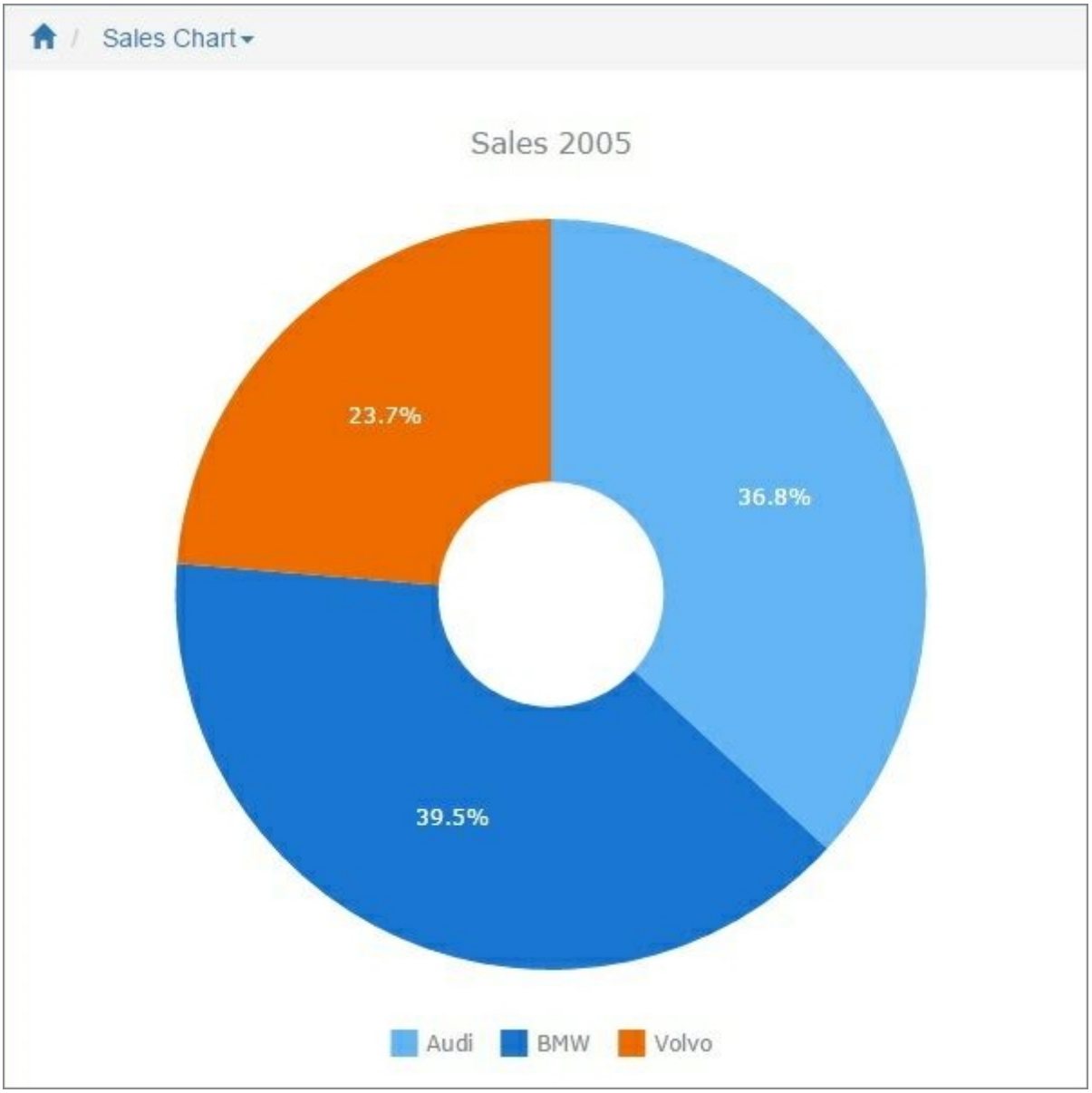
Make	Sales2005
Audi	14000
BMW	15000
Volvo	9000

In the examples, we chose *Sales2005* as the **Data Series** field, *Make* as the **Label** field.

Pie chart



Doughnut chart



Other chart types:

- Accumulation chart
- Column/Bar charts
- Line chart
- Gauge chart
- Bubble chart
- Area chart

- Financial OHLC/Candlestick charts
- Combined charts

See also:

- Creating charts
- A list of chart types
- Chart parameters
- Chart appearance
- Using SQL to shape chart data

2.9.2.3 Accumulation chart

Description

Accumulation charts are single-series charts that represent data in percentages. This type of charts does not utilize axes. The height of a chart segment is proportional to the y-axis value of the corresponding point.

Chart settings

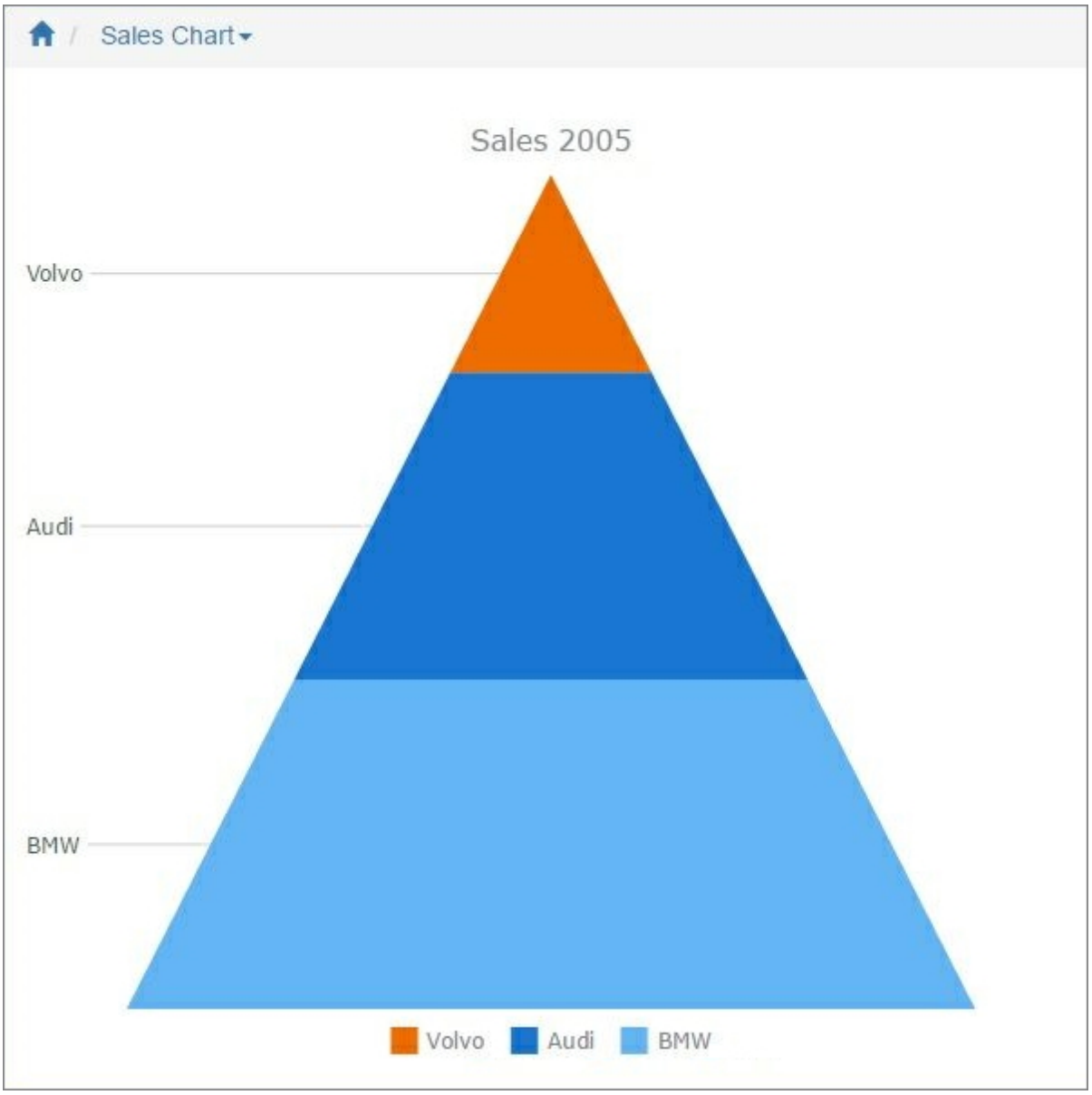
- **Accumulation inverted** - enable this option to make the chart image appear inverted.

Example

Sample data table:

Make	Sales2005
BMW	15000
Audi	14000
Volvo	9000

In this example, we chose *Sales2005* as the Data Series field, *Make* as the **Label** field.



Other chart types:

- Pie/Doughnut charts
- Column/Bar charts
- Line chart
- Gauge chart
- Bubble chart

- Area chart
- Financial OHLC/Candlestick charts
- Combined charts

See also:

- Creating charts
- A list of chart types
- Chart parameters
- Chart appearance
- Using SQL to shape chart data

2.9.2.4 Column/Bar charts

Description

A **Column**/**Bar** chart is a chart with rectangular bars. The height/length of the bars is proportional to the magnitudes/frequencies of the data they represent.

The **Column** chart has vertical bars. With **Column** charts, the categories are typically organized along the horizontal axis and values - along the vertical axis.

The **Bar** chart has horizontal bars. With **Bar** charts, the categories are typically organized along the vertical axis and values - along the horizontal axis.

Consider using a **Bar** chart when:

- the axis labels are long;
- the values represent durations.

You can create single-series (one Data series field and **Label** field selected) or multi-series (two or more **Data series** fields selected) **Column**/**Bar** charts.

Chart settings

- **Chart 3D** - this option allows building a 3D (three-dimensional) chart. With this option disabled, a 2D (two-dimensional) chart is built.
- **Chart stacked** - this option allows building a stacked chart where a single bar on the chart shows more than one category of data. The stacked chart requires two or more **Data series** fields selected.

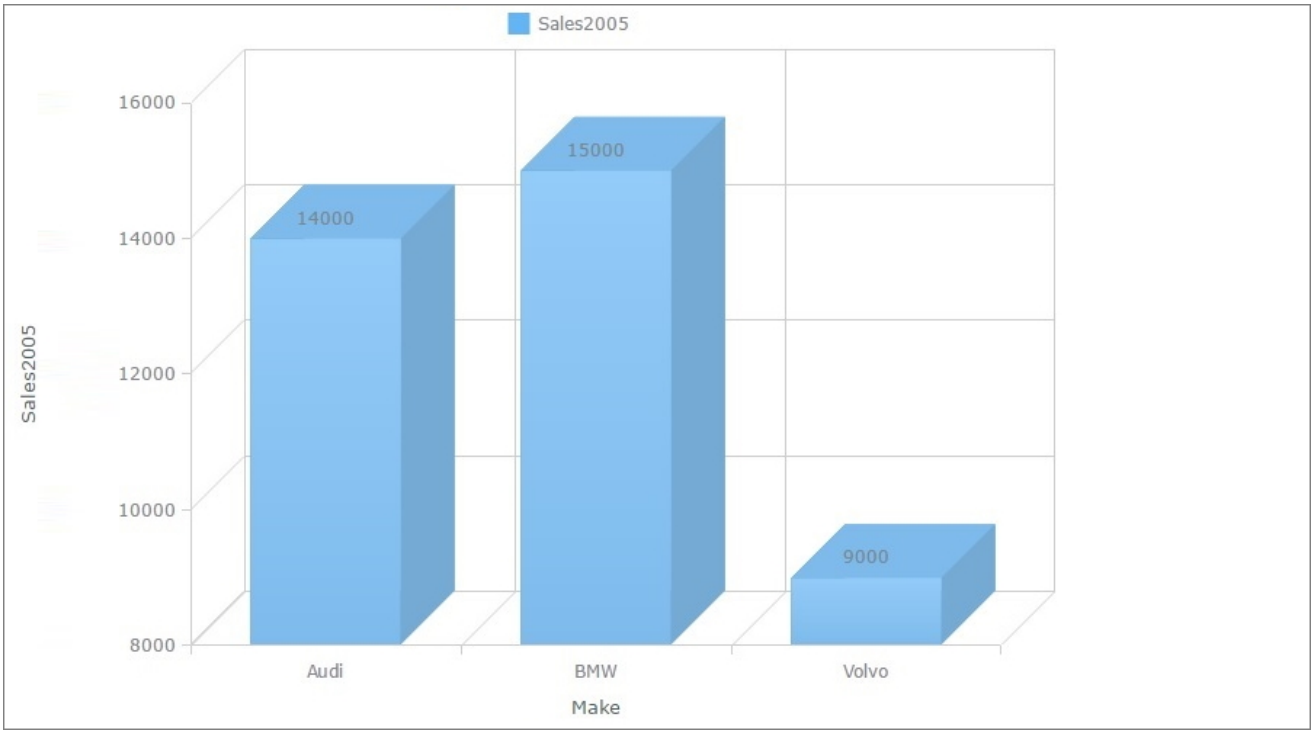
Examples

Sample data table:

Make	Sales2004	Sales2005
Audi	10000	14000
BMW	14000	15000
Volvo	10000	9000

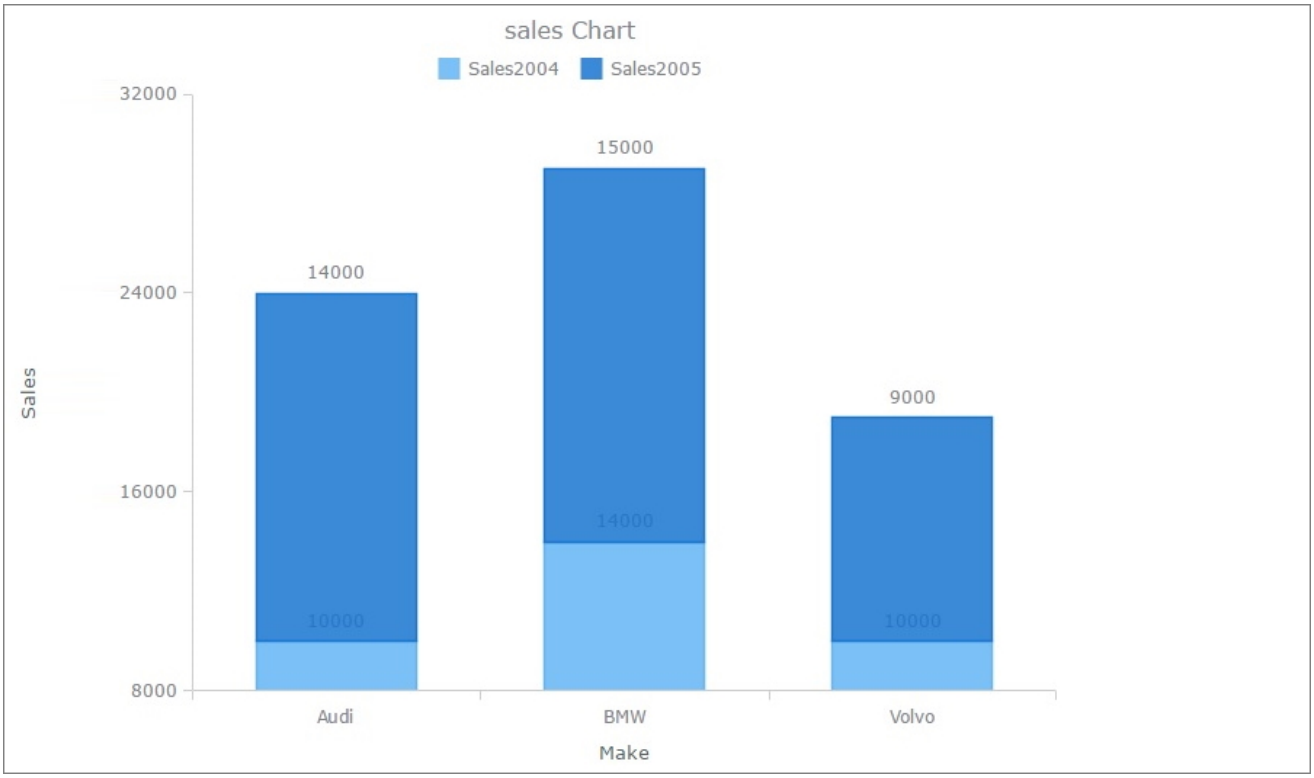
A 3D column chart

In this example, we chose *Sales2005* as the **Data Series** field, *Make* as the **Label** field.



A stacked column chart

In this example, we chose *Sales2004* and *Sales2005* as the **Data Series** fields, *Make* as the **Label** field.



Other chart types:

- Pie/Doughnut charts
- Accumulation chart
- Line chart
- Gauge chart
- Bubble chart
- Area chart
- Financial OHLC/Candlestick charts
- Combined charts

See also:

- Creating charts
- A list of chart types
- Chart parameters

- Chart appearance
- Using SQL to shape chart data

2.9.2.5 Line chart

Description

A **Line** chart displays information as a series of data points connected by line segments. The **Line** chart is often used to visualize a trend over of time.

With **Line** charts, the category data is distributed evenly along the horizontal axis, and the value data is distributed evenly along the vertical axis.

You can create single-series or multi-series **Line** charts.

Chart settings

- **Line style** - this option defines the line segments style (normal, spline, step).

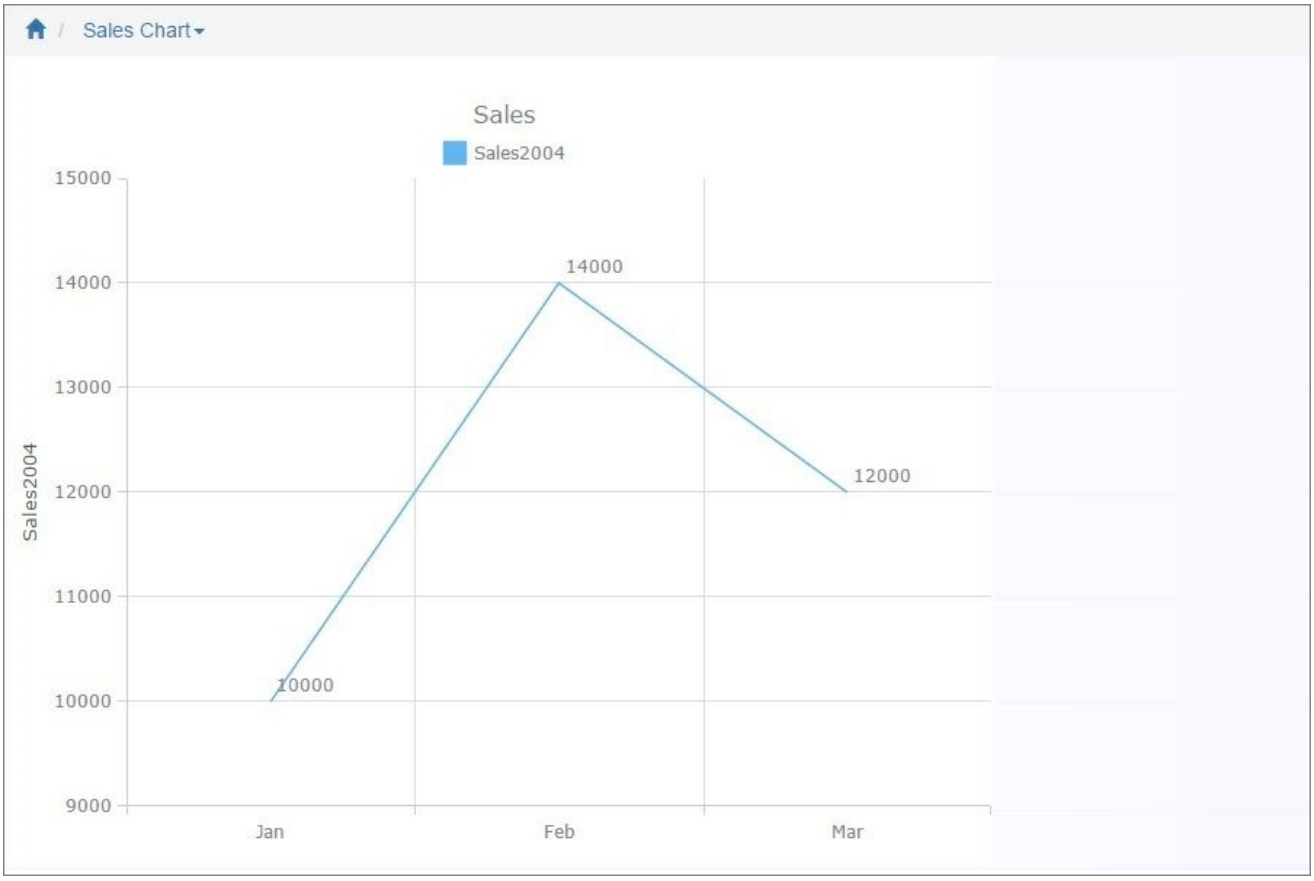
Examples

Sample data table:

Month	Sales2004	Sales2005
Jan	10000	14000
Feb	14000	15000
Mar	12000	9000

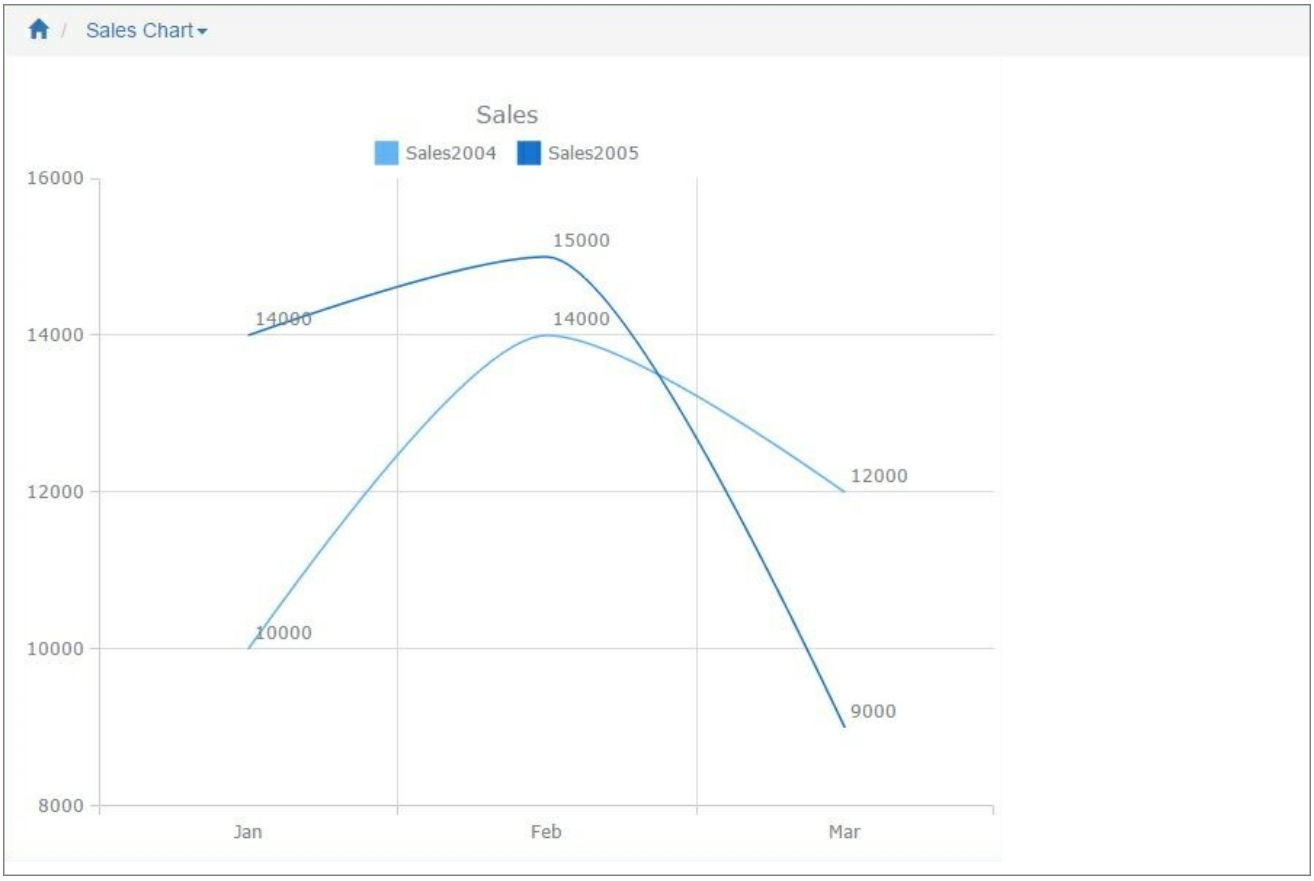
Single series normal chart

In this example, we chose *Sales2004* as the **Data Series** field, *Month* as the **Label** field.



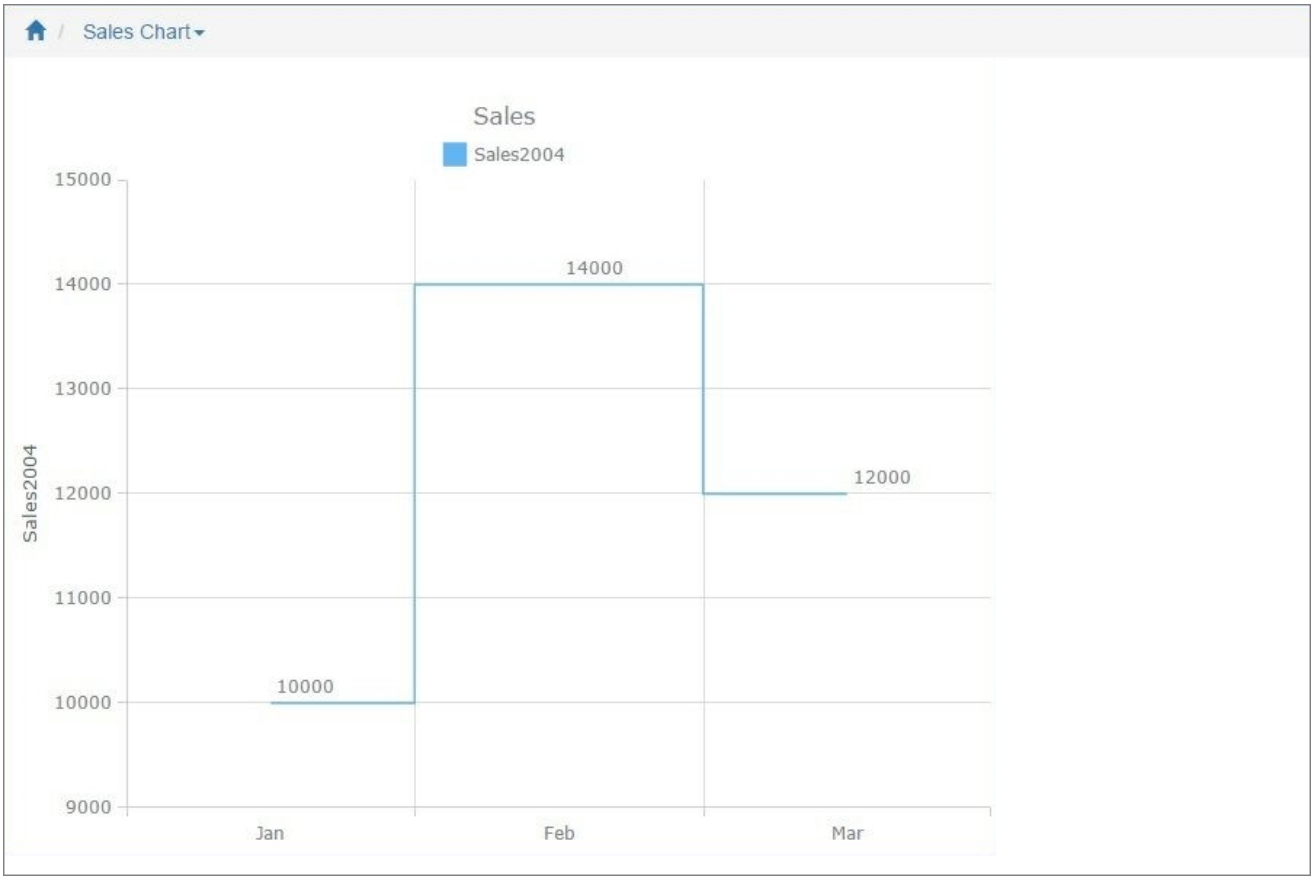
Multi-series spline chart

In this example, we chose *Sales2004* and *Sales2005* as the **Data Series** fields, *Month* as the **Label** field.



Single series step chart

In this example, we chose *Sales2004* as the **Data Series** field, *Month* as the **Label** field.



Other chart types:

- Pie/Doughnut charts
- Accumulation chart
- Column/Bar charts
- Gauge chart
- Bubble chart
- Area chart
- Financial OHLC/Candlestick charts
- Combined charts

See also:

- Creating charts

- A list of chart types
- Chart parameters
- Chart appearance
- Using SQL to shape chart data

2.9.2.6 Gauge chart

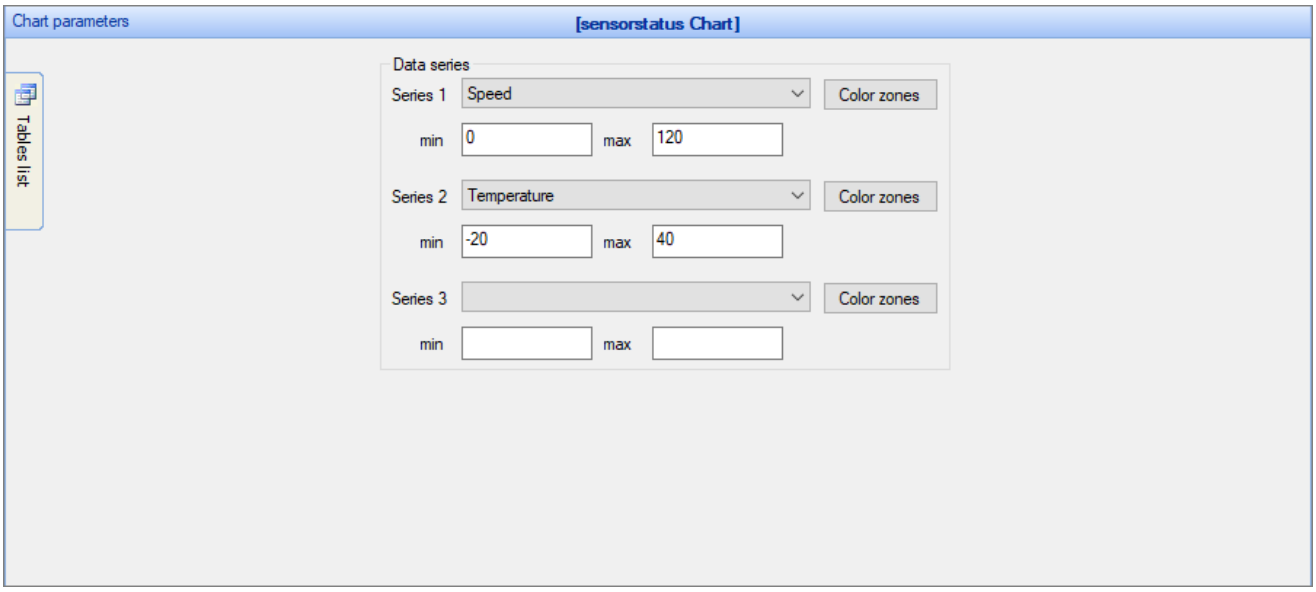
Description

A **Gauge** chart presents a value on a graduated scale or dial. There are two **Gauge** chart types: **Circular Gauge** and **Linear Gauge**.

The **Circular Gauge** looks like the gauges on car dashboards. It consists of a radial scale as the data range, and a pointer. The **Circular Gauge** can show different range intervals in different colors.

The **Linear Gauge** works similarly to the **Circular Gauge**, but presents the data in a straight line. It can be either vertical or horizontal.

You can display one or several gauges on a chart by choosing one or several Data series. Also you can define the minimum and maximum values.



Color zones can show different colored intervals on the gauge. You can set the interval colors and their borders.

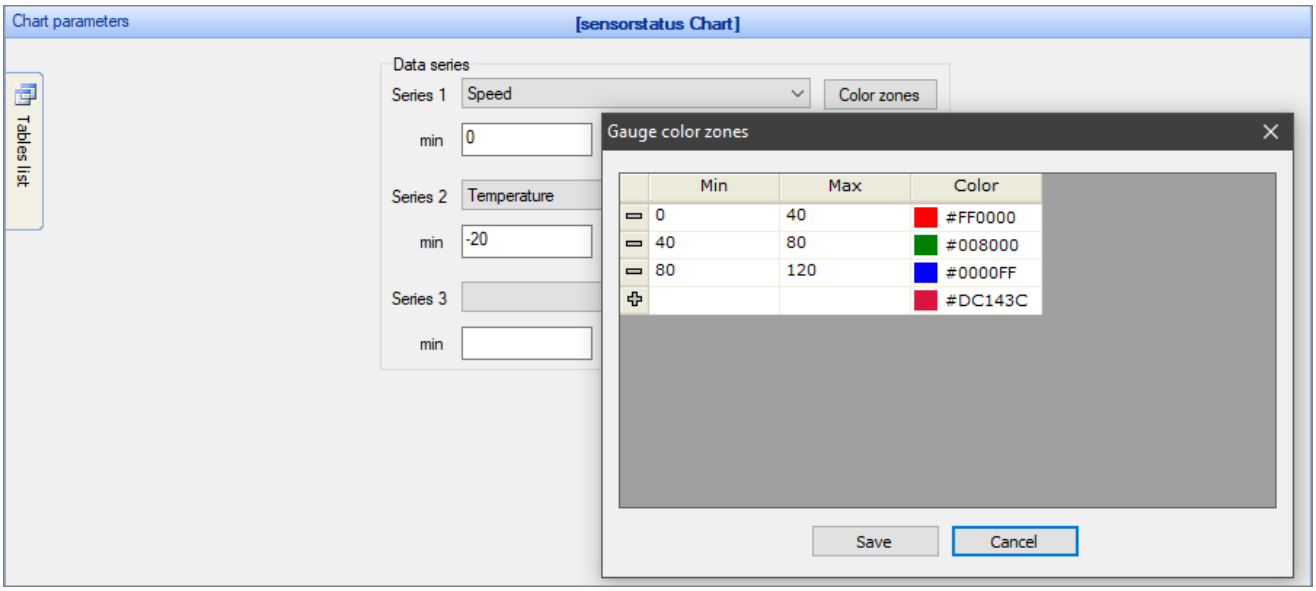
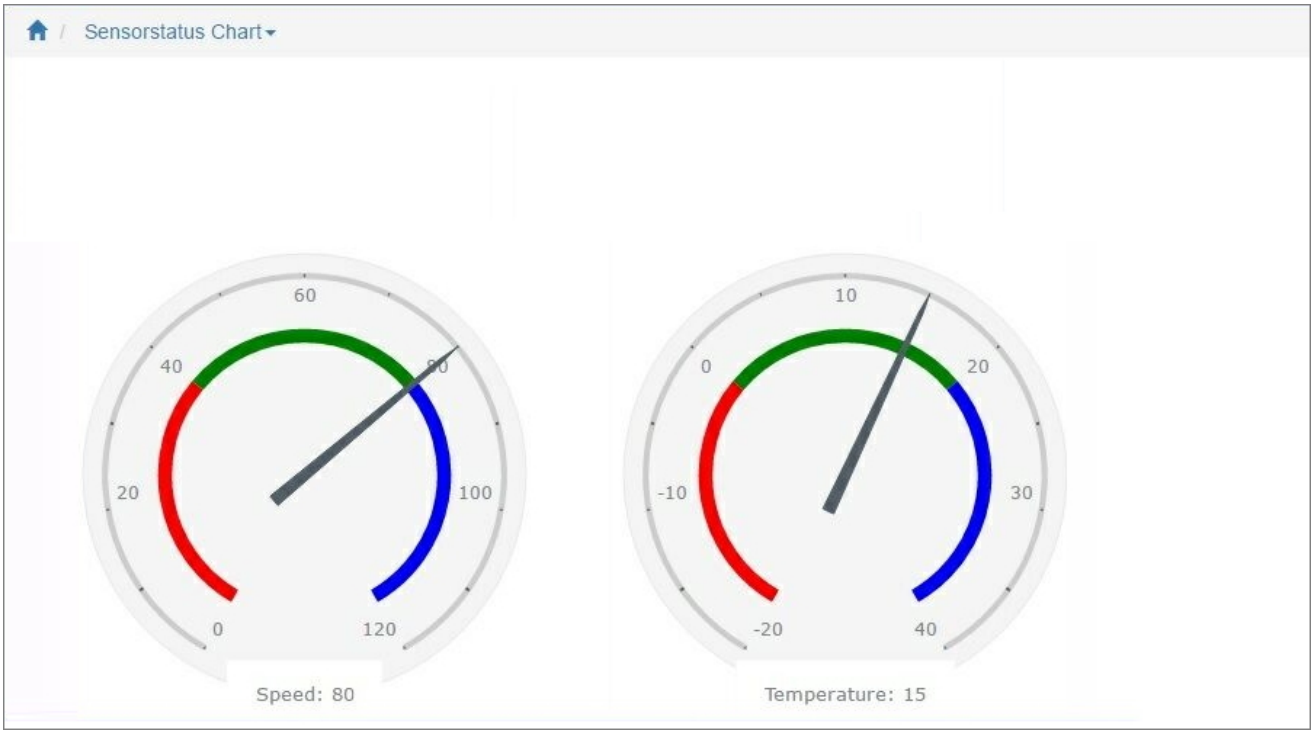


Chart settings

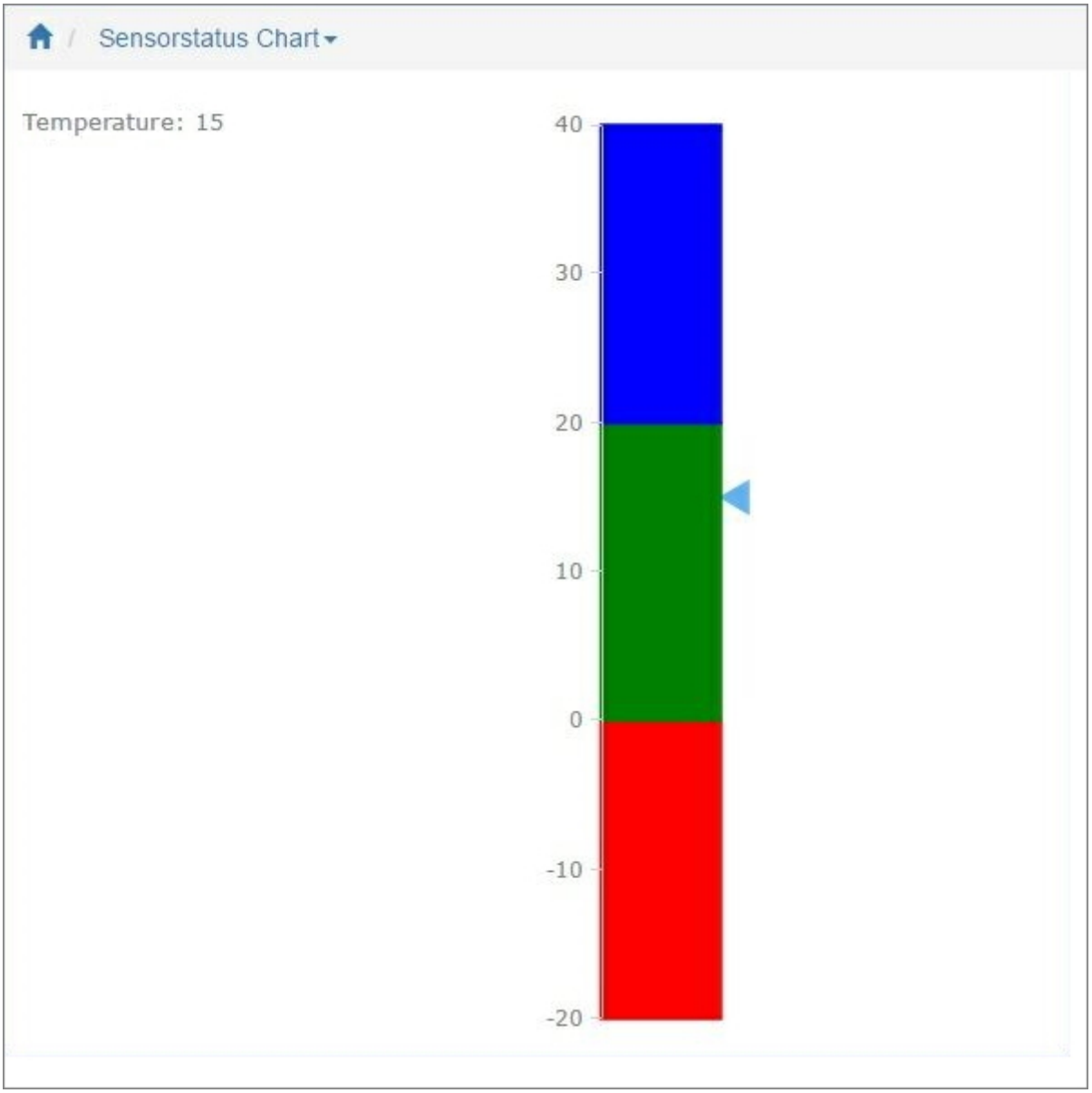
- **Gauge style** - this option defines the gauge appearance (circle, horizontal linear, vertical linear).

Examples of gauge chart types

Circular Gauge chart with two Data series selected



Vertical Linear Gauge chart



Other chart types:

- Pie/Doughnut charts
- Accumulation chart
- Column/Bar charts
- Line chart
- Bubble chart
- Area chart

- Financial OHLC/Candlestick charts
- Combined charts

See also:

- Creating charts
- A list of chart types
- Chart parameters
- Chart appearance
- Using SQL to shape chart data

2.9.2.7 Bubble chart

Description

A **Bubble** chart is a variation of a Scatter chart in which the data pointers are replaced with bubbles. The **Bubble** charts are often used to present financial data. Use the **Bubble** chart when you want to emphasize specific values.

The **Bubble** chart needs 3 values: x, y and size. Depending on the data model and visualization purpose, the bubble chart may contain single series or multi-series data fields.

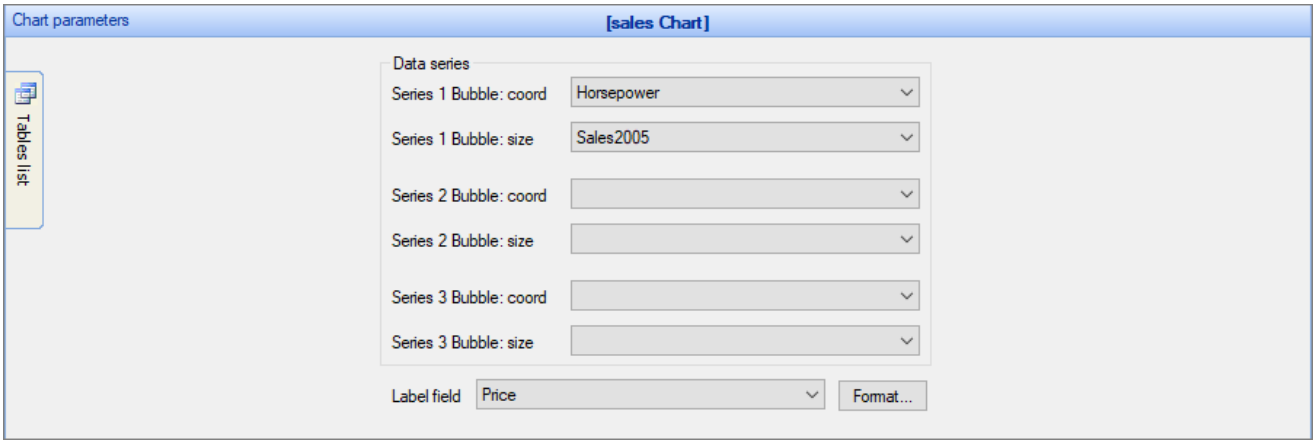


Chart settings

- **Chart 3D** - this option allows building a 3D (three-dimensional) chart. With this option is disabled, a 2D (two-dimensional) chart is built.

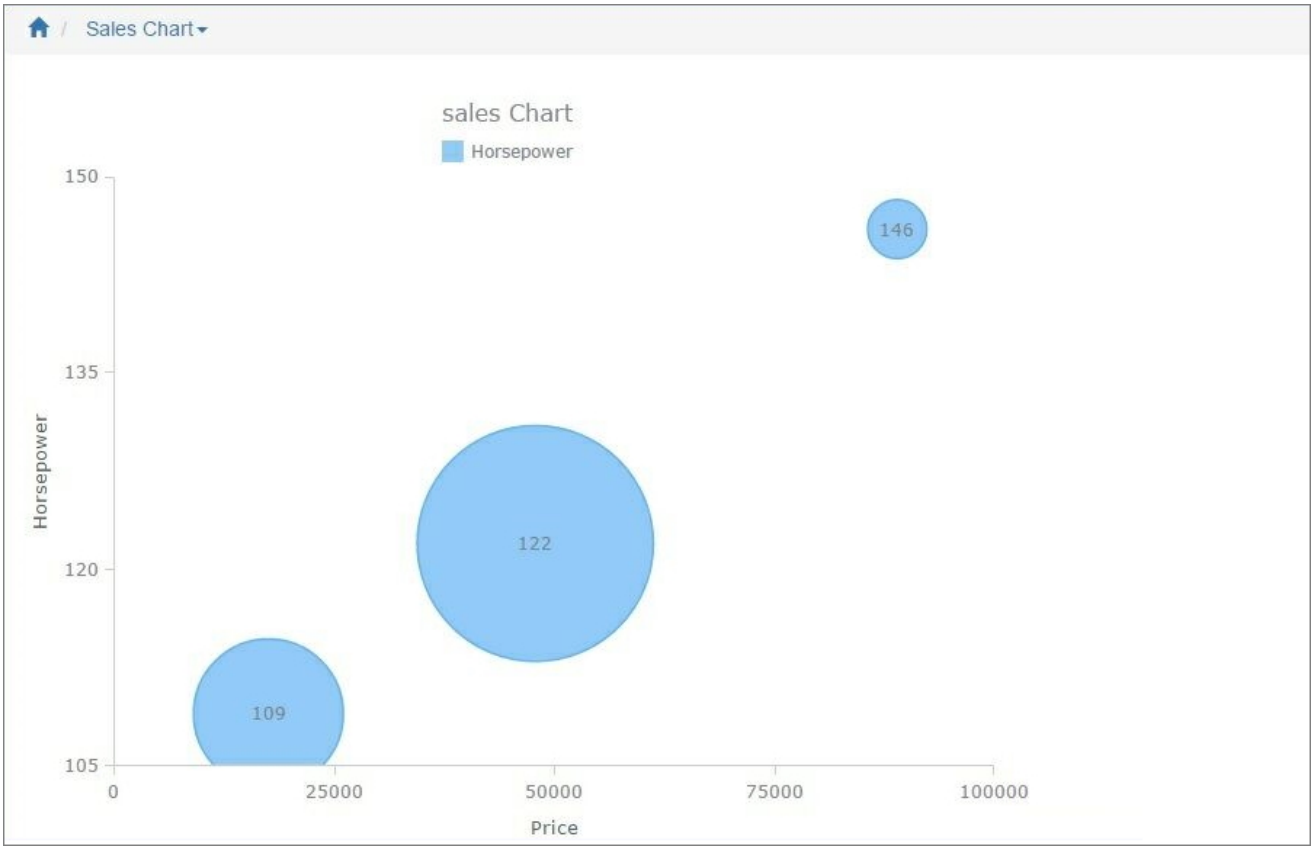
Examples

Example 1

Sample data table:

Make	Horsepower	Sales2005	Price
Audi	109	120000	17599
BMW	146	109500	88999
Volvo	122	130000	47899

In this example, we chose *Horsepower* as the **Y-axis** field, *Sales2005* as the **Bubble size** field, *Price* as the **Label** field (X-axis field).



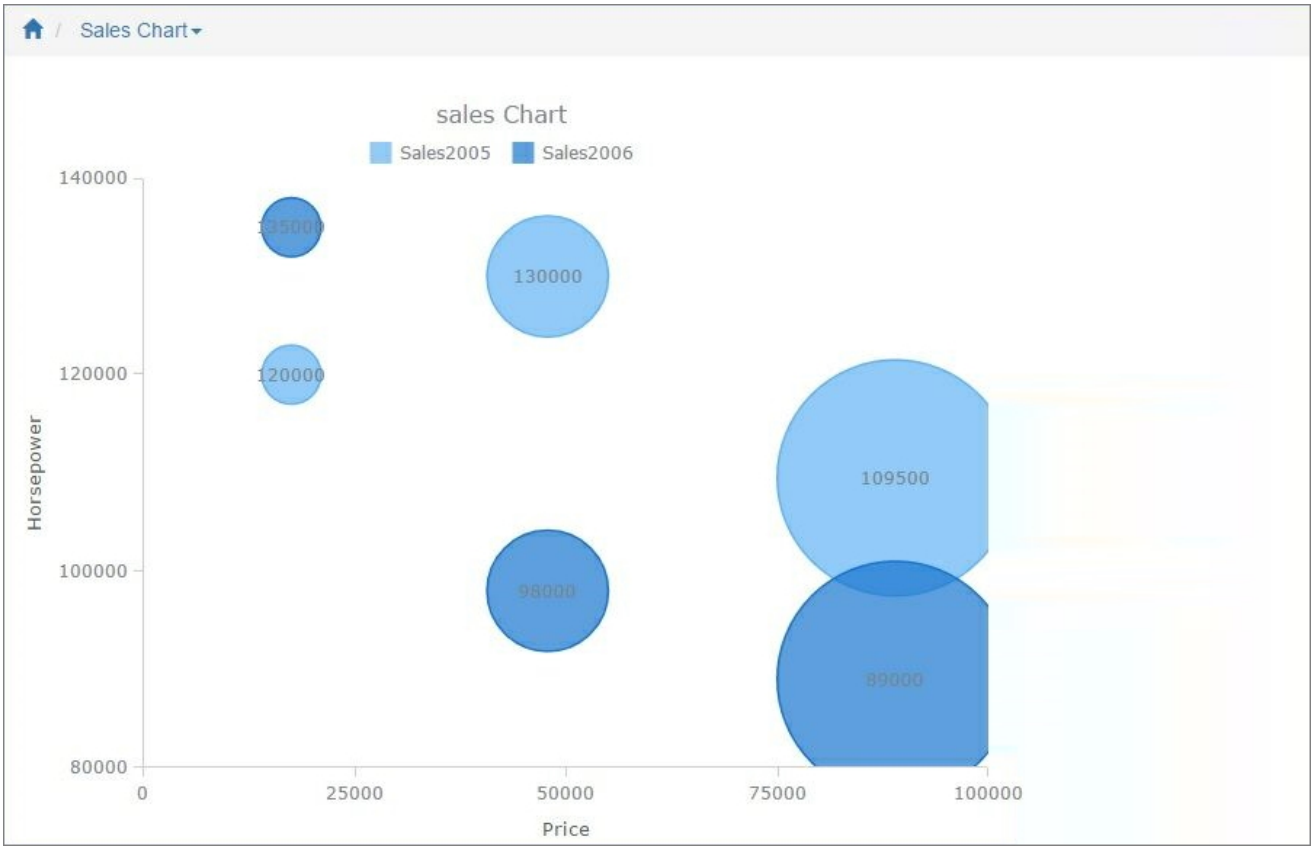
Example 2

Here is an example of a chart with two data series selected.

Sample data table:

Make	Horsepower	Sales2005	Sales2006	Price
Audi	109	120000	135000	17599
BMW	146	109500	89000	88999
Volvo	122	130000	98000	47899

In this example, we chose *Sales2005* and *Sales2006* as **Y-axis** fields, *Horsepower* as the **Bubble size** field, *Price* as the **Label** field (X-axis field).



Other chart types:

- Pie/Doughnut charts
- Accumulation chart
- Column/Bar charts
- Line chart
- Gauge chart
- Area chart
- Financial OHLC/Candlestick charts
- Combined charts

See also:

- Creating charts
- A list of chart types

- Chart parameters
- Chart appearance
- Using SQL to shape chart data

2.9.2.8 Area chart

Description

An **Area** chart is based on the Line chart, but the area between the line and the x-axis is colored. **Area** charts are used to represent the totals over time using numbers or percentages.

You can build single series or multi-series **Area** charts.

Chart settings

- **Chart stacked** - this option allows building stacked chart where a horizontal point on the chart can show several vertical points of data. The **Stacked** chart requires two or more **Data series**.

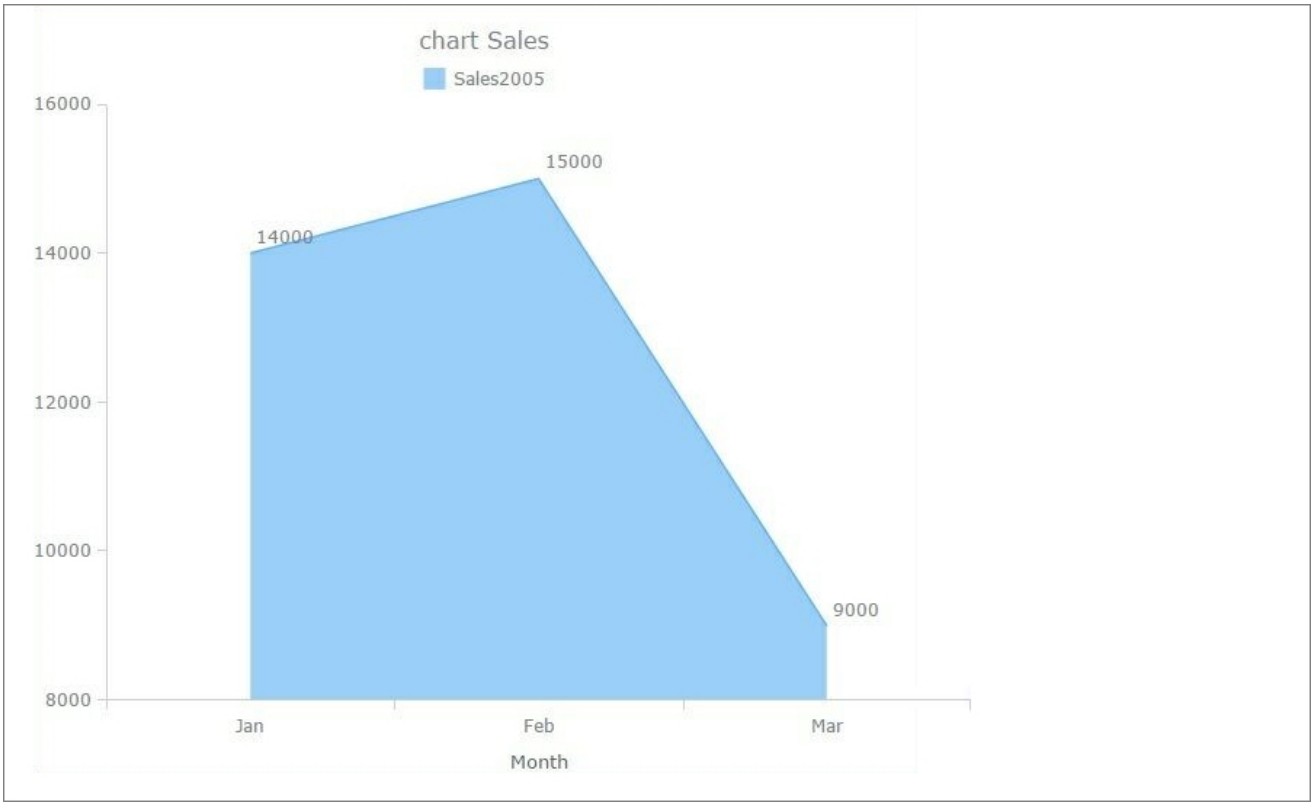
Examples

Sample data table:

Month	Sales2004	Sales2005
Jan	10000	14000
Feb	14000	15000
Mar	12000	9000

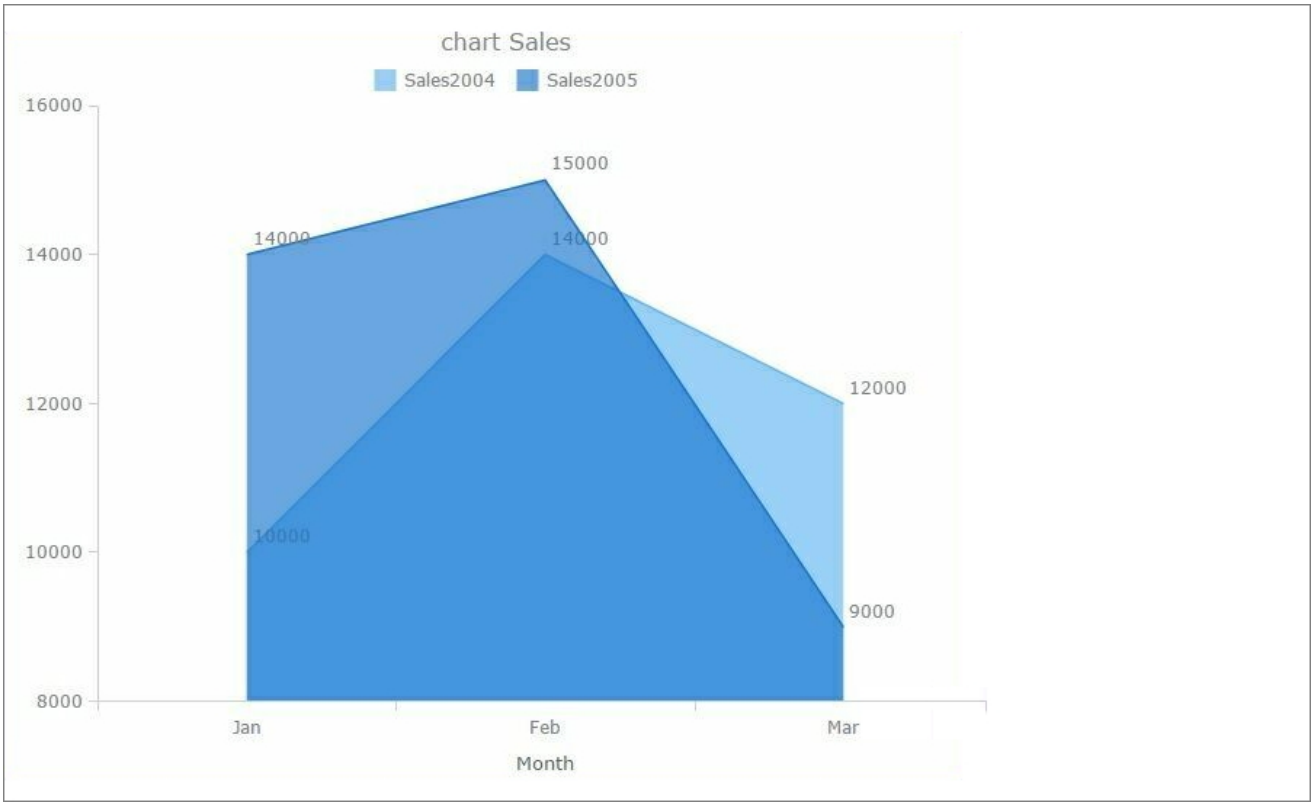
Single series area chart

In this example, we chose *Sales2005* as the **Data Series** field, *Month* as the **Label** field.



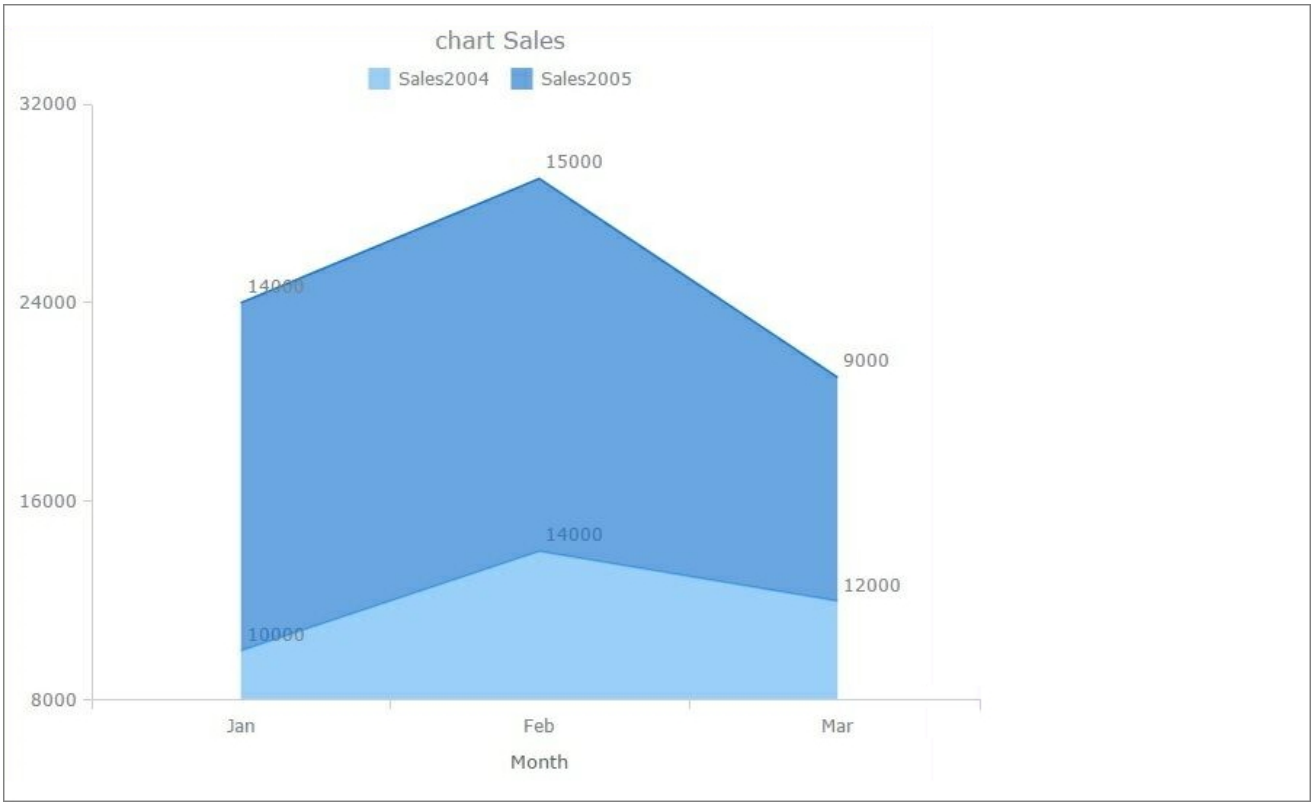
Multi-series area chart

In this example, we chose *Sales2004* and *Sales2005* as the **Data Series** fields, *Month* as the **Label** field.



Multi-series stacked area chart

In this example, we chose *Sales2004* and *Sales2005* as the **Data Series** fields, *Month* as the **Label** field.



Other chart types:

- Pie/Doughnut charts
- Accumulation chart
- Column/Bar charts
- Line chart
- Gauge chart
- Bubble chart
- Financial OHLC/Candlestick charts
- Combined charts

See also:

- Creating charts
- A list of chart types
- Chart parameters

- Chart appearance
- Using SQL to shape chart data

2.9.2.9 Financial OHLC/Candlestick charts

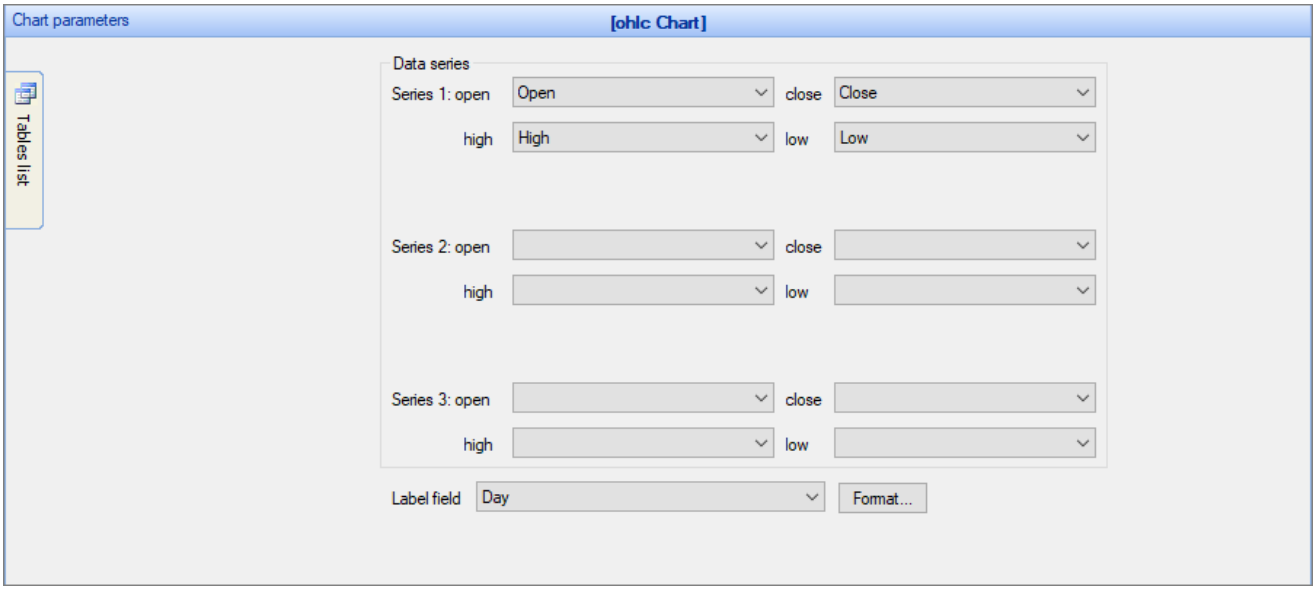
Description

An **OHLC** chart is a type of Bar chart that shows Open, High, Low, and Closing prices for each period. Each vertical line on the chart shows the price range (the highest and lowest prices) over one unit of time, e.g., one day or one hour. Tick marks project from each side of the line indicating the opening price (e.g., for a daily **OHLC** chart this would be the starting price for that day) on the left, and the closing price for that time period on the right.

The **Japanese Candlestick** chart is another way of displaying market price data, with the opening and closing prices defining a rectangle within the range for each time unit. The rectangles have different colors depending on whether prices rose or fell in that period.

Both charts show the exact same data, i.e., the opening, high, low, and closing prices during a particular time frame. Some traders find the candlestick chart easier to read.

OHLC/Candlestick chart may contain single series or multi-series data fields. These charts use four values: open, high, low and closing price values. You also need to select the **Label** field.

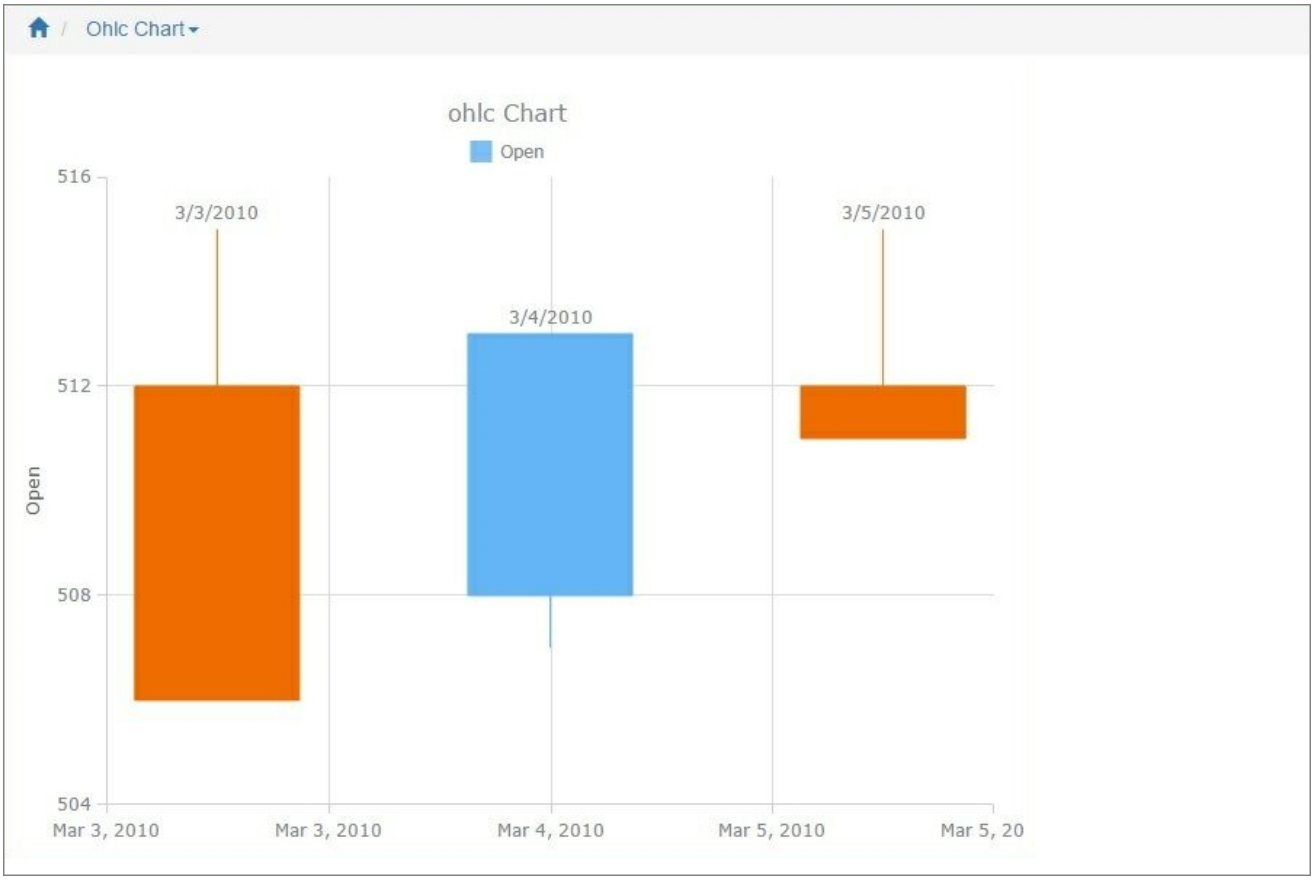


Example of a Candlestick chart

Sample data table:

Day	Open	High	Low	Close
03-Mar-10	512.00	515.00	506.10	506.00
04-Mar-10	508.00	513.00	507.00	513.00
05-Mar-10	512.00	515.00	511.00	511.00

In this example, we chose *Open*, *High*, *Low*, *Close* as the **Data Series** fields, *Day* as the **Label** field.



Other chart types:

- Pie/Doughnut charts
- Accumulation chart
- Column/Bar charts
- Line chart
- Gauge chart
- Bubble chart
- Area chart
- Combined charts

See also:

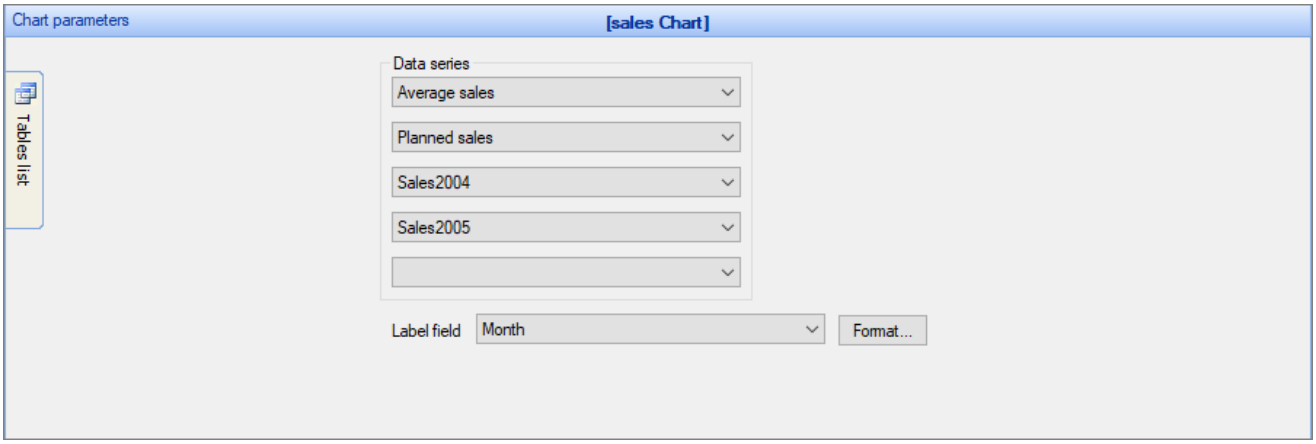
- Creating charts
- A list of chart types
- Chart parameters
- Chart appearance
- Using SQL to shape chart data

2.9.2.10 Combined chart

Description

A **Combined** chart allows you to combine several types of charts.

If you select one Data series field, you get a Line chart. If you select two **Data series** fields, you get a **Line**-Area chart. With three or more **Data series** fields selected - a **Line**-**Area**-Column chart.



Examples of combined charts

Sample data table:

Month	Average sales	Planned sales	Sales2004	Sales2005
Jan	12000	13000	10000	14000
Feb	14500	15000	14000	15000
Mar	10500	14000	12000	9000

Example 1

In this example, we chose *Average sales* and *Planned sales* as the **Data Series** fields, *Month* as the **Label** field.

Example 2

In this example, we chose *Average sales*, *Planned sales*, *Sales2004* and *Sales2005* as the **Data Series** fields, *Month* as the **Label** field.

Other chart types:

- Pie/Doughnut charts
- Accumulation chart
- Column/Bar charts
- Line chart
- Gauge chart
- Bubble chart
- Area chart
- Financial OHLC/Candlestick charts

See also:

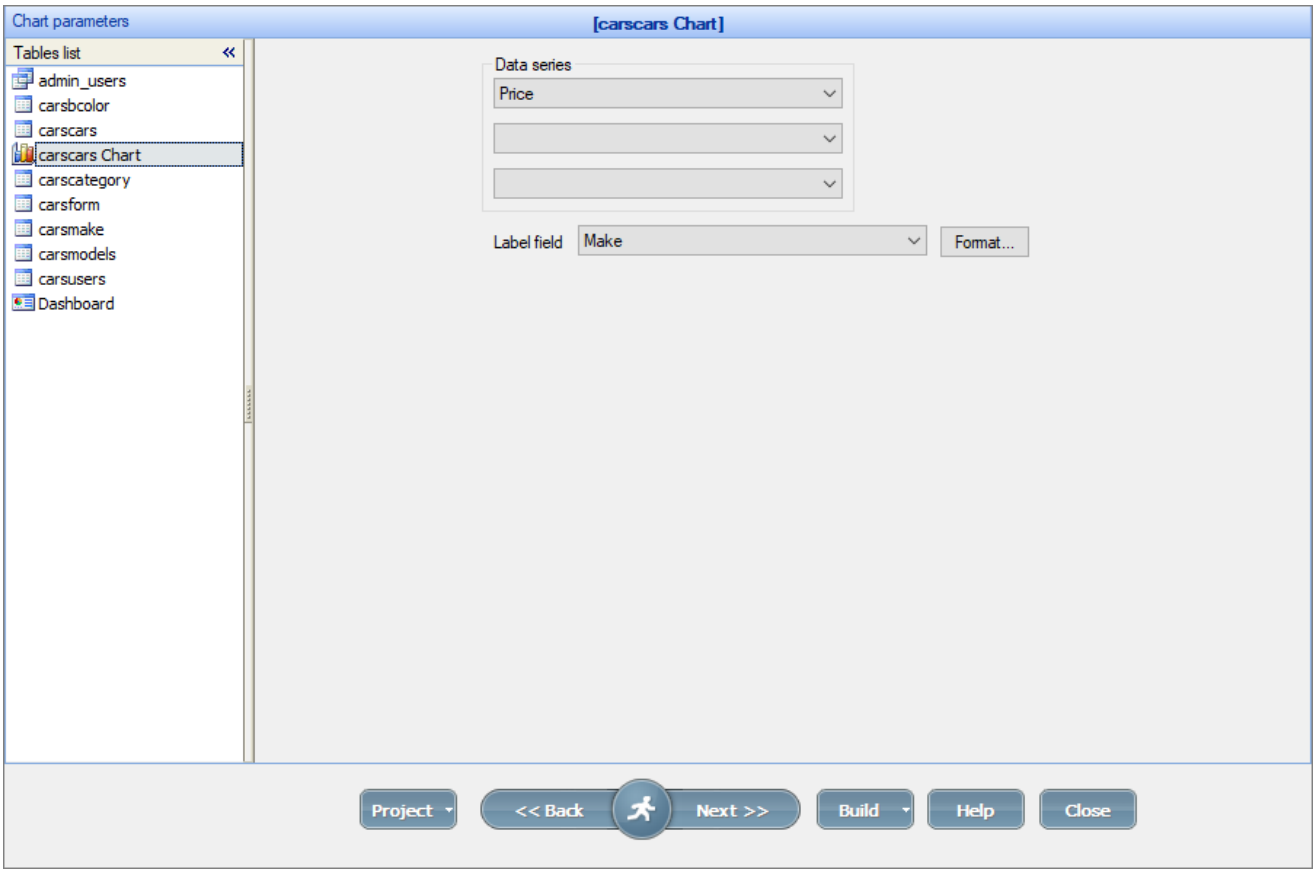
- Creating charts
- A list of chart types
- Chart parameters
- Chart appearance
- Using SQL to shape chart data

2.9.3 Chart parameters

The **Chart parameters** screen allows you to choose the **Data series** fields (fields with data) and the **Label** field (field with data labels).

You can add any number of **Data series** fields. Additional data series dropdown boxes appear automatically once you've used the available ones.

For more information about choosing the **Data series** fields for certain chart types, see Chart types.

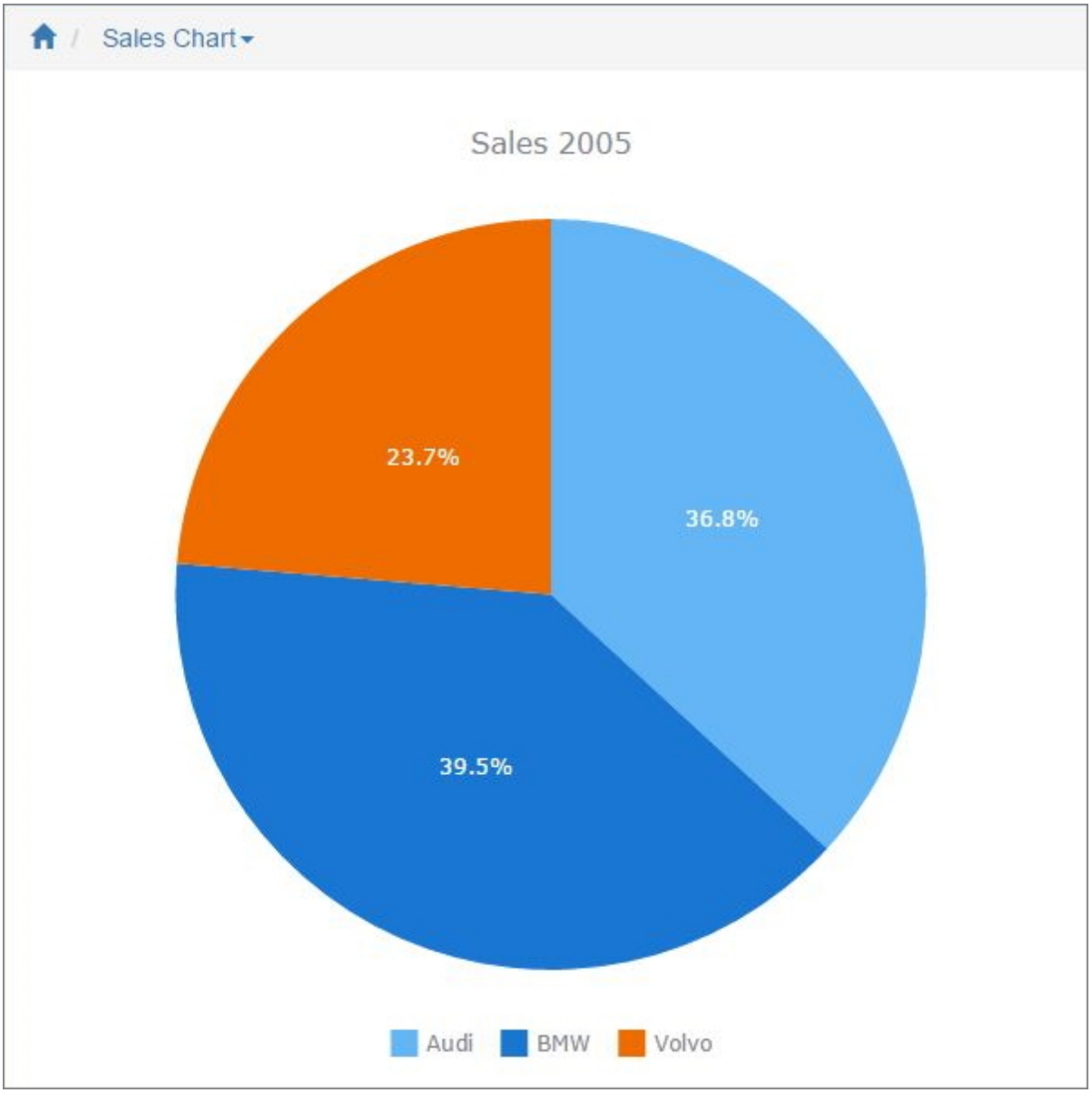


Note: only the numeric fields are available to be chosen as the **Data series**.

Here is a sample data table:

Make	Sales2005
Audi	14000
BMW	15000
Volvo	9000

In this example, we chose *Sales2005* as the **Data Series** field, *Make* as the **Label** field.



Sometimes the data in your database needs to be processed before it can be used in the chart. See Using SQL to shape chart data to learn more.

To further customize the chart appearance like colors, fonts, or the chart title, use the ChartModify event.

See also:

- Creating charts
- A list of chart types
- Chart appearance
- Using SQL to shape chart data
- ChartModify event
- Datasource tables
- Master-details relationship

2.9.4 Chart appearance

The **Chart Appearance** screen allows you to control the way your chart looks in the generated app.

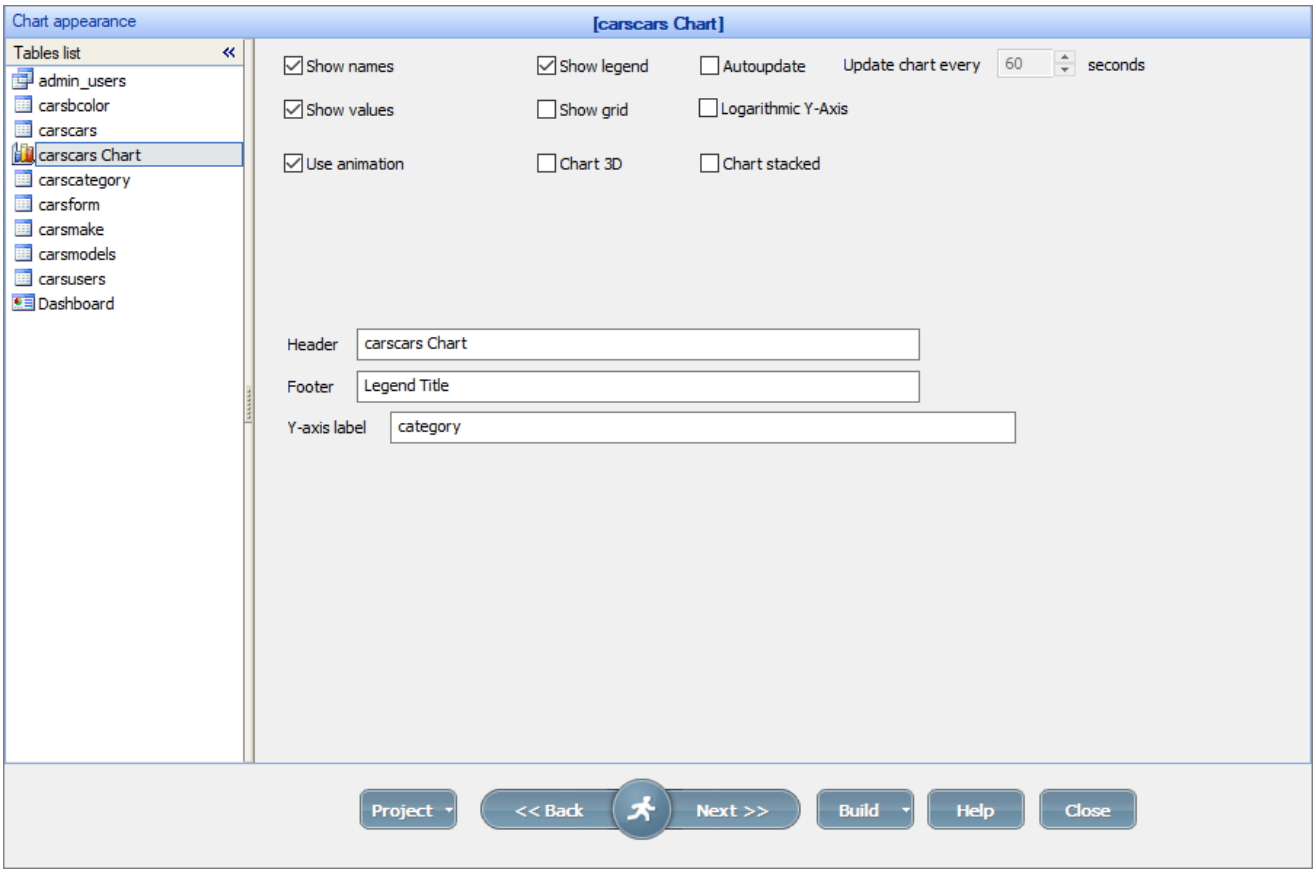


Chart appearance options

- Select the **Autoupdate checkbox** to auto-refresh the chart every N seconds.
- The **Use animation** checkbox enables the chart animation upon opening a chart.
- Use the **Logarithmic Y-Axis** option to convert a linear value axis to a logarithmic value axis.

If you select one of the 2D charts (e.g., 2D Column chart), additional options become available:

- Use the **Chart 3D** option to build a 3D chart instead of a 2D one.
- Use the **Chart stacked** option to display a chart where the chart elements are stacked on top of each other.

> **Note**: the **Y-axis label** input box appears only if you have multiple Data series fields.

For more information about each chart type settings, see Chart types.

Additional customization

1. Charts can be resized on the Editor screen.

2. With the **Editor**, you can copy the chart to another page (a **Report**/**List**/**View** or another chart page). It can be useful to build a dashboard containing several charts.

3. To further customize the chart appearance like colors, fonts, or the chart title, use the ChartModify event.

See also:

- Creating charts
- A list of chart types
- Chart parameters
- Using SQL to shape chart data

- ChartModify event
- Datasource tables
- Master-details relationship

2.9.5 Using SQL to shape chart data

Charts are all about visualizing the numbers or percentages - they require data fields that store numeric values. However, you can use SQL queries to build charts using almost any initial data.

Consider the following *Orders* table:

Customer	Country	Total
Andrew Peters	USA	$250
Katie Bradshow	Australia	$85
Jeff Simpson	USA	$150
Arnold Matteus	Germany	$120
Arnold Matteus	Germany	$160
Jeff Montgomery	GB	$150
Andrew Peters	USA	$65
Jeff Simpson	USA	$95
Luke Sohu	France	$40
Jeff Montgomery	GB	$120

Example 1: Total sales per country

SQL query:

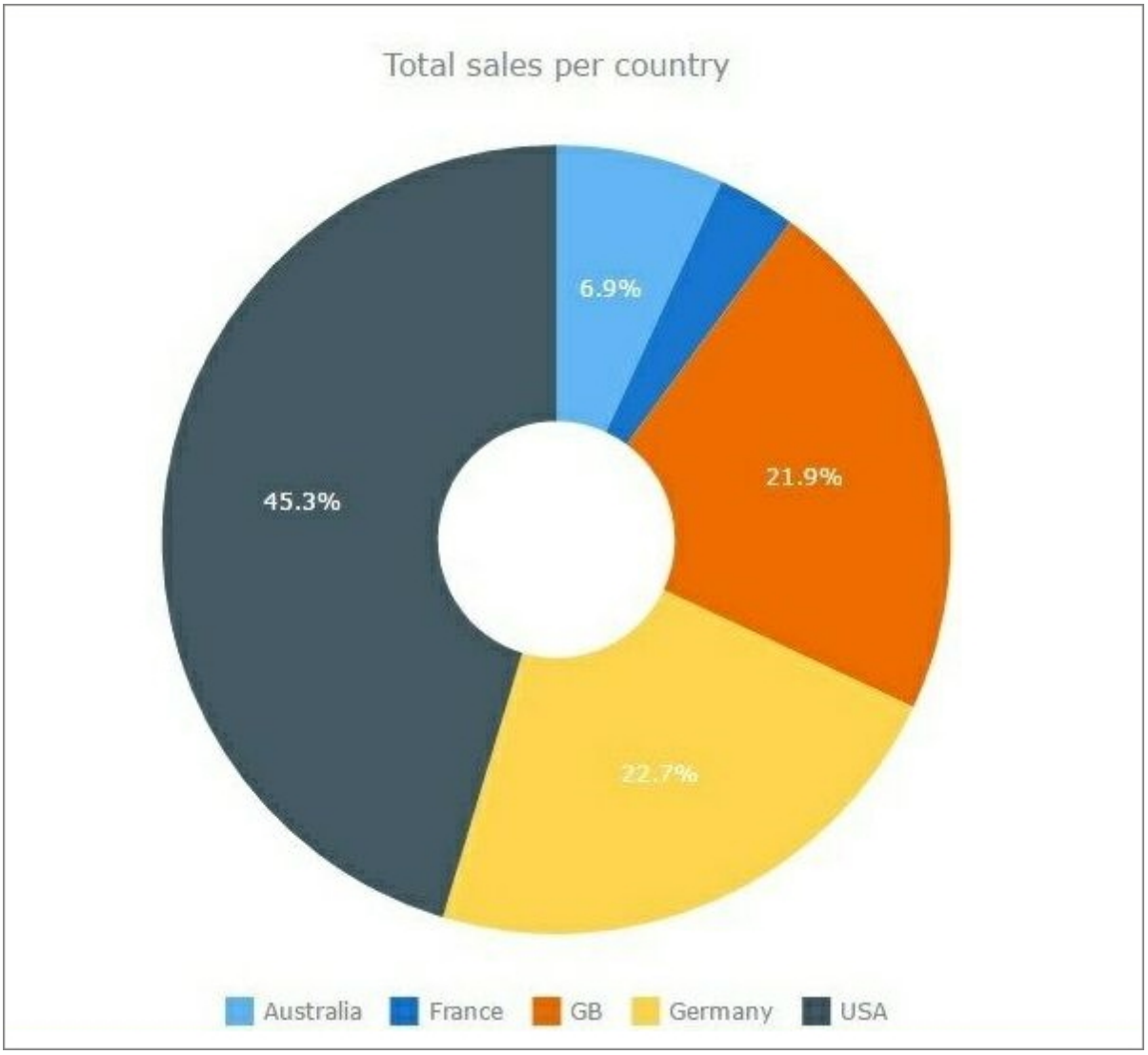
```
select Country, sum(total) as STotal
from Orders
```

```
group by country
order by 2 desc
```

Results:

Country	STotal
USA	$560
Germany	$280
GB	$270
Australia	$85
France	$40

The resulting chart:



Example 2: Number of orders per country

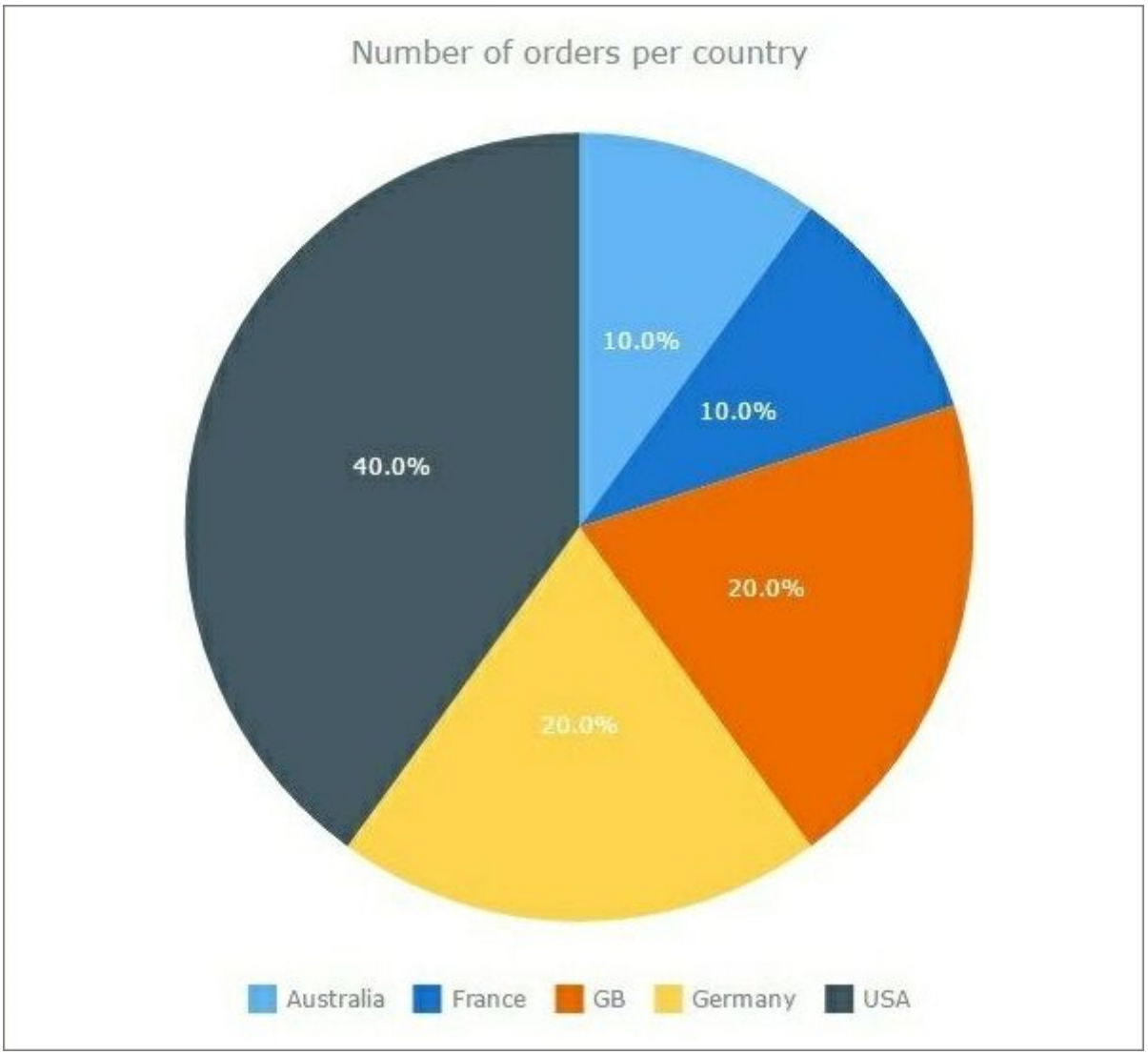
SQL query:

```
select Country, count(total) as CTotal
from Orders
group by country
order by 2 desc
```

Results:

Country	CTotal
USA	4
Germany	2
GB	2
Australia	1
France	1

The resulting chart:



Example 3: Shaping your data in a more complex way

This example shows how to use GROUP BY in conjunction with INNER JOIN. For example, we have the following data and would like to display a diagram illustrating how many flags each client has.

clientid	flag
1001	Green
1001	Green
1001	Green
1001	Green
1001	Amber
1001	Amber
1001	Red
1002	Green
1002	Amber
1002	Amber
1002	Amber
1002	Red
1003	Green
1003	Amber
1003	Red

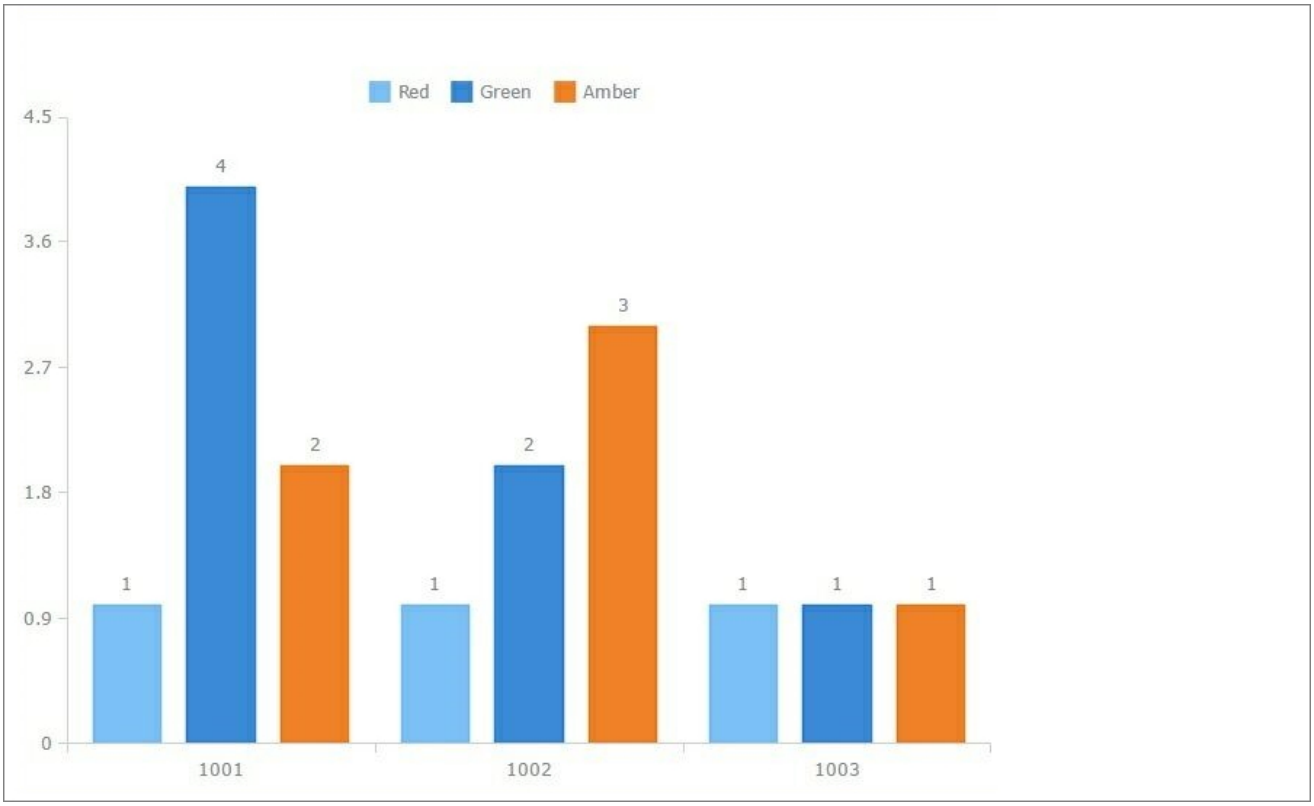
SQL query:

```
select a.clientid, a.green,b.red, c.amber
from (select count(flag) as green, clientid from sensorstatus
where flag='Green' group by clientid) a
inner join (select count(flag) as red, clientid from sensorstatus
where flag='Red' group by clientid) b on a.clientid=b.clientid
inner join (select count(flag) as amber, clientid from sensorstatus
where flag='Amber' group by clientid) c on a.clientid=c.clientid
```

Results:

clientid	green	red	amber
1001	4	1	2
1002	2	1	3
1003	1	1	1

The resulting chart:



See also:

- Interactive SQL tutorial
- Creating charts
- A list of chart types
- Chart parameters

- Using SQL to shape chart data
- ChartModify event
- Datasource tables
- Master-details relationship
- SQL query screen

2.10 Reports

2.10.1 Creating and configuring reports

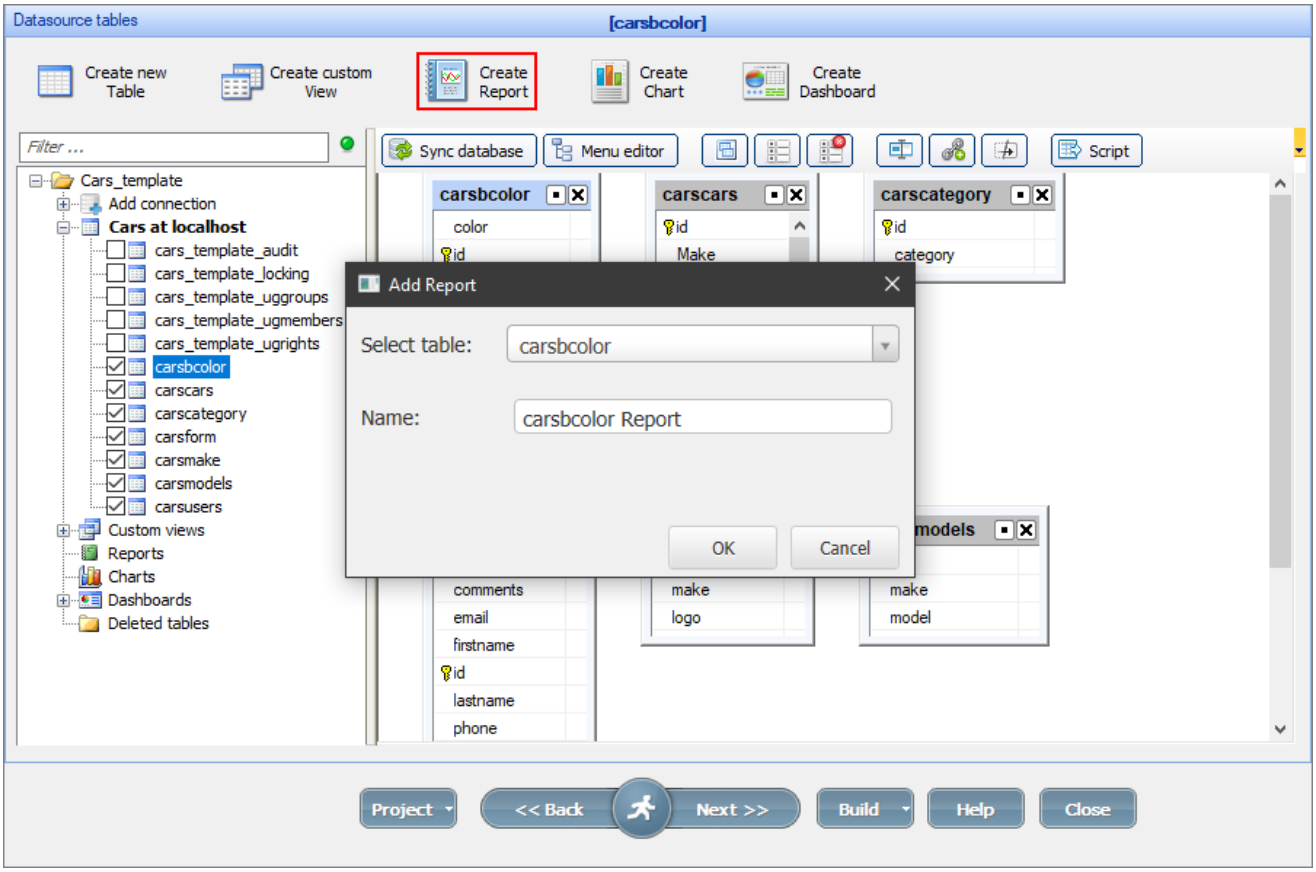
Quick jump

Creating a report

To create a report:

1. Proceed to the Datasource tables screen and click **Create Report**.

2. Select the table, type in the report name, and click **OK**.



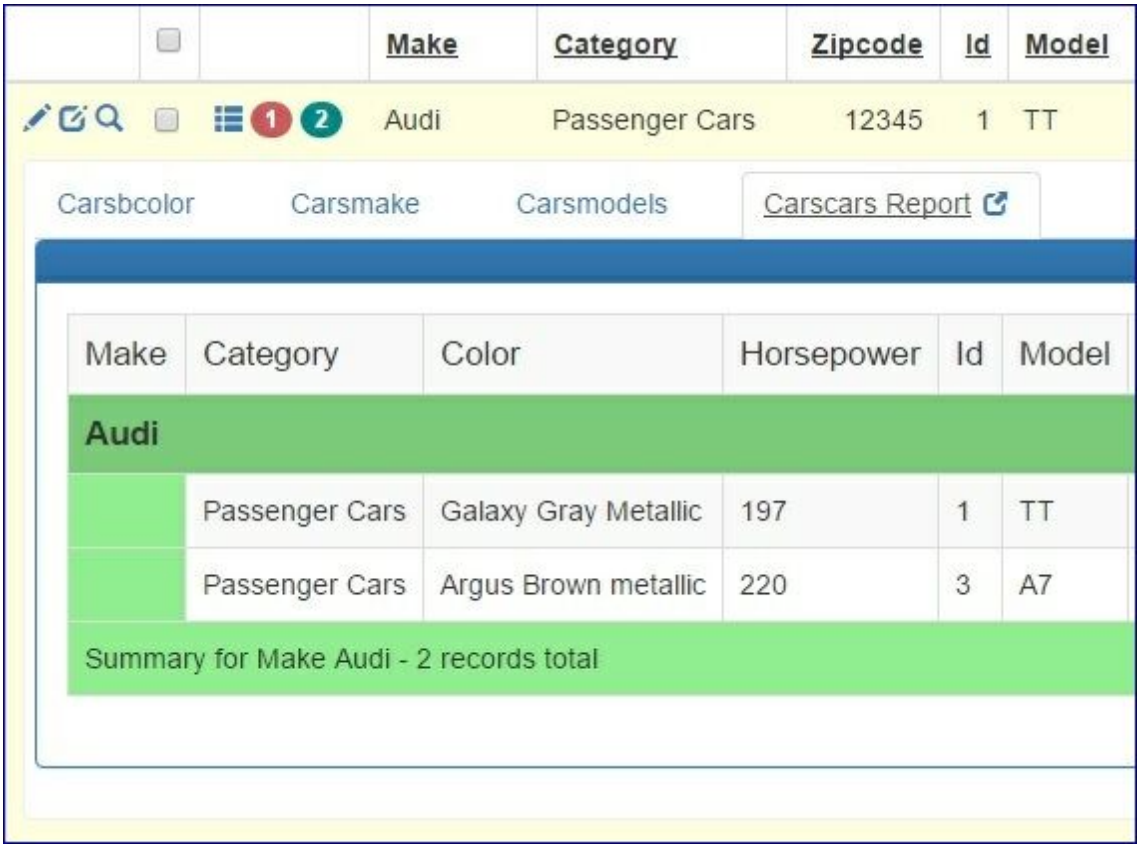
> **Note**: you can create a copy of an existing report (right-click the report and select **Copy**).

On the next several screens (use the **Next** button to navigate) you can:

- make the changes to the SQL query. More info about editing SQL queries;
- choose group fields and grouping types;
- set up the report totals and layout.

> **Note**: you can use reports with both master and details tables. For more information, see Master-details relationship between tables.

A report as a details table:

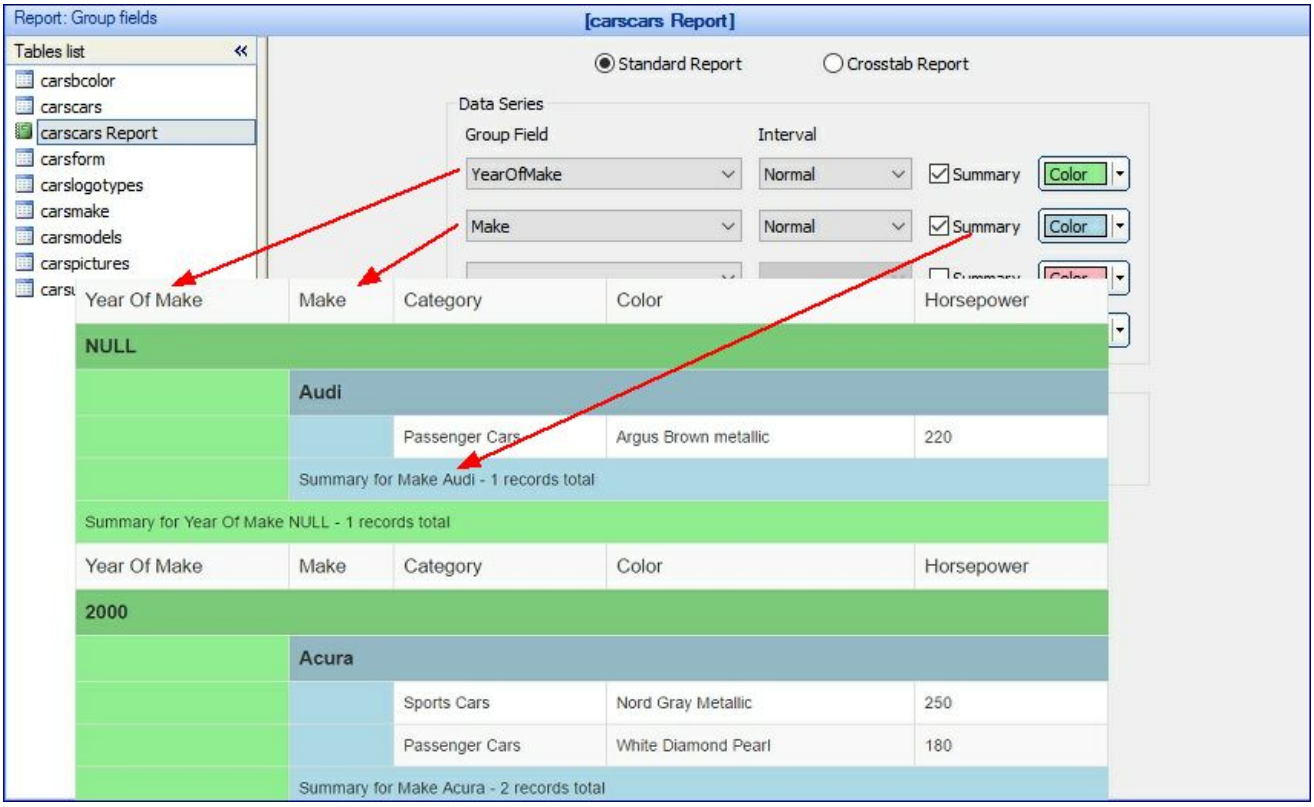


Group fields

On the **Report: Group fields** screen you can select the group fields, grouping intervals, and summary types.

You can choose between standard and cross-tab reports.

Standard reports



Clear the **Show details and summary** checkbox to make the report page show only the summary.

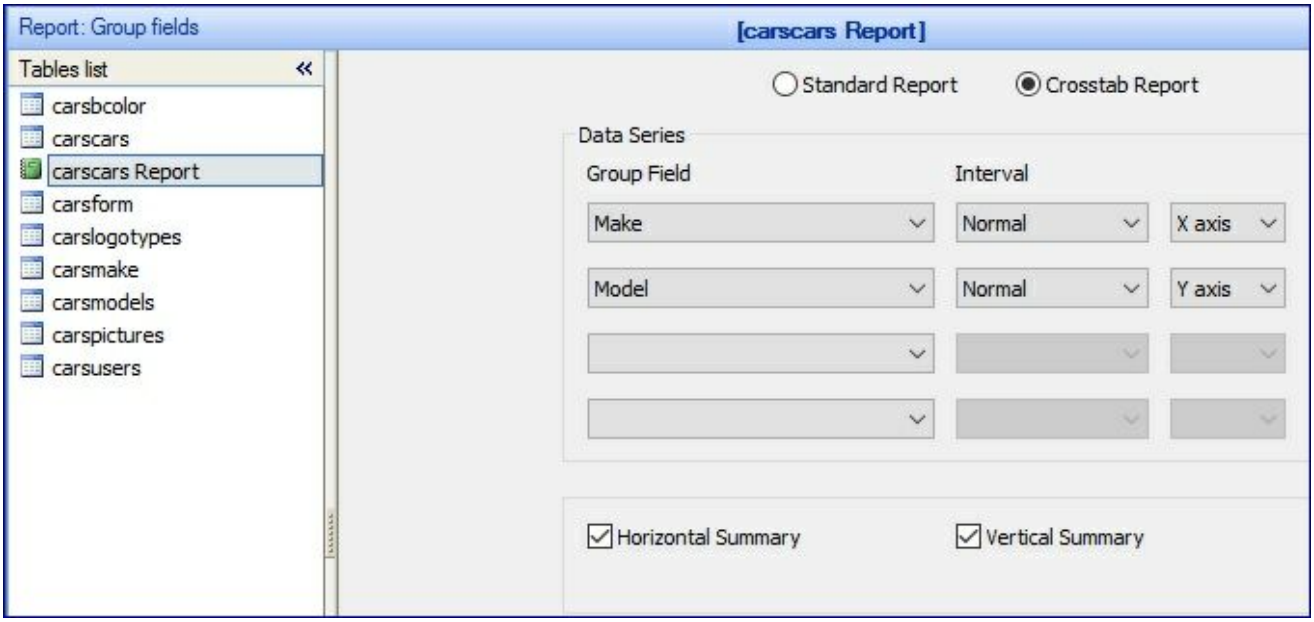
You can choose the **Interval** for the group field. Available interval types are different for each type of data.

Here is an example of a report with a text group field using the first letter as an interval:

Make	Year Of Make	Model	Price	Category	Color	Horsepower	Id
A							
	2000						
		MDX					
Acura			58000	Passenger Cars	White Diamond Pearl	180	5
		Summary for Model MDX - 1 records total					
		NSX-T					
Acura			58000	Sports Cars	Nord Gray Metallic	250	4
		Summary for Model NSX-T - 1 records total					
		TT					
Audi			57900	Passenger Cars	Galaxy Gray Metallic	197	1
		Summary for Model TT - 1 records total					
	Summary for Year Of Make 2000 - 3 records total						
	2003						
		A7					
Audi			62000	Passenger Cars	Argus Brown metallic	220	3
		Summary for Model A7 - 1 records total					
	Summary for Year Of Make 2003 - 1 records total						
Summary for Make A - 4 records total							
Make	Year Of Make	Model	Price	Category	Color	Horsepower	Id
B							
	2001						
		750iL					
BMW				Passenger Cars	Glacier Silver Metallic	300	7
		Summary for Model 750iL - 1 records total					
		X6					

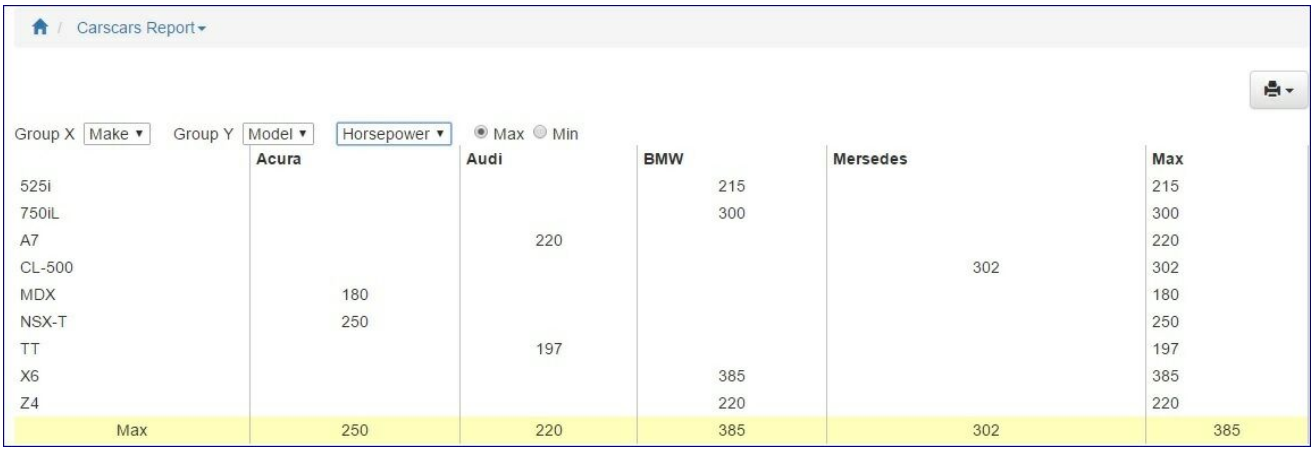
Cross-tab reports

PHPRunner supports cross-tab reports (often called pivot tables). On the **Group fields** screen, you can choose two or more variables (columns) and assign them to the X or Y-axis. You can assign more than one variable to the axis and switch between them in the generated report.



On the **Report: Totals** screen, you can choose the totals options like *Min*, *Max*, *Sum*, and *Average*.

Here is how a cross-tab report looks like in the generated application. You can select the variables, totals types, and data fields to display.



	Acura	Audi	BMW	Mersedes	Max
525i			215		215
750iL			300		300
A7		220			220
CL-500				302	302
MDX	180				180
NSX-T	250				250
TT		197			197
X6			385		385
Z4			220		220
Max	250	220	385	302	385

See also:

- Report totals and layout
- Datasource tables

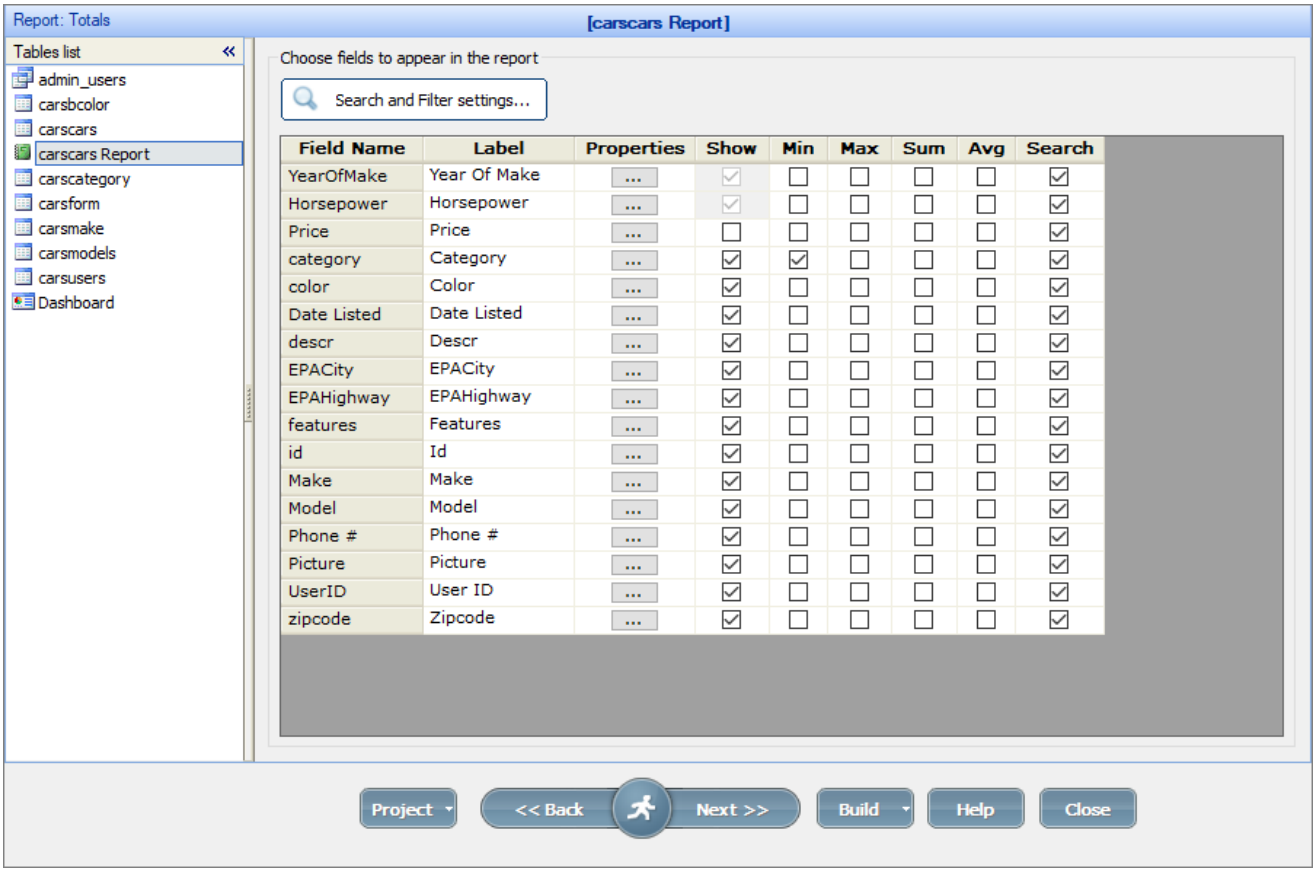
- Master-details relationship between tables
- SQL query screen
- Creating charts
- Creating dashboards

2.10.2 Report totals and layout

Quick jump

Report: Totals

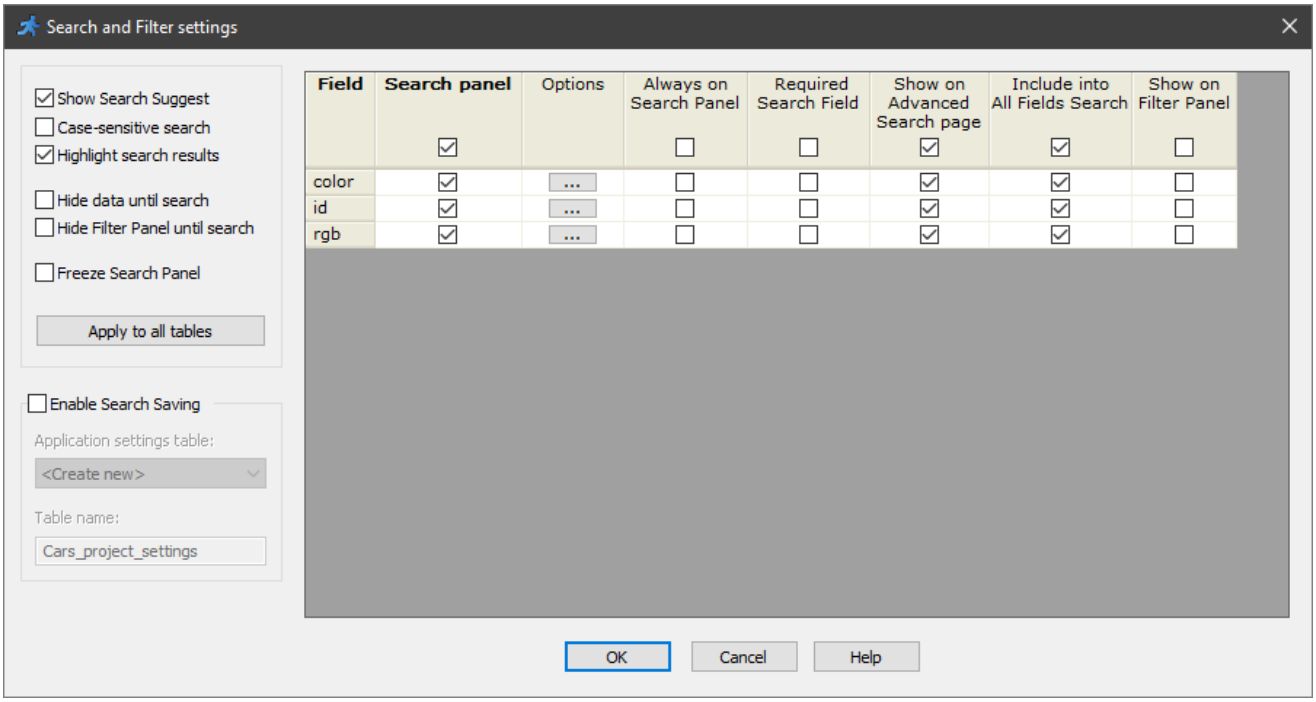
On the **Report: Totals** screen, you can choose the fields to display on the report/search pages.



Apply the aggregate functions like *Min*, *Max*, *Sum*, and *Average* by selecting the corresponding checkboxes. The results of these calculations are displayed after each group and at the end of the page/report.

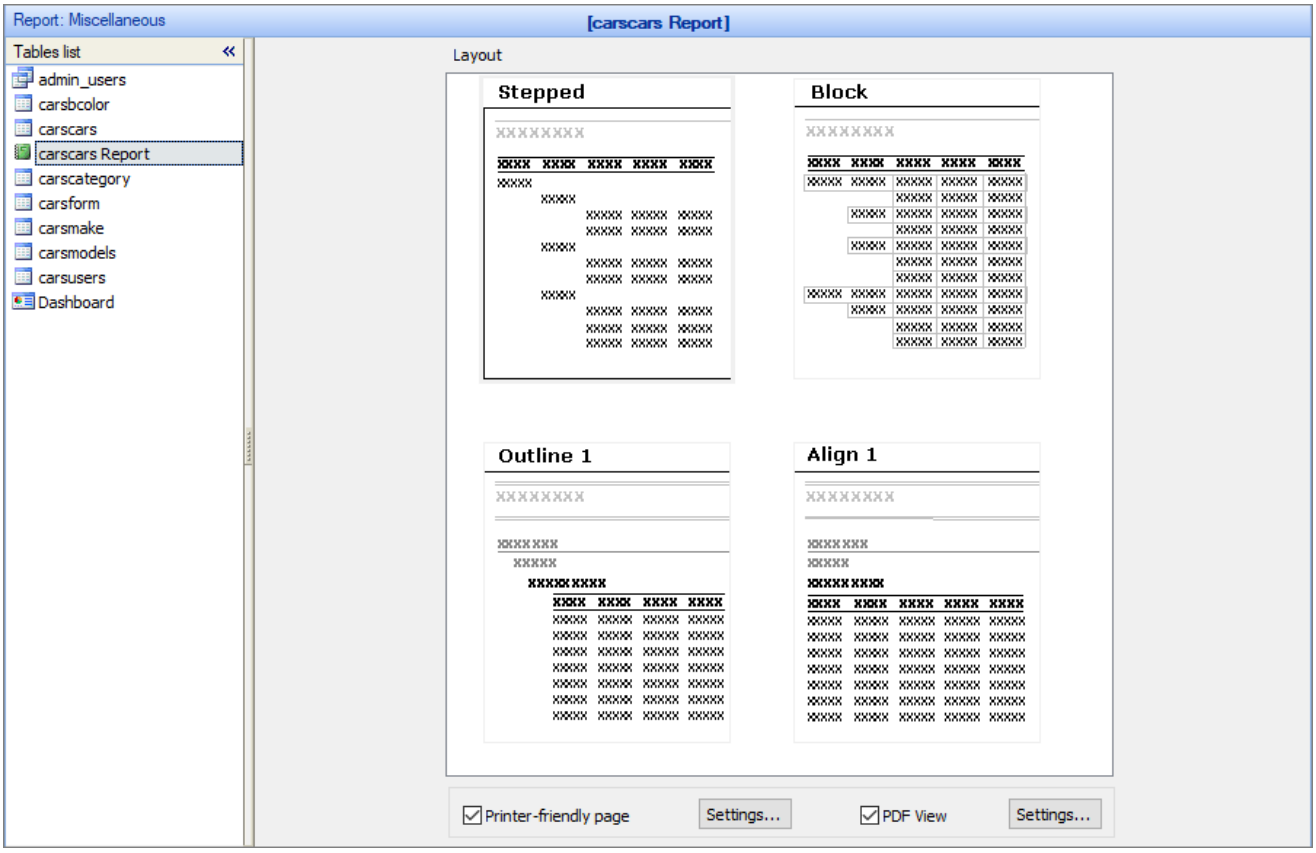
Order Number	Customer Email	Billing Name	Billing Address	Billing City	Weight Total	Order Cost	Taxable Total	Tax Amount	Order Total	Pay Type	Order Comments	Order Status	Order Number
4													
	user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4
	user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4
	user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4
	user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4
	user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4
	user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4
Summary for Order Number 4 - 6 records total													
Sum					76.2								
Min					12.70								
Max					12.70								

You can also click the **Search and Filter settings** button to open a window with corresponding settings in a popup. For more information on these settings, see [Choose fields screen].



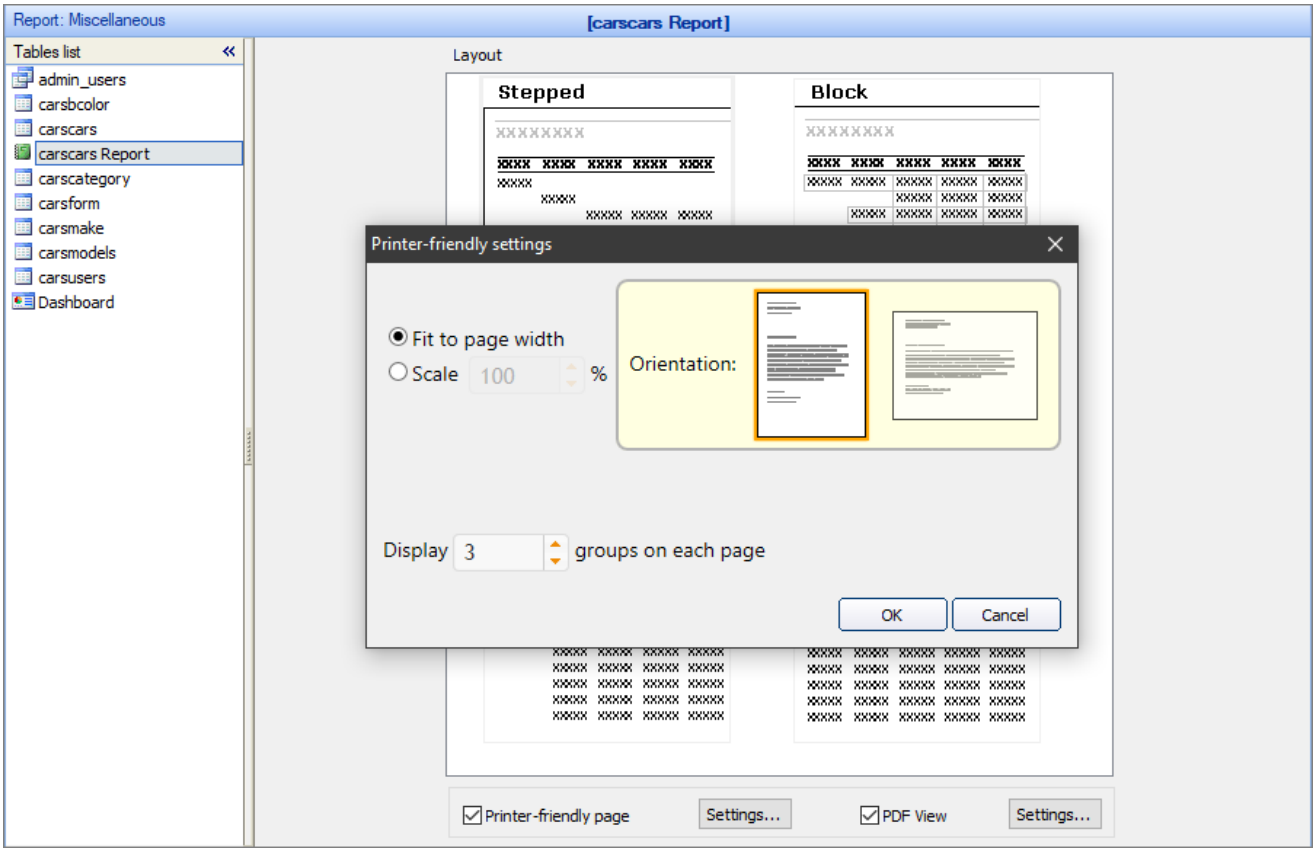
Report: Miscellaneous

The **Report: Miscellaneous** screen allows you to select the layout of the report. If you use grouping, you can choose between **Stepped**, **Block**, **Outline**, and **Align** layouts.



If you don't use grouping, you can only use the **Tabular** layout, which is similar to the basic **List** page.

Use the **Printer-friendly page** and **PDF View** options to make the report printer- and PDF-friendly. Click the **Settings** button to configure these options:



- **Fit to page width/Scale**. You can choose to fit the report to page width or to scale it to a set percentage.
- **Orientation**. Select the portrait or landscape orientation for the report pages.
- The **Display N groups per page** option determines where to insert the page break when you print the whole report.

Displaying lookup values in reports

If you need to display a value from a lookup table instead of the ID on the **Report** page, you have two options:

1. Modify the SQL query to include the fields from joined tables.

2. Enable the report search page, proceed to the Page Designer or Editor screen, open the **Search** page, double-click the field and turn it into a Lookup Wizard on the Edit as tab.

Either option allows you to display values from another table on the **Report** page (i.e., *Customer Name* instead of *Customer ID*).

See also:

- Creating and configuring reports
- Datasource tables
- Master-details relationship between tables
- SQL query screen
- Choose fields screen
- Creating charts
- Creating dashboards

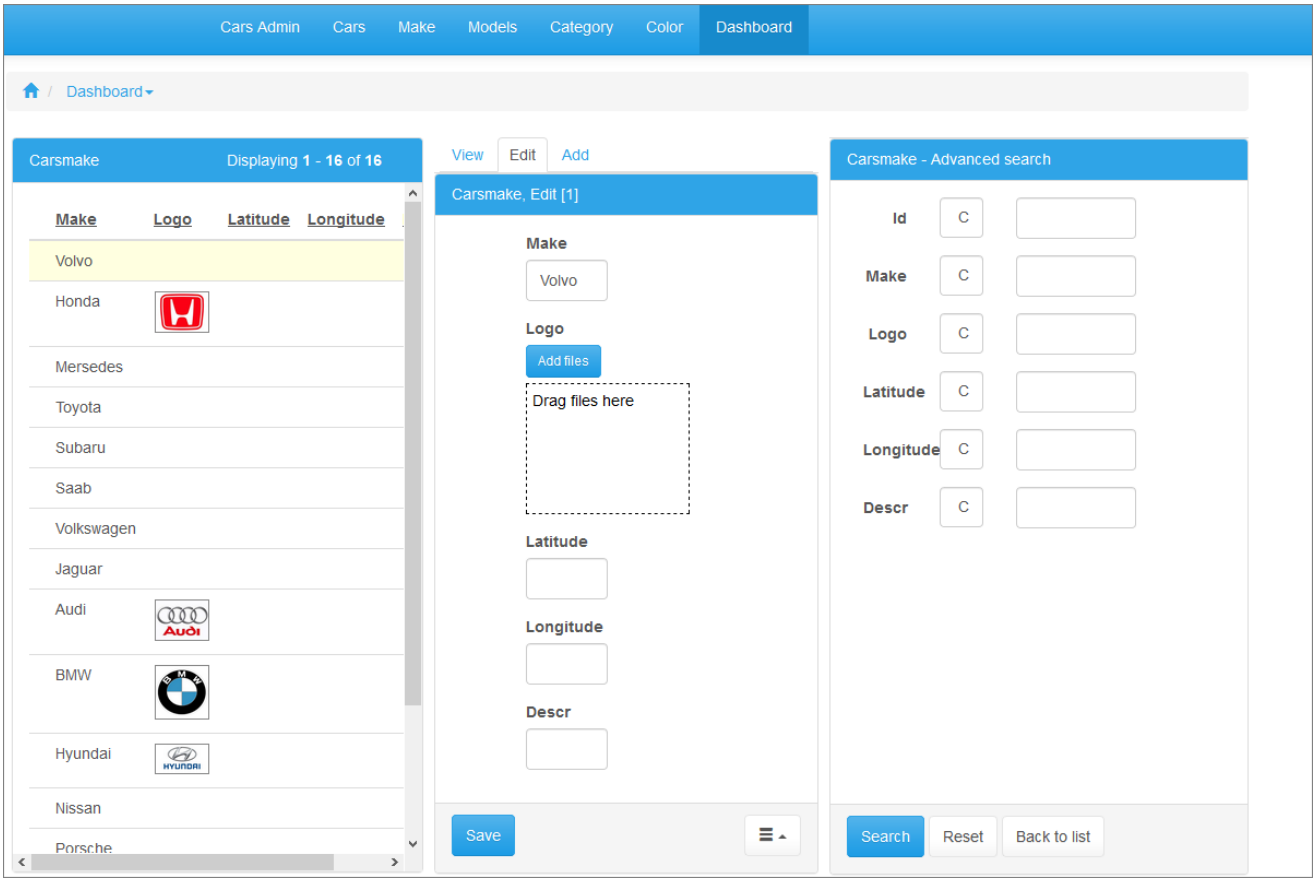
2.11 Dashboards

2.11.1 Creating dashboards

Quick jump

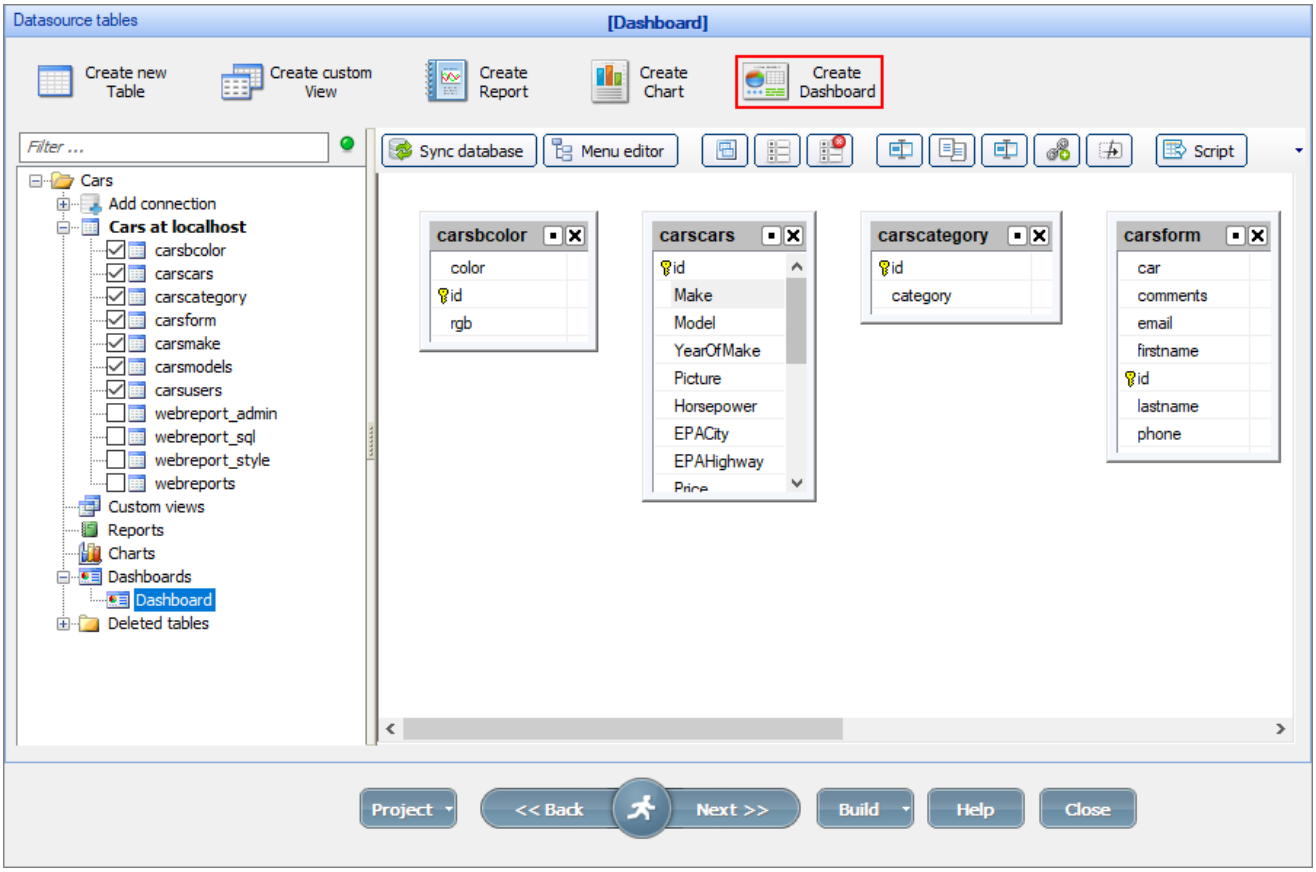
Dashboards allow you to display multiple related or unrelated objects on the same page such as grids, single record views, search pages, master and details tables together, maps, and custom code snippets.

Here is an example of a dashboard with a **grid**, **single record**, and **search**:



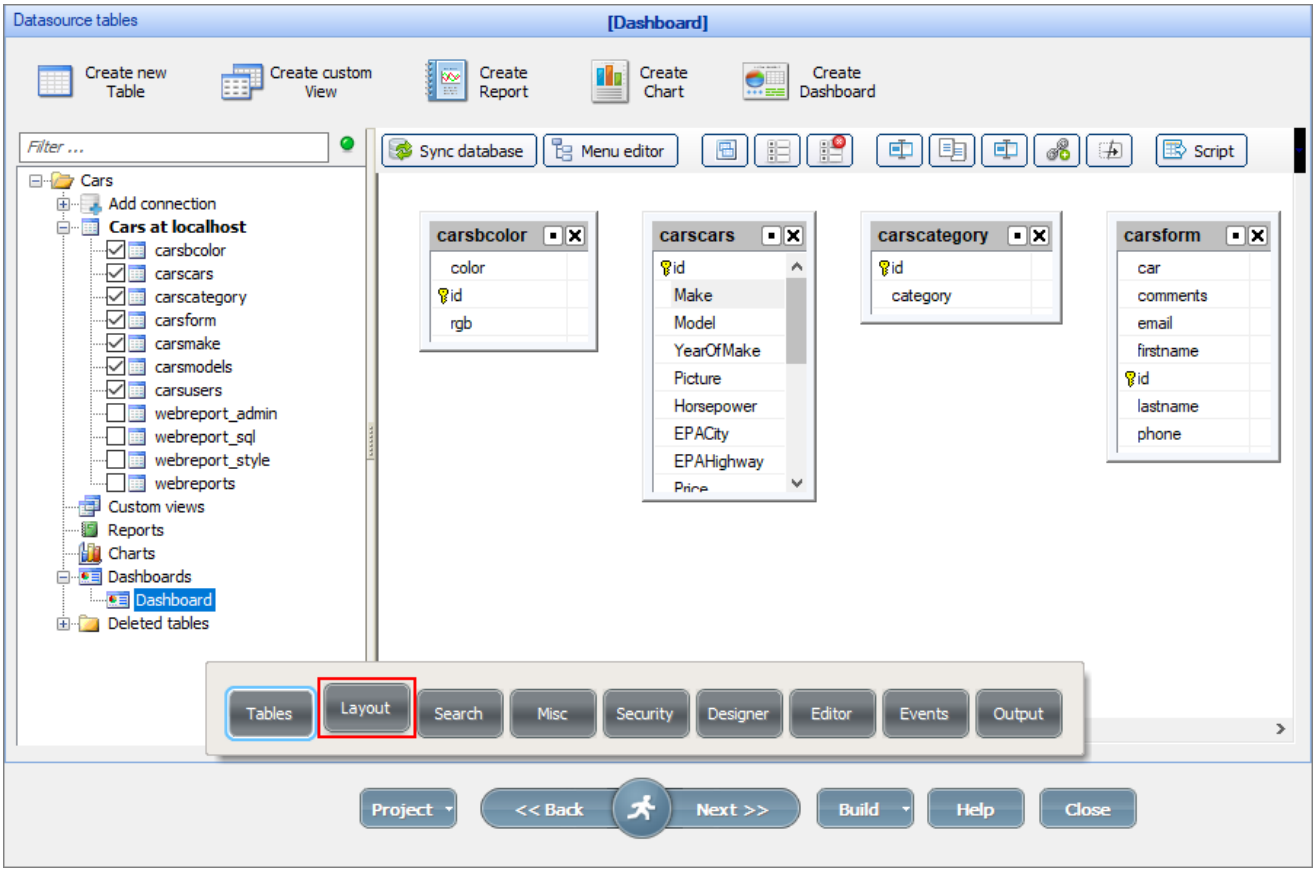
Creating a new dashboard

To create a dashboard, proceed to the **Tables** page, click **Create dashboard**, name it, and click **OK**.



Note: It is possible to create a copy of an existing dashboard (right-click the dashboard and select **Copy**).

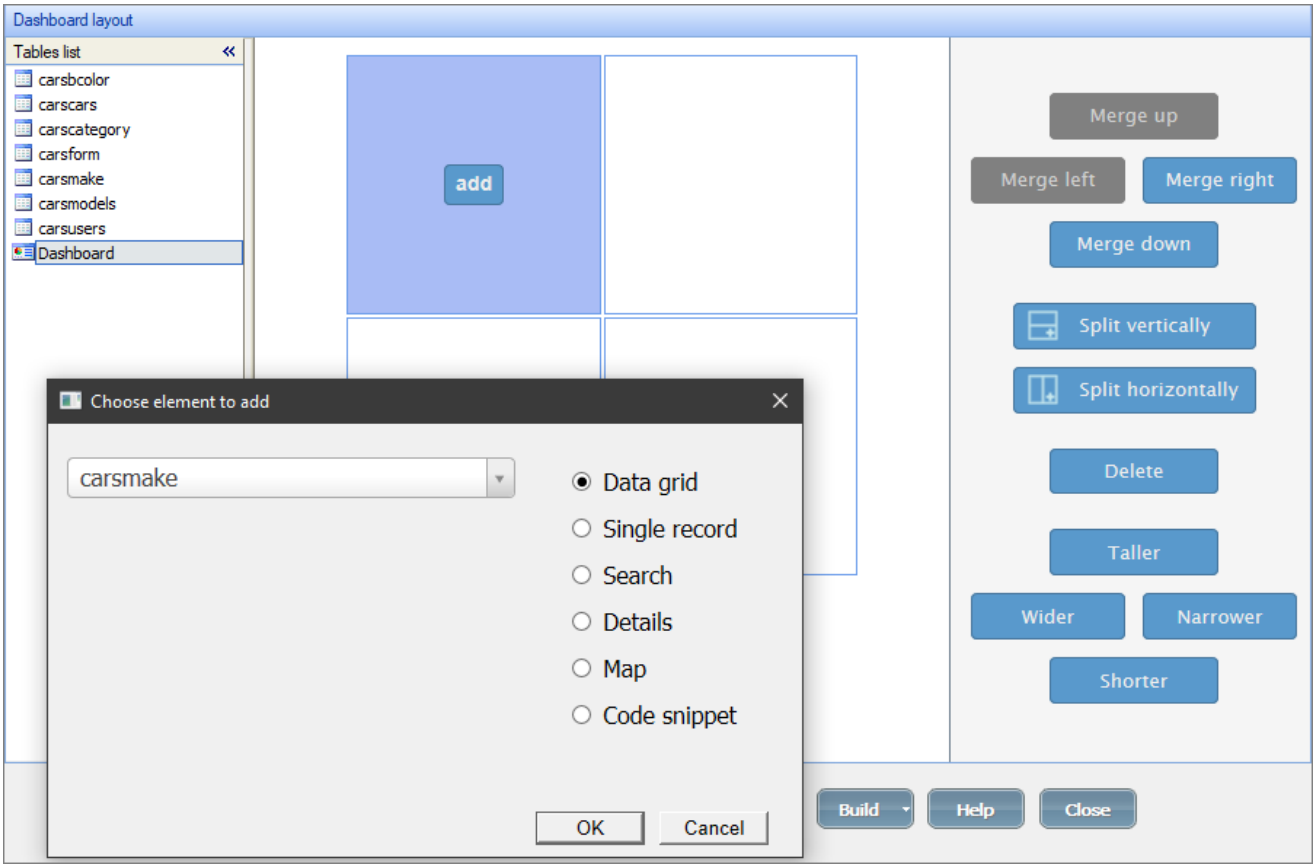
To edit the layout of the dashboard, select the dashboard and go to the **Layout** page. Alternatively, you can select the dashboard on the **Tables** page and click **Next**.



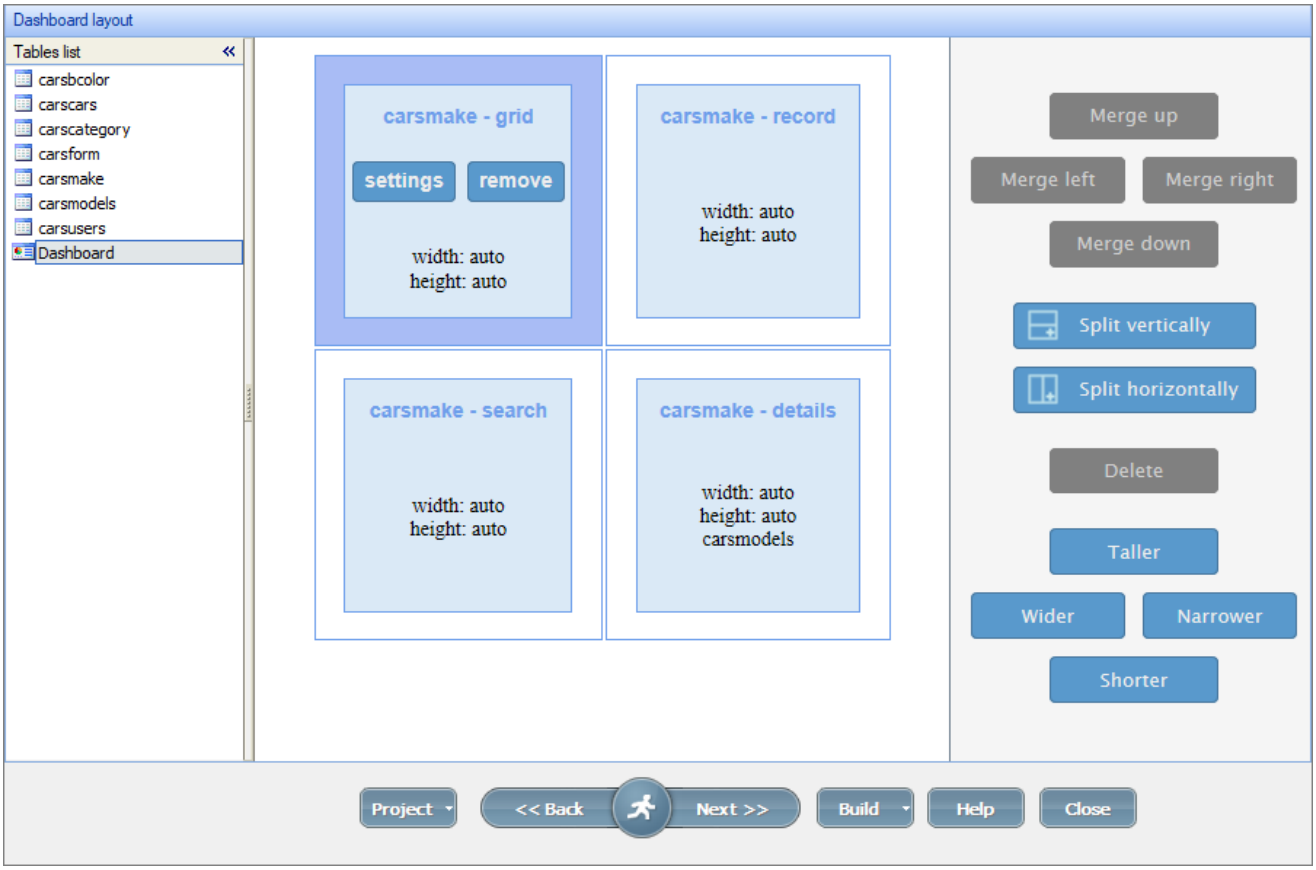
Customizing the layout of the dashboard

On the **Dashboard layout** screen, you may select dashboard elements and use the buttons on the right-side panel to change the layout. You can change the number of available dashboard elements with the **Merge** and **Split** buttons.

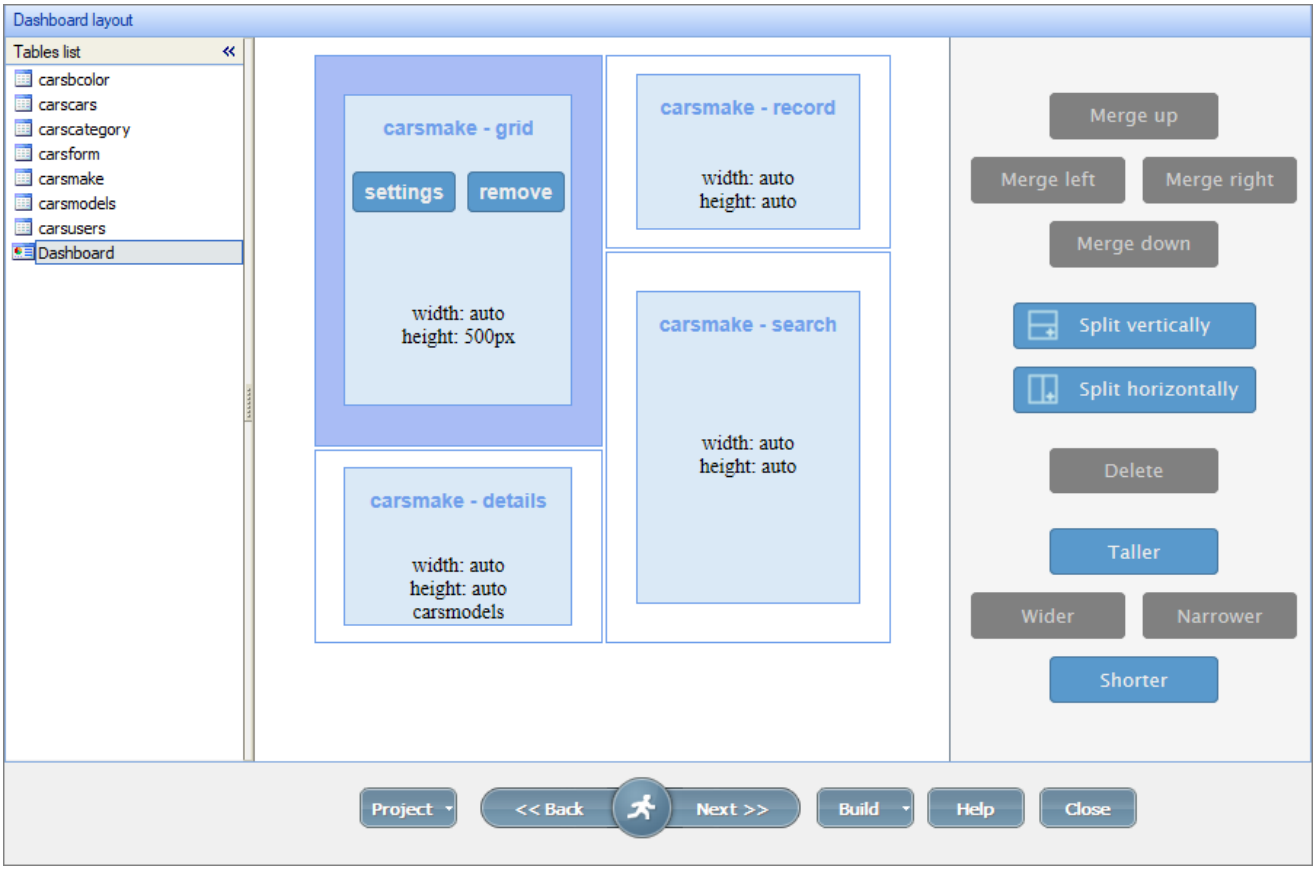
Click **Add** to add an element to the dashboard. You can choose between a data grid, single record, search, details, map or code snippet to use with the selected table in the dropdown.



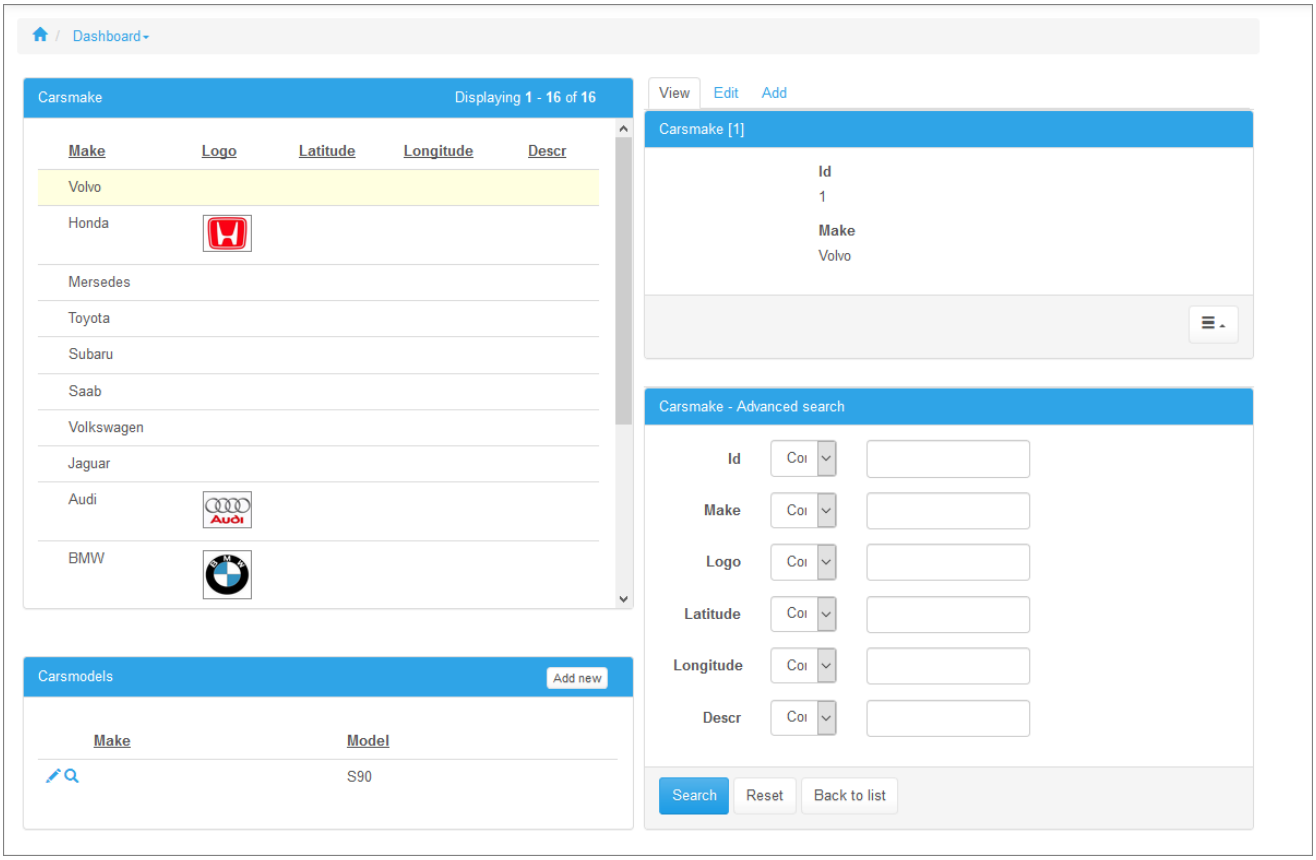
Click **Settings** on the added dashboard element to customize its appearance. Click **Remove** to delete it.



If a dashboard element has a strongly differing width or height, you may use the **Taller**/**Shorter** and **Wider**/**Narrower** buttons to change the dimensions of the element. In the example below, the *carsmake*-record element is made shorter.



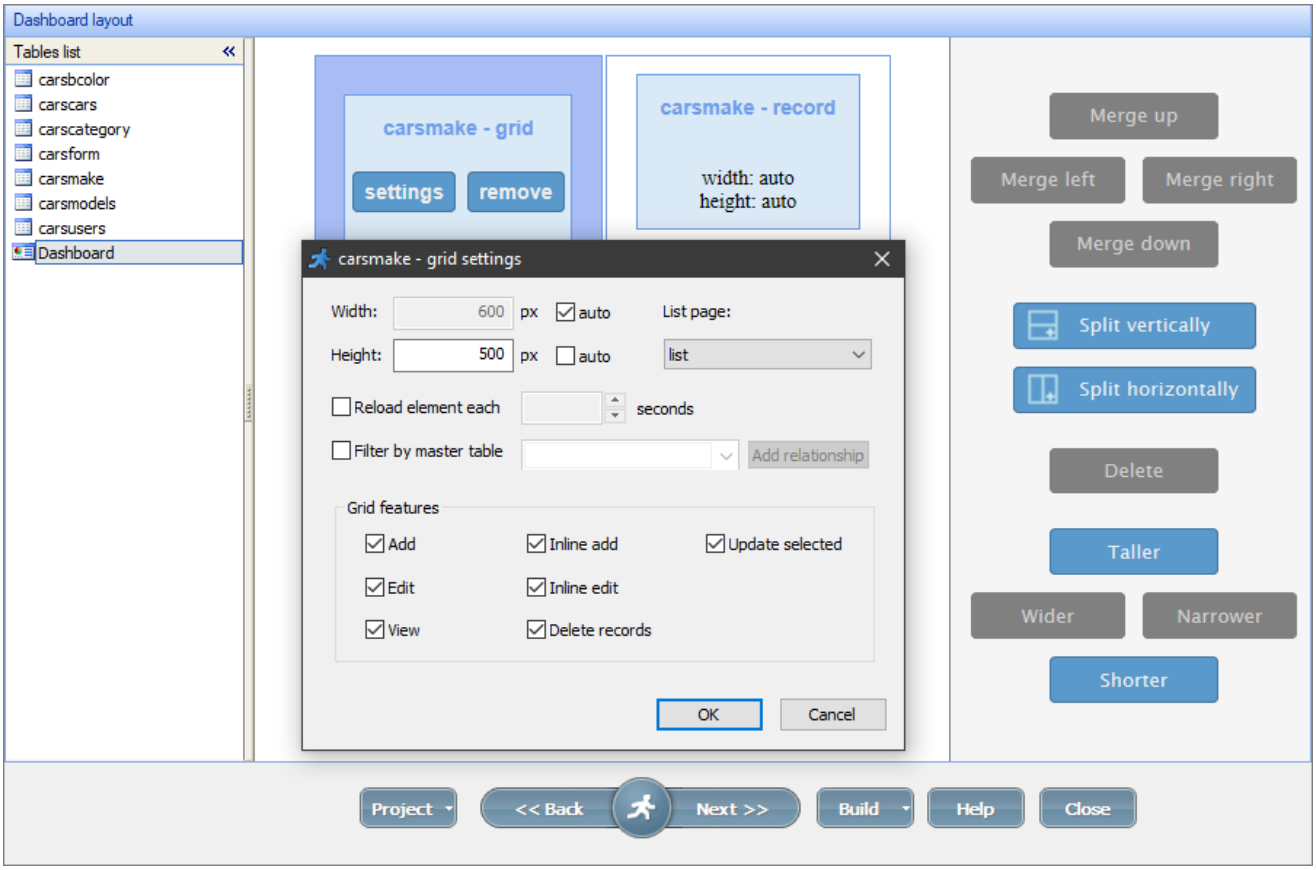
Here is how it looks like in the browser:



Types of dashboard elements

Data grid

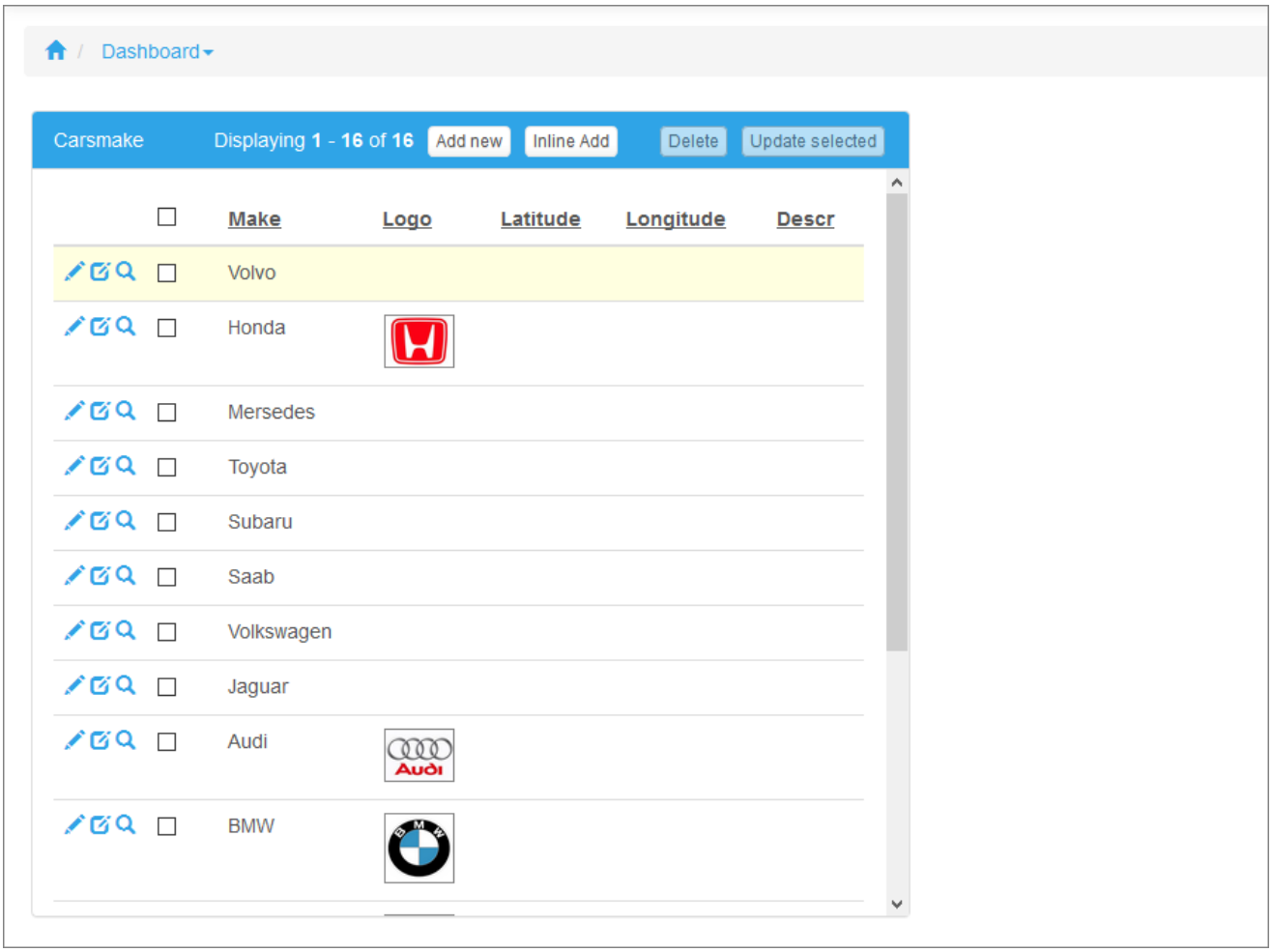
This element displays a data grid of the selected table.



Settings

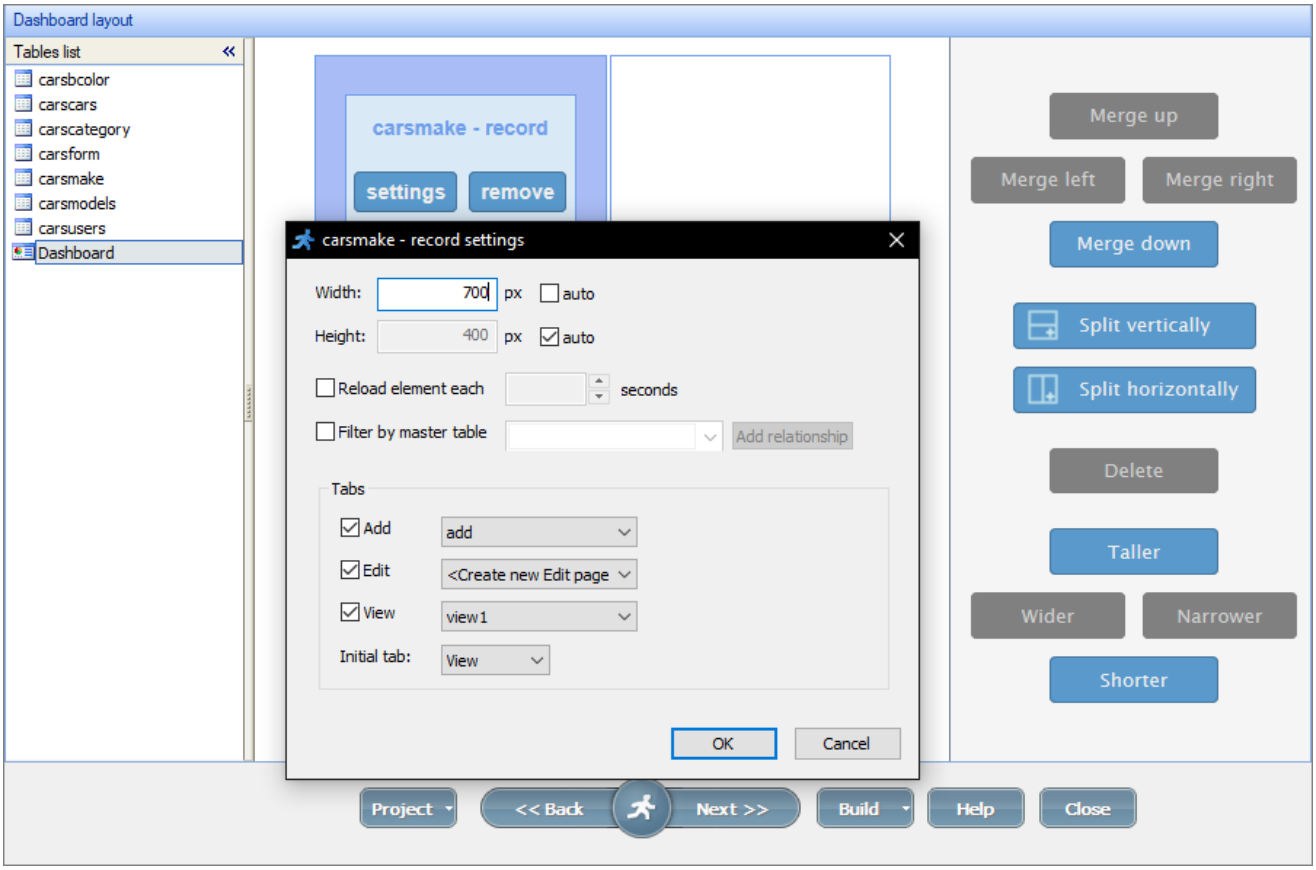
- **Width/Height**. These options define the width and height of the data grid. Select the **auto** checkboxes to make the grid take the space needed to show all of its elements.
- **List page**. If you have several **List** pages for the table, you can select the page to show in the dashboard element. Alternatively, you can create a new page with the default appearance. For more information, see Working with additional pages.
- **Reload element each X seconds**. You can set the element to reload each X seconds so that it accurately shows the current records.
- **Filter by master table**. This option makes dashboard elements display the data depending on other dashboard elements. You can use this option for cascading master-details elements (for example, *customers -> orders -> order details*), or when a map is used as the details. For more information, see Master-details dashboard.
- **Grid features**. You can select the grid features to be displayed in the data grid element. For more information, see Choose pages screen.

Here is an example of a dashboard with a data grid:



Single record

This element displays the **View**/**Add**/**Edit** page for a single record of the selected table.

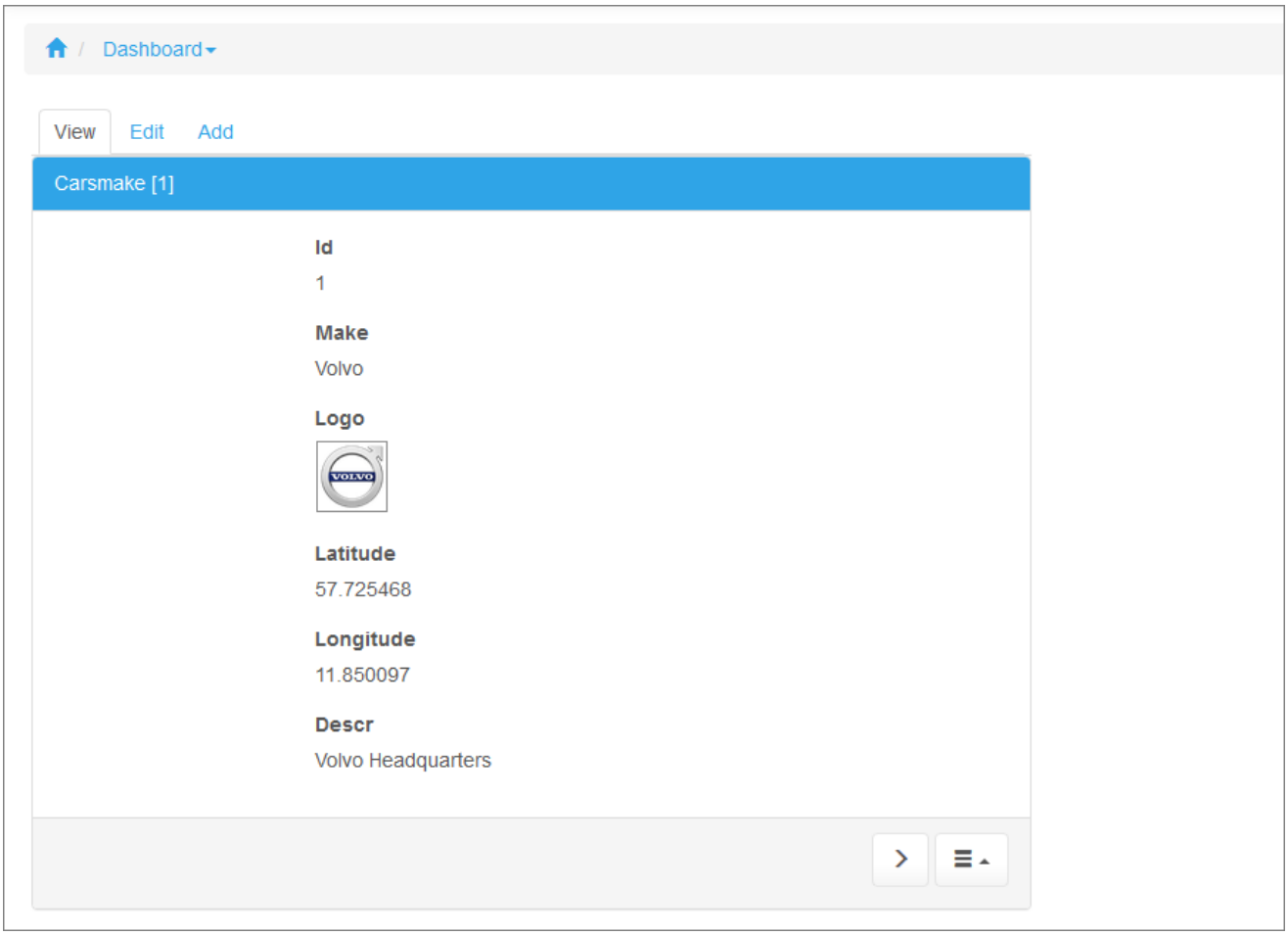


Settings

- **Width/Height**. These options define the width and height of the single record element. Select the **auto** checkboxes to make the single record take the space needed to show all of its elements.
- **Reload element each X seconds**. You can set the element to reload each X seconds so that it accurately shows the current records.
- **Filter by master table**. This option makes dashboard elements display the data depending on other dashboard elements. You can use this option for cascading master-details elements (for example, *customers -> orders -> order details*), or when a map is used as the details. For more information, see Master-details dashboard.
- **Tabs**. You can select the **View**/**Add**/**Edit** tabs to be displayed in the element. If you have several **View**, **Add**, or **Edit** pages, you can select the pages to show in the dashboard element with the dropdown next to the respective checkbox. Alternatively, you can create a new page with the default appearance. For more information, see Working with additional pages.

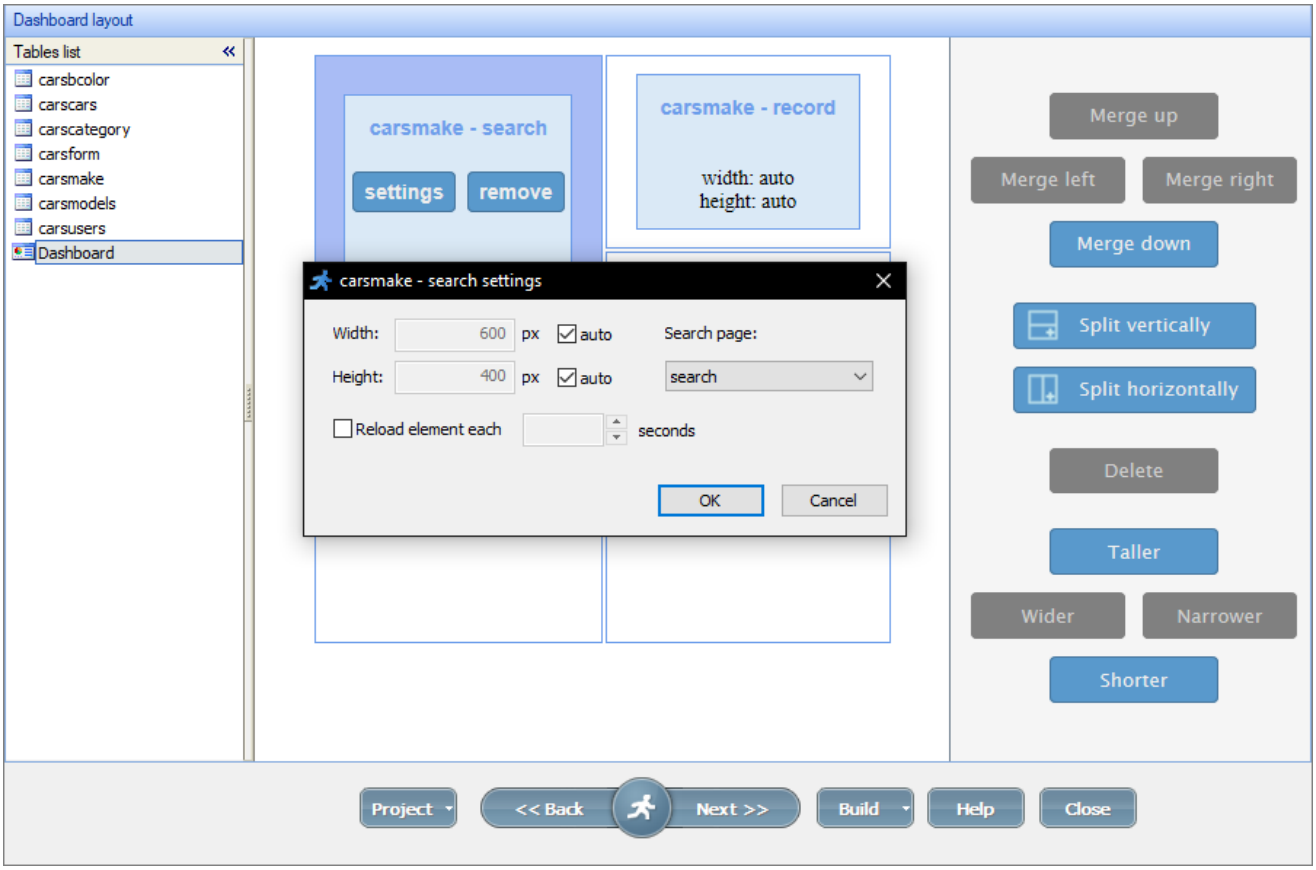
Note: you can also select the initial tab - that tab will be active when the page is loaded.

Here is an example of a dashboard with a single record:



Search

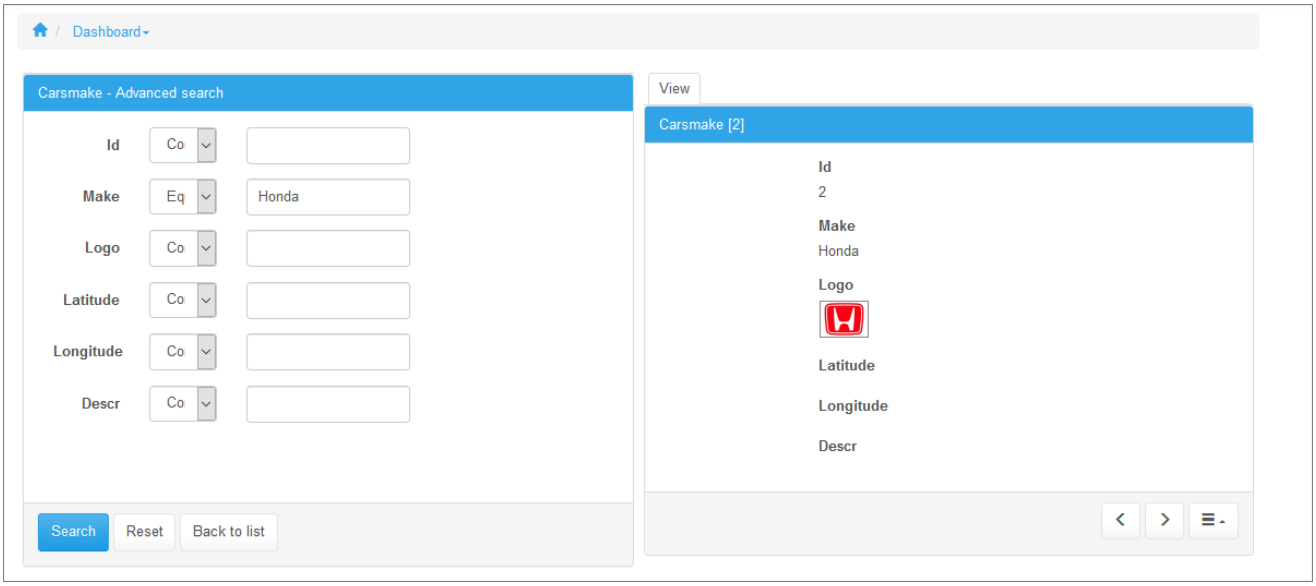
This element displays the **Advanced search** page for the selected table. The search results are displayed in a **Data grid** or a **Single record** element, so you need to have one of them on the dashboard page as well.



Settings

- **Width/Height**. These options define the width and height of the search element. Select the **auto** checkboxes to make the element take the space needed to show all of its content.
- **Reload element each X seconds**. You can set the element to reload each X seconds so that it accurately shows the current records.
- **Search page**. If you have several **Search** pages, you can select the page to show in the dashboard element. Alternatively, you can create a new page with the default appearance. For more information, see Working with additional pages.

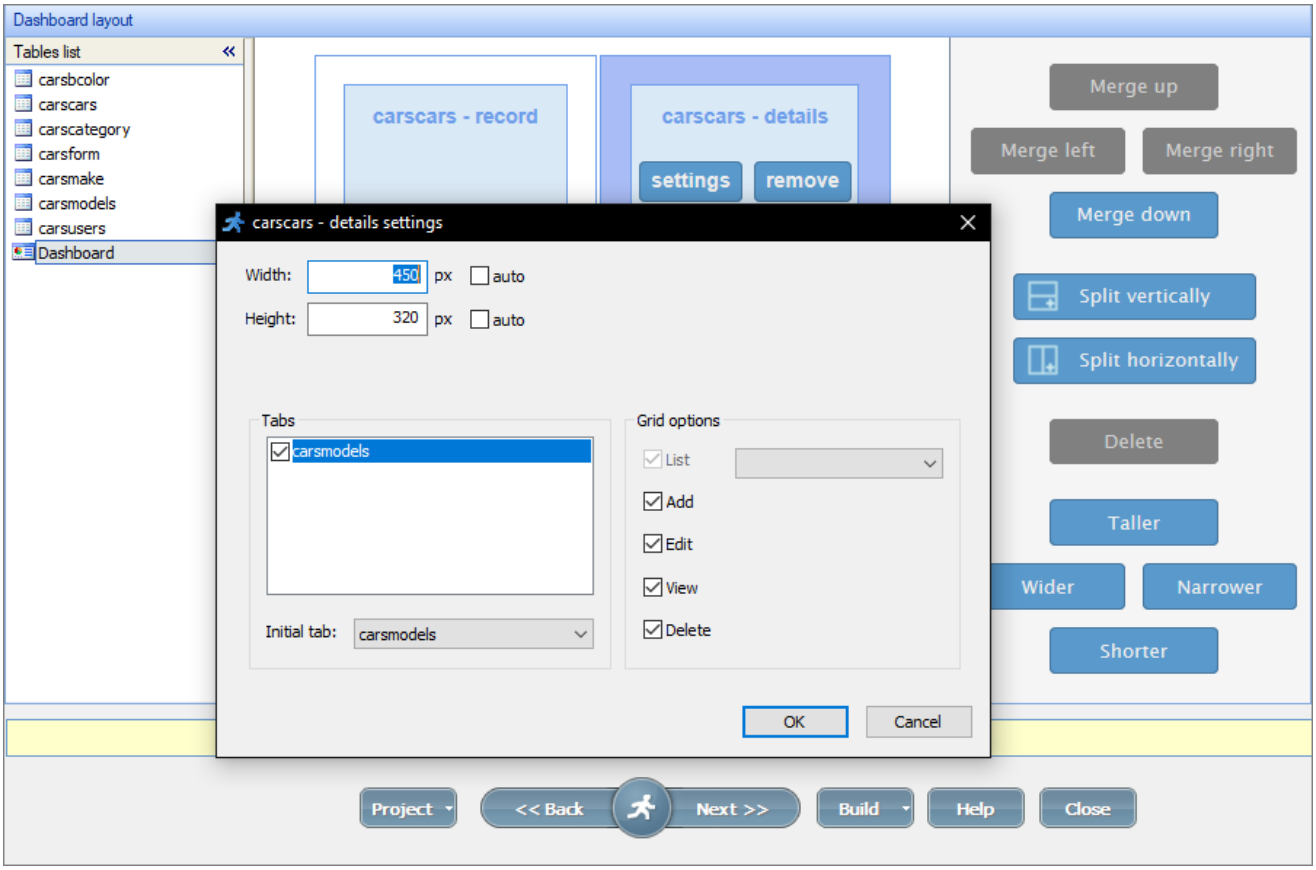
Here is an example of a dashboard with a search and a single record:



Details

This element displays the **details** tables for the selected master table. If the selected table doesn't have a details table, the PHPRunner warns you about it with an error message on the dashboard element. For more information about the master-details relationship, see Master-details relationship between tables.

The contents of the **details** element depend on the selected record in a **Data grid** or a **Single record** element, so you need to have one them on the dashboard page as well.



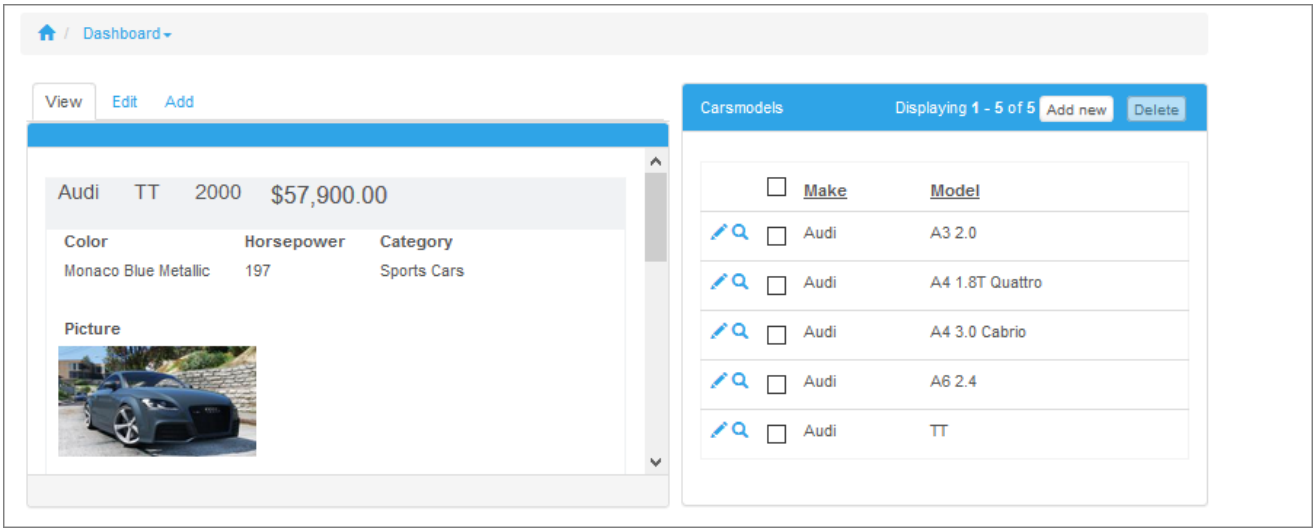
Settings

- **Width/Height**. These options define the width and height of the details element. Select the **auto** checkboxes to make the element take the space needed to show all of its records.
- **Tabs**. This window shows all of the details tables for the selected master table. You can switch them on and off with checkboxes. The selected details tables appear as tabs in the generated app.

> **Note**: you can also select the initial tab - that tab will be active when the page is loaded.

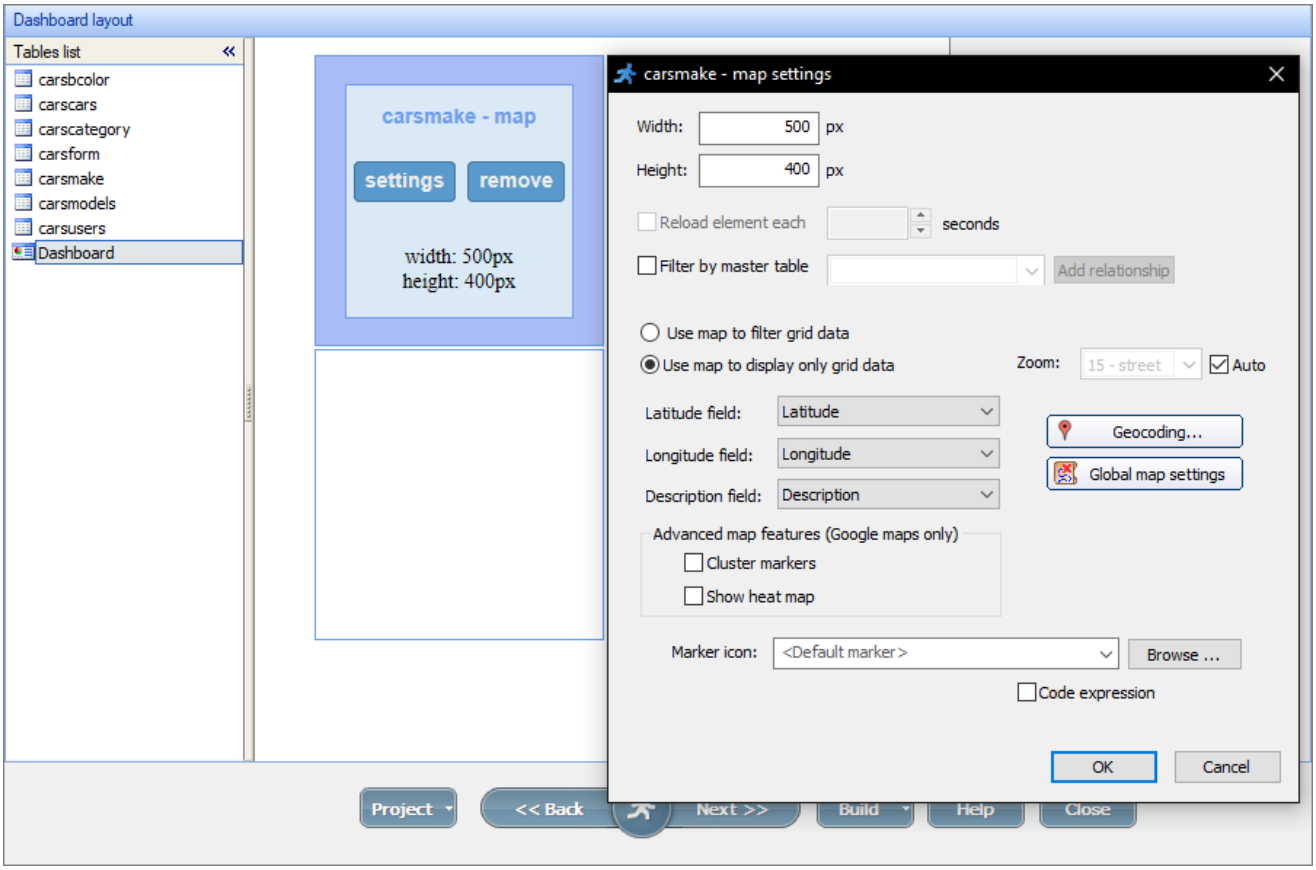
- **Grid options**. Select the buttons/pages to appear on the details element. If you have several **List** pages, you can select the page to show in the dashboard element. Alternatively, you can create a new page with the default appearance. For more information, see Working with additional pages.

Here is an example of a dashboard with a **data grid** and a **details** element. In this example, the tables are linked through the *Make* field, with *carscars* as a master table:



Map

This element displays the **map** for the selected table. The table should have the *latitude* and *longitude* fields for the **map** to function correctly. Alternatively, you can set up Geocoding for the address fields of the table. For more information about maps, see Insert Map.

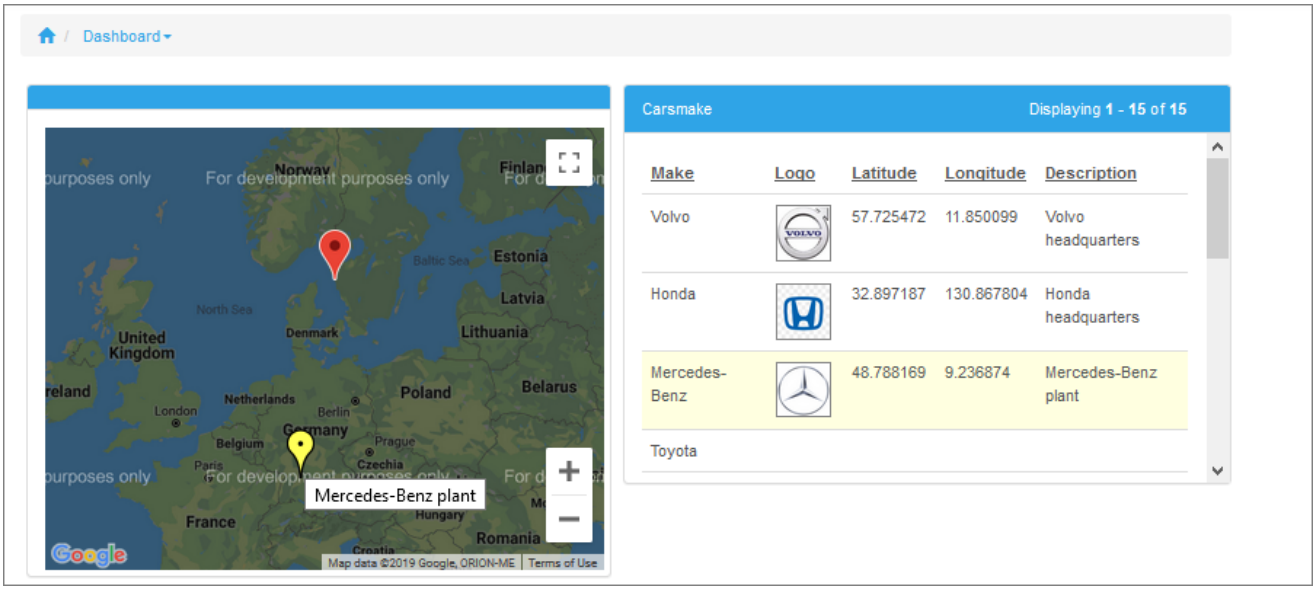


Settings

- **Width/Height**. These options define the width and height of the map element.
- **Reload element each X seconds**. You can set the element to reload each X seconds so that it accurately shows the current records.
- **Filter by master table**. This option makes dashboard elements display the data depending on other dashboard elements. You can use this option for cascading master-details elements (for example, *customers -> orders -> order details*), or when a map is used as the details. For more information, see Master-details dashboard.
- **Use map to filter/display only grid data**. The **filter** option makes the grid/record show only the records with marker icons that are visible on the map screen. The display option shows all of the records in the grid/record. In both cases, clicking on the marker icon in the map highlights the record in the data grid or the **View** tab of the single record element.
- **Zoom**. This option sets the zoom of the map to a specified level, or you can select the **Auto** checkbox to set the zoom level that shows all of the marker icons.

- **Latitude/Longitude/Description field**. Select the fields that contain latitude, longitude, and the marker icon description, respectively.
- **Geocoding**. This option lets the user automatically convert the address fields into the latitude and longitude to show on the map each time the records are added or updated. For more information, see Geocoding and Insert map.
- **Global map settings**. This option allows you to select the map provider and insert its API key if required.
- **Advanced map features**. The **Cluster markers** and **Show heat map** options are available only for Google maps. For more information, see Insert map.
- **Marker icon**. You can select and use custom marker icons instead of the default ones. Alternatively, you can use the PHP expression option to set the custom icons.

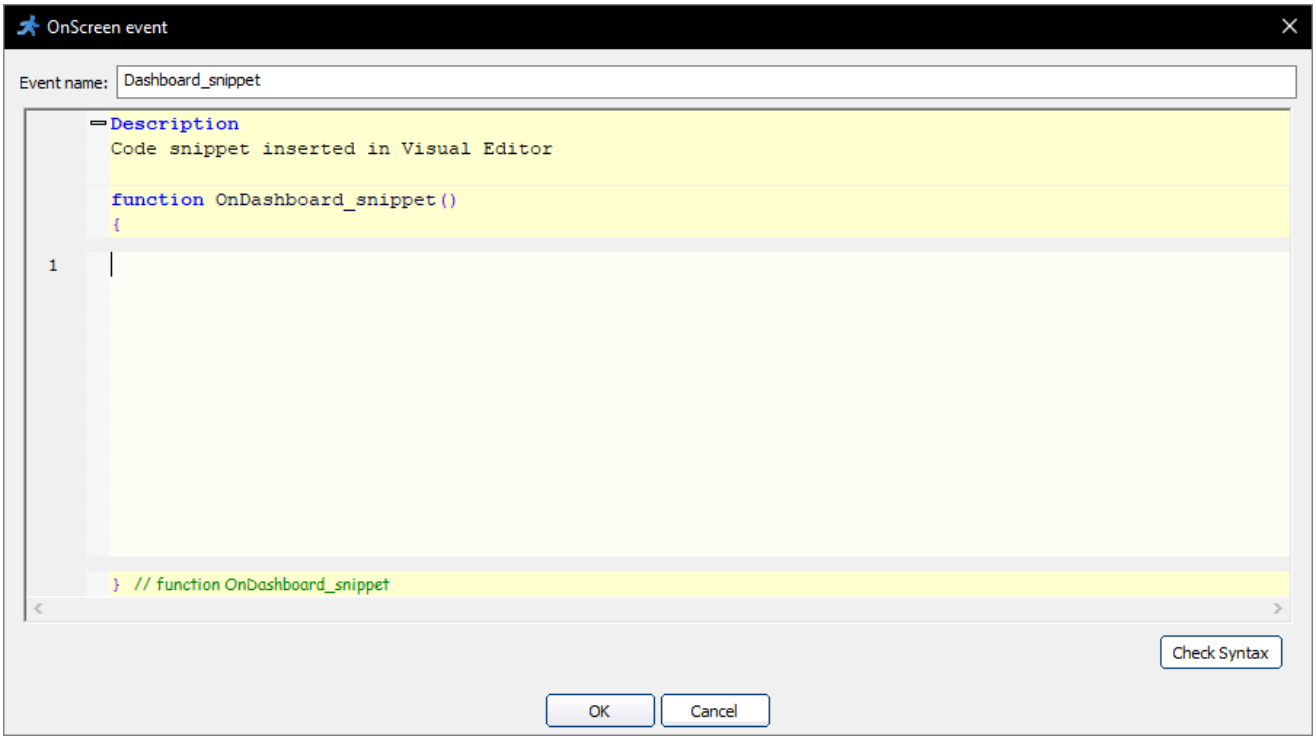
Here is an example of a dashboard with a **map** and a **data grid** with the **Use map to display only grid data** option enabled:



Code snippet

You can insert any custom code snippet as a dashboard element. To do so, click **Add** and select **Code snippet**. This code can be used to display current weather, calculate order totals, show the number of active users, or even embed a Youtube video.

When you add a **Code snippet** element, a popup window appears. Insert the code into this window and click OK.

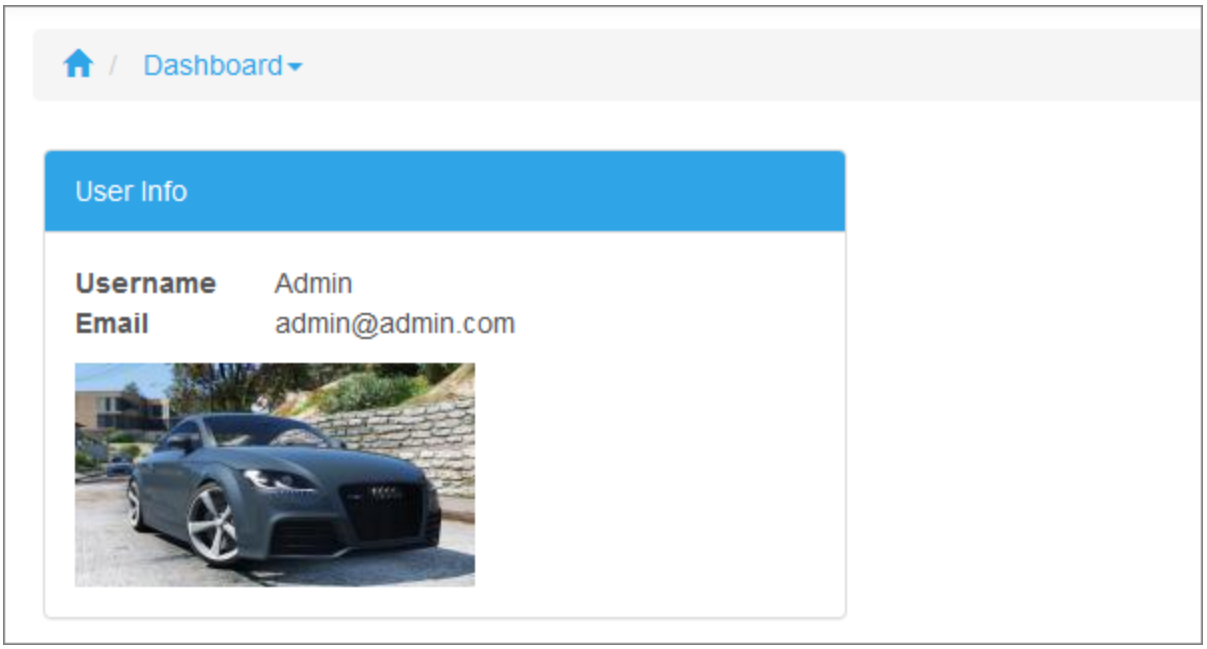


The **Code snippet** element only has the **Width/Height** settings. These settings define the width and height of the element. Select the **auto** checkbox to make the element take the space needed to show all of its contents.

To create a custom dashboard element called "User Info" to view the current username, email and avatar, insert the code below as a Code snippet:

```
$header = "User Info";
if($_SESSION["user_id"]){
  $rs = DB::Query("select * from users where id=".$_SESSION["user_id"]);
  $data = $rs->fetchAssoc();
  echo "<table cellpadding=10px cellspacing=0px border=0>";
  echo "<tr><td
width=100px><b>Username</b></td><td>".$data["displayname"]."</td></tr>";
  echo "<tr><td ><b>Email</b></td><td>".$data["email"]."</td></tr>";
  echo "<tr><td colspan=2 style='padding-top:10px'><img width=200px
src='avatars/".$data["image"]."'></td></tr>";
  echo "</table>";
}
else
  echo "No information";
```

As a result, users can see their user info in the dashboard:



You can also customize the dashboard elements further with Custom CSS. For example, to show the total number of orders, add this code snippet to the dashboard element:

```
$res=DB::query("select count(id) as cnt FROM orders");
$data = $res->fetchAssoc();
$res = "<div class='info-box'>";
$res.=  "<span class='info-img img_bgcolor_red glyphicon glyphicon-star'></span>";
$res.= "<div class='dashtext'>";
$res.= "<b>Orders</b><br>";
$res.= $data["cnt"];
$res.= "</div>";
$res.= "</div>";
echo $res;
```

Proceed to the **Style Editor** for the dashboard page and insert this code as a **Custom CSS**:

```
.panel-heading{
   display:none;
}
.panel-body{
   padding:0px !important;
}
```

```
.panel-primary {
 border-color: white !important;
}
.panel {
  margin-bottom: 0px !important;
   border: 0px !important;
}
.info-box{
      display: block;
    min-height: 90px;
    width: 100%;
    border-radius: 2px;
}
.info-img{
    display: block;
    float: left;
    height: 90px;
    width: 90px;
    text-align: center;
    font-size: 45px;
    line-height: 90px;
      color:white;
}
.img_bgcolor_red{
   background-color:#dd4b39 !important;
}
.dashtext{
    border-radius: 2px;
    border: 1px solid #e6e6e6;
    min-height: 91px;
    padding:7px;
    padding-left:97px;
    line-height: 1.6em;
    background-color:#f5f5f5;
}
```

Here is how the result looks like:

Note: See this live demo for inspiration.

See also:

- Master-details dashboard
- Dashboard search
- Master-details relationship between tables
- Choose pages screen
- Insert map
- Creating charts
- Creating and configuring reports

2.11.2 Master-details dashboard

Quick jump

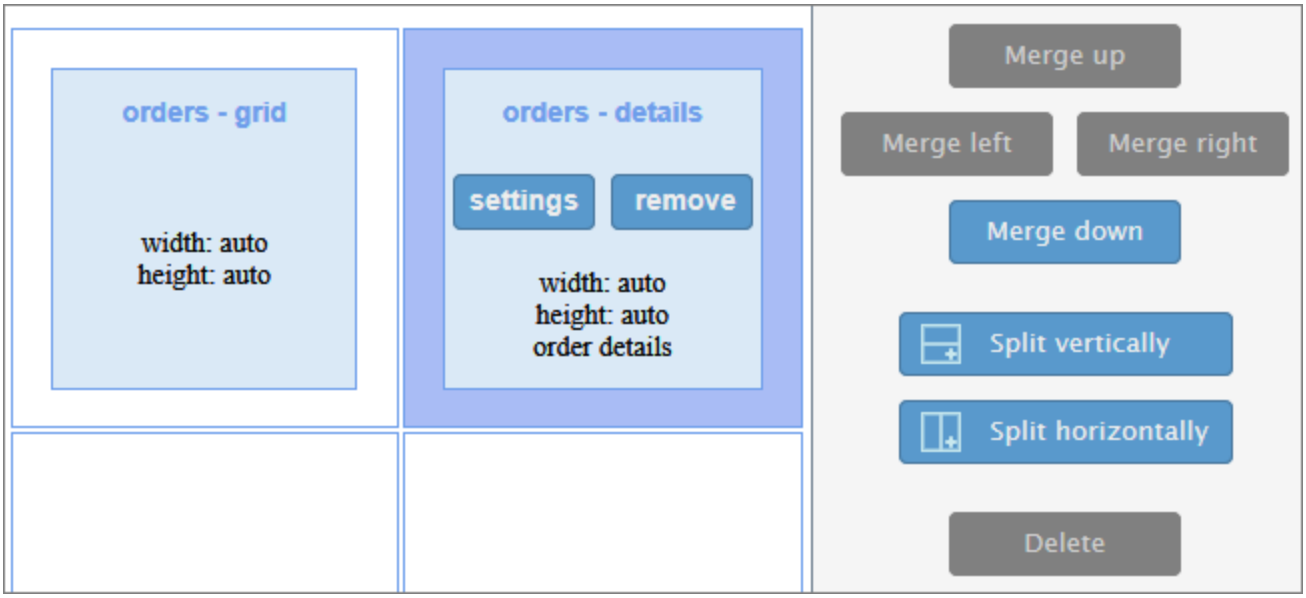
With master-details dashboards, the data you select in the master table filters the records in the details table. Make sure you have a master-details relationship between two or more tables before creating a master-details dashboard.

There are two ways to organize your master-details dashboard:

- Add a master table to the dashboard twice: as a data grid and then as details. This way, if the master table has several details tables, you can display all the details tables (each on a separate tab) or select the tables to display.
- Add a master table and a details table as data grids and enable the **Filter by master table** option for the details table. Use this option to display nested master-details tables, such as *Customers-Orders-Order details*. You can also display the master-details relationship between other elements, for example, between a table and a map.

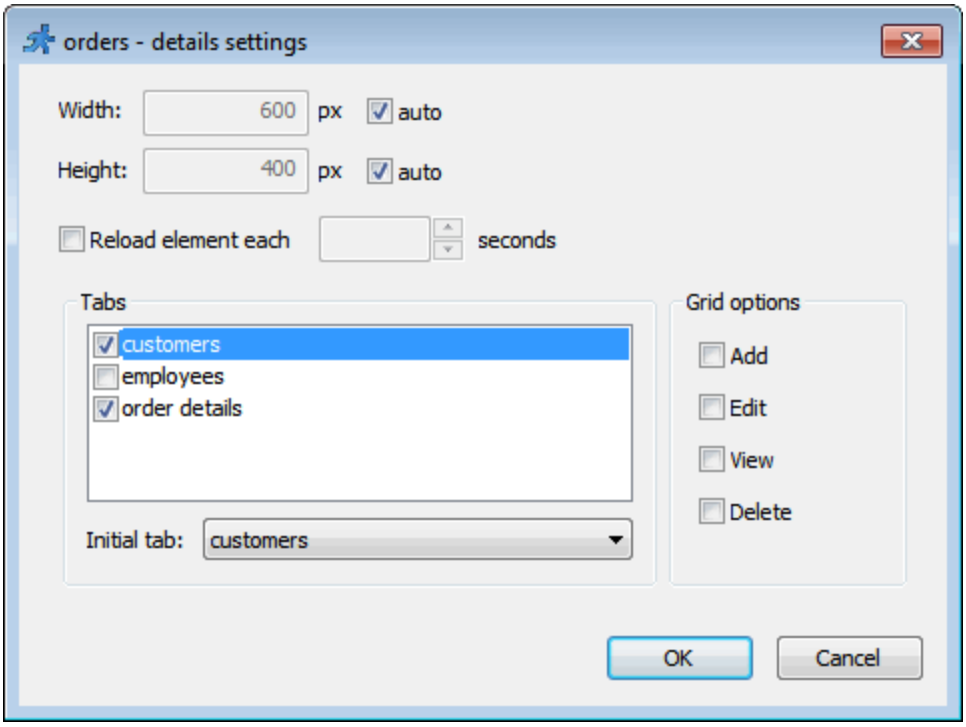
Master table as a data grid and as details

1. Add a master table to the dashboard as a data grid and then once again - as details.

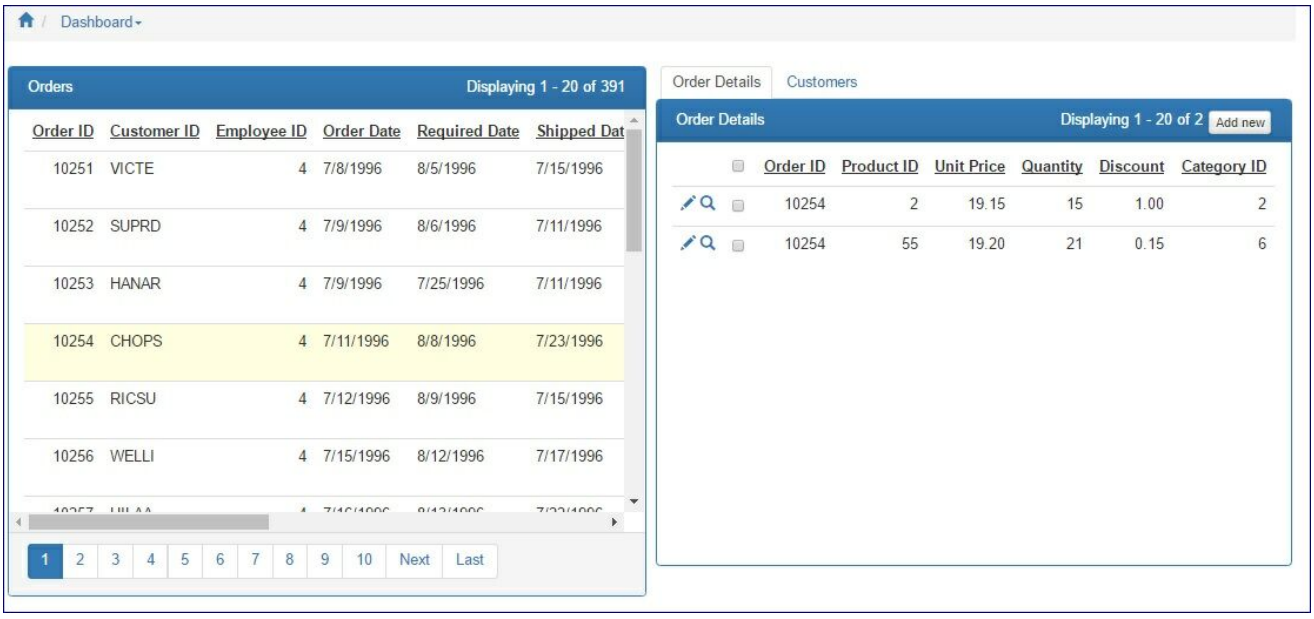


2. Click settings on the grid element and select the grid features to be displayed in a browser (**Add**/**Edit**/**View**/**Delete record** checkboxes).

3. If the master table has several details tables, click settings on the details element, select the details tables to display, and set the initial one. Select the grid features to be displayed for the details table (**Add**/**Edit**/**View**/**Delete record** checkboxes).



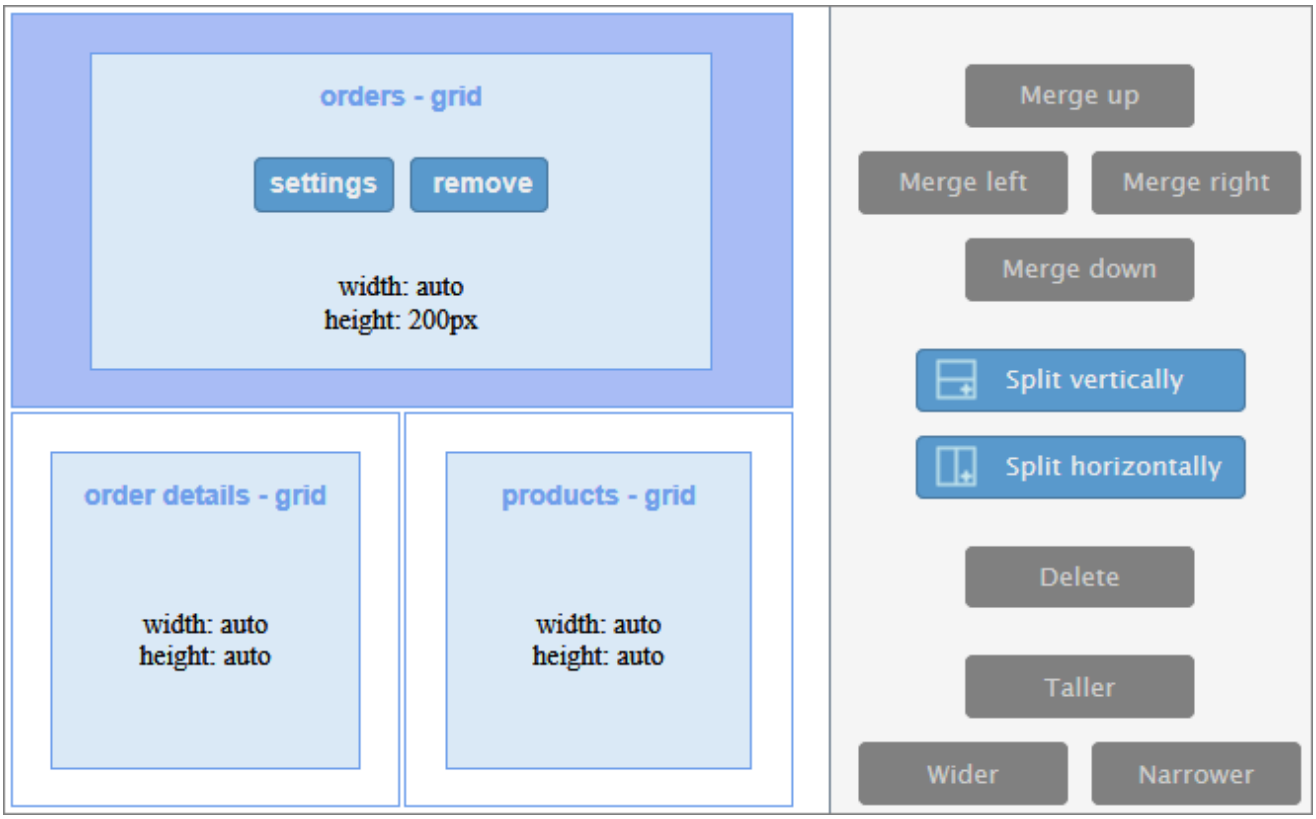
Here is an example of a master-details dashboard; you can see the list of orders and the order/customer details:



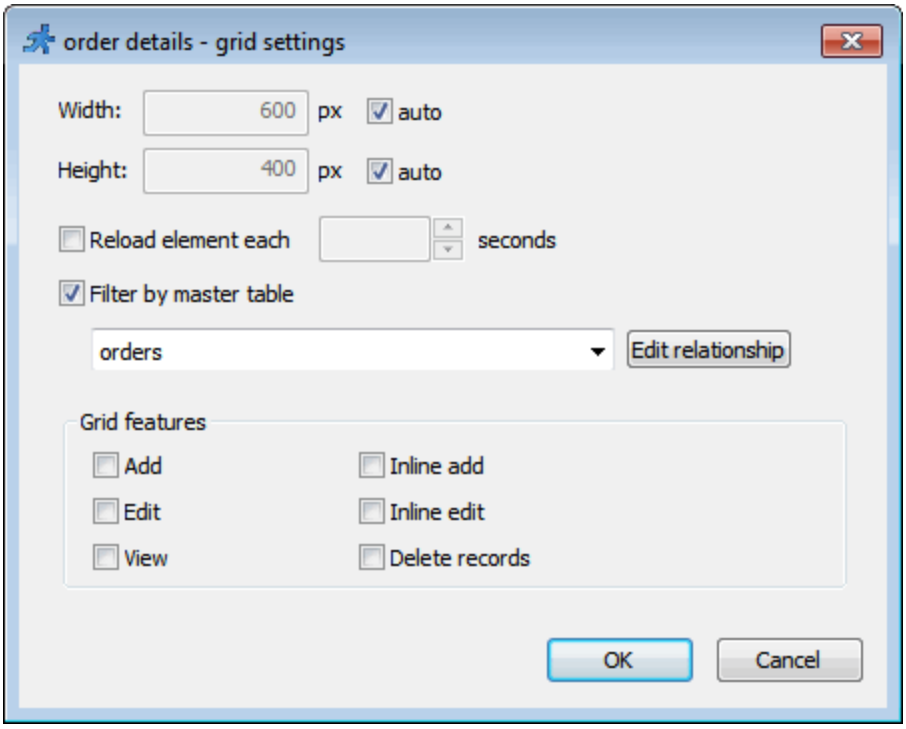
Master table and details tables added as data grids

Let's create a dashboard that displays data from nested master-details tables *Orders -> Order details -> Products*. This dashboard updates the *Order details* grid upon choosing a specific *Order* record, and *Products* grid - upon choosing a specific *Order details* record.

1. Add the *Order* (master), *Order details* and *Products* (details) tables as data grids.

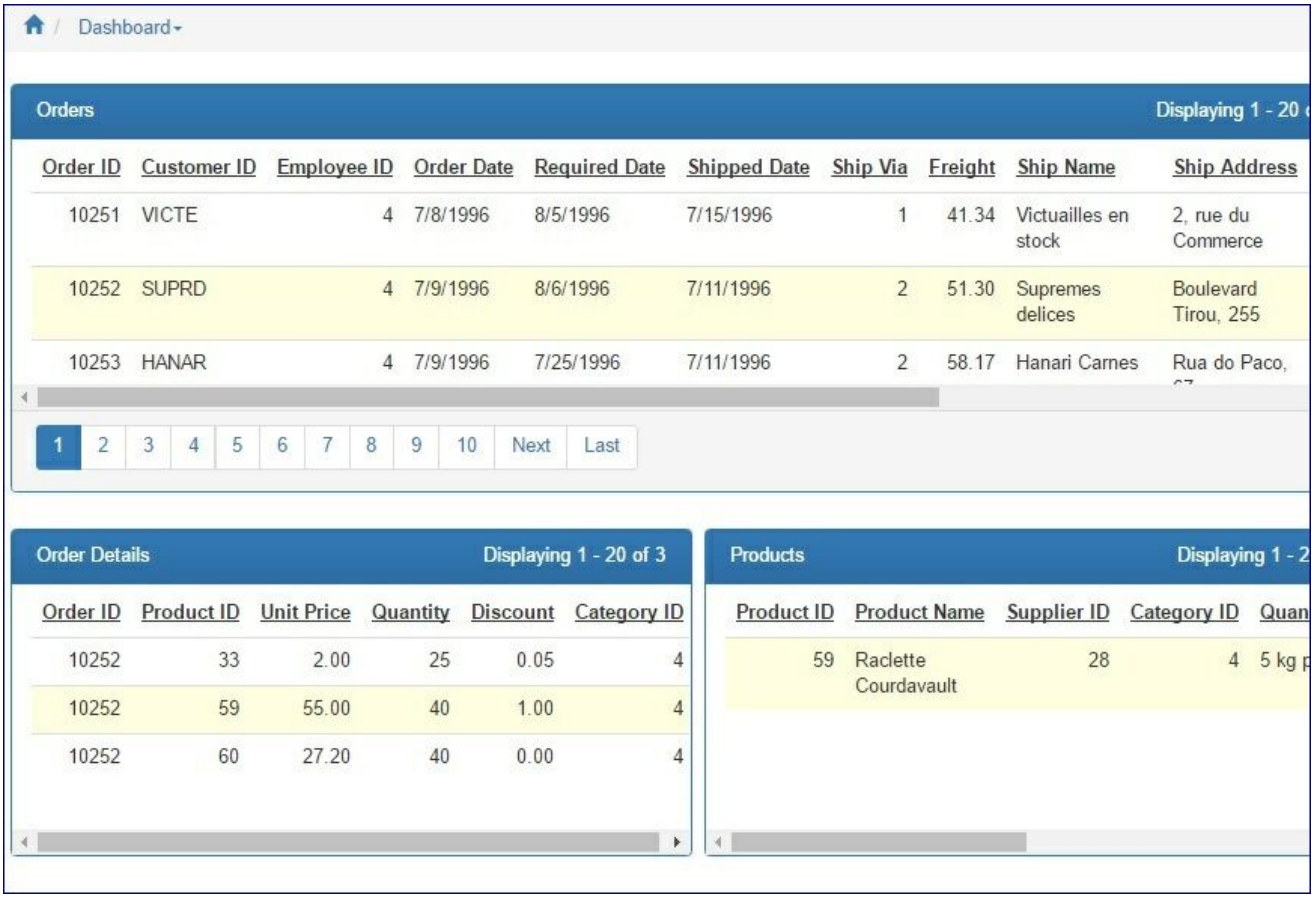


2. Click settings on the *Order details* grid element and enable the **Filter by master table** option. Select *Orders* as the master table.



3. Click settings on the *Products* grid element and enable the **Filter by master table** option. Select *Order details* as the master table.

Here is how the dashboard page looks like in the browser:



Master-details relationship between a chart and a data grid

You can apply the same principles when creating the dashboard with a chart as a master and a grid as details.

1. Create a chart as a master and add the details table to it as described in the article Charts and reports as master-details tables.

2. Create a dashboard and add the chart to it.

3. Add the same chart to the dashboard again, but this time choose details in the **Choose element to add** dialogue.

4. Adjust chart or grid settings, if necessary.

The resulting dashboard page looks like this in the browser. When you click on the chart bar, you filter the details grid.

See also:

- Creating dashboards
- Dashboard search
- Master-details relationship between tables
- Choose pages screen
- Creating charts
- Creating and configuring reports

2.11.3 Dashboard search

Quick jump

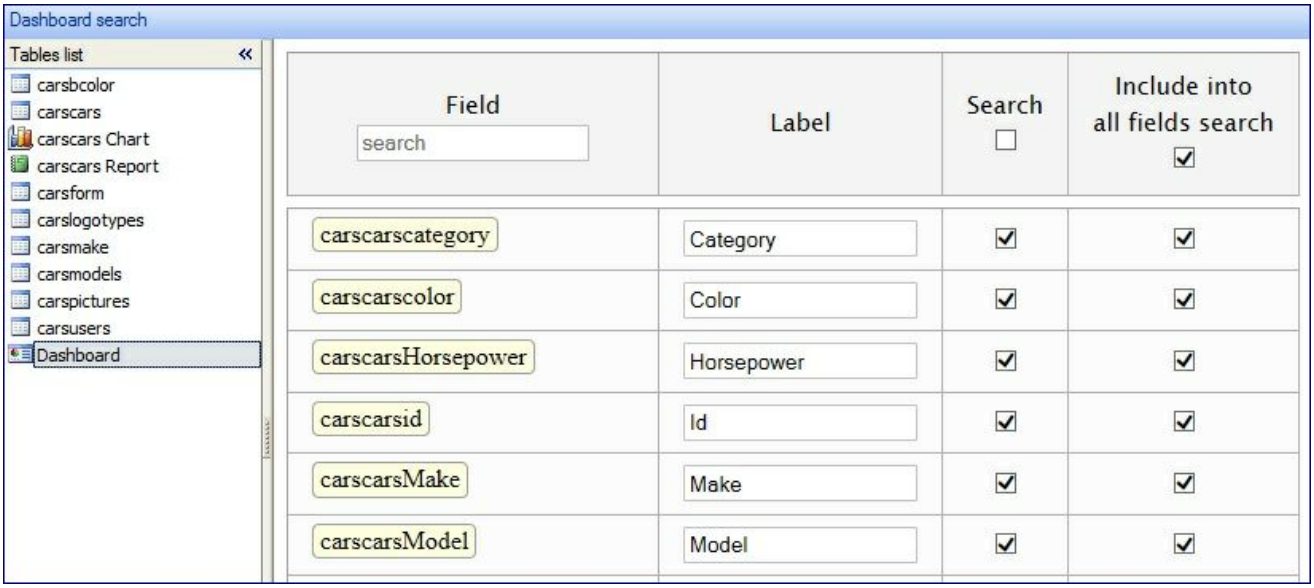
Dashboards feature two different types of search:

1. Search across all dashboard items. You can set this option up on the **Dashboard search** page by choosing the searchable fields. This type of search is available as a separate page in the generated application and also as the Search all fields search box on the dashboard page.

2. Add the search page for any dashboard table as a dashboard item. When you add a new dashboard item, select any table, view, chart, or report, and choose **Search** as an option. This option works only if you also add this table/view/chart/report to the dashboard.

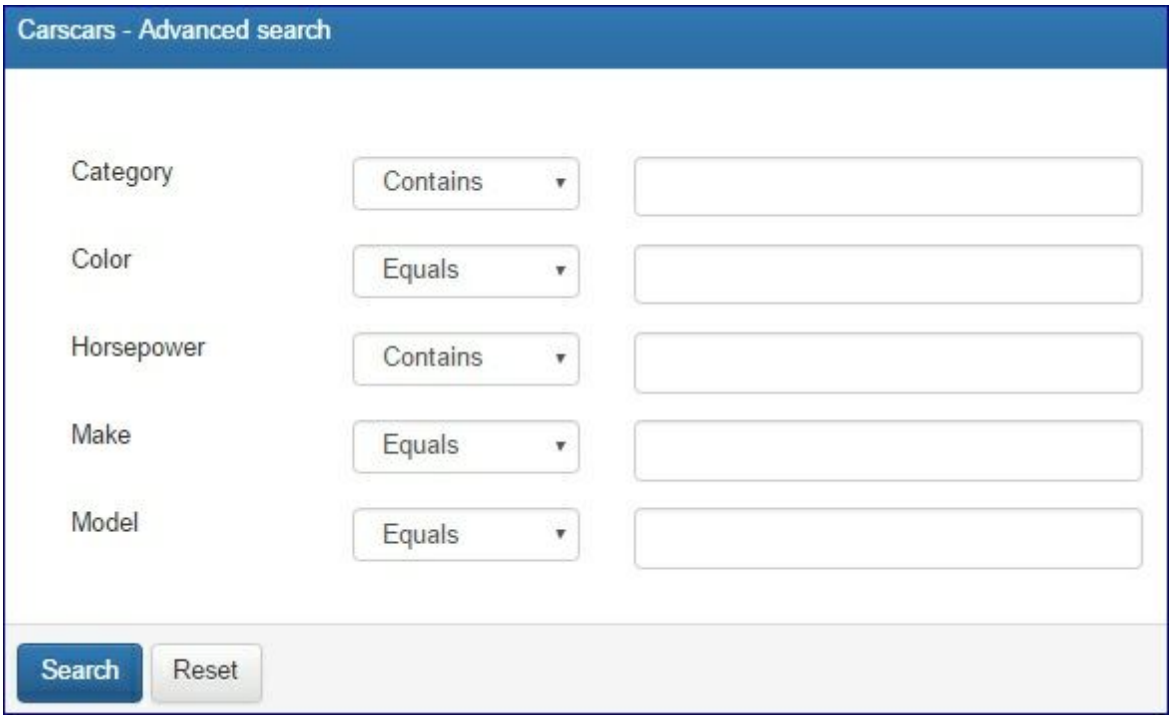
Search across all dashboard items

On the **Dashboard search** screen, select the fields you want to be searchable.



Drag fields from different tables/elements into a single cell to enable searching through all the elements that belong to the cell at the same time. For example, you can simultaneously search the *Orders* table and *Orders* Chart.

The **Search** option allows adding fields to the **Advanced search** page.

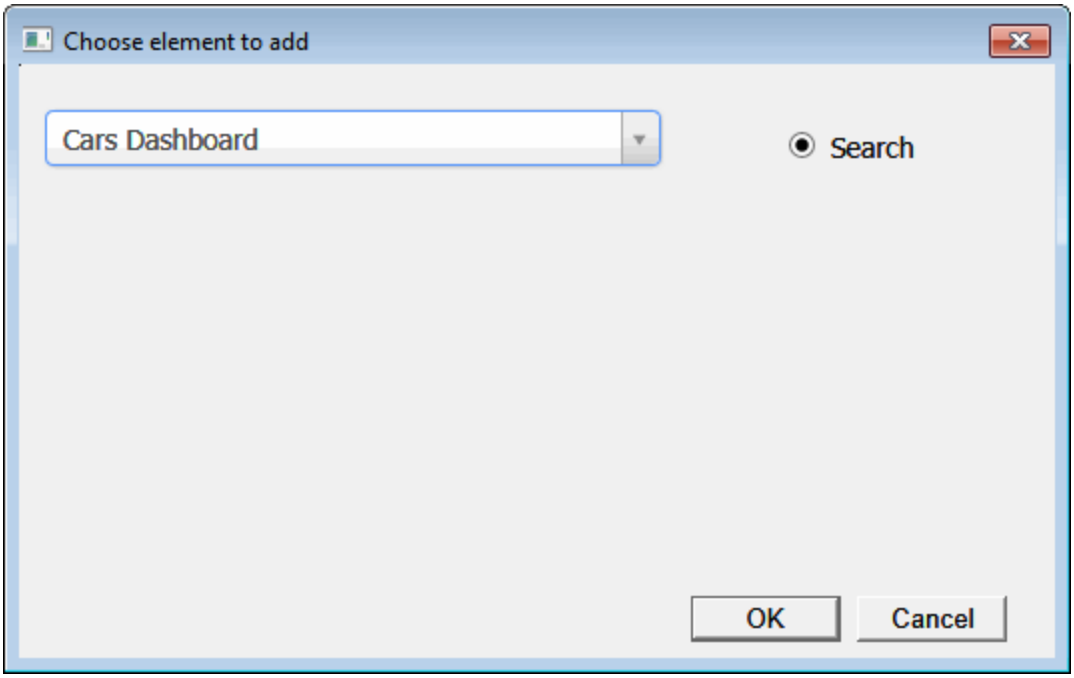


The **Include into all fields search** option allows adding fields to the **All fields quick search**.

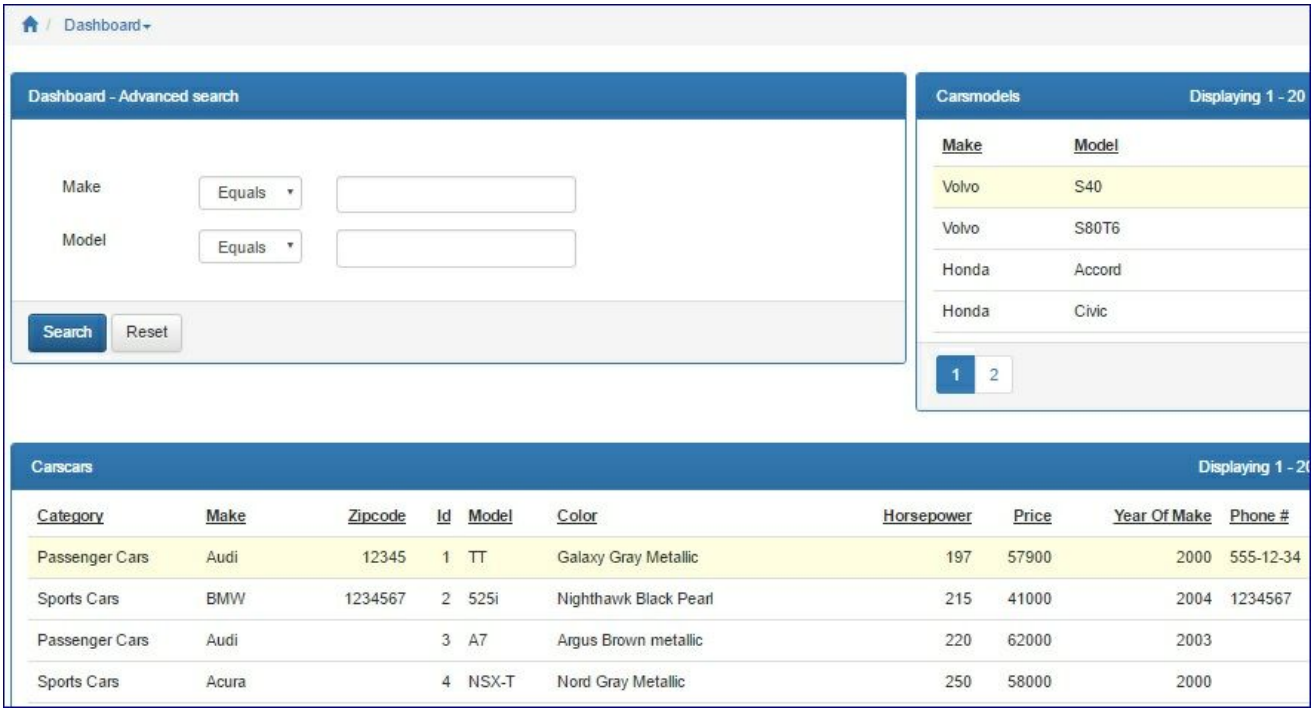
If you want some dashboard elements not to display the data initially, apply the **Hide data until search** option to the elements in the Search and Filter settings on the Choose fields screen.

Add a dashboard search page to the dashboard

You can add a dashboard advanced search page to the dashboard as one of the elements. On the **Dashboard layout** screen, click **add** and select the dashboard itself.

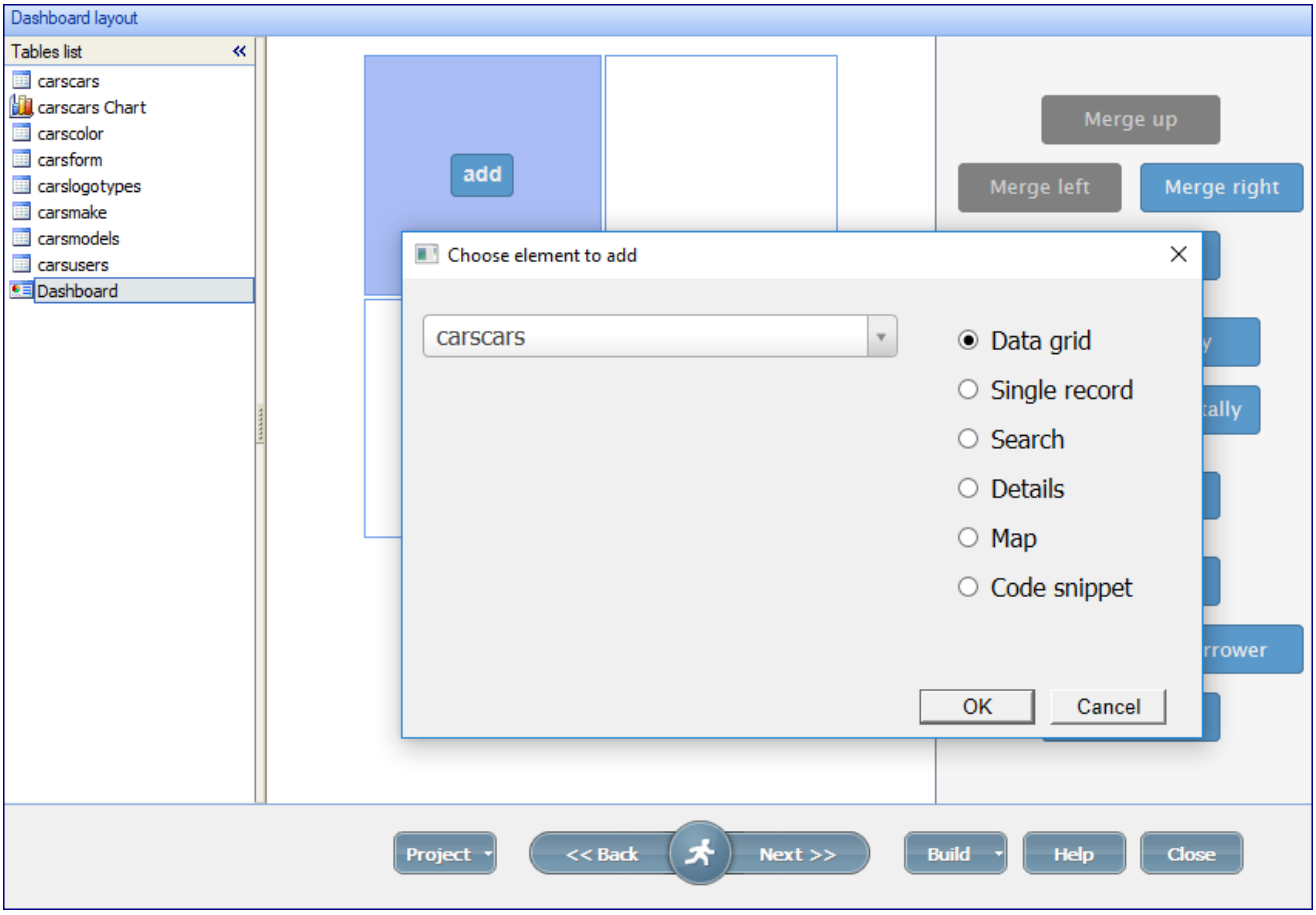


An example of a dashboard with the search element:

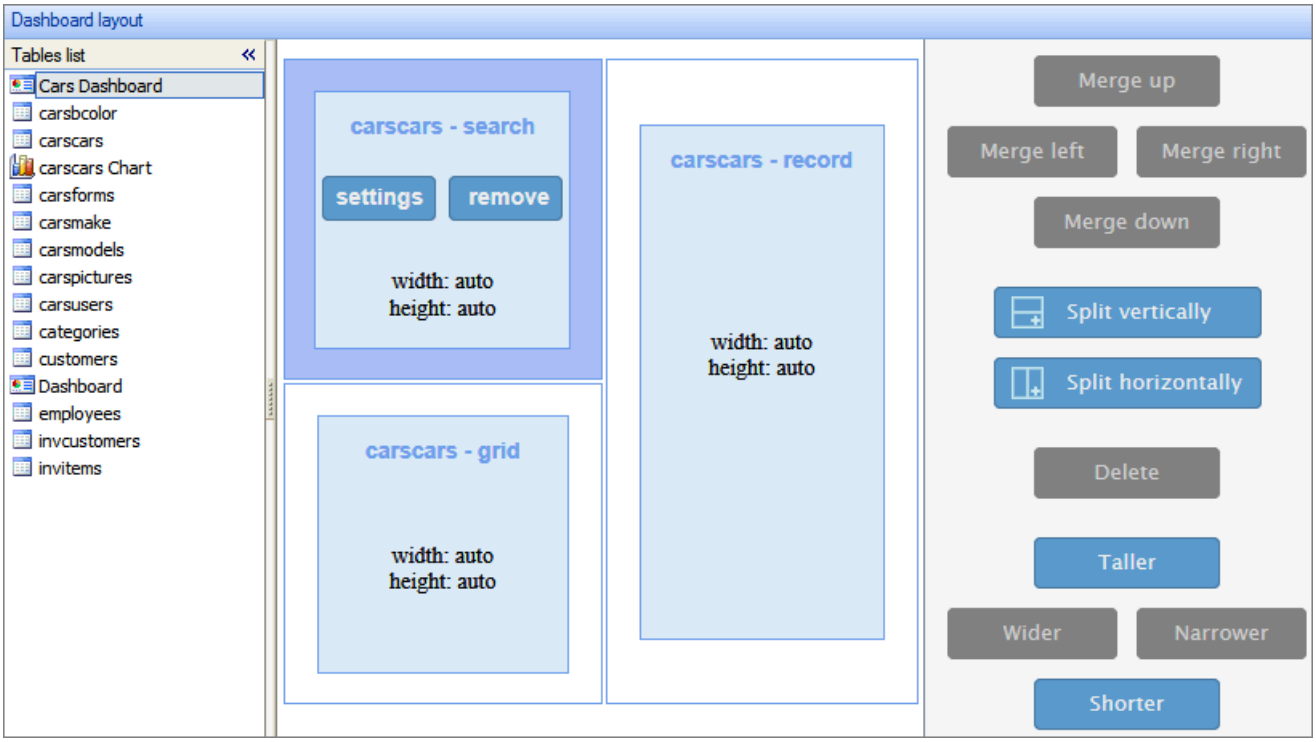


Add the search page for any dashboard table as a dashboard item

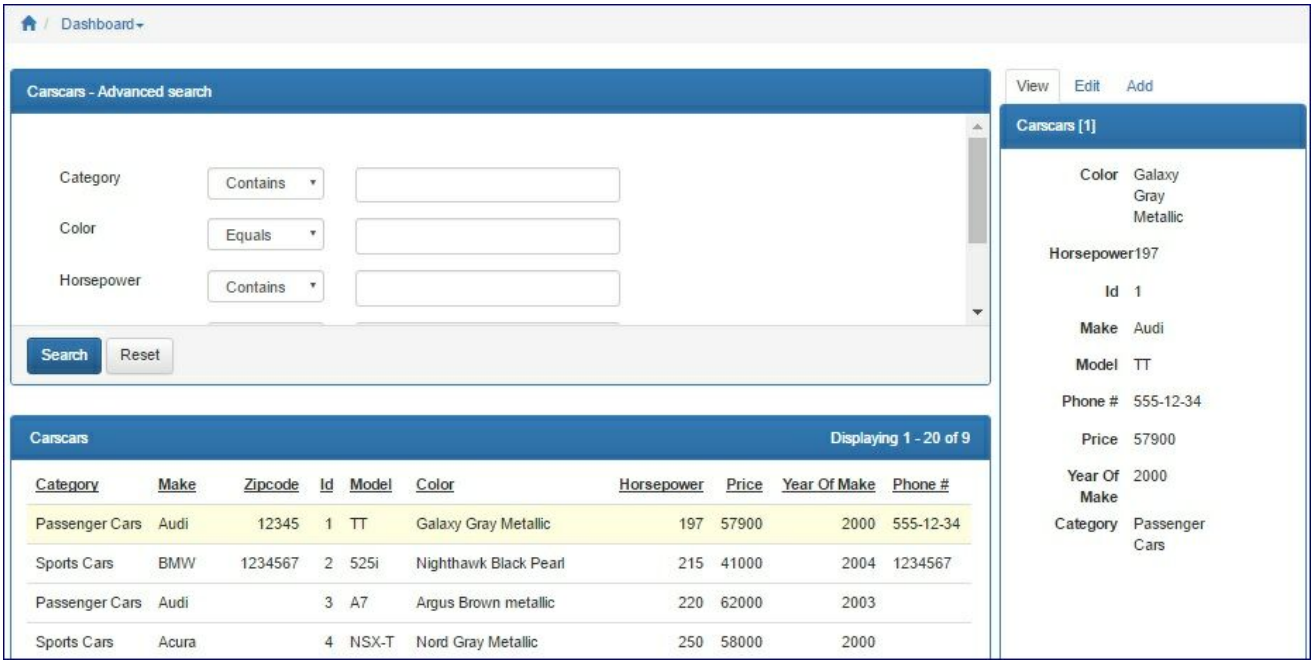
To add a search page to the dashboard, click **add**, select any table/view/chart/report, and choose **Search**.



Make sure you've also added the selected table/view/chart/report to the dashboard as a data grid, single record, chart or report.



How it looks like in the browser:



See also:

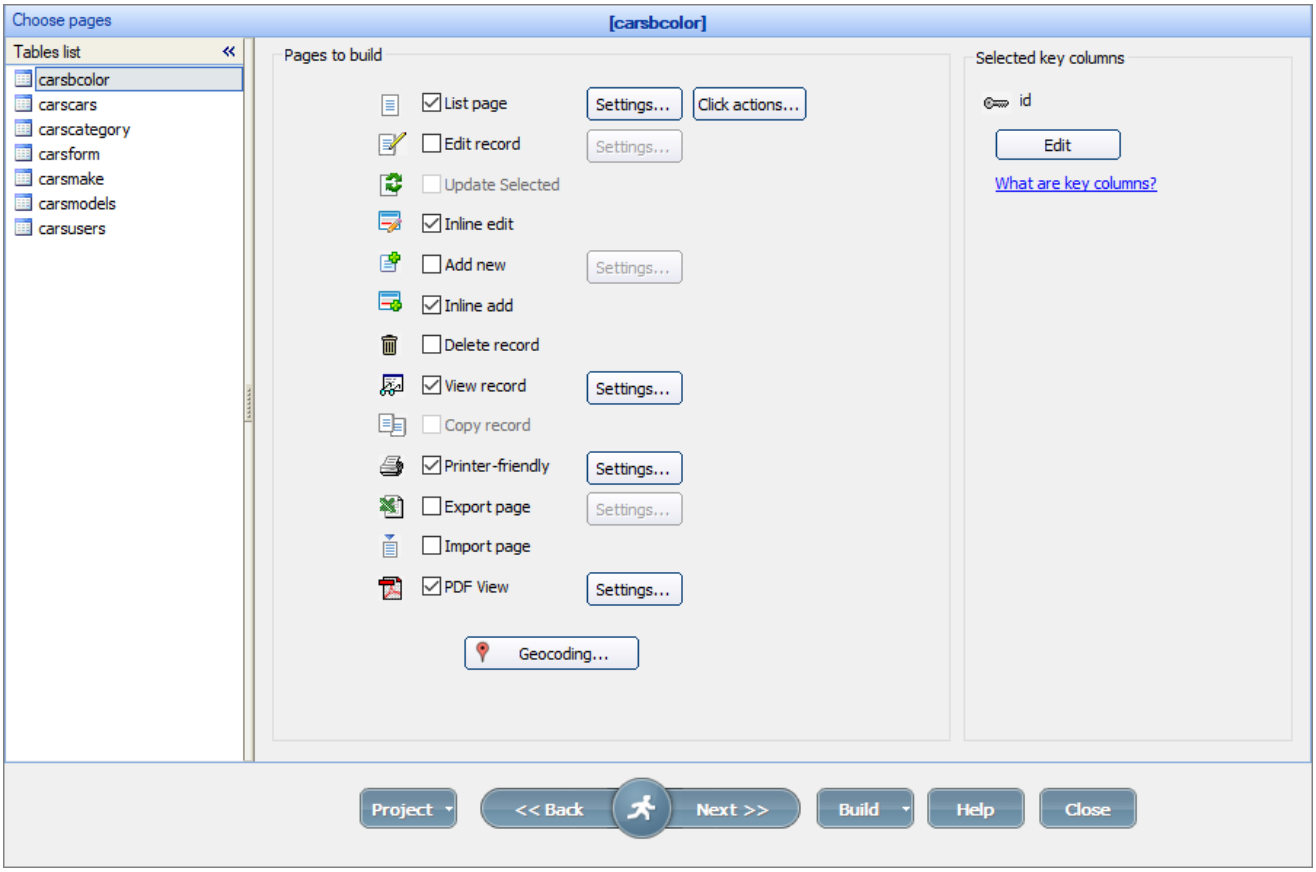
- Creating dashboards
- Master-details dashboard
- Master-details relationship between tables
- Choose pages screen
- Insert map
- Creating charts
- Creating and configuring reports

2.12 Choose pages screen

2.12.1 Choose pages screen

Quick jump

The **Choose pages screen** allows you to select the pages and functionality you want to include in the generated app.



Enable the options under **Pages to build** to add pages or functionality into the build.

Table pages

The checkboxes that add pages into the generated app are:

Checkbox	Description of the added page
List page	A page that shows the table records and the buttons to work with these records.
Edit record	A page/popup that edits the selected record.
Add new	A page/popup that adds new records to the table.
View record	A page that shows the selected record only, without the additional buttons.

Printer-friendly	A page that shows the **List**/**View** pages in a printer-friendly way.
Export page	A page/popup that allows exporting the records into a DOC, XLSX, or CSV file.
Import page	A page/popup that allows importing records into the table from a CSV, XLS/XLSX file, or plain text.

These pages are called **Table pages** because they show the table records or allow working with the tables. To learn more about working with table pages in Page Designer, see Working with table pages.

You can click the **Settings** buttons next to the checkboxes to adjust the settings for the corresponding pages.

> **Note**: if the **Edit record**, **View record** checkboxes are disabled, you may need to select key columns for the table.

Functionality

The checkboxes that add functionality to the **List**/**View** pages are:

Checkbox	Description
Update selected	Adds the **Update selected** button to the **List** page that allows editing multiple records at once.
Inline edit	Adds the **Inline edit** button to the **List** page that allows you to edit multiple records without leaving the **List** page.
Inline add	Adds the **Inline add** button to the **List** page that allows you to add multiple records without leaving the **List** page.
Delete record	Adds the **Delete records** button to the **List** page that allows you to delete the selected records.
Copy record	Adds the **Copy records** button to the **List** page that allows you to copy the selected records.
PDF View	Adds the **PDF View** button to the **List**/**View** pages that generates a PDF document with the contents of the page.

As the **Update selected** function requires the **Edit** page to work, you need to have the **Edit** record checkbox enabled.

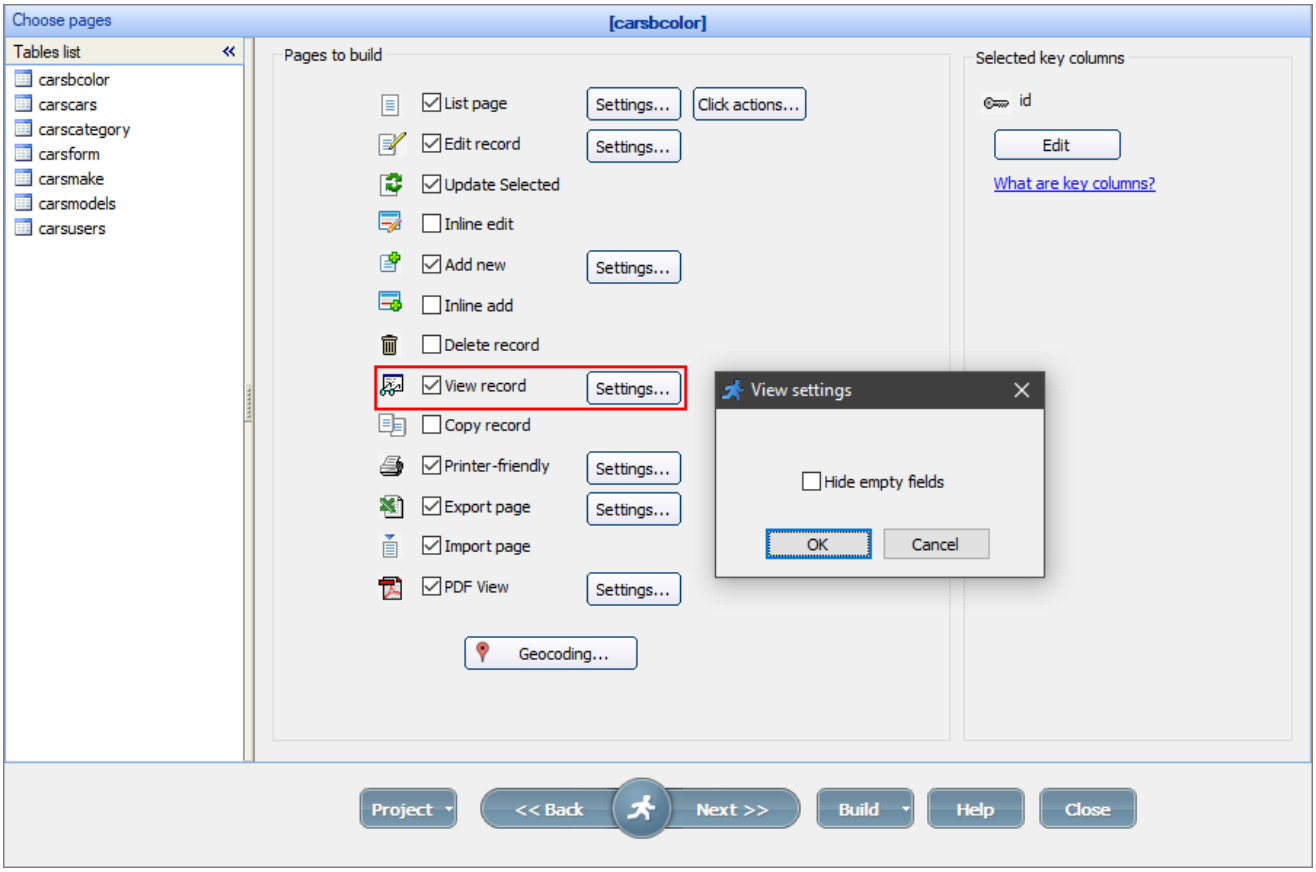
You can click the **Settings** button next to the **PDF View** checkbox to adjust settings for the generated PDF document.

> **Note**: the **Inline add**/**edit** functions use the same Edit as types as the regular **Add**/**Edit** pages.

> **Note**: if the **Delete record**, **Copy record** checkboxes are disabled, you may need to select key columns for the table.

View record settings

Click the **Settings** button next to the **View record** checkbox to edit the **Hide empty fields** option. When selected, this option shows only the fields with values on the **View** page.



See also:

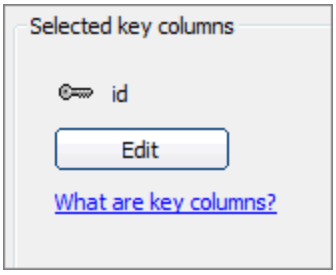
- Key columns
- List page settings / Click actions
- Add/Edit page settings
- CAPTCHA on Add/Edit pages
- Update selected
- Export/Import pages
- Printer-friendly/PDF view settings
- Geocoding

2.12.2 Key columns

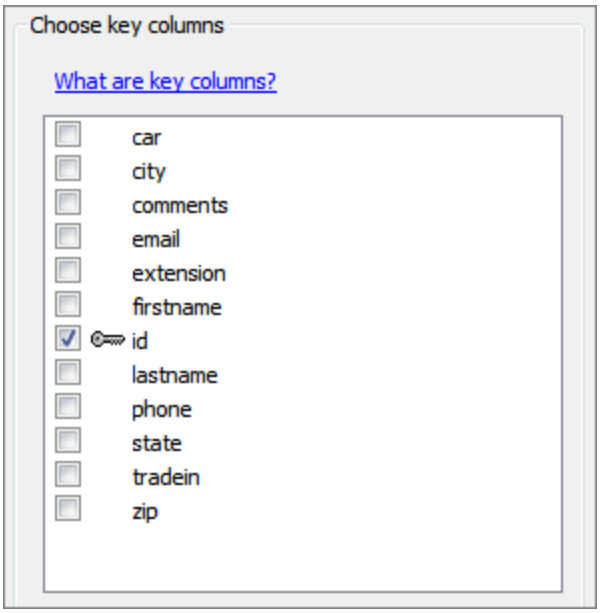
The **Key column** is a table field that lets you query and modify each record individually, without altering other records in the same table. The values of a key column are unique.

You can specify any number of the columns that form a key. In general, one key column is sufficient, as it forms the **primary key**.

The key column is required for the tables that need the **Edit**, **Delete** or **View** functionality, work with images, or **Print**/**Export** the selected records. Do not remove the key columns from the query.



To change the selected key columns, proceed to the Choose pages screen and click **Edit** under the **Selected key columns**.



The best option is to use an auto-incremented field as a **primary key** field so that the database generates a unique key value every time you add a new record.

With MS Access, the best choice for a key column is an *AutoNumber* field.

	Field Name	Data Type
🔑	ID	AutoNumber
	Make	Text
	Model	Text
	YearOfMake	Number
	Picture	OLE Object
	Horsepower	Number
	EPACity	Number
	EPAHighway	Number
	Price	Currency
	Date Listed	Date/Time
	Phone #	Text
	UserID	Number

With SQL Server, use an INT IDENTITY field; with MySQL - use an INT AUTO_INCREMENT field.

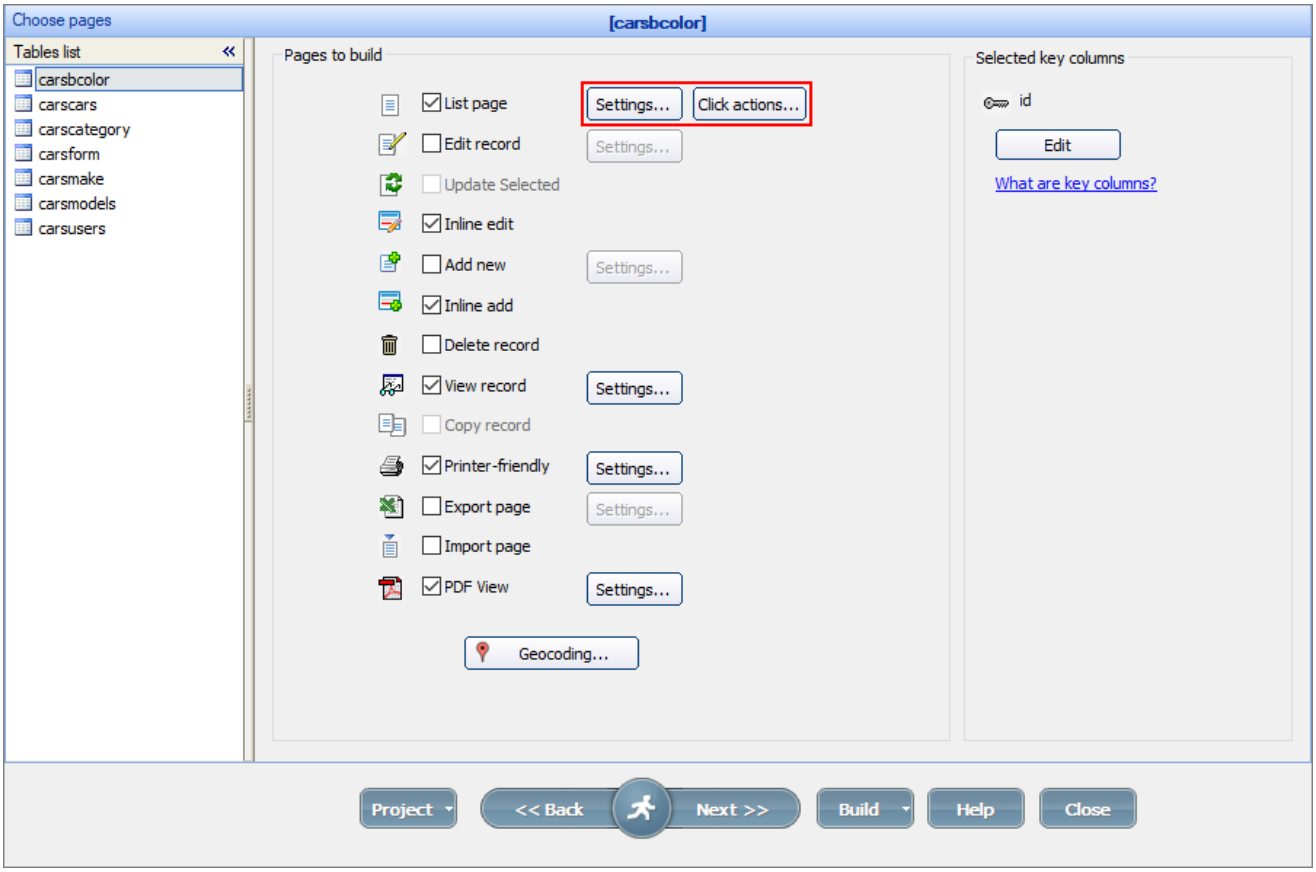
See also:

- Choose pages screen
- List page settings / Click actions
- Add/Edit page settings
- CAPTCHA on Add/Edit pages
- Update selected
- Export/Import pages
- Printer-friendly/PDF view settings
- Geocoding

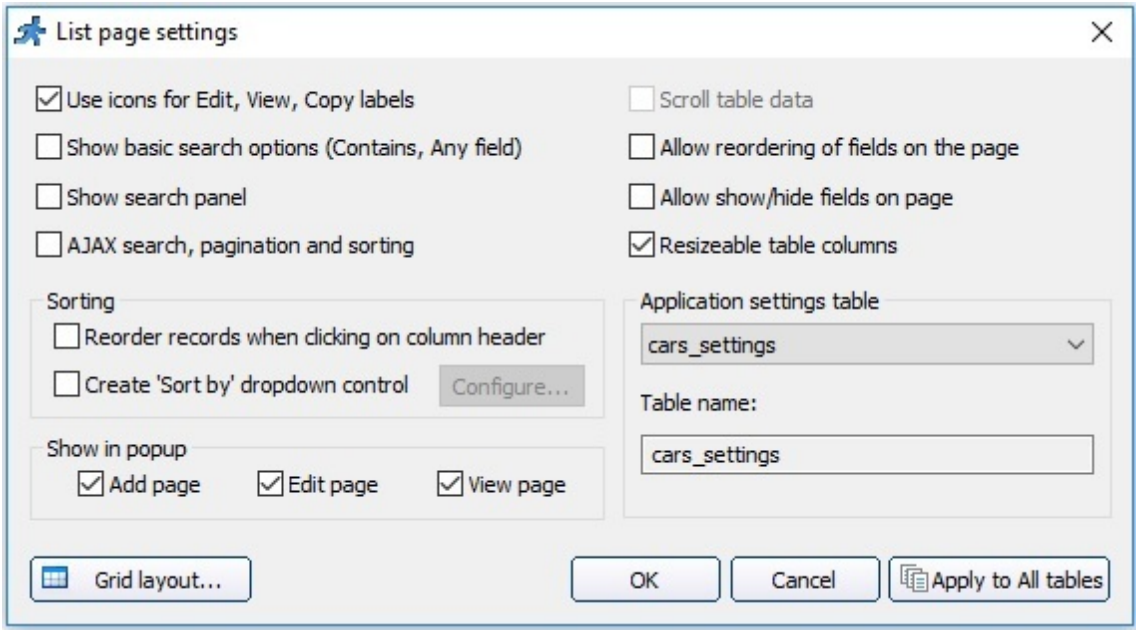
2.12.3 List page settings / Click actions

Quick jump

To access the **List page settings** and **Click actions**, proceed to the Choose pages screen and click the corresponding buttons next to the **List page** checkbox.

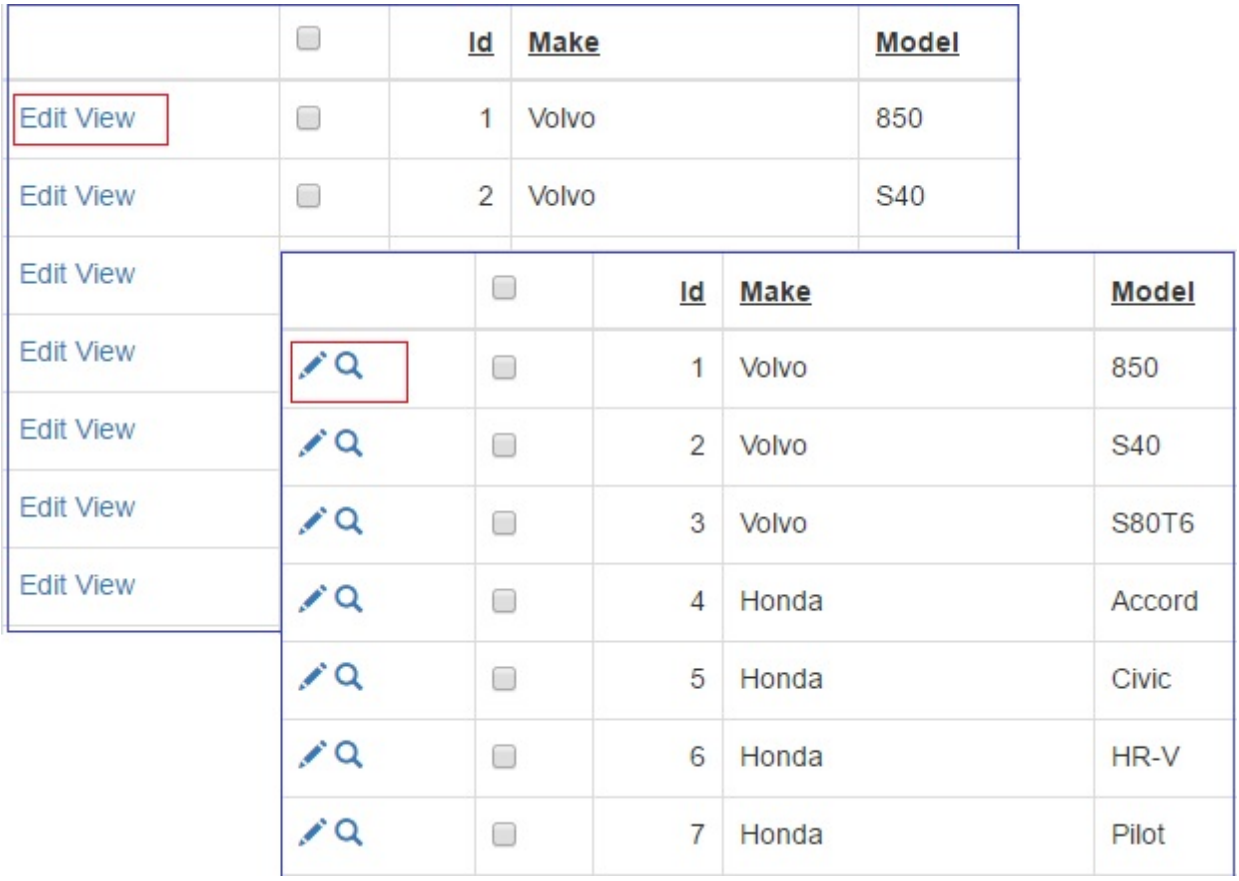


List page settings

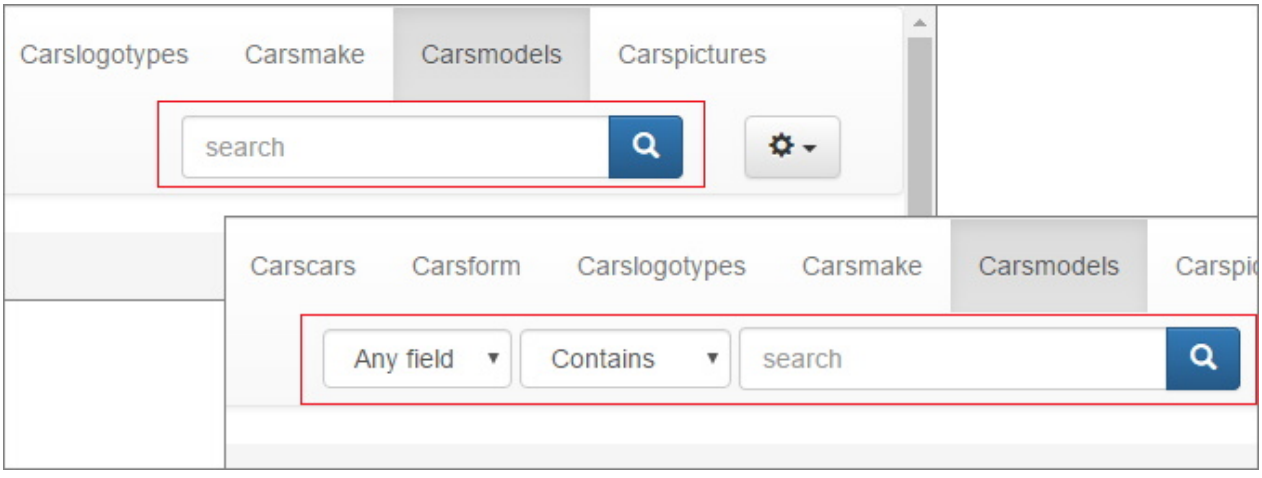


You can select the table to store the application settings like the order of columns, show/hide states, column sizes, and saved searches. These settings are user-specific.

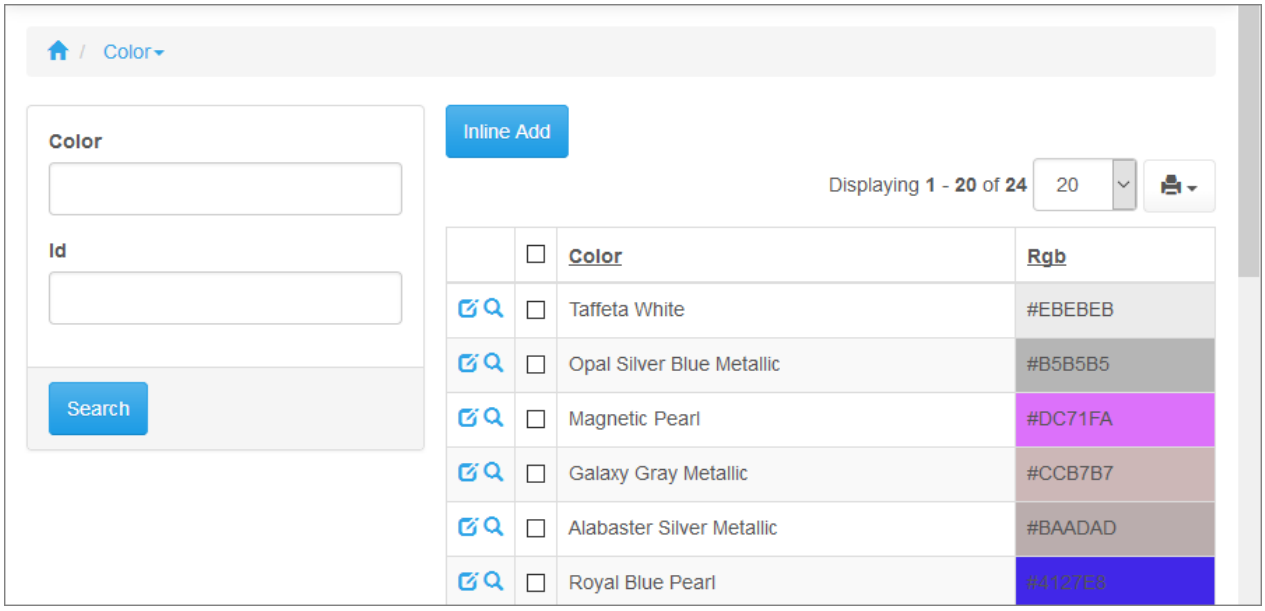
- **Use icons for Edit, View, Copy labels** - this option displays the icons for the **Edit**, **View**, **Copy** actions instead of the text labels.



- **Show basic search options** - this option displays the basic search options (**Any field**, **Contains**), instead of a single **Search field**.



- **Show search panel** - this option displays the search panel. The search panel allows you to search for specific values on the **List** page.

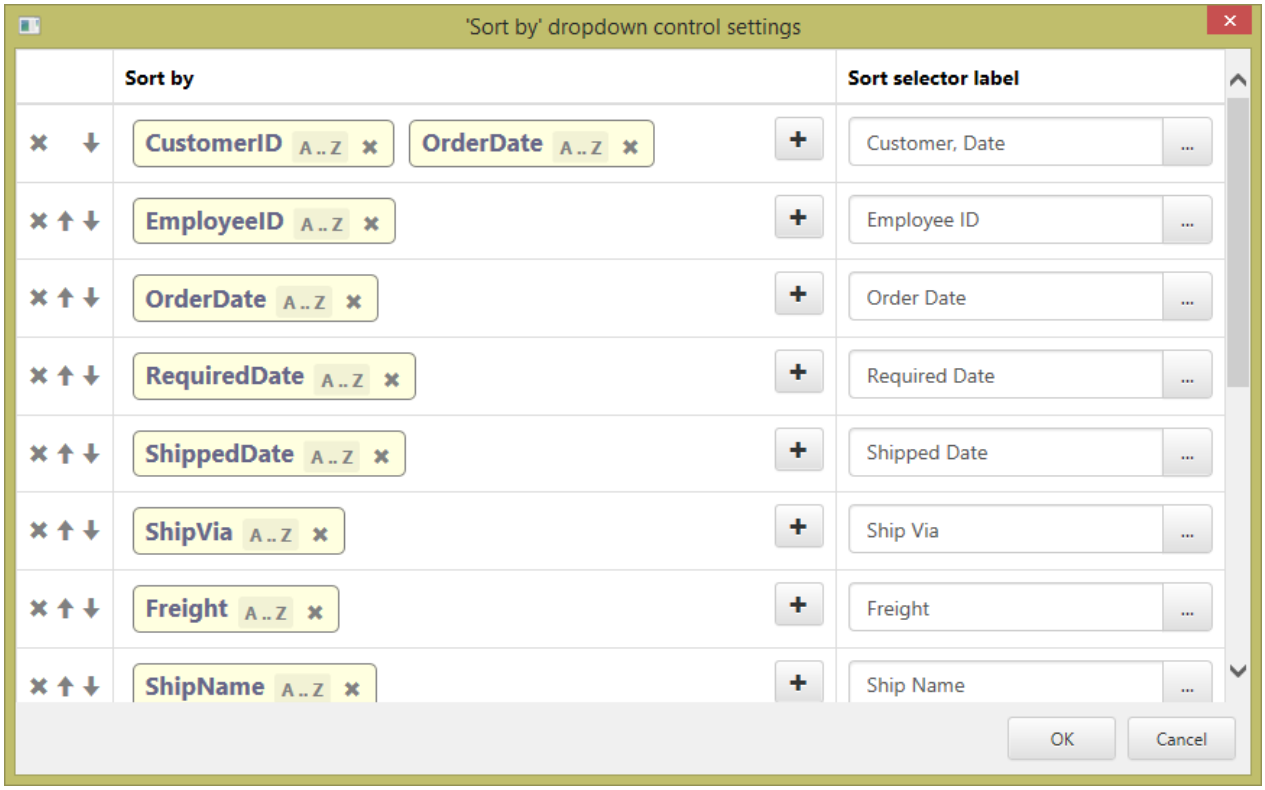


- **AJAX search, pagination and sorting** - this option enables the AJAX search, pagination and sorting so that the data is updated without needing to reload the entire page.

- **'Sort by' dropdown control**. This option is useful when:

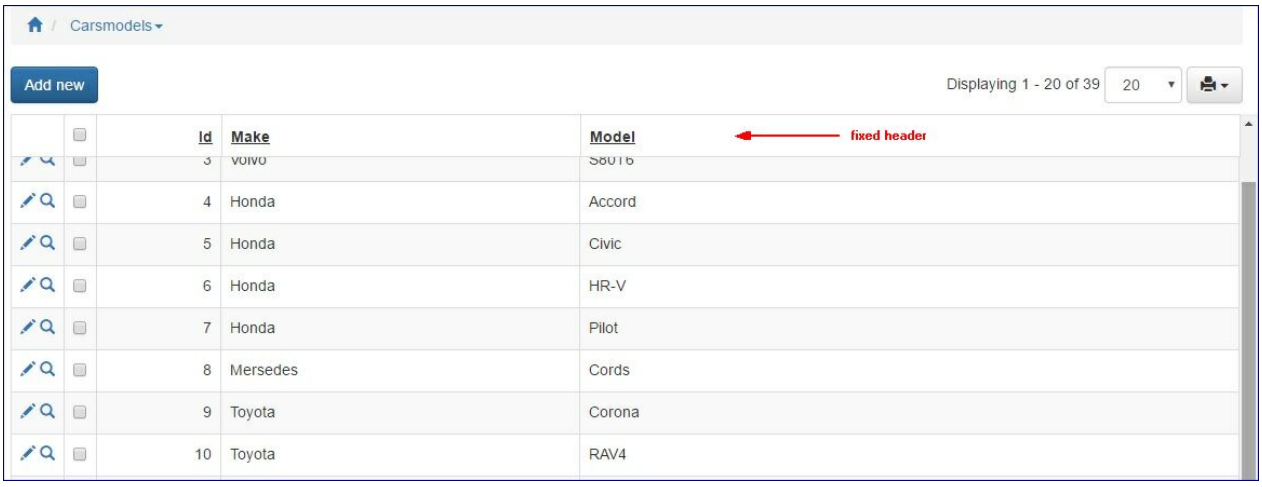
  - The **List** page is in a vertical/columns mode, and previously it didn't have any sorting options;
  - You need a **'Sort by'** control in the mobile mode;
  - You need to set up the application-specific sorting modes.

You can also choose whether users can sort the data by clicking on the column headers.

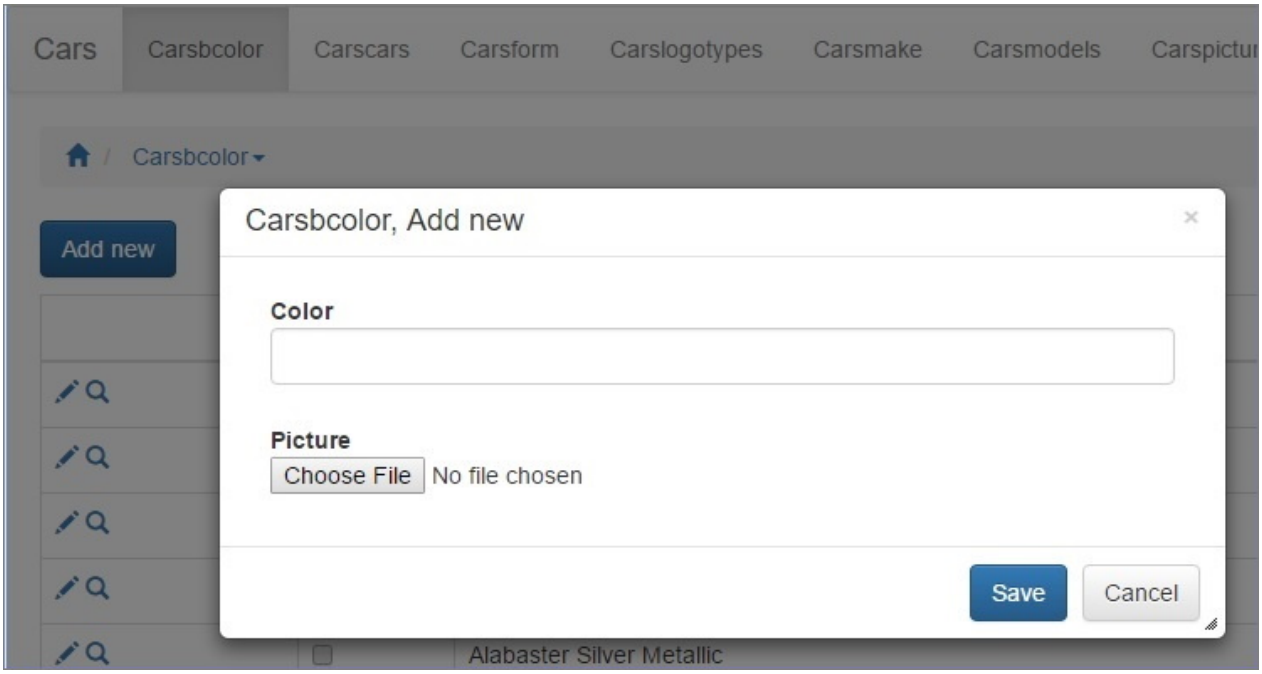


Check how it looks like in the generated application.

- **Scroll table data** - this option displays the data records as a scrollable table with a fixed table header.

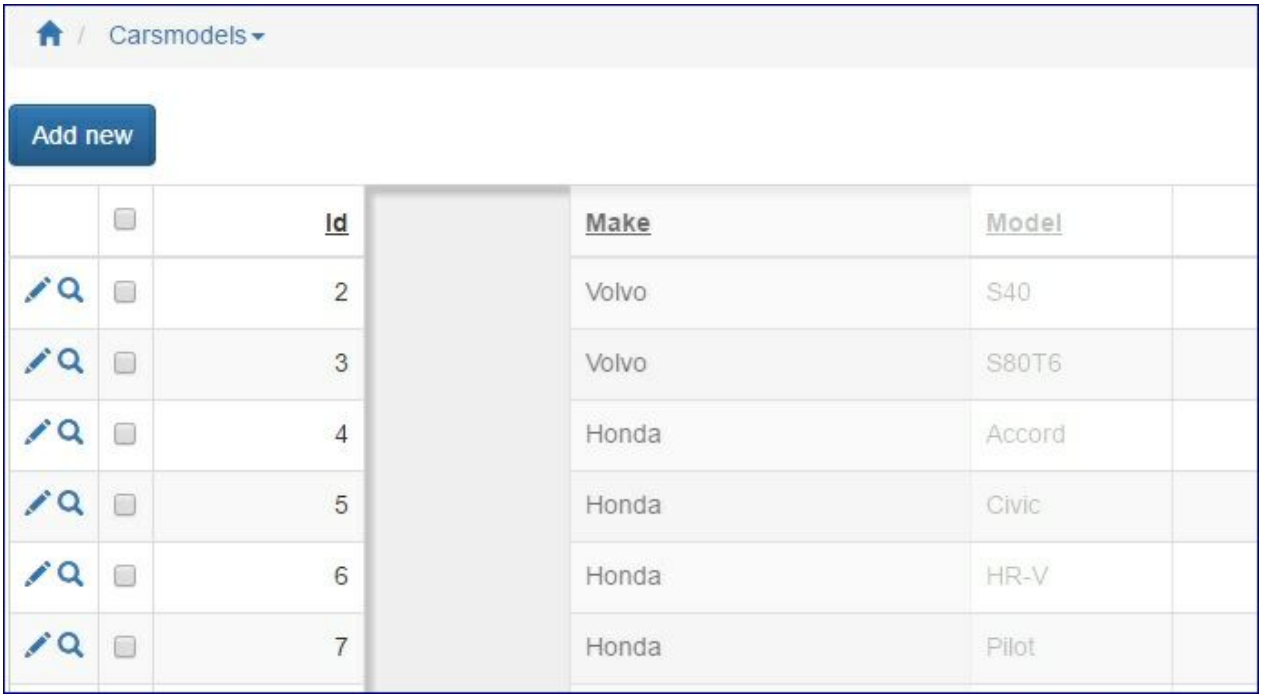


- **Show in popup** - this option allows showing the **Add**/**Edit**/**View** pages in a popup window.



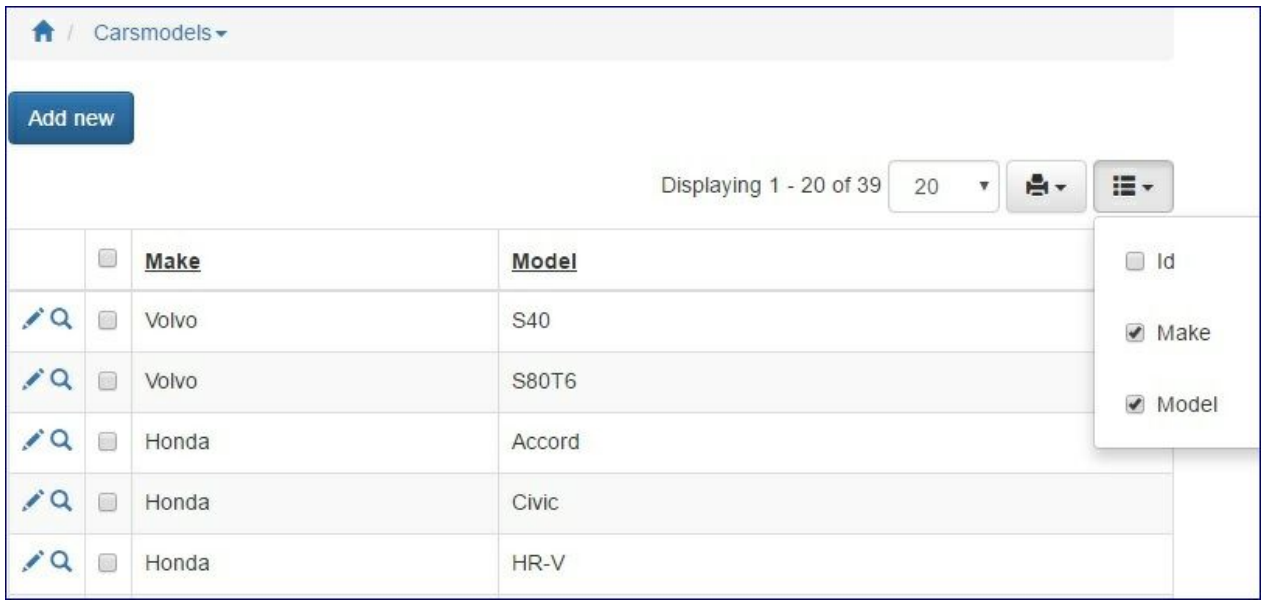
- **Allow reordering of fields on the page.**

Drag-n-drop the columns on the **List** page to reorder them. These settings are saved in the database and preserved between the sessions for each user.

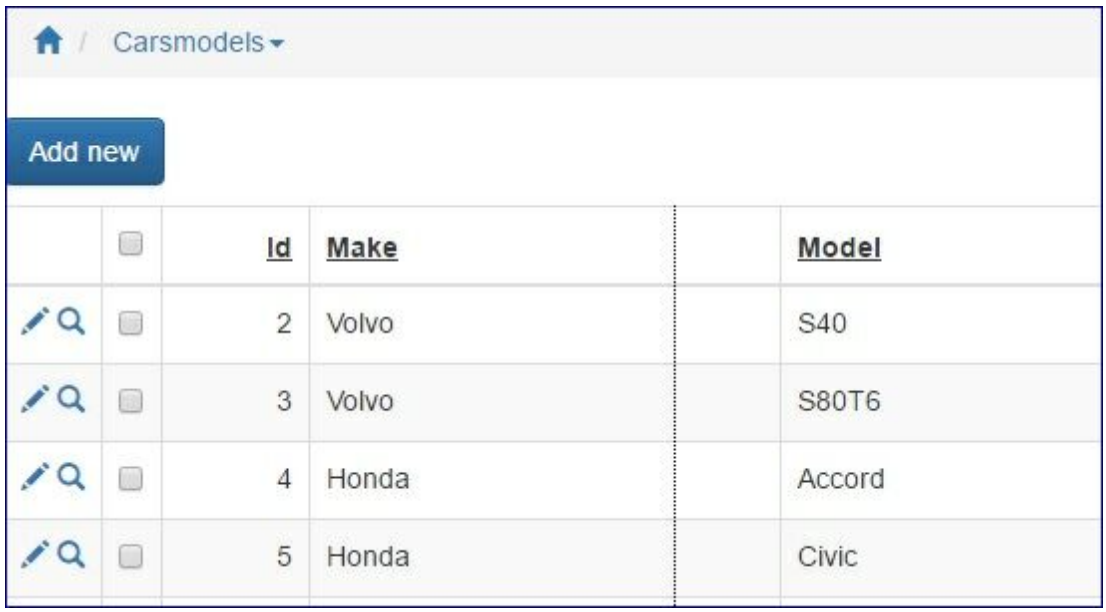


- **Allow show/hide fields on page.**

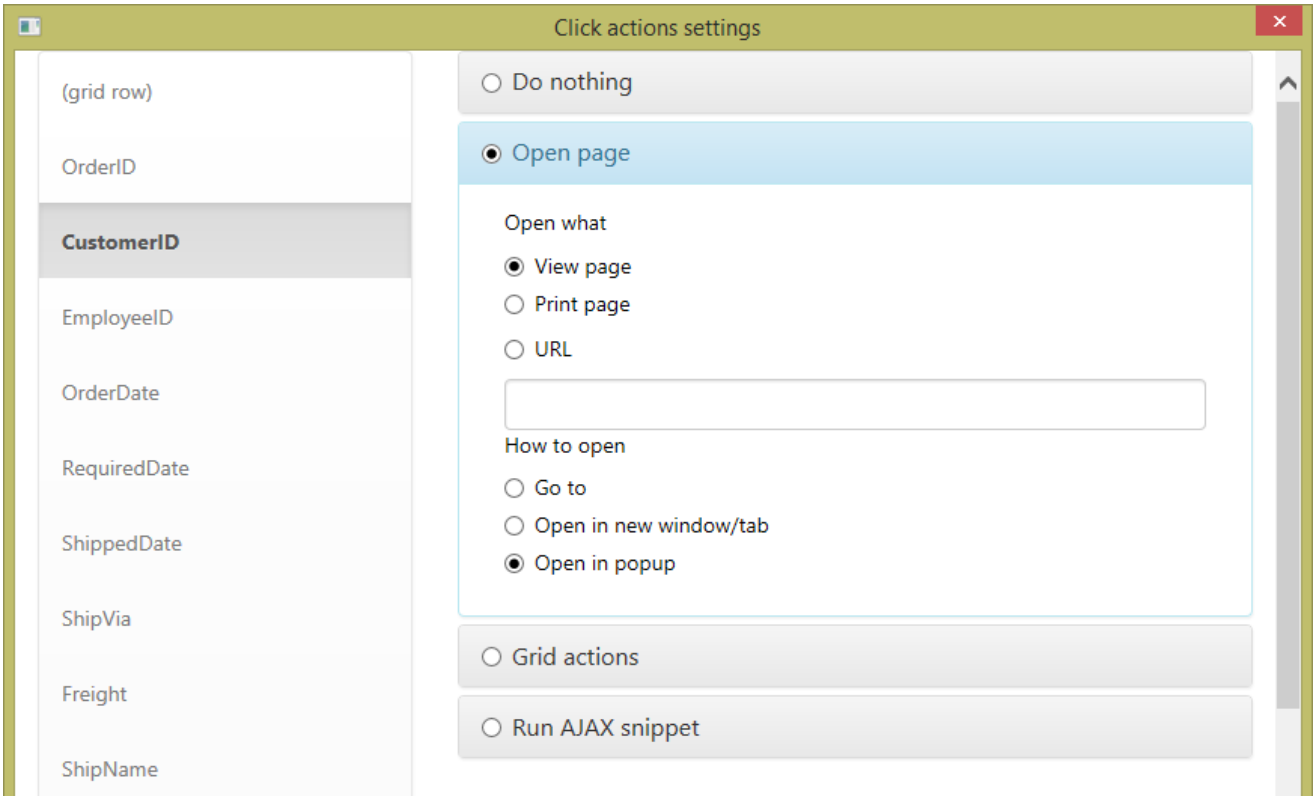
Choose the columns to show on each page in the generated app. These settings are also saved in the database and preserved between the sessions for each user.



- **Resizeable table columns** - this option allows resizing the table columns.



List page Click actions



With the **List** page **Click actions**, you can assign the actions like open a specific page, make the record selected, expand/collapse the details, or execute a custom code upon clicking the row/cell.

> **Note**: **Click actions** utilize the Tri-part event system, that consists of three parts: Client Before, Server, Client After.
>
> The **Client Before** part runs JavaScript code in the browser, then passes parameters to the **Server** part that runs PHP code, and then back to the browser to run the JavaScript code of the **Client After** part.
>
> To learn more about these types of events, see Tri-part events.

Check this live demo to try it in the app:

- Click the *CustomerID* cell to open the *Orders* view page in a popup.
- Click the *OrderID* field to retrieve the current order total and display it in the *OrderID* field.

The code used in the second example:

Client Before:

```
// pass OrderID to Server event
params["OrderID"] = row.getFieldValue("OrderID");
```

Server:

```
// run SQL Query to retrieve order total
$result["total"] = DBLookup("select sum(Quantity*UnitPrice) from `Order Details`
where OrderID=".$params["OrderID"]);
```

Client After:

```
// change cell background
row.fieldCell("OrderID").css('background', 'yellow' );
// add order total to OrderID cell content
row.fieldCell("OrderID").html(
   row.fieldCell("OrderID").html()+"<br>Total: "+result["total"]);
```

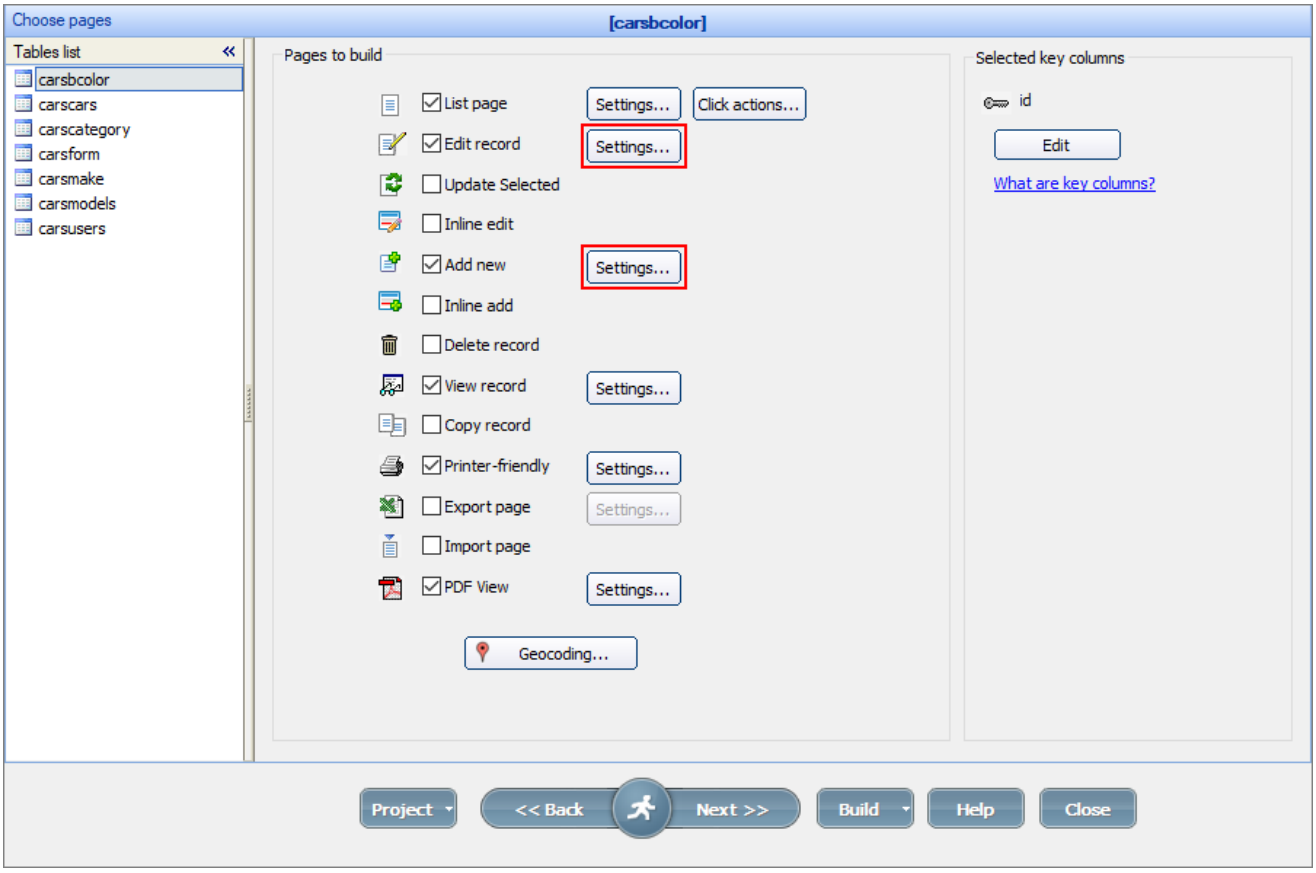
See also:

- row.getFieldValue
- row.fieldCell
- DBLookup
- Tri-part events
- GridRow JavaScript API
- Choose pages screen
- Key columns
- Add/Edit page settings
- CAPTCHA on Add/Edit pages
- Update selected
- Export/Import pages

- Printer-friendly/PDF view settings
- Geocoding

2.12.4 Add/Edit page settings

To access the **Add**/**Edit** page settings, proceed to the Choose pages screen, enable the corresponding checkboxes, and click the **Settings** buttons next to these checkboxes.



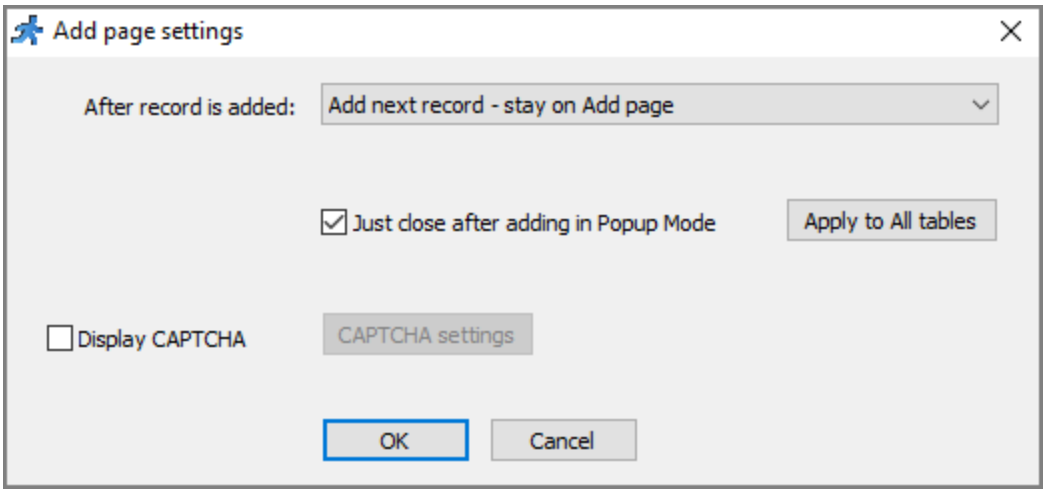
You can choose the action to be performed after the record is added or edited.

> **Note**: enable the Display CAPTCHA checkbox and configure its settings if you want to use CAPTCHA on the **Add** and **Edit** pages.

Add page settings

After record is added options:

- Return to the **List** page;
- Add next record - stay on **Add** page (the default action);
- Open the new record **View** page;
- Open the new record **Edit** page.

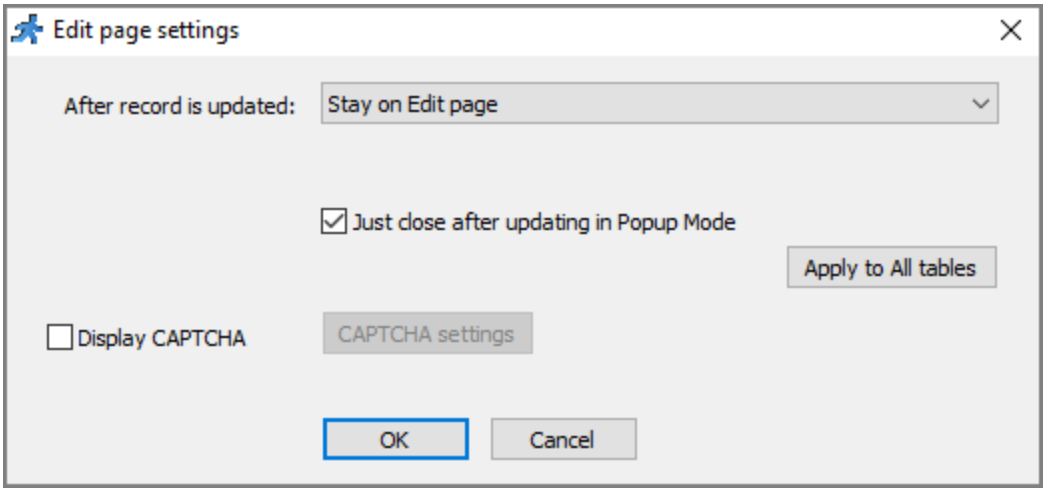


You can apply these settings to all tables using the **Apply to All tables** button.

Edit page settings

After record is updated options:

- Return to the **List** page;
- Stay on **Edit** page (the default action);
- Go to the **View** page;
- Edit next record;
- Edit previous record.



You can apply these settings to all tables with the **Apply to All tables** button.

See also:

- Choose pages screen
- Key columns
- List page settings / Click actions
- CAPTCHA on Add/Edit pages
- Update selected
- Export/Import pages
- Printer-friendly/PDF view settings
- Geocoding

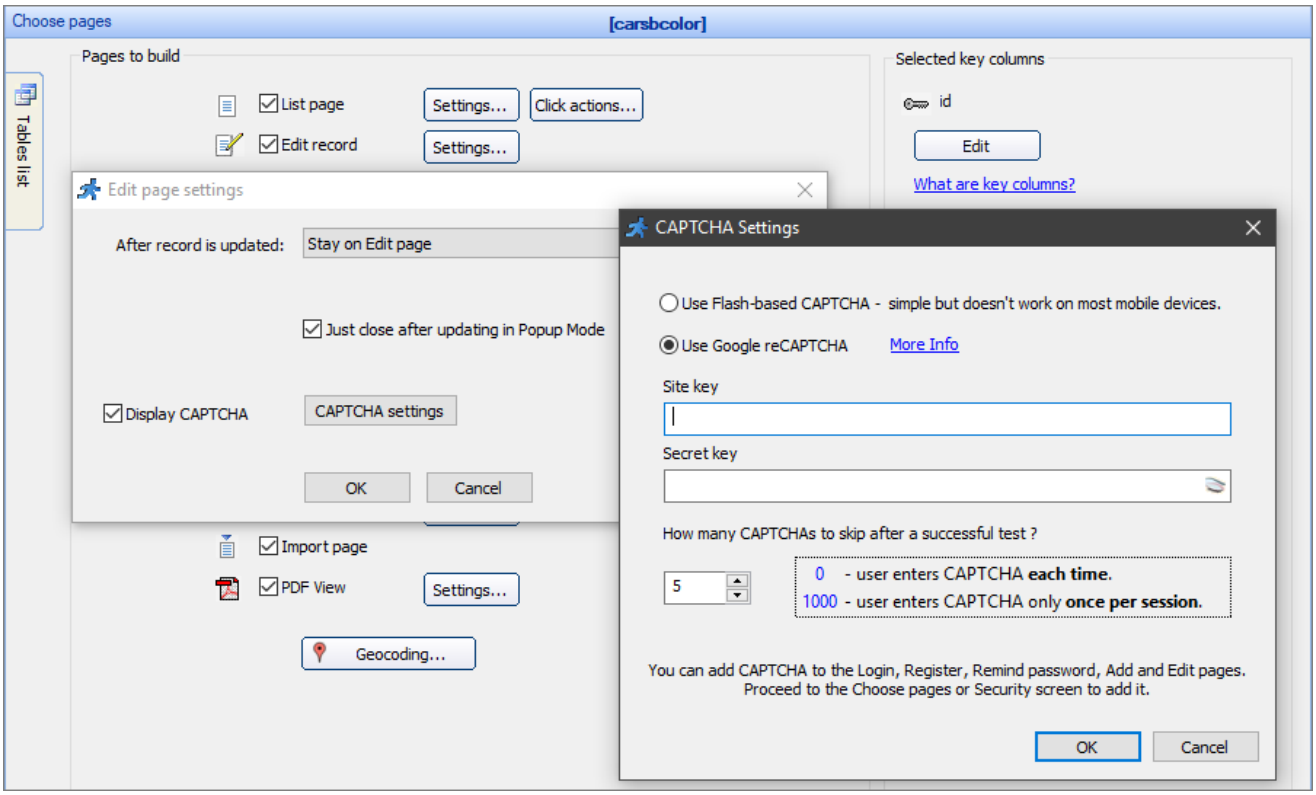
2.12.5 CAPTCHA on Add/Edit pages

CAPTCHA is a simple test to determine whether the user is a human or a bot. It is used to prevent spamming and other automated abusive behavior on websites.

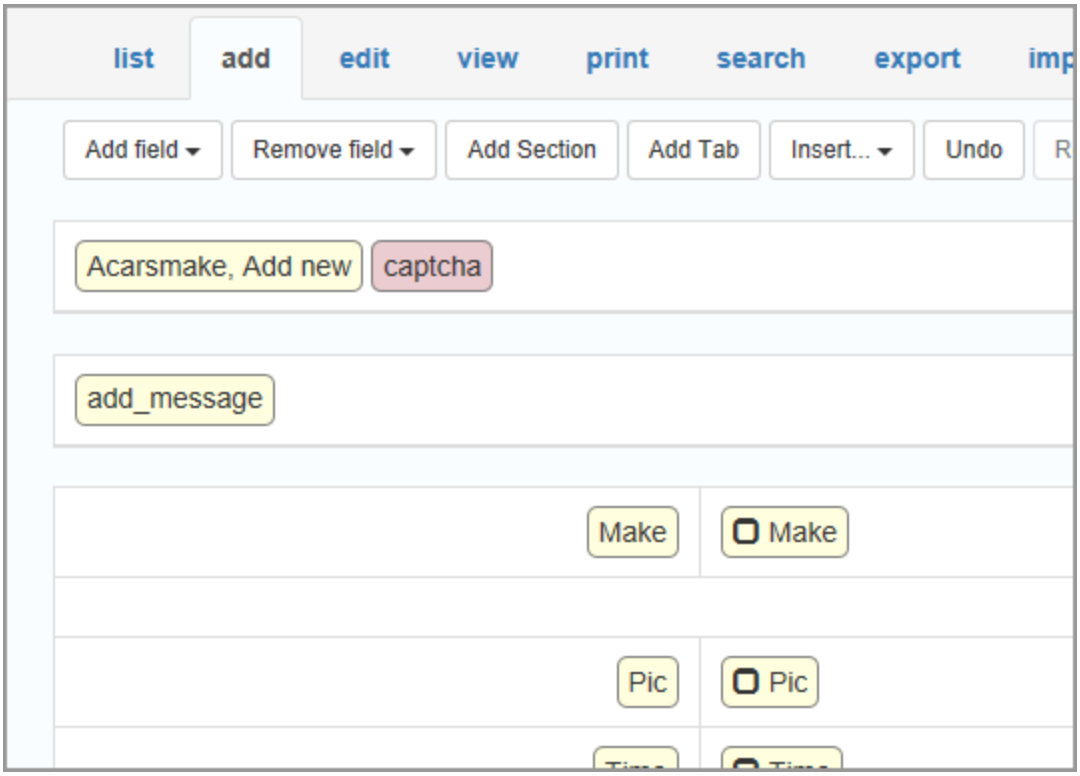
You can add CAPTCHA to the **Add** and **Edit** pages by selecting the **Display CAPTCHA checkbox** on the Add/Edit page settings that can be accessed on the Choose pages screen.

Click **CAPTCHA settings** and choose what type of CAPTCHA to use: Flash-based CAPTCHA (simple, but doesn't work on most mobile devices) or Google reCAPTCHA.

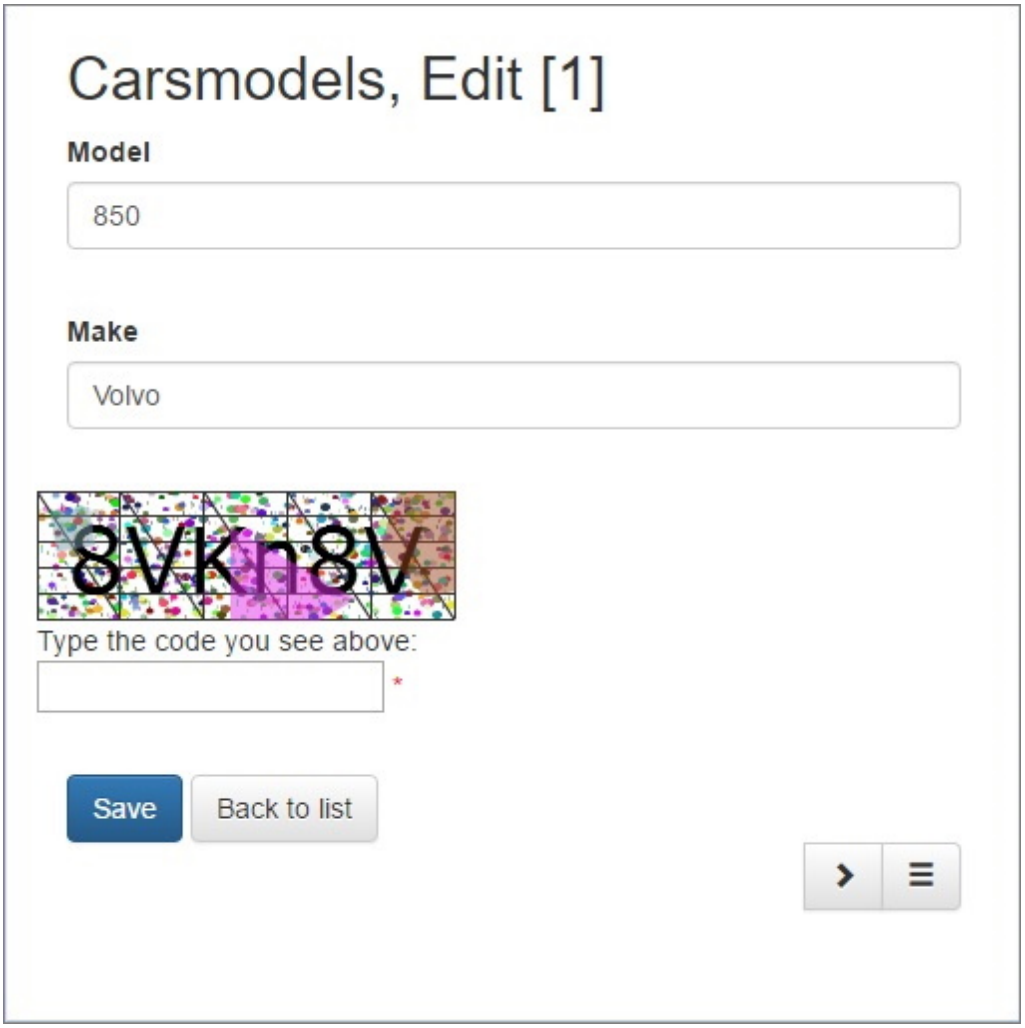
To use Google reCAPTCHA, register your web site at https://www.google.com/recaptcha/intro/index.html, copy the site key and secret key, and paste them into the respective fields in the CAPTCHA settings popup.



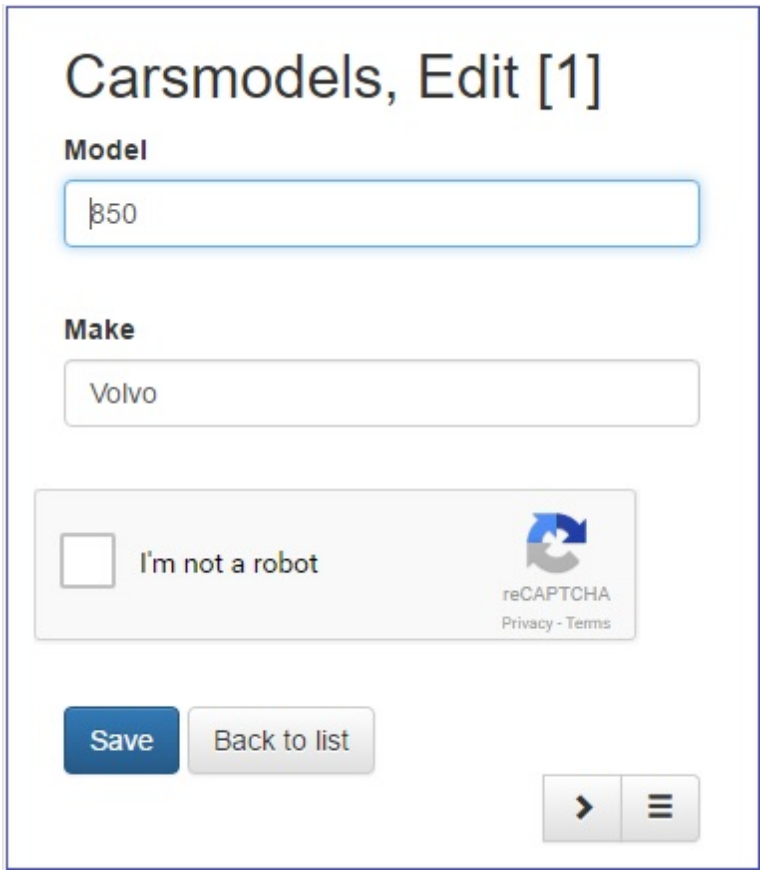
When you open a page with the CAPTCHA in the Page Designer, you can see a page element labeled *captcha*. Drag this element to change its location.



Here is an example of a Flash-based CAPTCHA:



Here is an example of a Google reCAPTCHA:

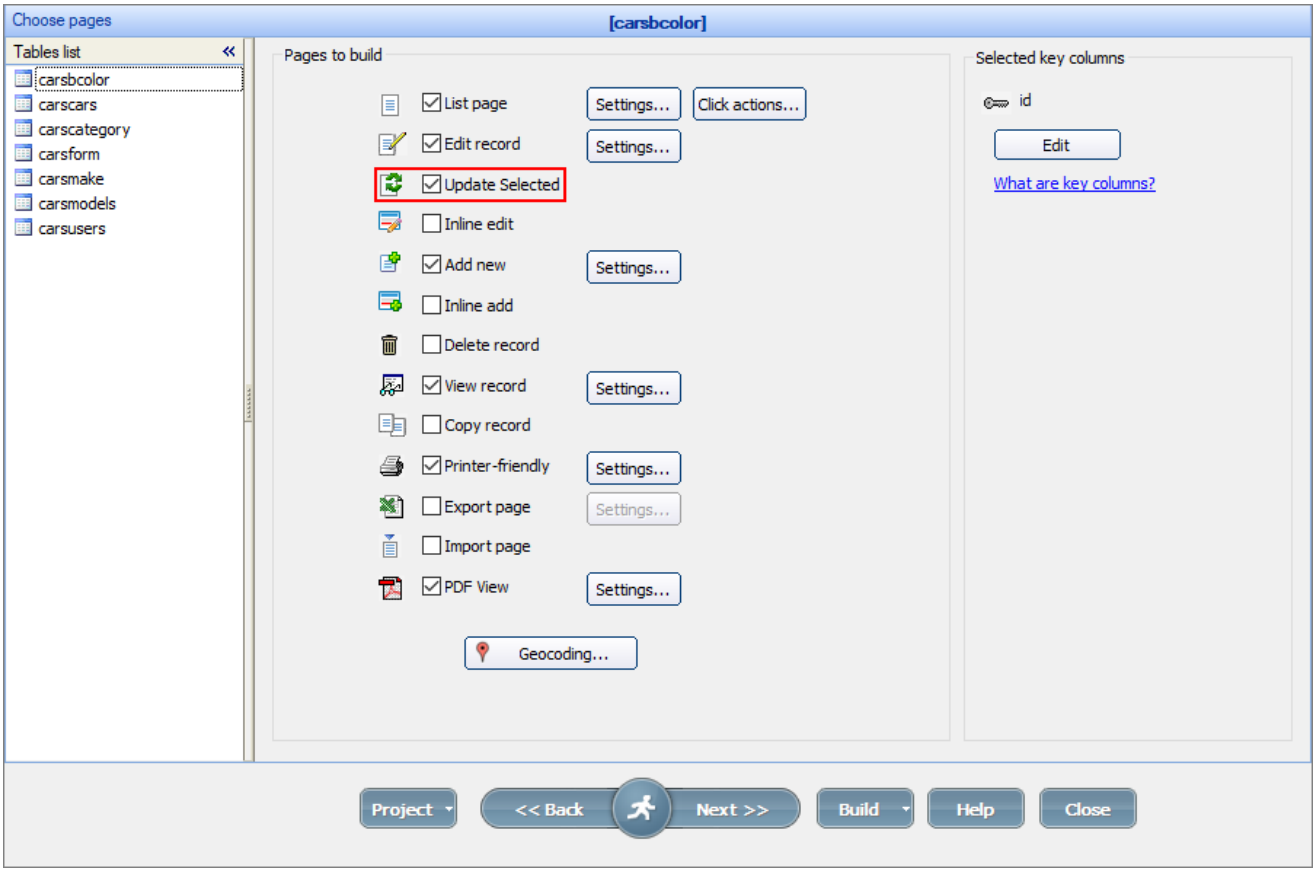


See also:

- Choose pages screen
- Key columns
- List page settings / Click actions
- Add/Edit page settings
- Update selected
- Export/Import pages
- Printer-friendly/PDF view settings
- Geocoding

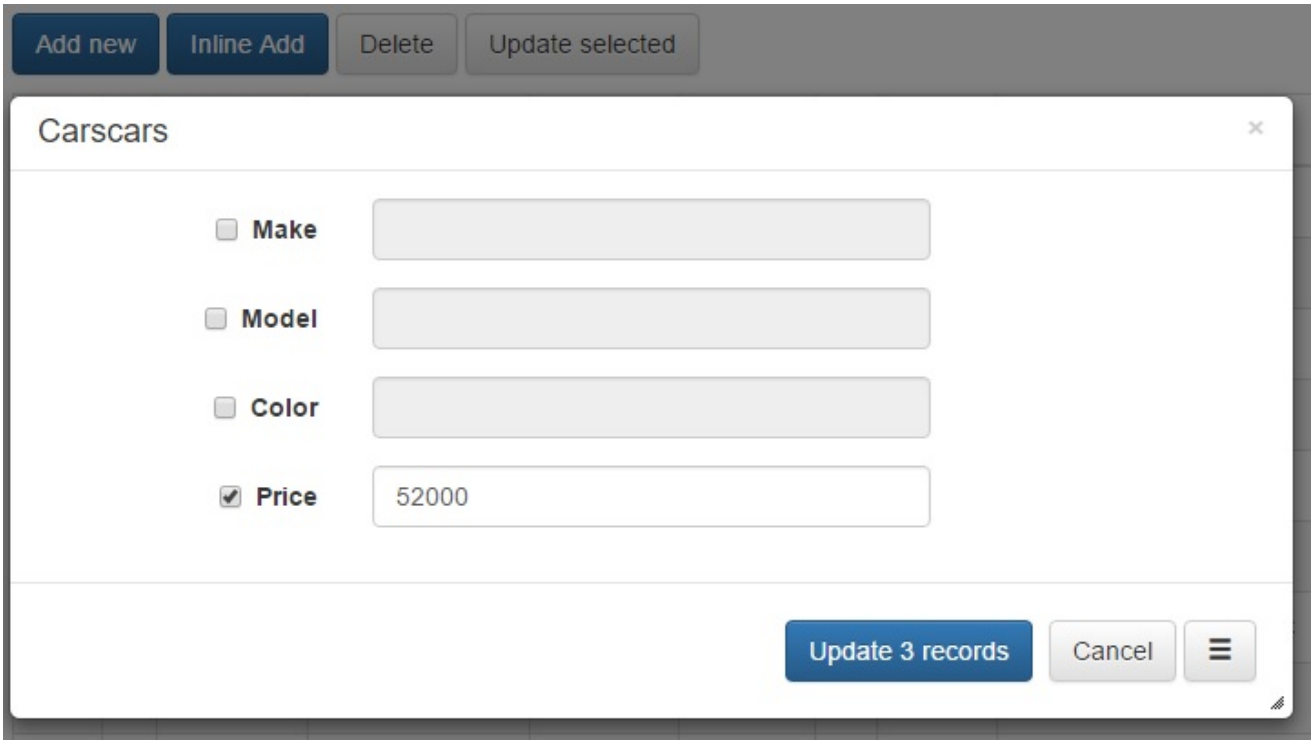
2.12.6 Update selected

Enable the **Update selected** option on the Choose pages screen to add the **Update selected** button to the **List** page.



Select several records and click this button to edit the selected records in the generated app. Choose the fields to appear with the **Update selected** dialog.

Depending on the **Edit** page settings, the **Update selected** page can be shown either in a popup or as a separate page.



See also:

- Choose pages screen
- Key columns
- List page settings / Click actions
- Add/Edit page settings
- CAPTCHA on Add/Edit pages
- Export/Import pages
- Printer-friendly/PDF view settings
- Geocoding

2.12.7 Export/Import pages

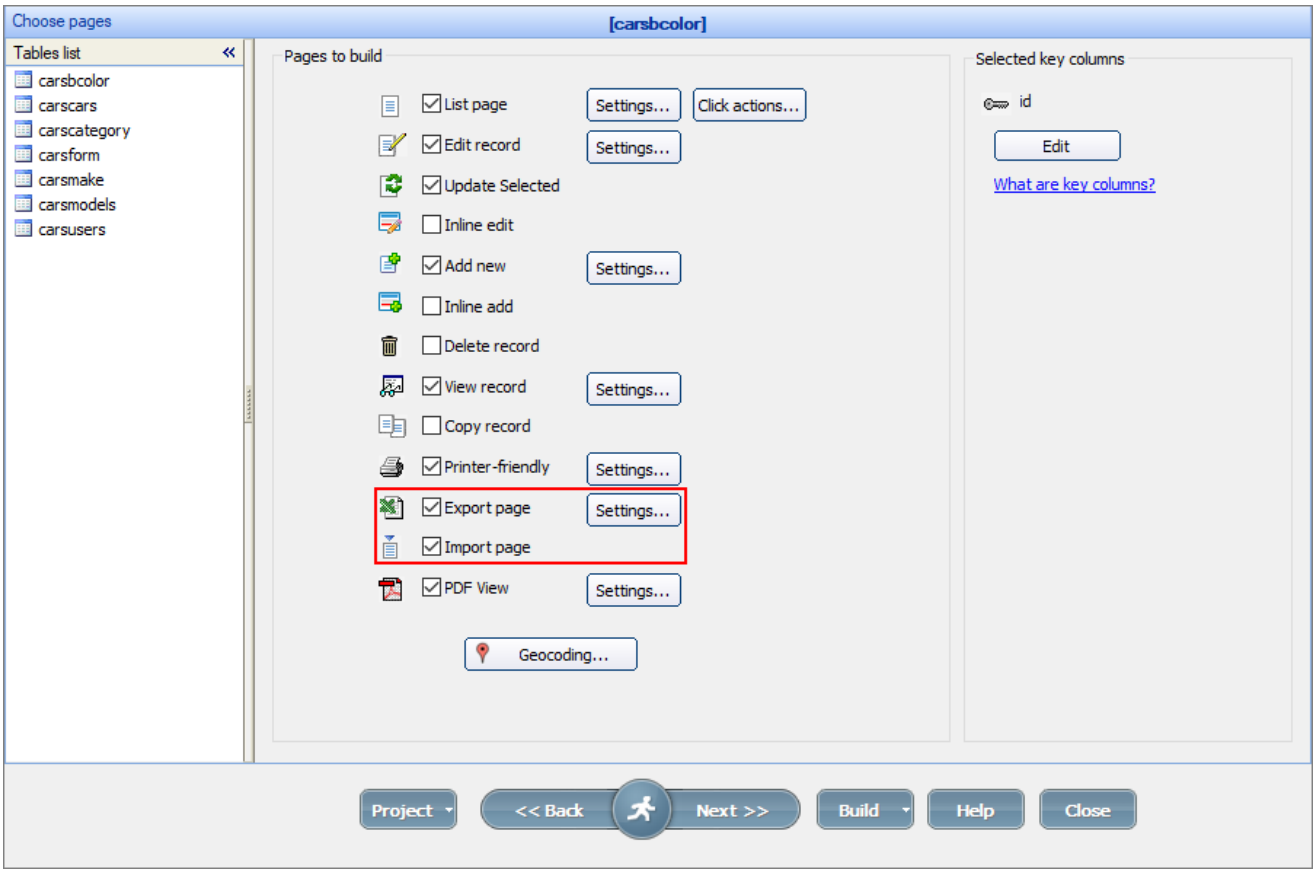
Quick jump

Export page

Import page

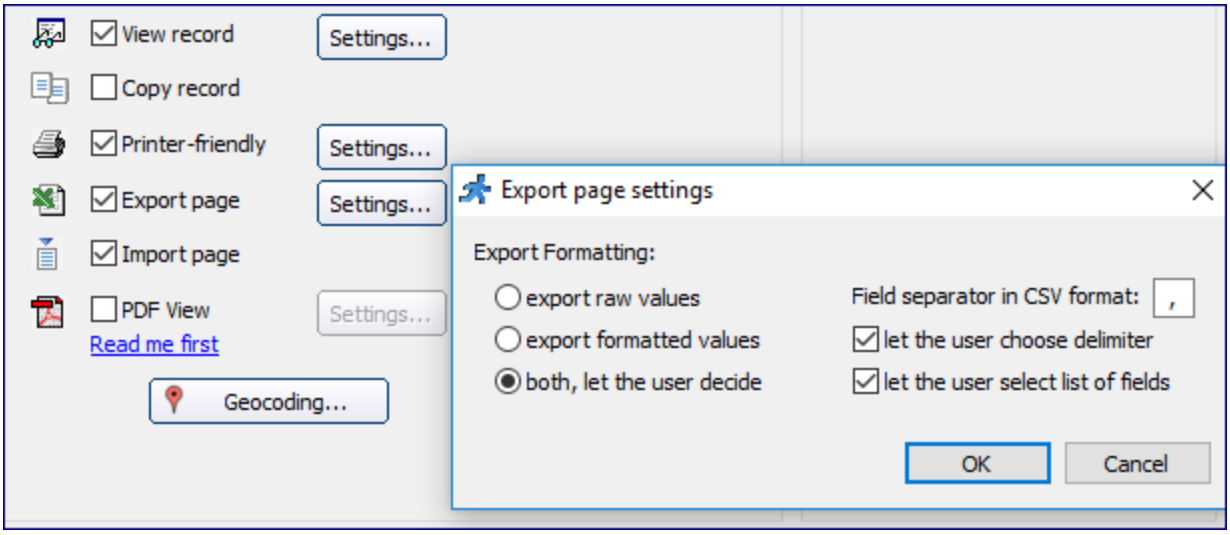
Copy and paste text

To access the **Export page settings**, proceed to the Choose pages screen and click the **Settings** button next to the **Export page** checkbox. To enable the **Import page**, click the **Import page** checkbox on the same screen.

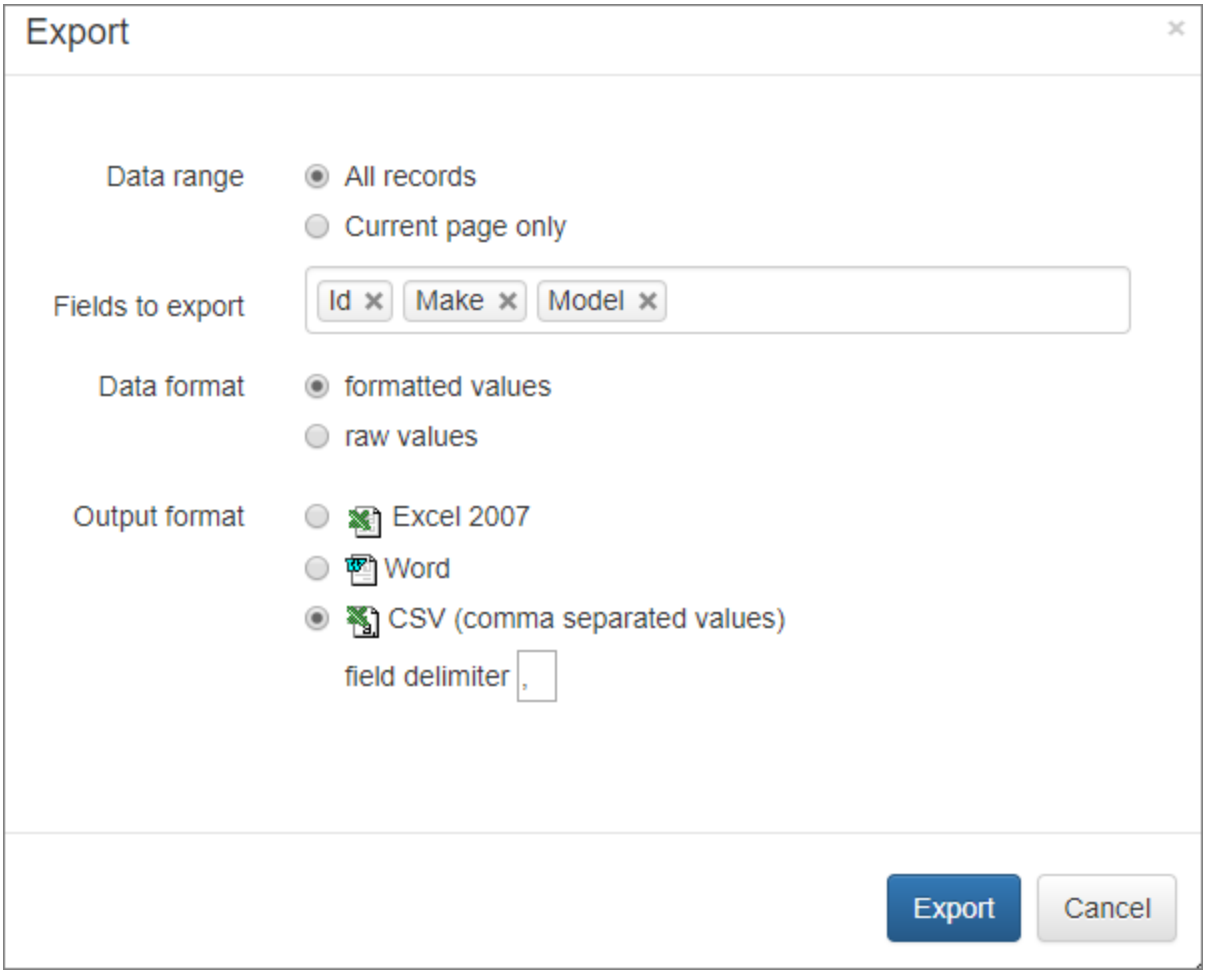


Export page

These are the **Export page settings**:



The **Export** page is opened in a popup by default. This is how it looks like in the generated application:

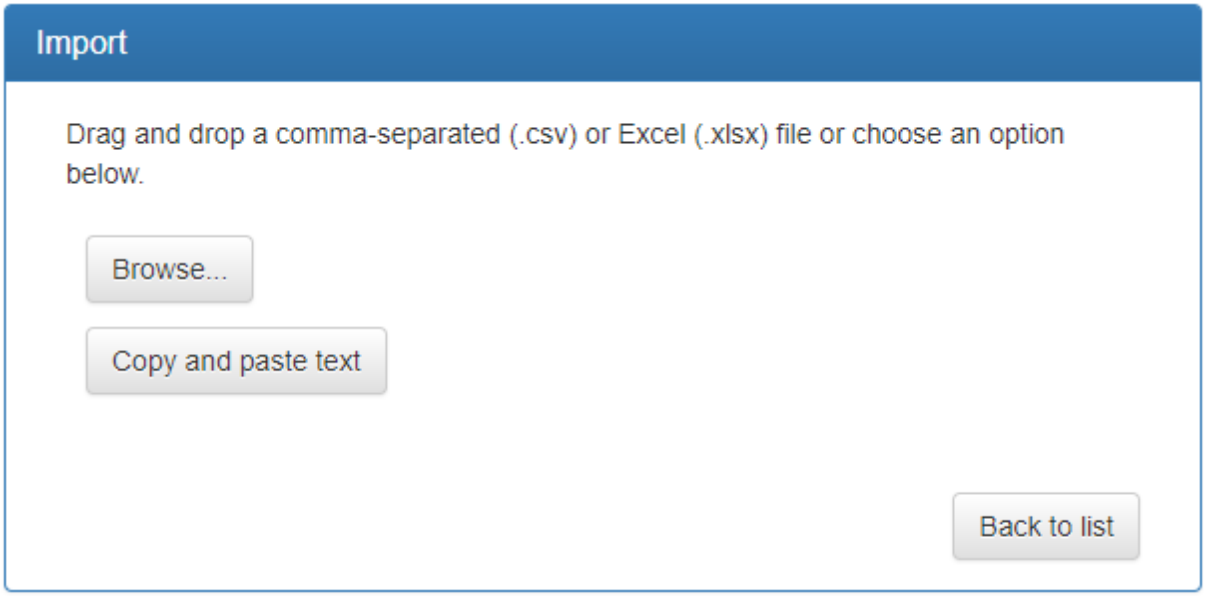


The supported file formats are:

- Excel 2007 (.xlsx);
- Word (.doc);
- CSV (comma separated values).

The field labels are used as the headers for the Excel/Word files. The field names are used as the headers for the CSV files so that the exported file can be imported into other software.

Import page



Supported file formats are:

- CSV;
- Excel 2007 (.xlsx);
- Excel 97-2003 (.xls).

The field names or labels can be used as the headers for the Excel files. For the CSV files, the field names should be used as the headers.

PHPRunner adds new records or updates the existing ones when importing. During an import, PHPRunner tries to insert new records first. If the insert fails for any reason (for example, when there is a duplicate primary key), it then tries to locate and update the existing records.

PHPRunner updates the existing records instead of adding new ones when:

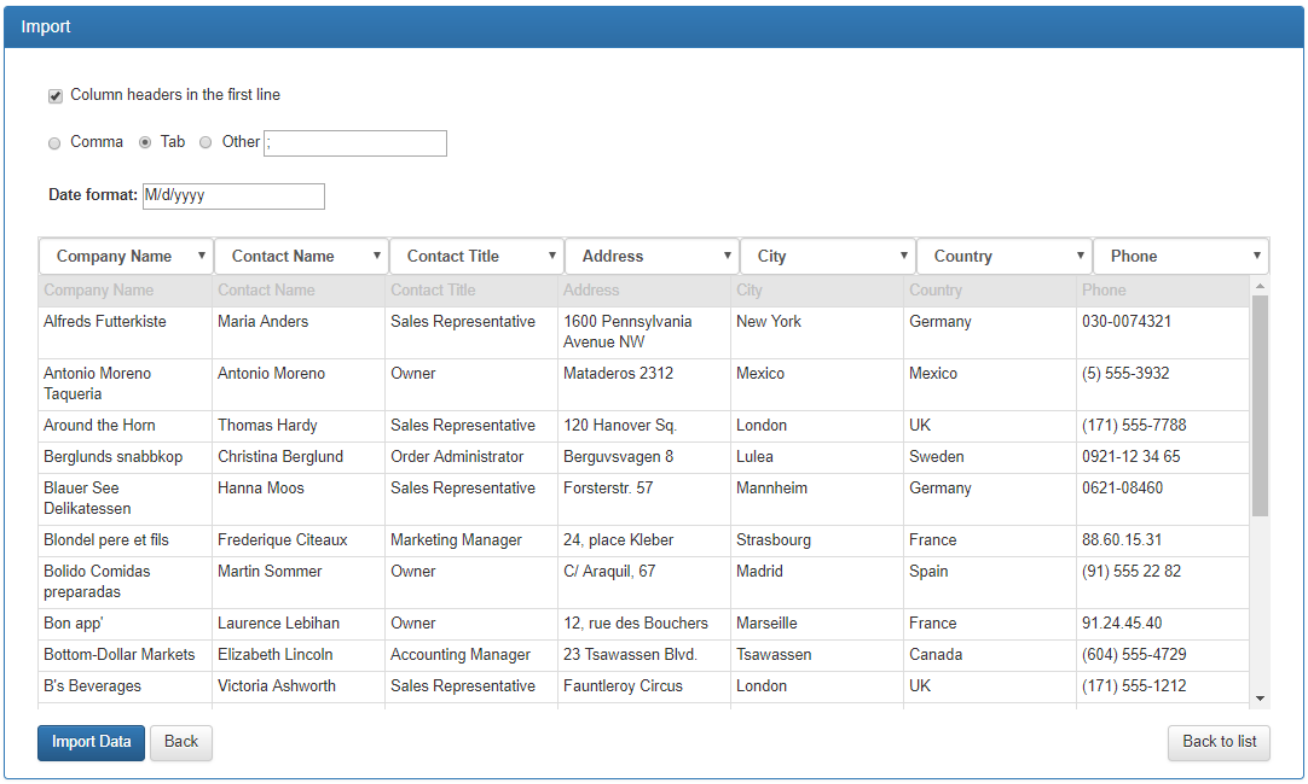
1. A key column is defined for the table in question.

2. Key columns selected on the Choose pages screen match the primary key in the database.

3. The key column/columns exist in the imported file.

When importing the data, set the Date format mask, so that PHPRunner recognizes the date correctly. Here are a few examples of supported date formats:

- dd.mm.yyyy
- mm/dd/yyyy
- yyyy-mm-dd
- dd/mm/yyyy



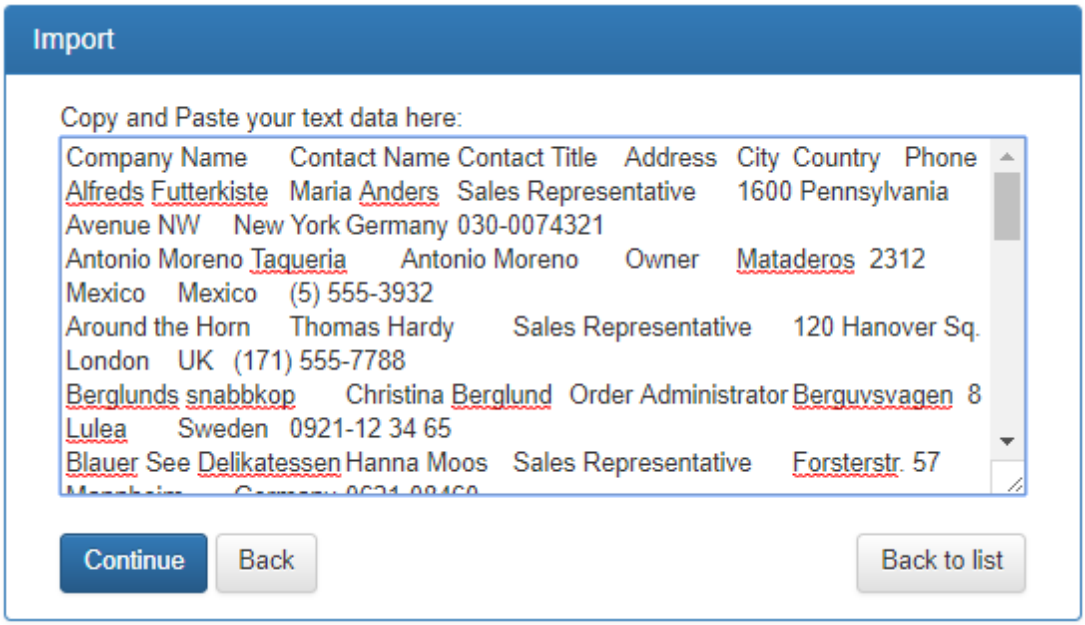
If you wish to combine the import with any extra actions, use the BeforeInsert event. For example, you may want to specify the file creation date or fill in the *OwnerID* field.

> **Note**: PHPRunner creates temporary files to preview the import results in the *templates_c* folder under the output directory. You need to set the writing permissions for this folder in the web server.

Copy and paste text

You can copy and paste import data instead of uploading the whole file.

Copy a few lines of data, then paste it into the import page after clicking the **Copy and paste text** button.



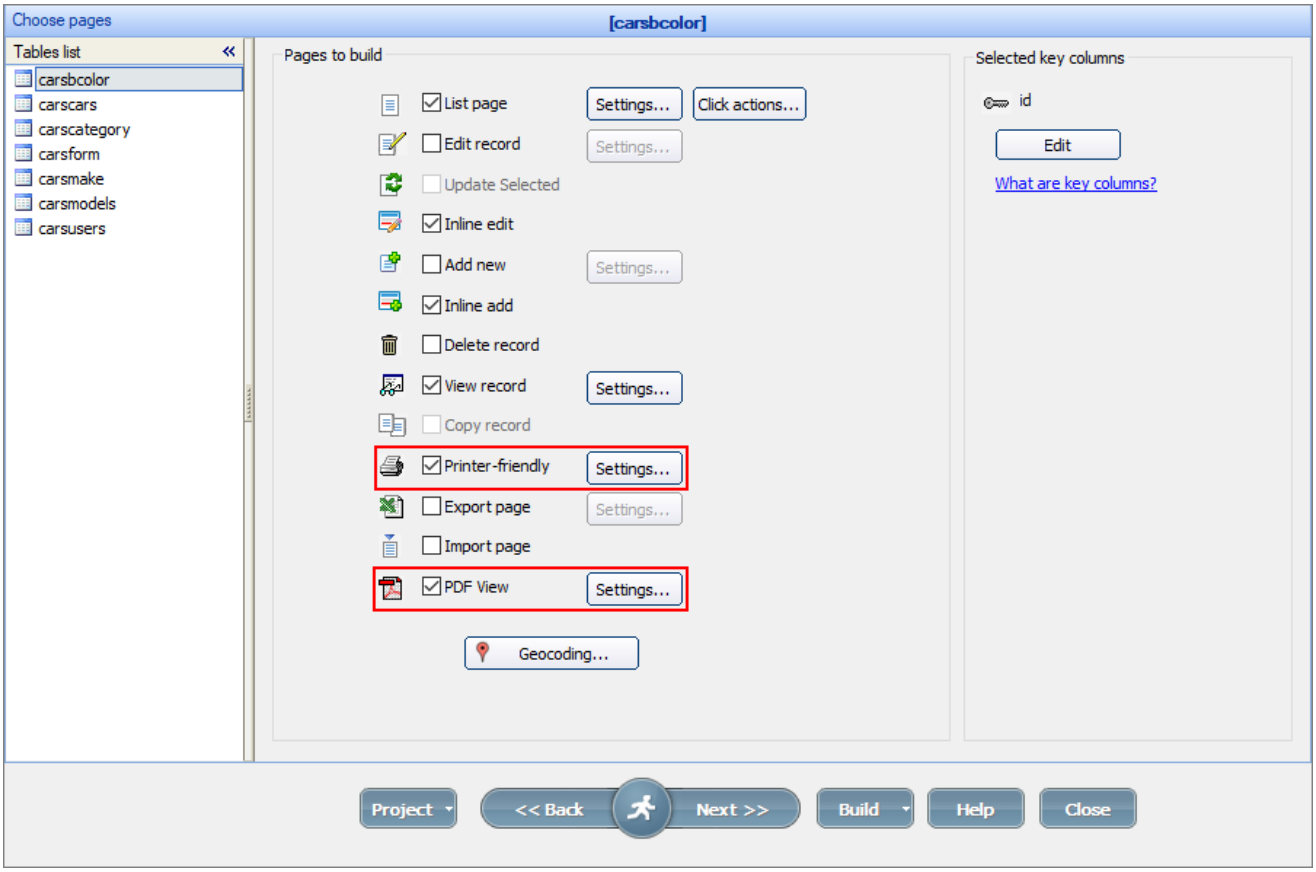
> **Note**: you do not need to include the column headers.

See also:

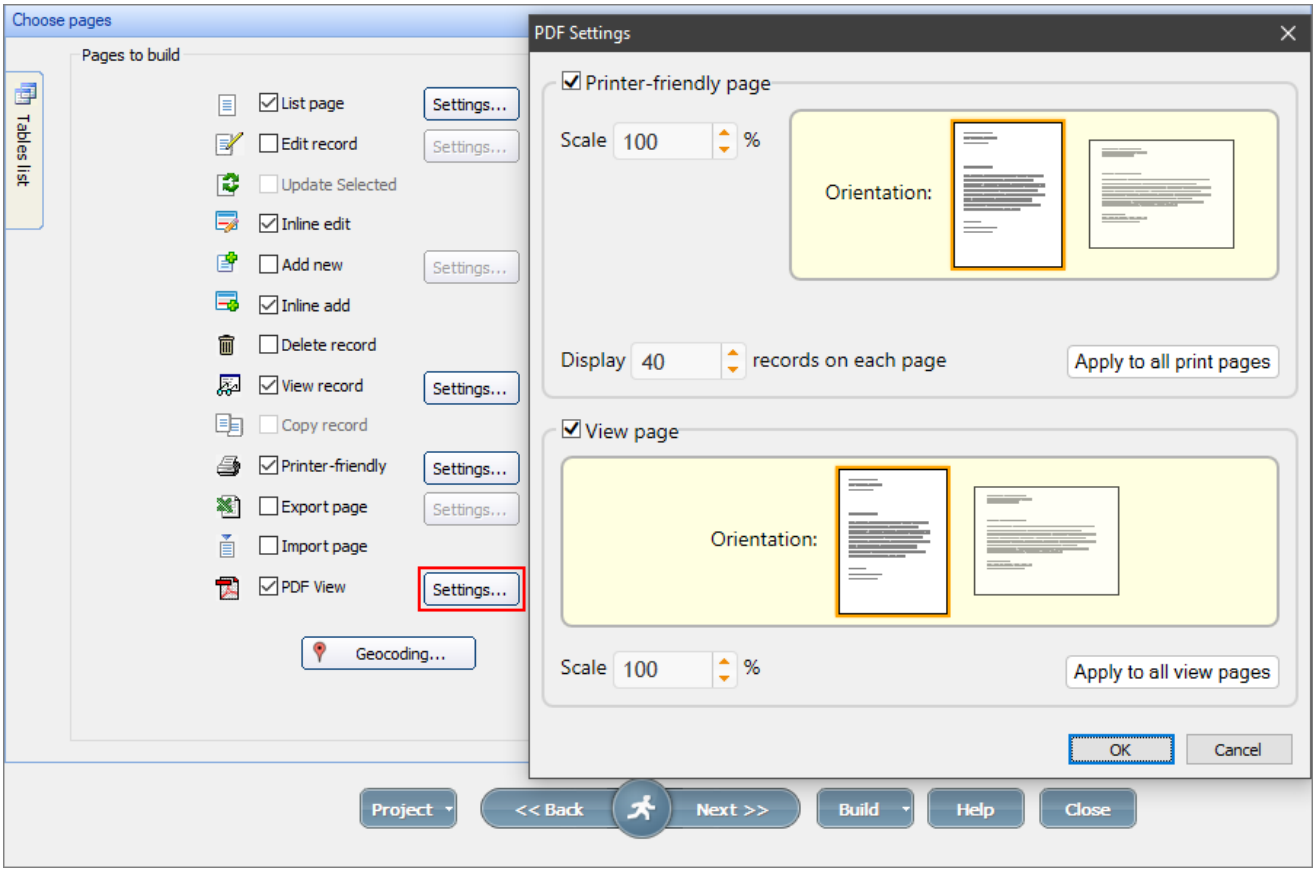
- Choose pages screen
- Key columns
- List page settings / Click actions
- Add/Edit page settings
- CAPTCHA on Add/Edit pages
- Update selected
- Printer-friendly/PDF view settings
- Geocoding

2.12.8 Printer-friendly/PDF view settings

Select the **Printer-friendly** and **PDF View** checkboxes on the Choose pages screen to make the page printer and PDF-friendly. The **PDF View** option allows downloading the page as a PDF document.



Click the **Settings** button to configure these options:



- **Scale**. You can choose to scale the pages to a set percentage.
- **Orientation**. Select the portrait or landscape orientation for the pages.
- The **Display N records per page** option determines where to insert the page break when you print the contents of the table.

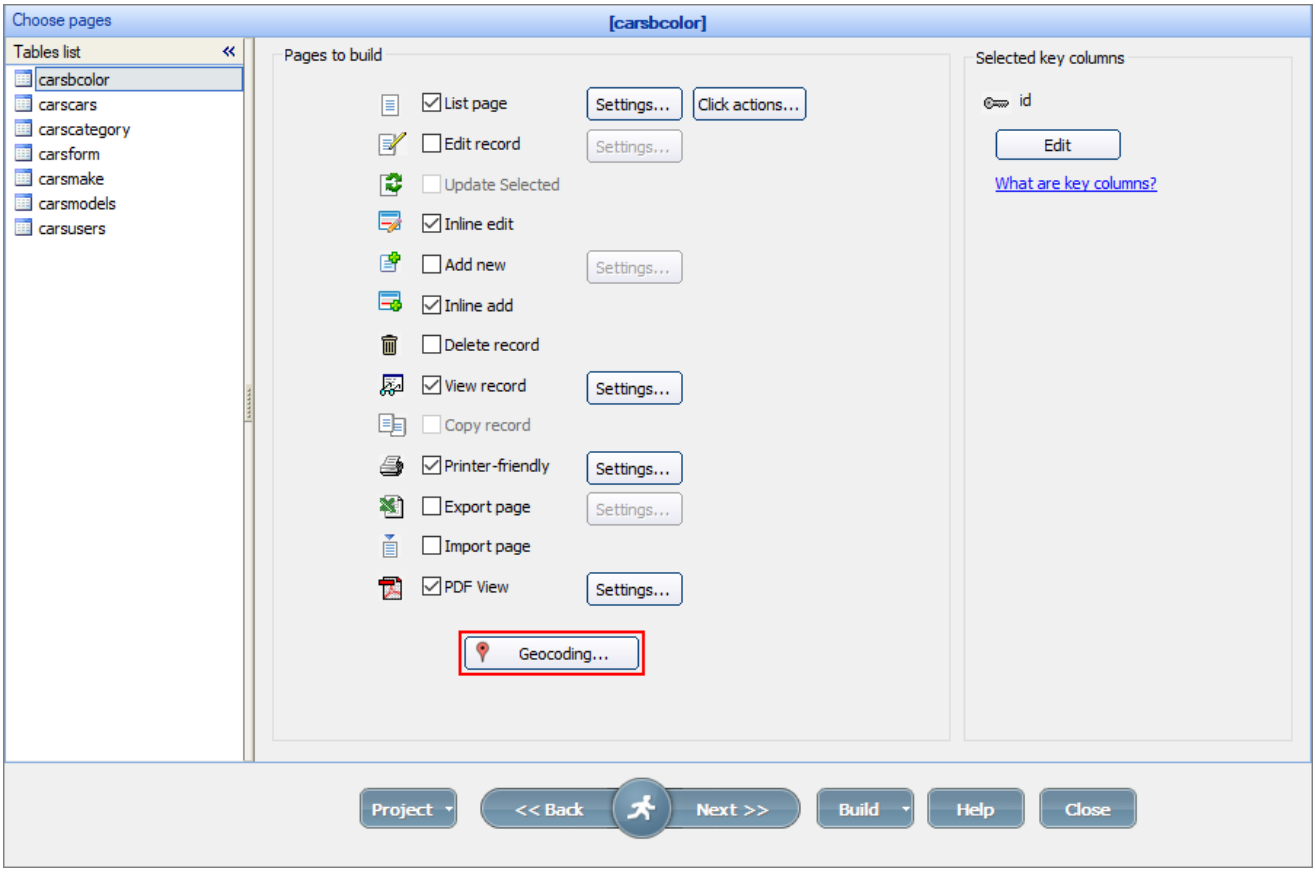
See also:

- PDF View settings
- Choose pages screen
- Key columns
- List page settings / Click actions
- Add/Edit page settings
- CAPTCHA on Add/Edit pages
- Update selected

- Export/Import pages
- Geocoding

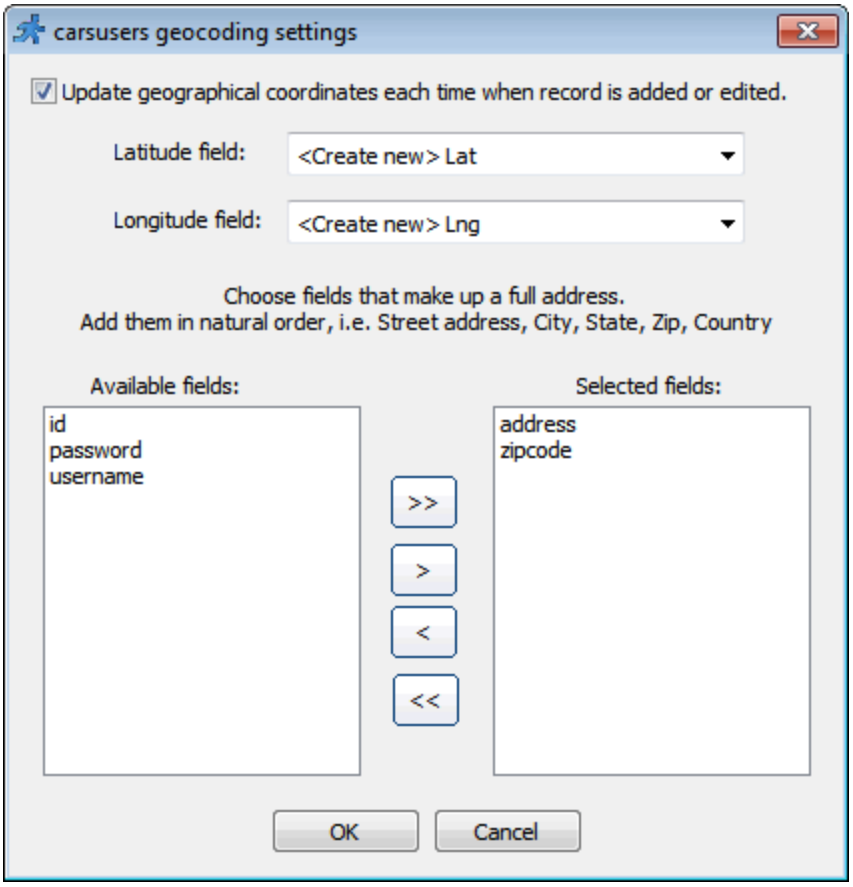
2.12.9 Geocoding

Geocoding is used for the tables where the location data is displayed as maps. To configure the **Geocoding** settings, click the corresponding button on the Choose pages screen.



It is recommended to use the *Latitude/Longitude* fields to show coordinates on the map. The **Geocoding** function takes the address fields and automatically converts them into the *Latitude/Longitude* fields each time the records are added or updated.

Select the existing fields for storing latitude/longitude data or create new ones. Then select the address field(s) to be used during geocoding.



For more information about displaying data as maps, see Insert map.

See also:

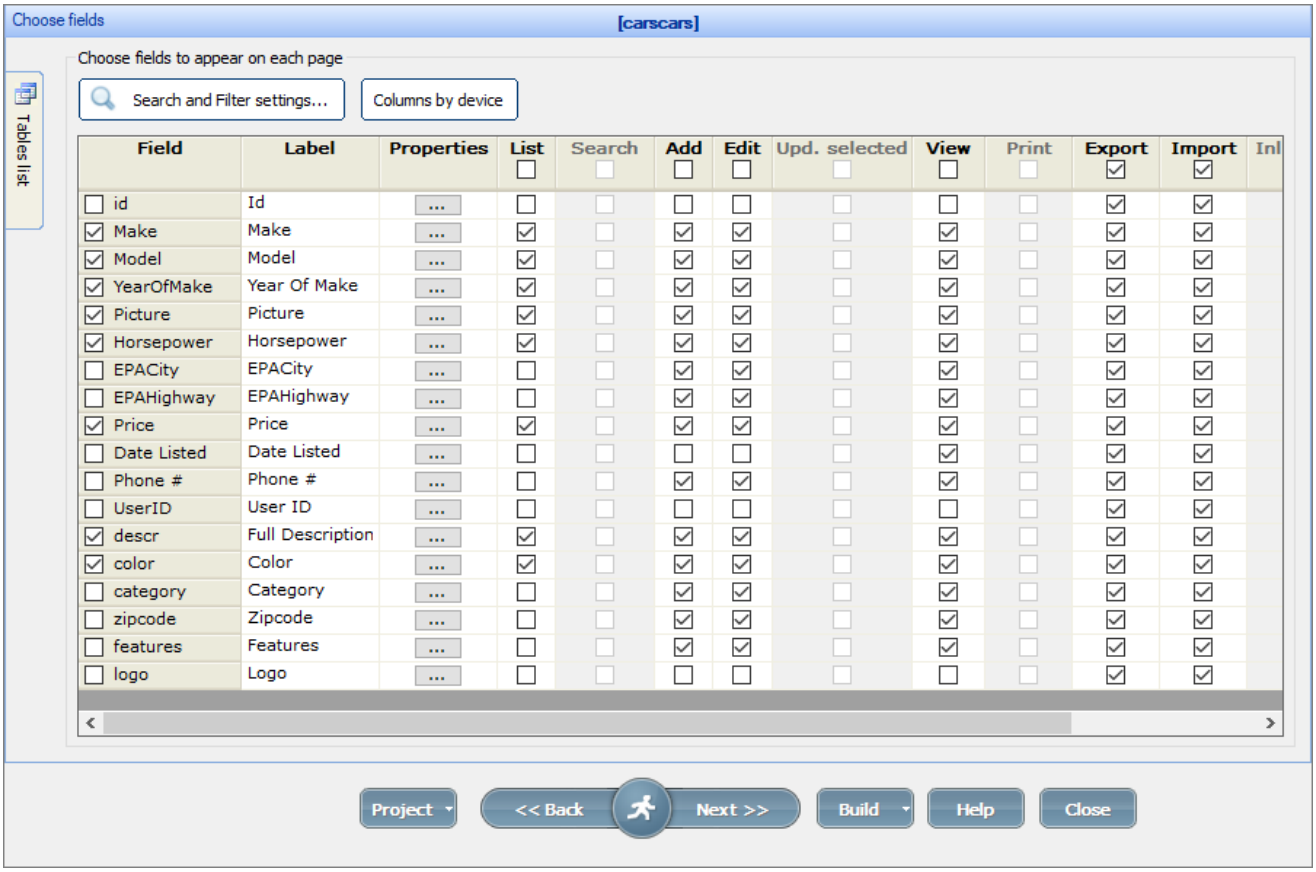
- Choose pages screen
- Key columns
- List page settings / Click actions
- Add/Edit page settings
- CAPTCHA on Add/Edit pages
- Update selected
- Export/Import pages
- Printer-friendly/PDF view settings

2.13 Choose fields screen

Quick jump

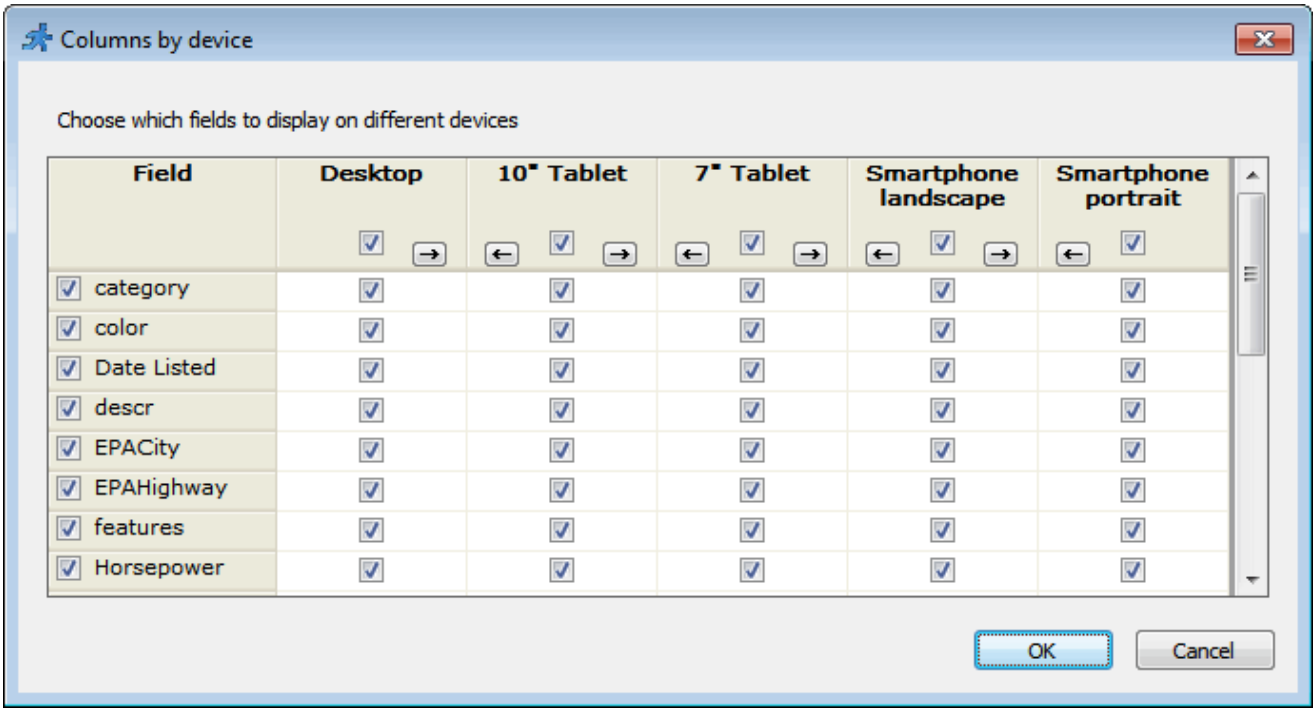
Choose fields screen

Use the **Choose fields screen** to select the fields to appear on each page. You can display different fields on different devices. You can also adjust search and filter settings.



Columns by device

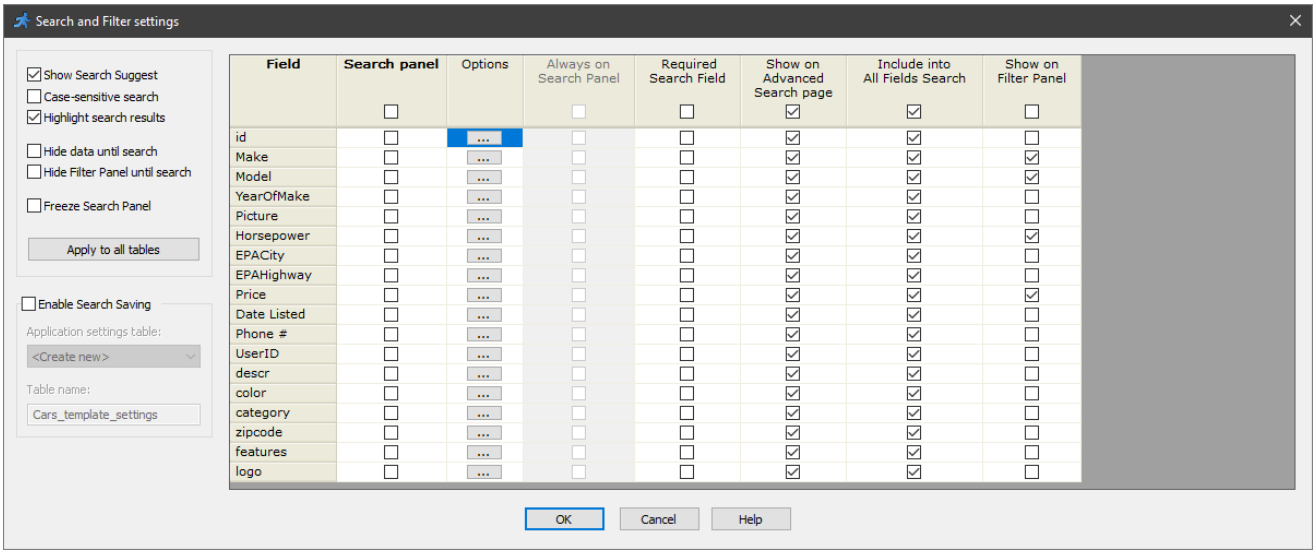
You can select different fields to display on different devices. This feature helps you customize your **List** pages by hiding specific columns on devices with smaller screens. You can easily copy selected fields from one column (device) to another using the left/right arrow buttons.



Search and Filter settings

Users can search by each word or by the whole phrase if double-quoted (Google-like search).

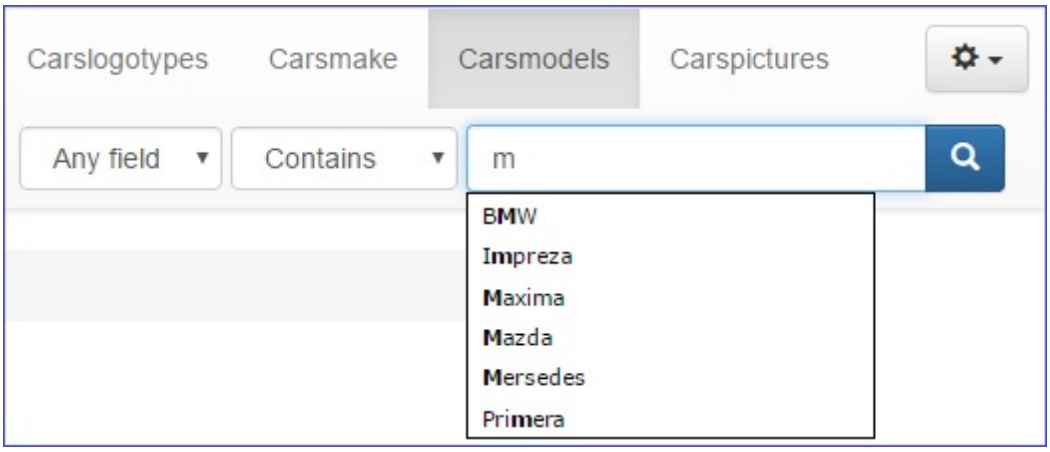
Click **Apply to all tables** button to apply the selected options to all tables.



Show Search suggest

When you start typing in the search box, an AJAX popup displays search suggestions. For more information, see [AJAX-based Functionality: AJAX Auto Suggest](#).

Example of the search suggest feature on the basic search page:

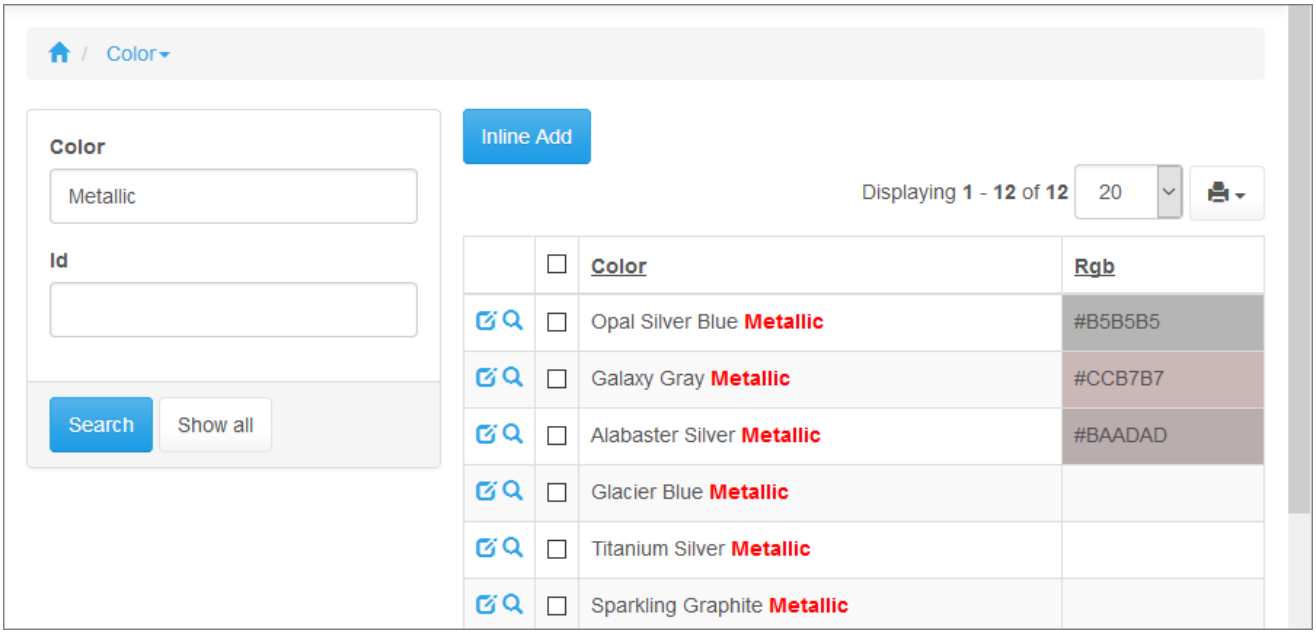


Case-sensitive search

Use this option to display only the search results that match the case of your search query. If this option is not selected, the search becomes case insensitive.

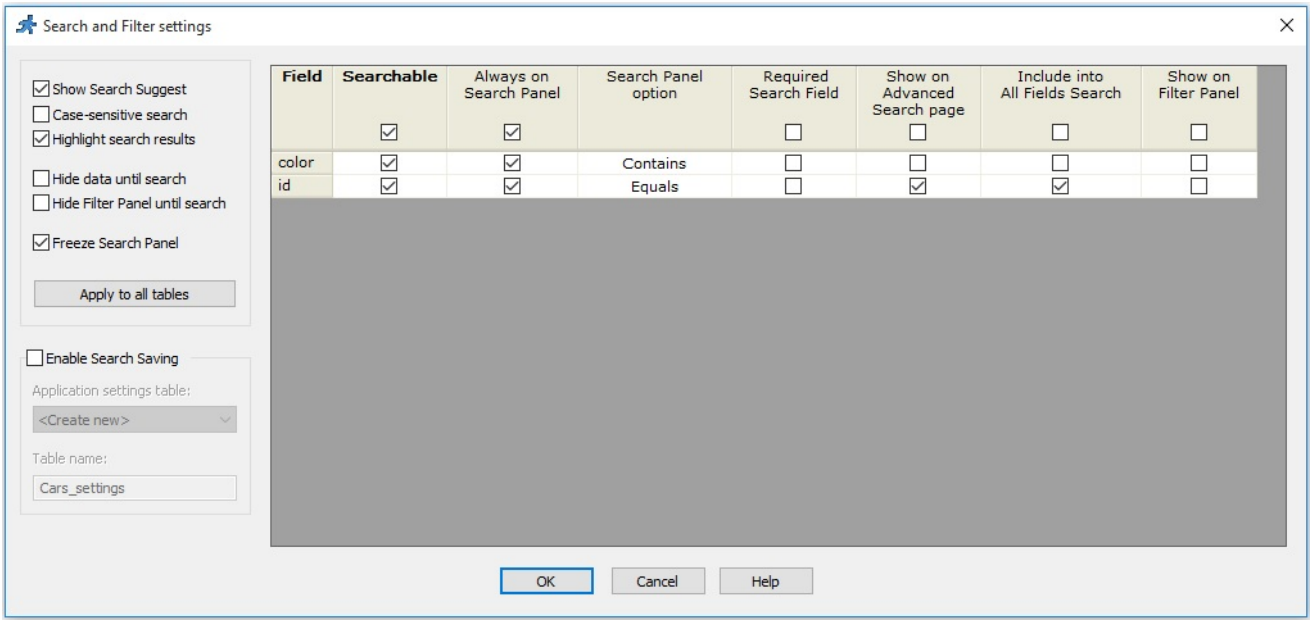
Highlight search results

Use this option to highlight search results in the database grid.



Freeze Search Panel

You can define the default search option (*Equals*, *Contains*, etc.) for each field on the search panel.



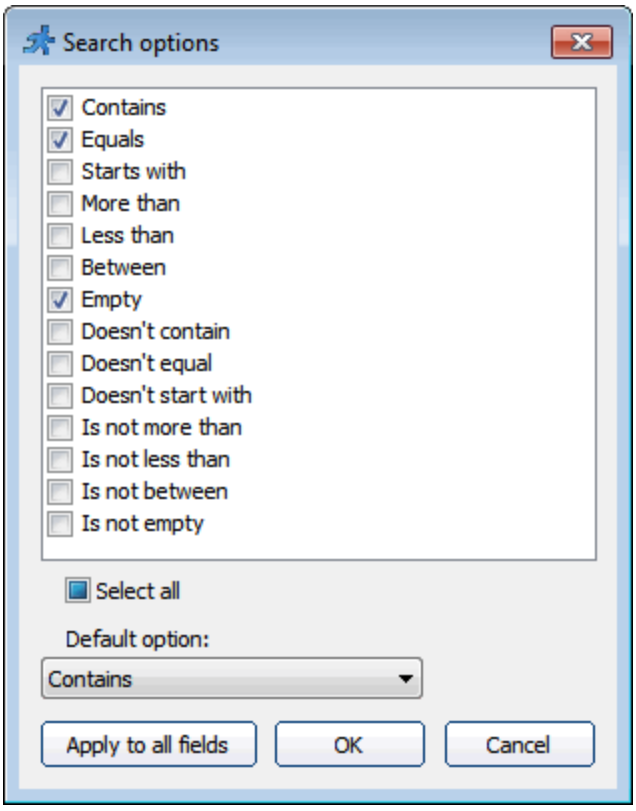
Searchable fields

When you make a field **searchable**, you allow searching by this field in the **Quick search panel**. You can also define other search options such as **Always on Search panel**, **Show on Advanced search page** and **Include into All fields Search** for this field.

> **Note**: to be able to make the field **searchable**, enable the **Show basic search options** option for the **List** page on the Choose pages screen.

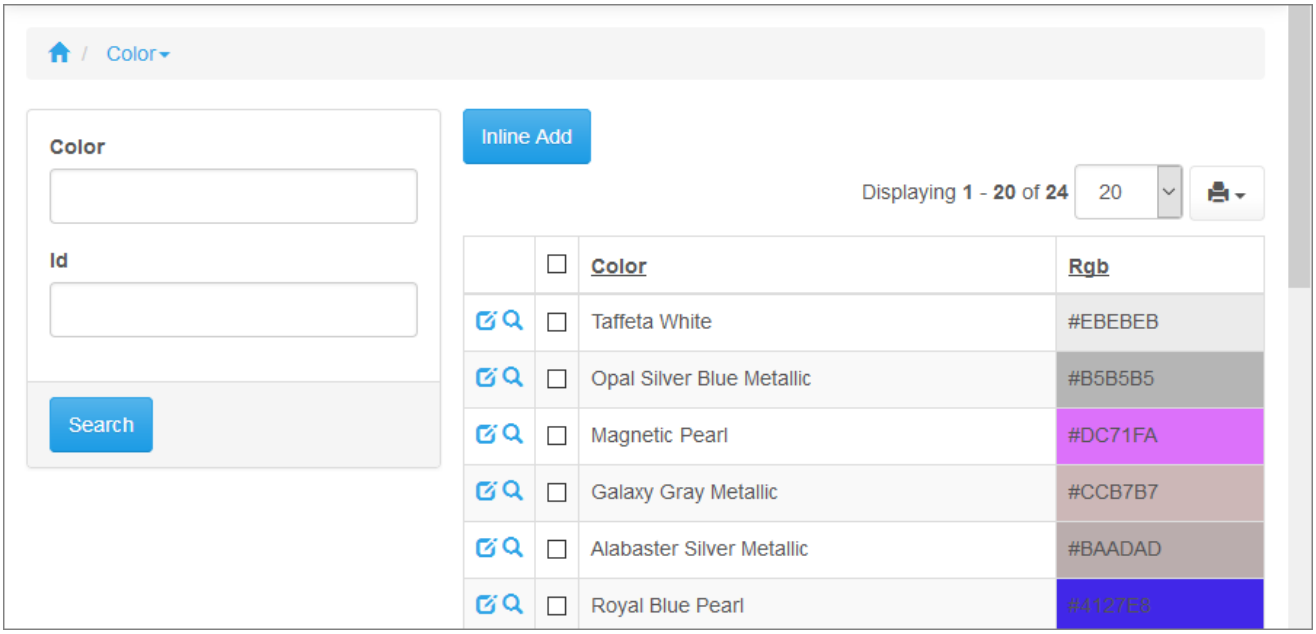
Options

You can select what search options are available for each field on the search panel. You can also define the default search option for each field.



Always on Search panel

Use this option to add fields to the search panel on the **List** page. The search panel allows you to define the search criteria.

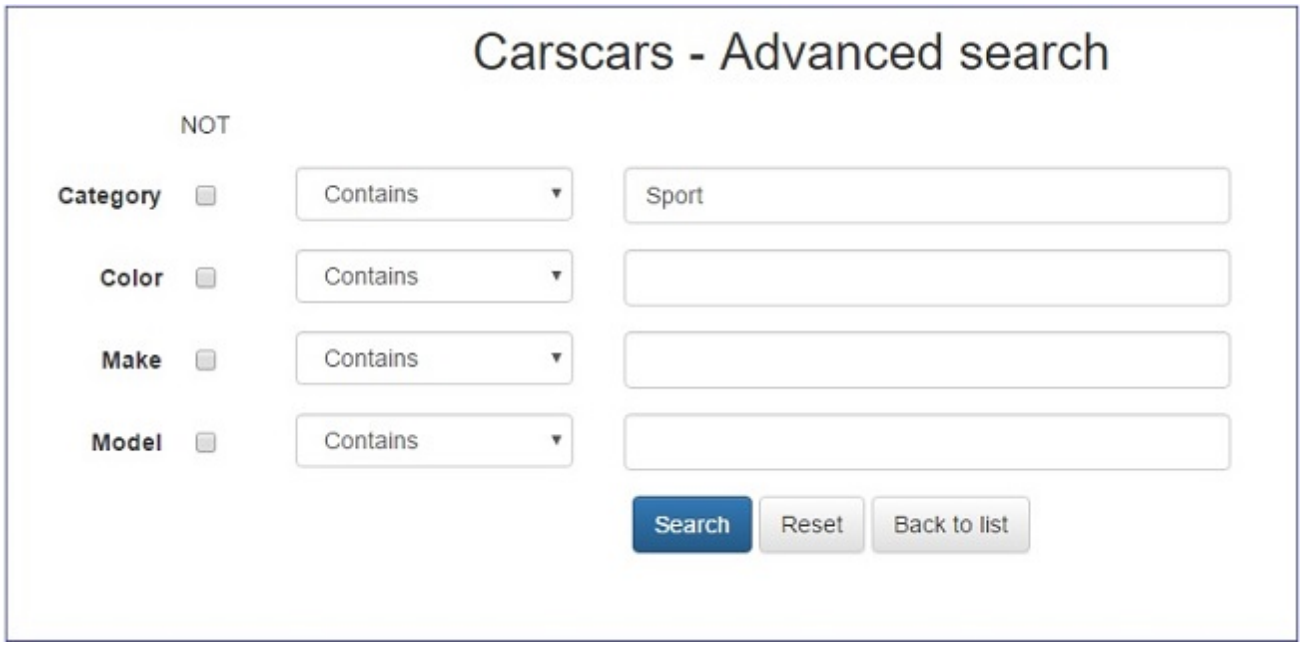


Required Search field

This option restricts search until users fill in the required search fields. For example, the Real estate template does not show any data until the zip code is entered.

Show on Advanced Search page

Allows displaying the fields on the **Advanced search** page.

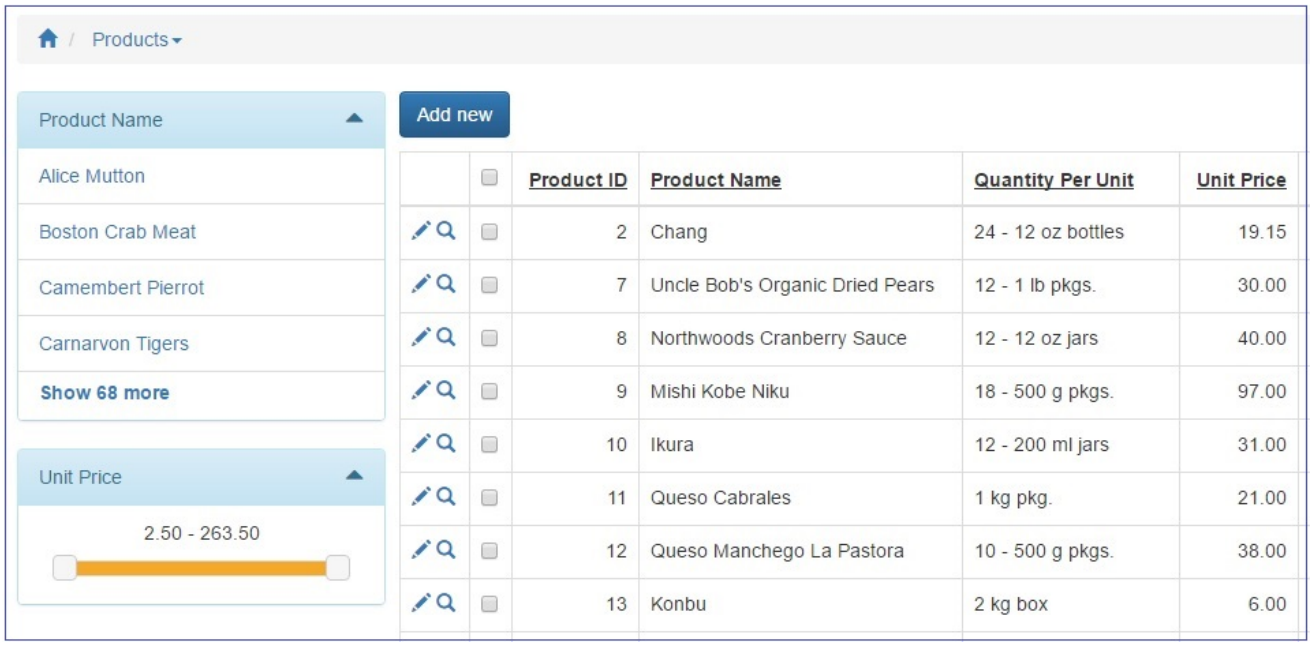


All fields search

Use this option to add fields to the 'All fields' quick search.

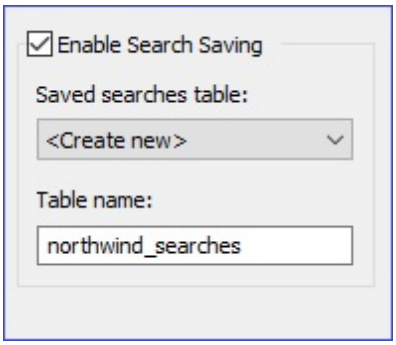
Show on Filter Panel

Allows displaying the fields on the **Filter Panel**. Filters provide instant feedback on the number of records matching each criterion. You can select multiple criteria or multiple intervals for each filter. The **Filter panel** is located on the left side of the generated app. For more information about filter settings, see "Filter as" settings.

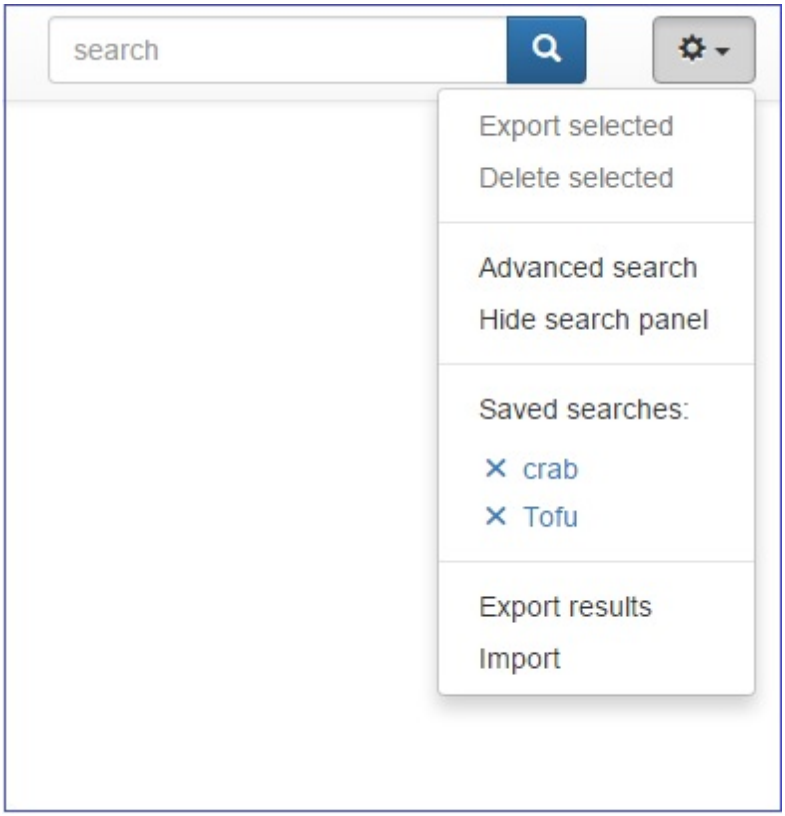


Enable Search Saving

When searching for data using the **Search panel**, you can save searches in the database for later retrieval (use the **Save search** button in the browser). Select *<project_name>_searches* as the table for saving searches or create a new one.



To view the saved searches, click **Saved searches** on the **Search panel**.



See also:

- Choose pages screen
- Miscellaneous settings
- Page Designer
- AJAX-based functionality
- Search API

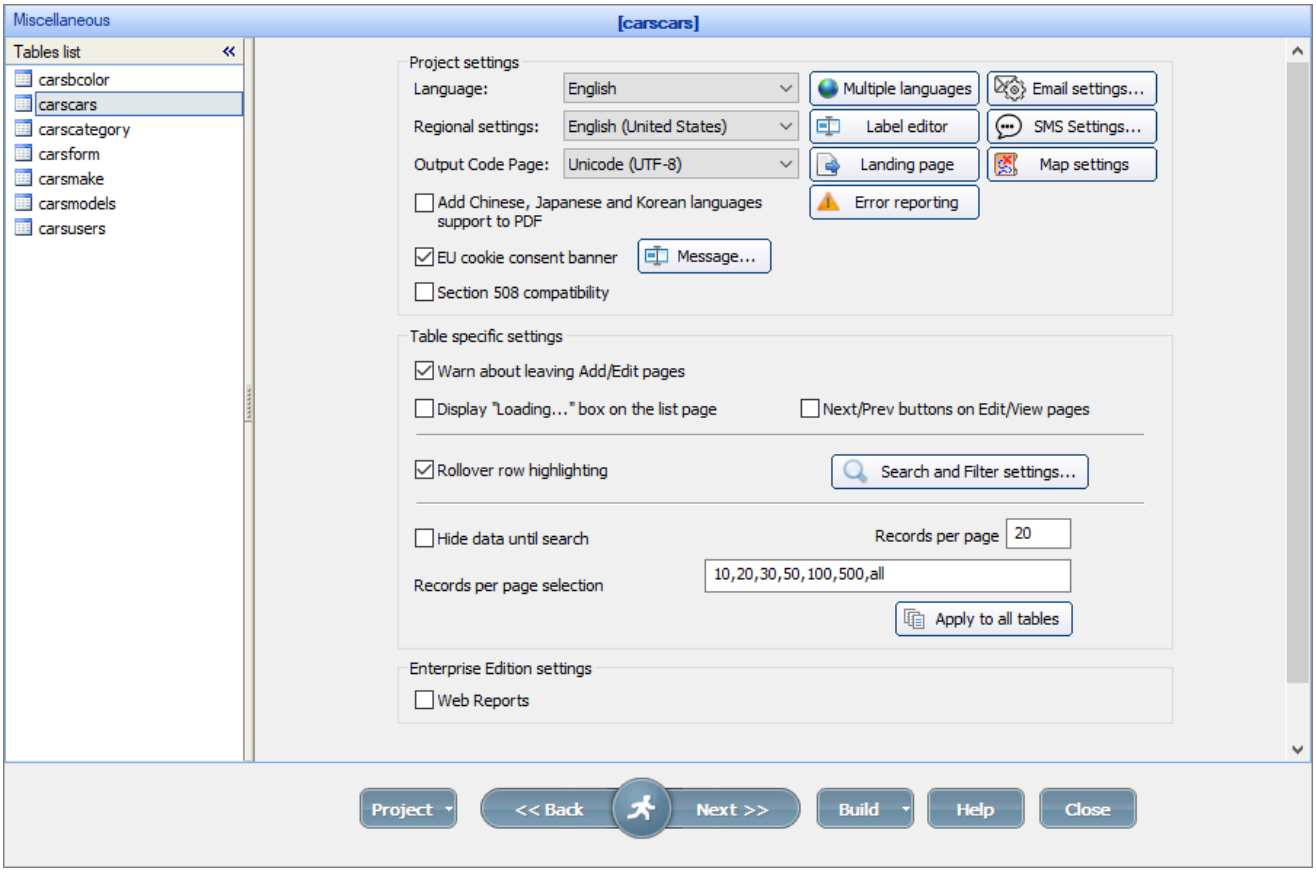
2.14 Miscellaneous settings

Quick jump

Project settings
Language
Regional settings
Landing page
Map settings
Error reporting

Miscellaneous settings screen

The Miscellaneous settings screen allows setting Project, Table specific, and Enterprise Edition settings.

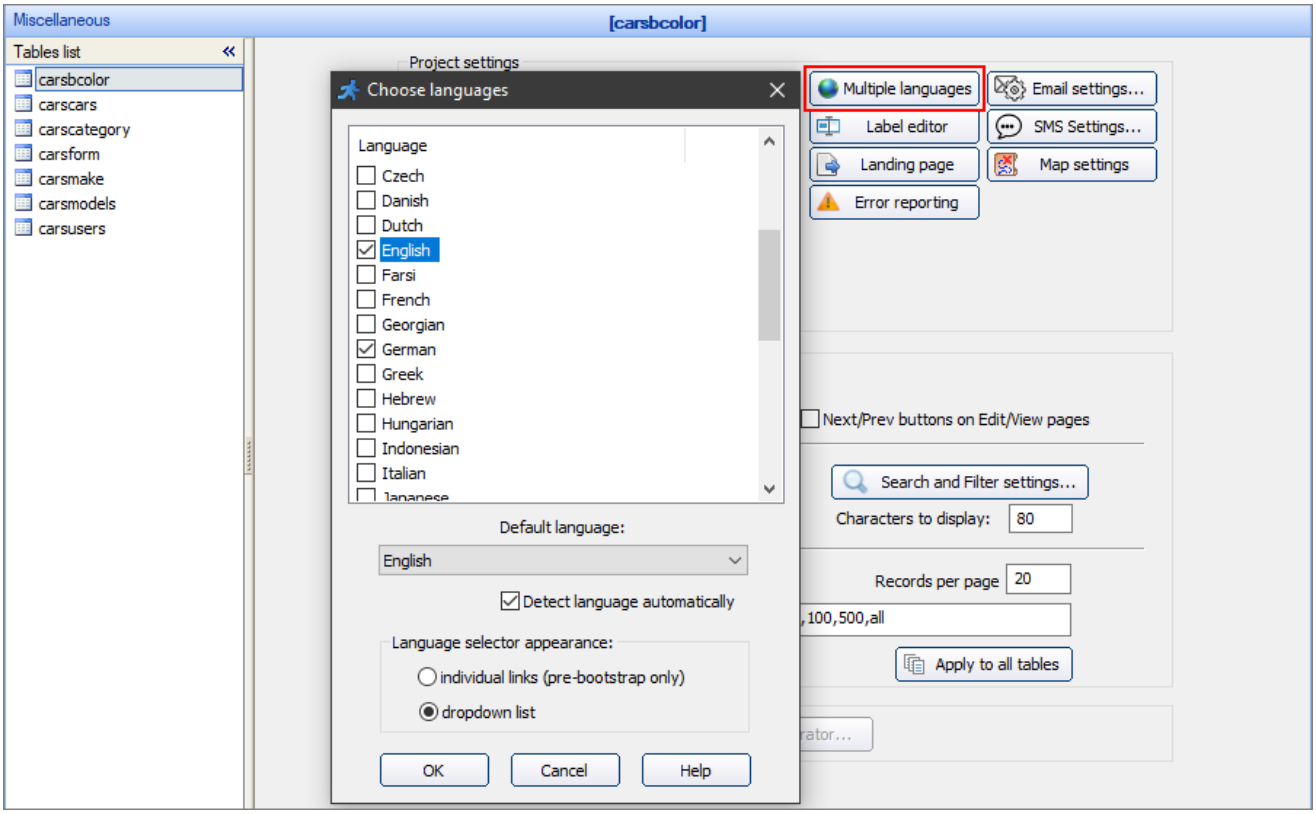


Project settings

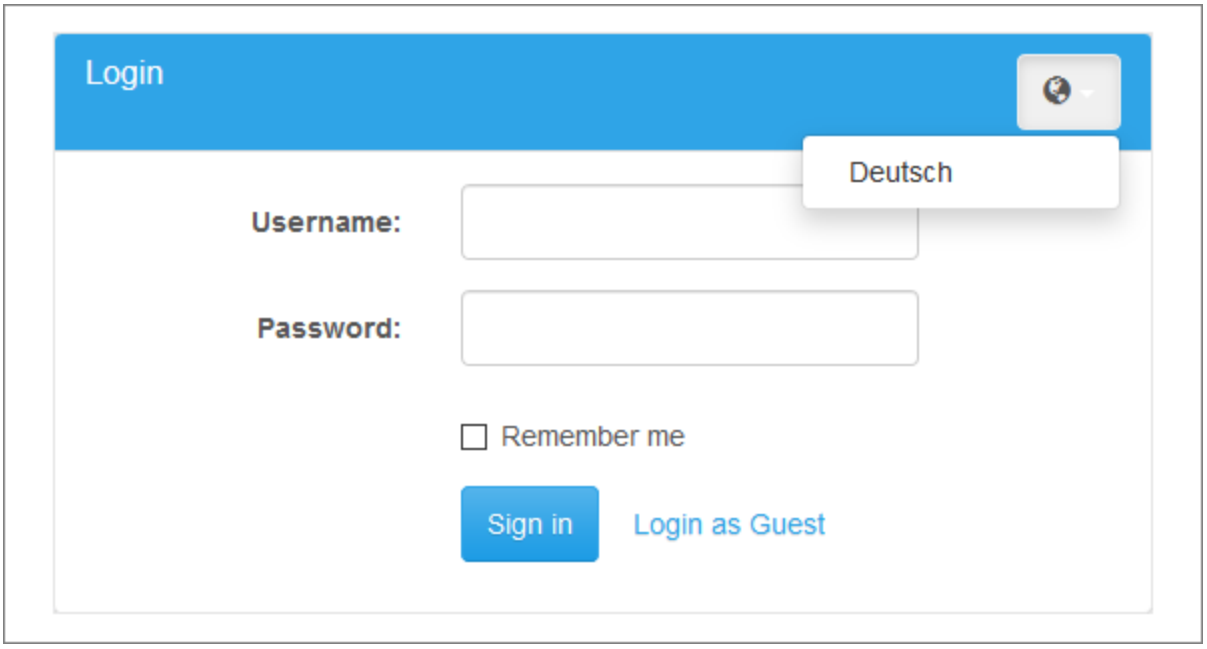
The **Project settings** block includes settings that affect the entire project.

Language

Use the **Language** dropdown box to set the project language. By clicking the **Multiple languages** button, you can select several project languages and give users the ability to select the language when logging in. The translation of standard buttons and elements is already included in the PHPRunner, so you don't need to translate them yourself.



This is how the **dropdown list** option looks like in the generated project.



Regional settings

Choose the country from the dropdown box to select its regional settings. These settings affect the year-month-day order, the first day of the week, etc.

Output code page

This dropdown allows selecting the character encoding for the project.

Label Editor

Label Editor allows you to:

- edit table and field labels (Table labels tab);
- edit custom labels (Custom labels tab);
- add tooltips to the Edit forms (Edit form tooltips tab);
- edit web page titles (Page titles tab);
- edit the project logo (Project Logo tab);
- setting the placeholders for the fields (Input placeholders tab);
- editing the EU cookie consent banner text (EU cookie banner tab).

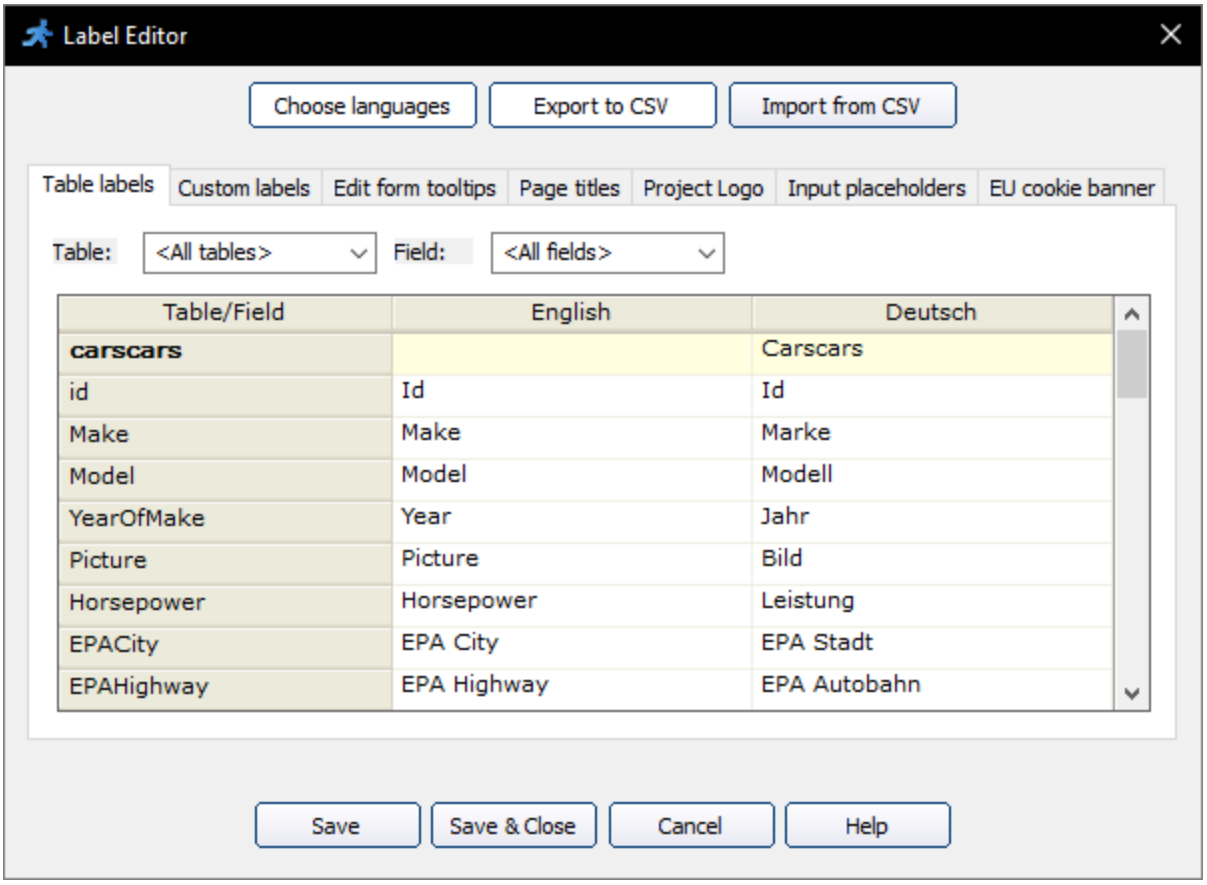
You can translate labels into several languages. Click **Choose languages** to add more languages to the project.

You can export multilanguage labels to a CSV file (use the **Export to CSV** button), then edit the labels in the external editor and import the file back (use **Import from CSV** button).

> **Note**: if you use languages other than English in the project, we recommend editing the CSV files using Google Docs or OpenOffice.
>
> Due to the problems with the character encoding, we do not recommend using Microsoft Excel for projects with multiple languages.

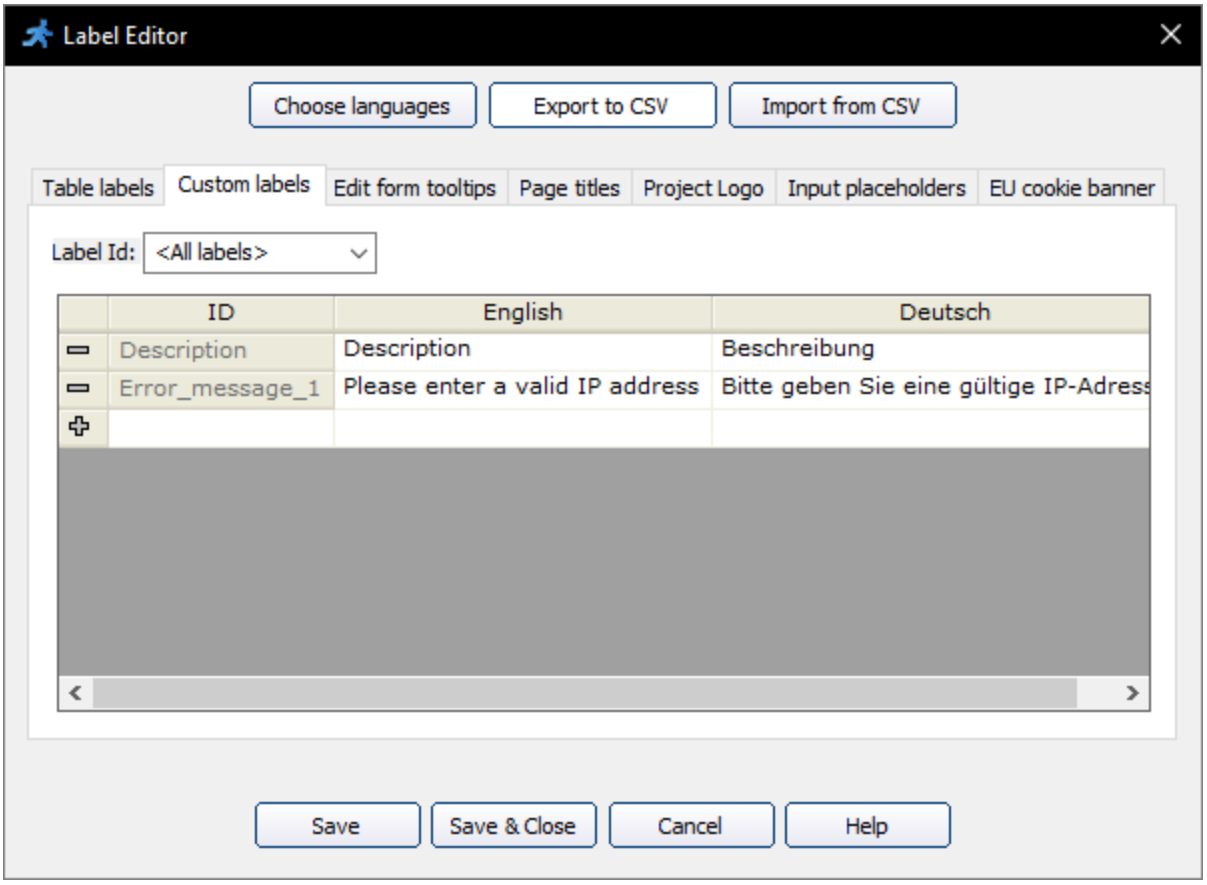
Table labels

Modify and translate the table and field labels to make them more user-friendly. Filter the labels by table or field to limit the data displayed. Use *
* to insert a line break in the field label.



Custom labels

Use the **Custom labels** to translate error messages in regular expressions and custom validation plugins. You can also create custom labels to display messages to a user. Filter the labels by the *Label Id* to limit the data displayed.



If you have created **Custom labels** and want to place the language-dependent text on a page, you can add a PHP code snippet to the page and use the following code in it:

```
echo GetCustomLabel ("CustomLabelID");
```

Replace "*CustomLabelID*" with the correct custom label identifier.

You can also use custom labels in a page template. Open **Editor** and switch to HTML mode. Then add the following code:

```
{$custom CustomLabelID}
```

And in JavaScript code:

```
Runner.getCustomLabel('CustomLabelID')
```

While editing the page title, you can use the field values, e.g. {%ID} and {%Make} for **View** and **Edit** pages. For example:

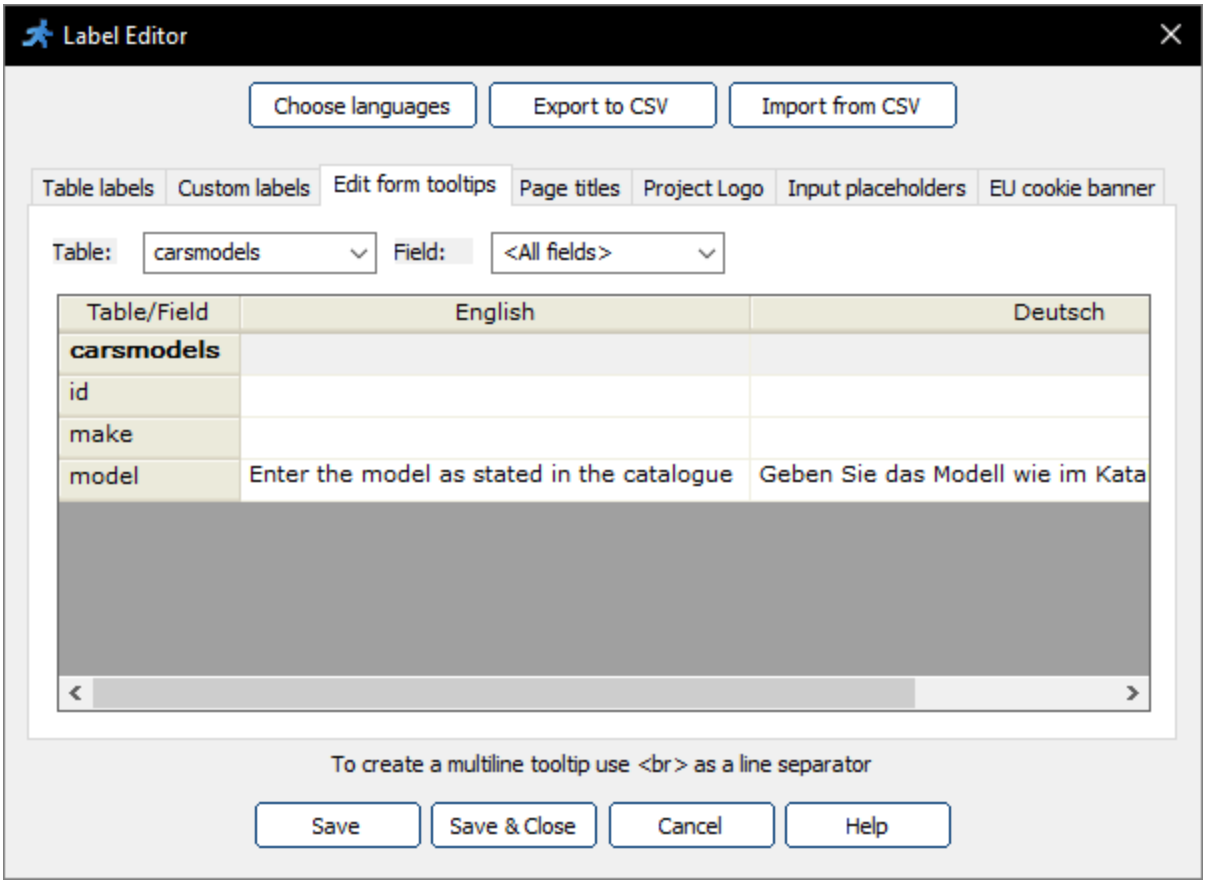
```
Cars, Edit record [ID:{%ID}]
Customers, {%FirstName} {%LastName}
```

You can also use the field values of the master table on the **List** page of the details table. Here is the example of a page title for the *Order details* table:

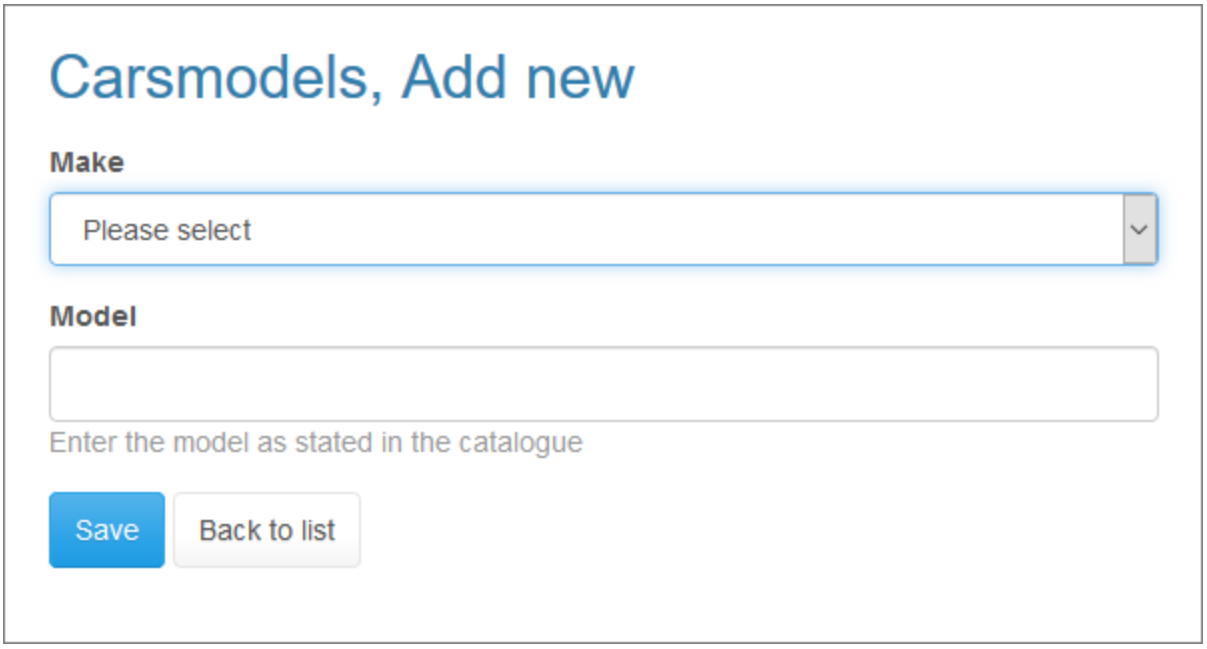
```
Orders, ID: {%master.OrderID}
```

Edit form tooltips

Add a tooltip to a field to display additional information about it on the **Add**/**Edit** page, or during inline editing on the **List** page. To create a multiline tooltip, use <br> as a line separator.

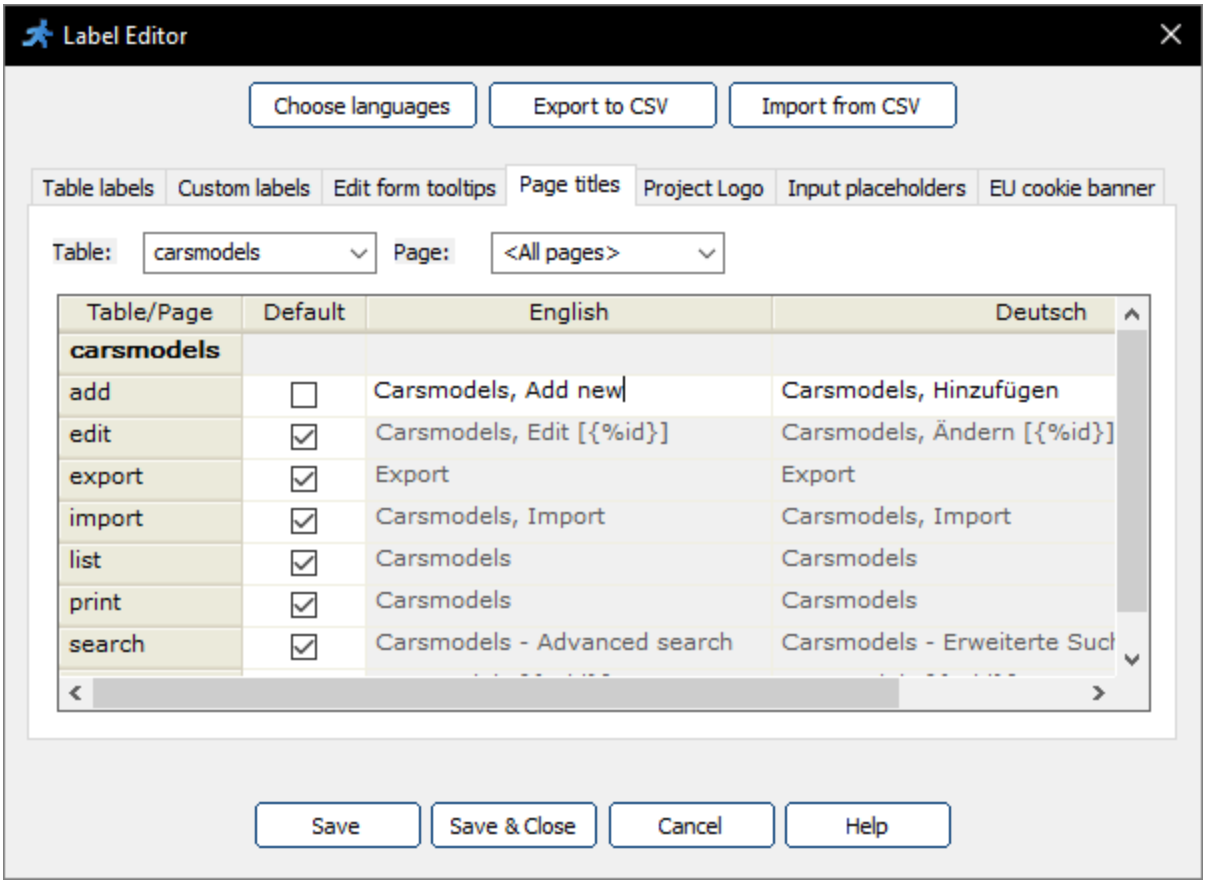


Here is how it looks in the browser:



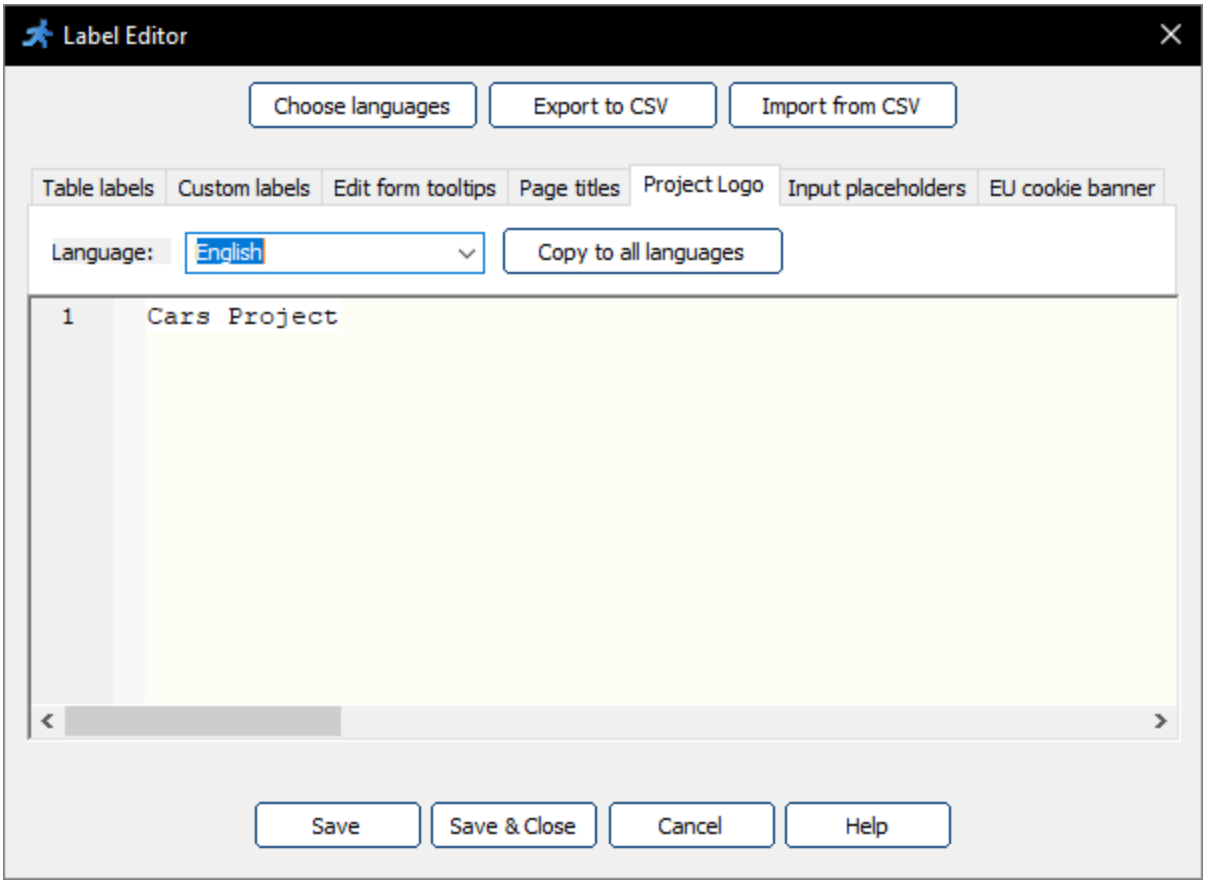
Page titles

To edit a web page title, deselect the **Default** checkbox next to it and make the changes. Filter the page titles by the table or page to limit the data displayed.



Project logo

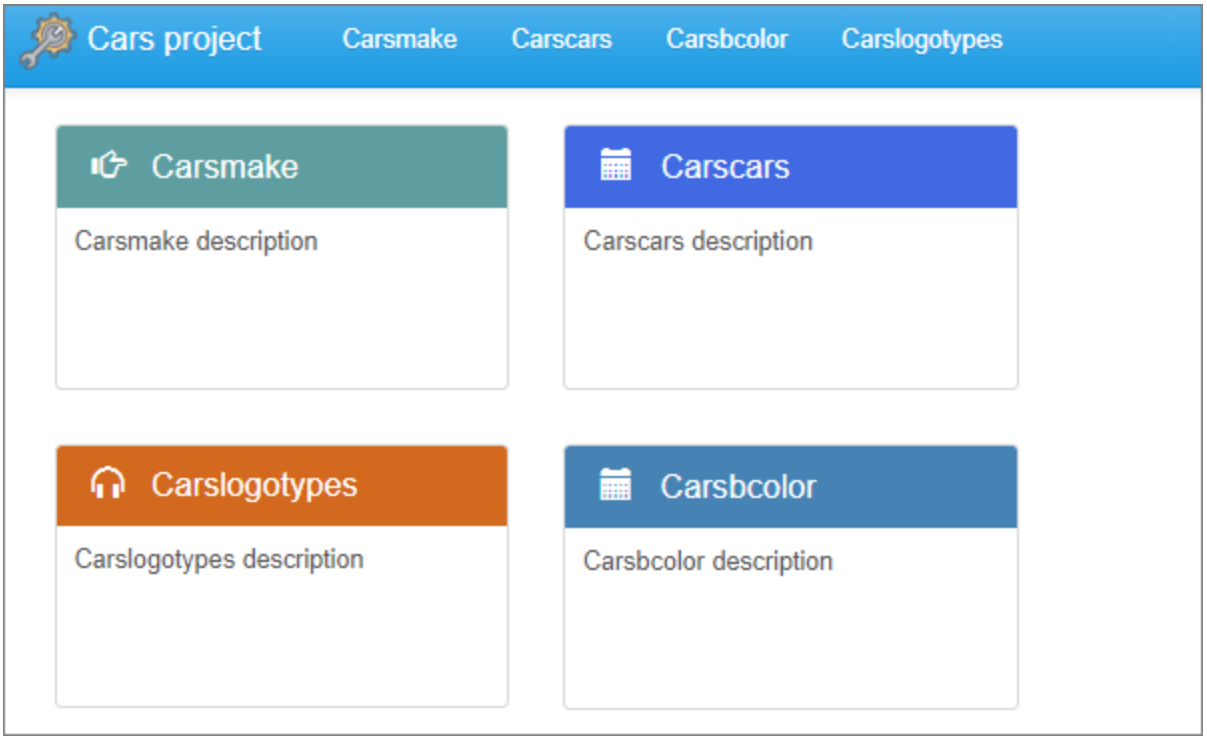
Change the project logo with the **Project logo** tab. You can enter plain text or HTML code here.



Here is an example of the code that can be used to display a logo icon and text:

```
<DIV style="position: relative; height: 40px; top: -6px; white-space: nowrap;">
      <IMG height="80%" id="navbarlogo" alt="Project logo"
src="http://mywebsite.com/images/logo.png">
      Cars Project
</DIV>
```

That's how it looks like on the generated page:



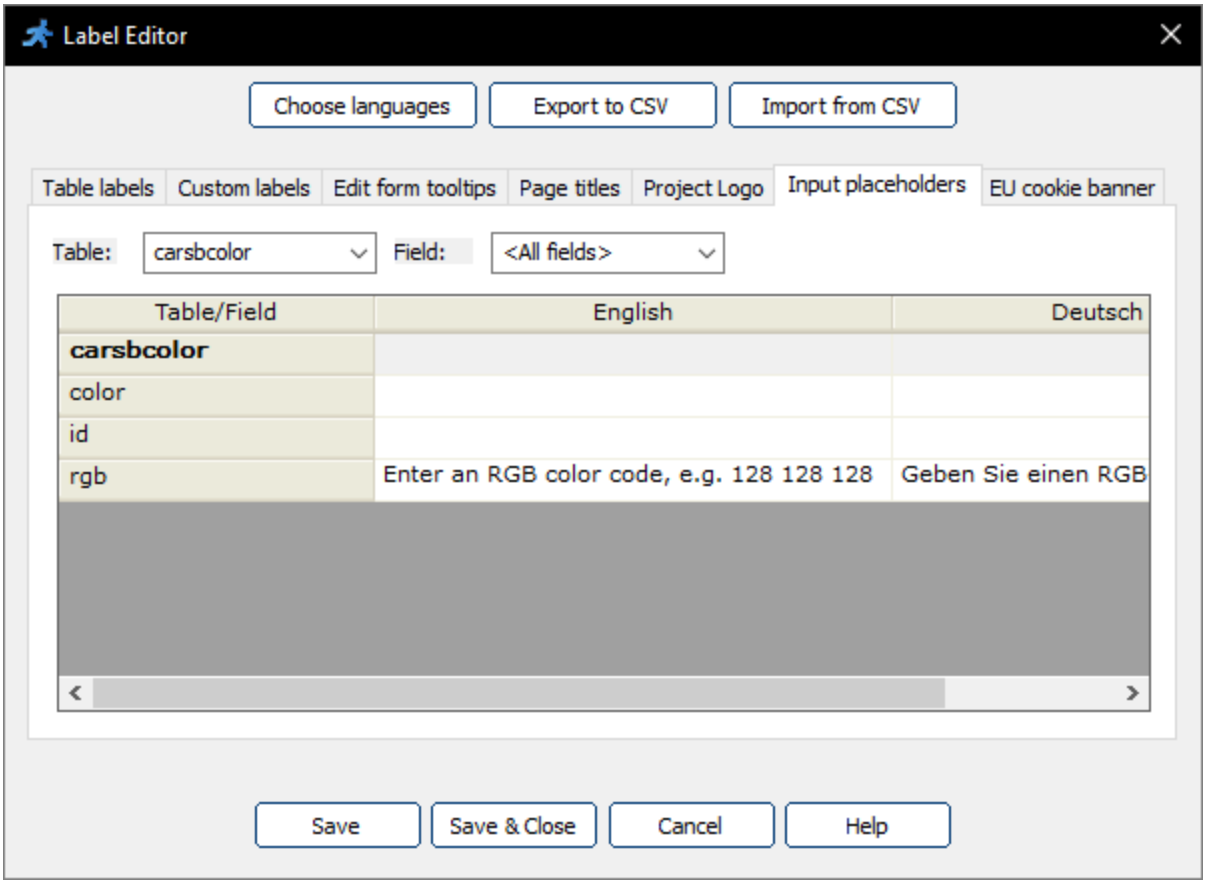
You can do even more by using the Labels/Titles API function setProjectLogo. You may, for example, add the logo image and the current year to the project logo.

Put the logo.png into the images folder of your application, and add the following code to AfterAppInit event:

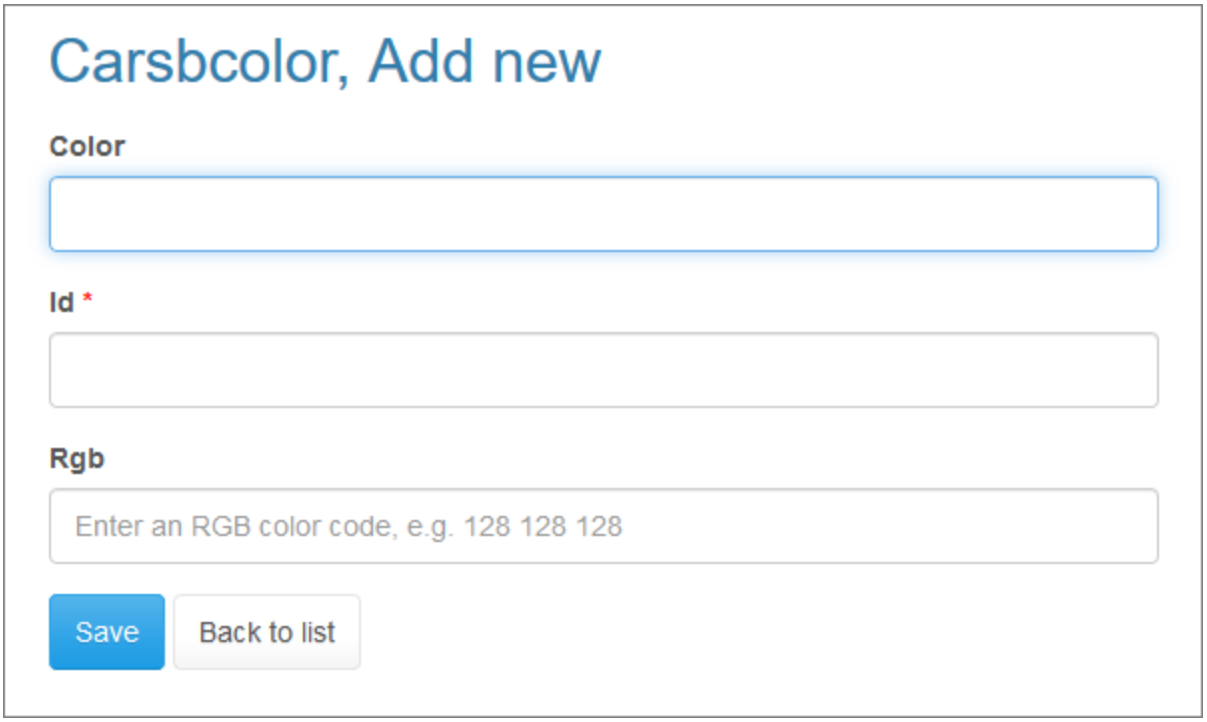
```
$currentYear = date("Y");
Labels.setProjectLogo('
  <DIV style="position: relative; height: 40px; top: -6px; white-space: nowrap;">
      <IMG height="80%" id="navbarlogo" alt="Project logo"
src="http://mywebsite.com/images/logo.png">
      Cars project'.$currentYear.'
  </DIV>
');
```

Input placeholders

The placeholder specifies a short hint that describes the expected value of an input field (e.g., a sample value or a short description of the expected format). Placeholders are displayed in the input field before the user enters a value.



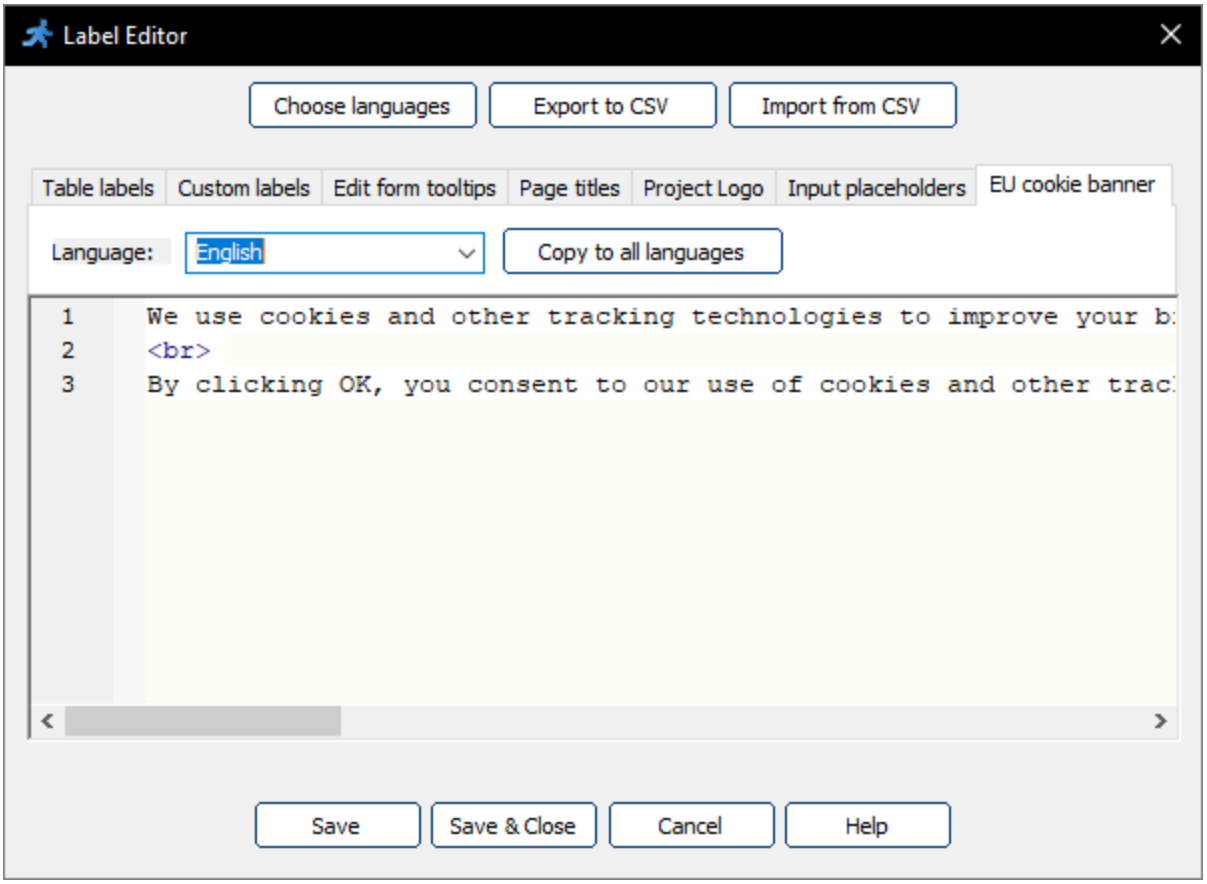
Here is how it looks like in the browser:



Check this Live Demo for more information.

EU cookie banner

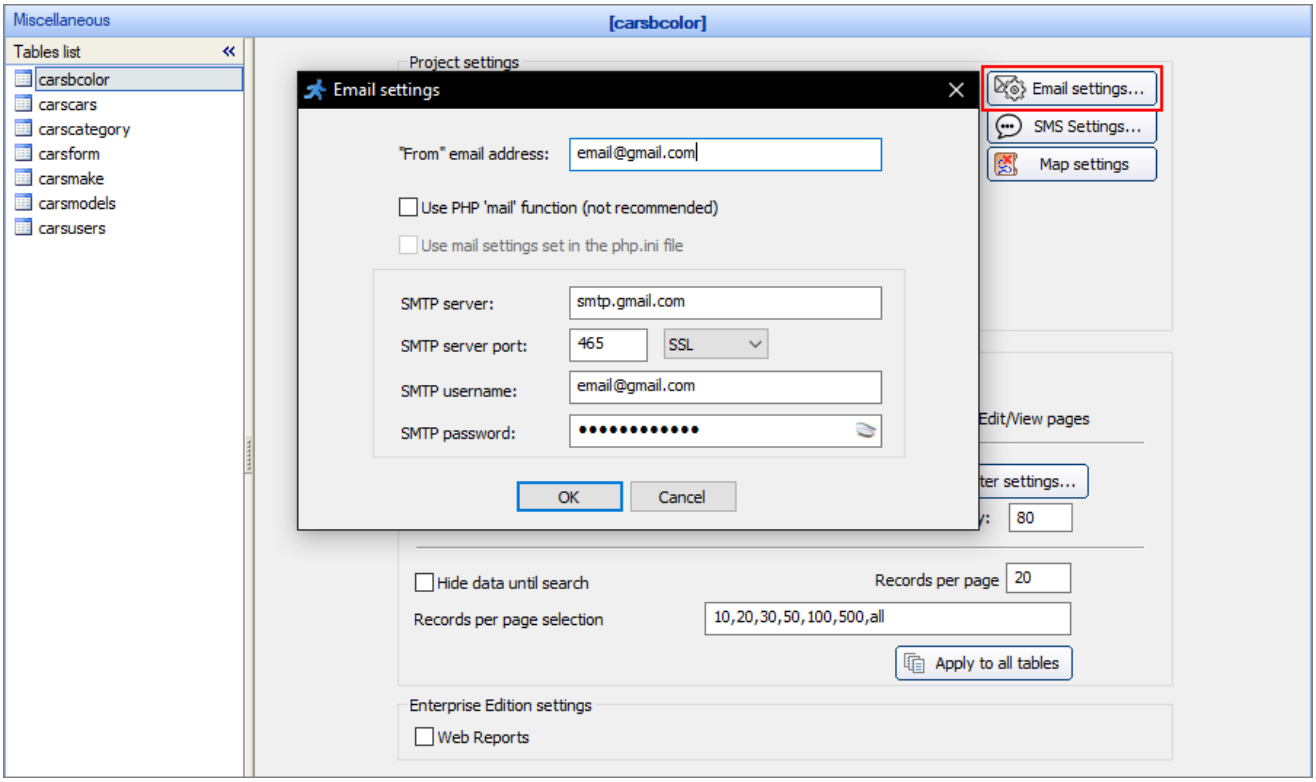
This tab sets the text for the EU cookie consent banner option. You can use *
* as a line separator.



Email settings

With **Email settings**, you can enter the email from which to send emails to the users, and define the settings of the custom mail server, if you do not use the built-in mail server. The Two-factor authentication must be turned off to send emails.

You can use the PHP 'mail' function by selecting the corresponding checkbox, although it is not recommended.



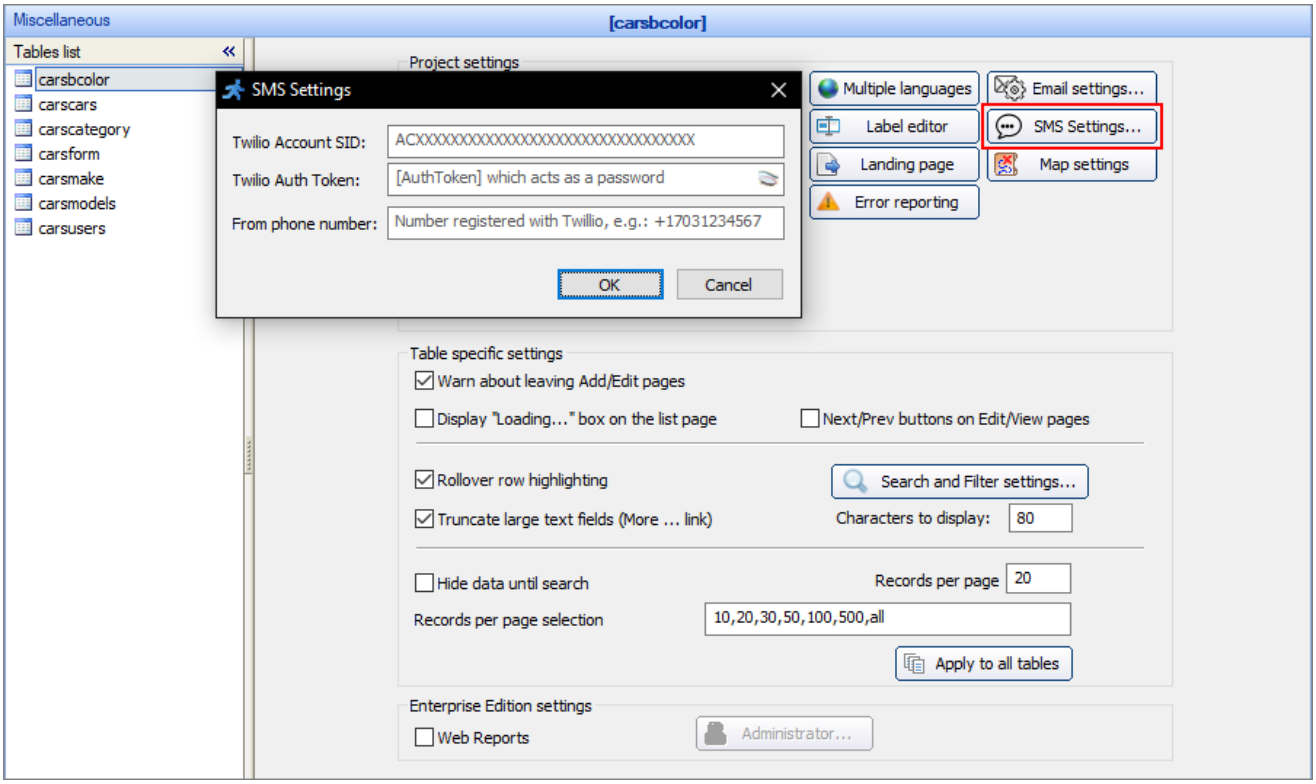
To send emails via Gmail, use the following settings:

- SMTP server: smtp.gmail.com
- SMTP server port: 465
- Secure connection: SSL
- SMTP username: <your gmail address>
- SMTP password: <Gmail password>

> **Note**: SMTP server, SMTP server port, as well as Secure connection (SSL) settings, may differ from the ones stated in the example. Contact your Mail Service Provider to get the necessary information.

SMS settings

The **SMS Settings** option lets you use your Twilio account to send the SMS to users. SMS are required to set up the Two-factor authentication.



You can open a Twilio account here: https://www.twilio.com/try-twilio.

> **Note**: Twilio is not free. Their rates depend on the number of messages you send among many other things.

Multiple SMS providers

PHPRunner supports several SMS providers, you can choose the most suitable one.

List of the providers:

- 1s2u.com
- easysendsms.com
- gatewayapi.com
- messagebird.com
- twilio.com

Here is a step-by-step instruction on how to use these providers:

1) Proceed to the "*smsapi*" folder. You can find it in your installation folder (*source**include**smsapi*).

2) Copy an SMS provider file, e.g., "*messagebird.php*", from that folder and replace the existing *sms.php* file with it. The path to the *SMS file* is *source**include**sms.php*.

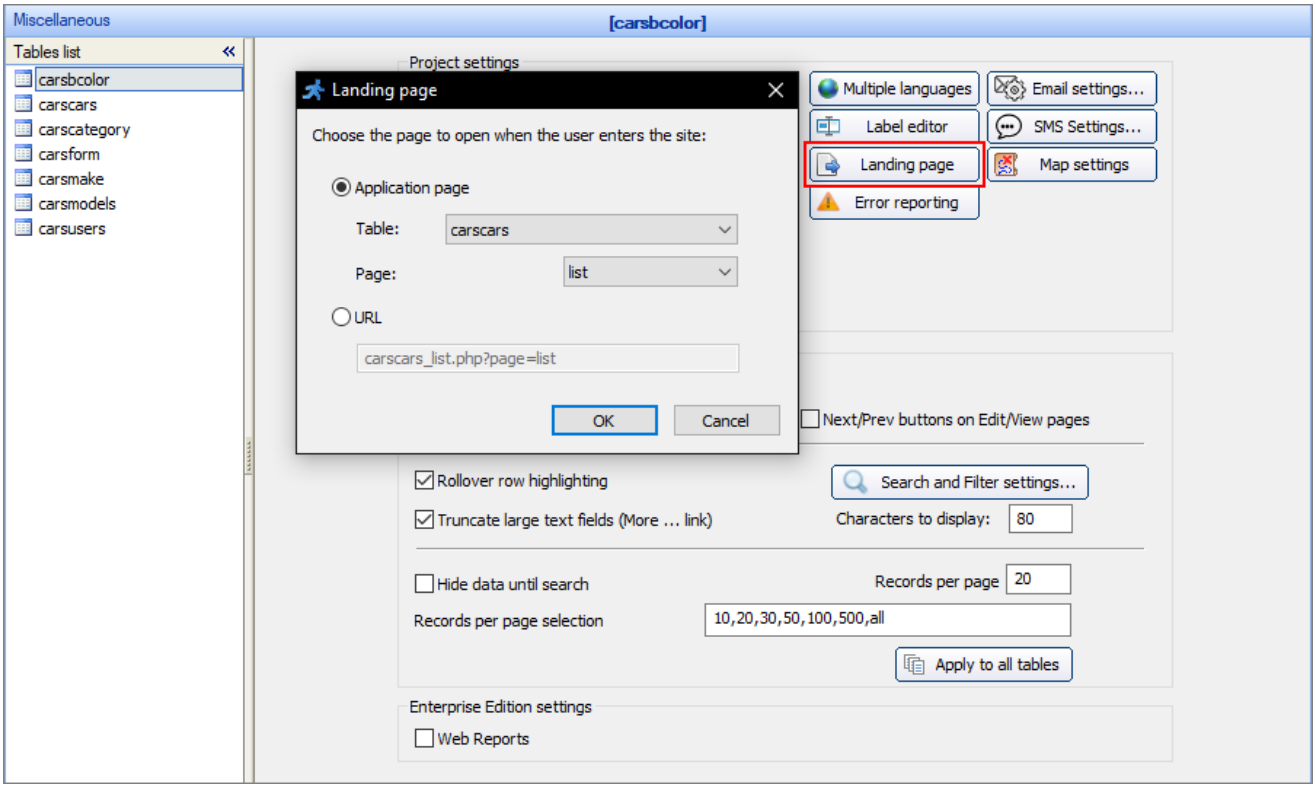
3) Create an account on the website of the chosen provider.

4) Insert your credentials into the new SMS provider file.

After you do these steps, the messages will go through the selected SMS provider. It is used in the Two-factor authentication mechanism, and also, each time the Runner_sms function function is invoked.

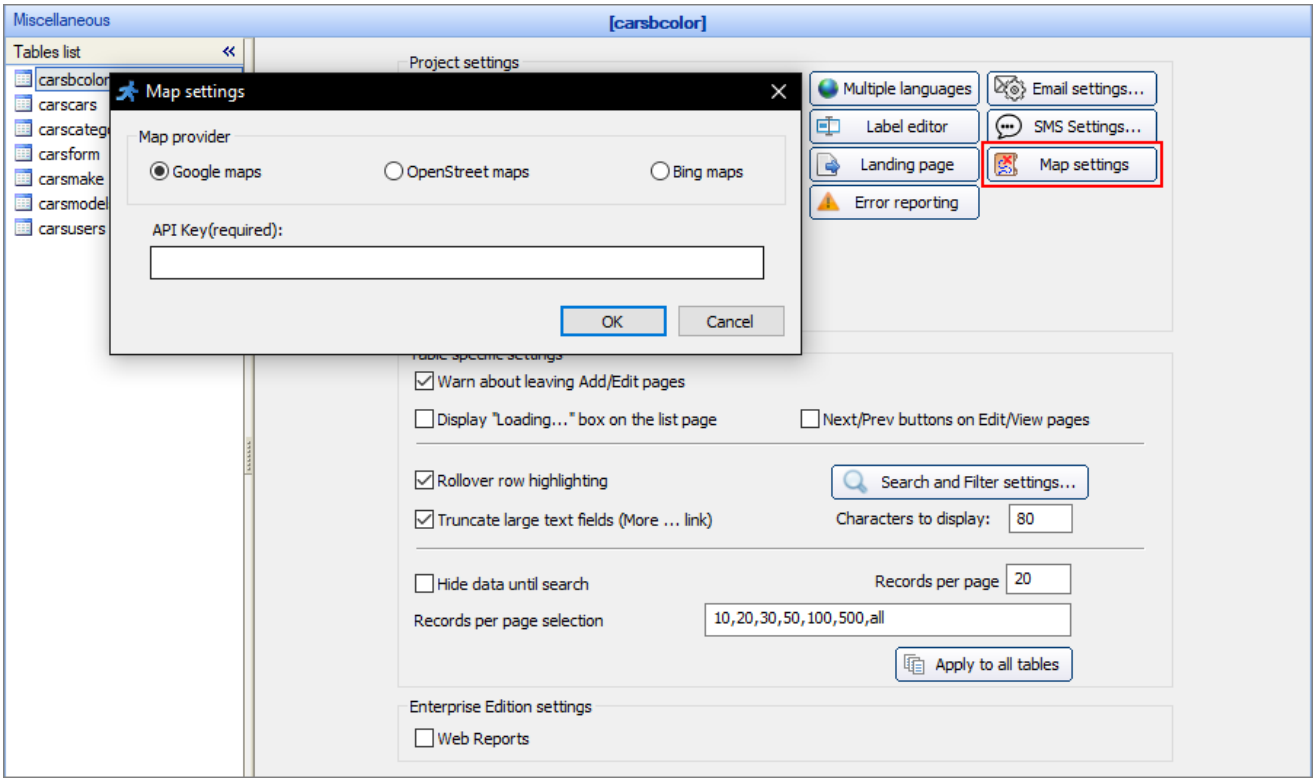
Landing page

Select the application page to open after the Login page, or when the user enters the site, if the login is disabled or a guest user login is enabled. Alternatively, you can enter a custom URL, for example, a URL with search options *carscars_list.php?qs=audi*.



Map settings

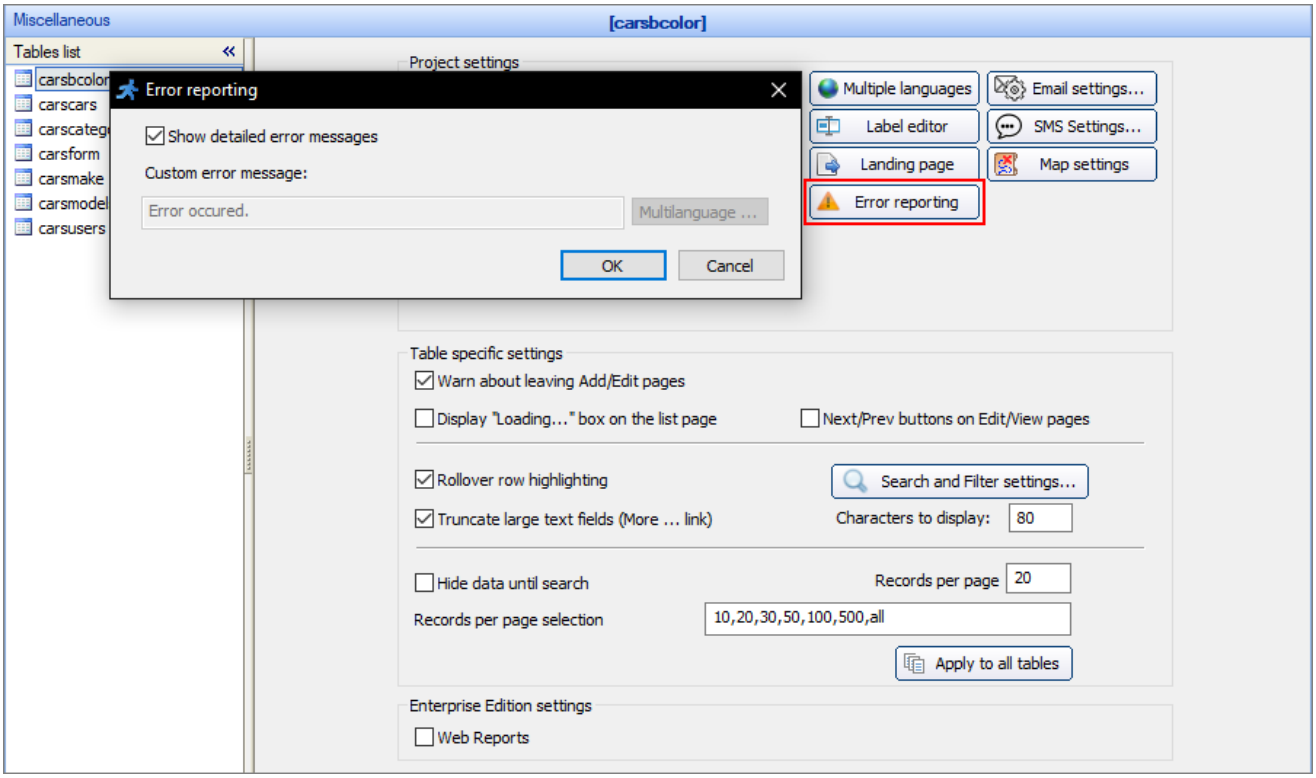
Select the map provider and enter the API Key (if required). You can get Bing maps API key for development purposes for free at bingmapsportal.com.



To learn more about inserting maps and working with them, see Insert Map.

Error reporting

You can select **Show detailed error messages** to display detailed error messages or deselect it and type in the custom error message. You can also translate the custom error message into all of the languages of the project.



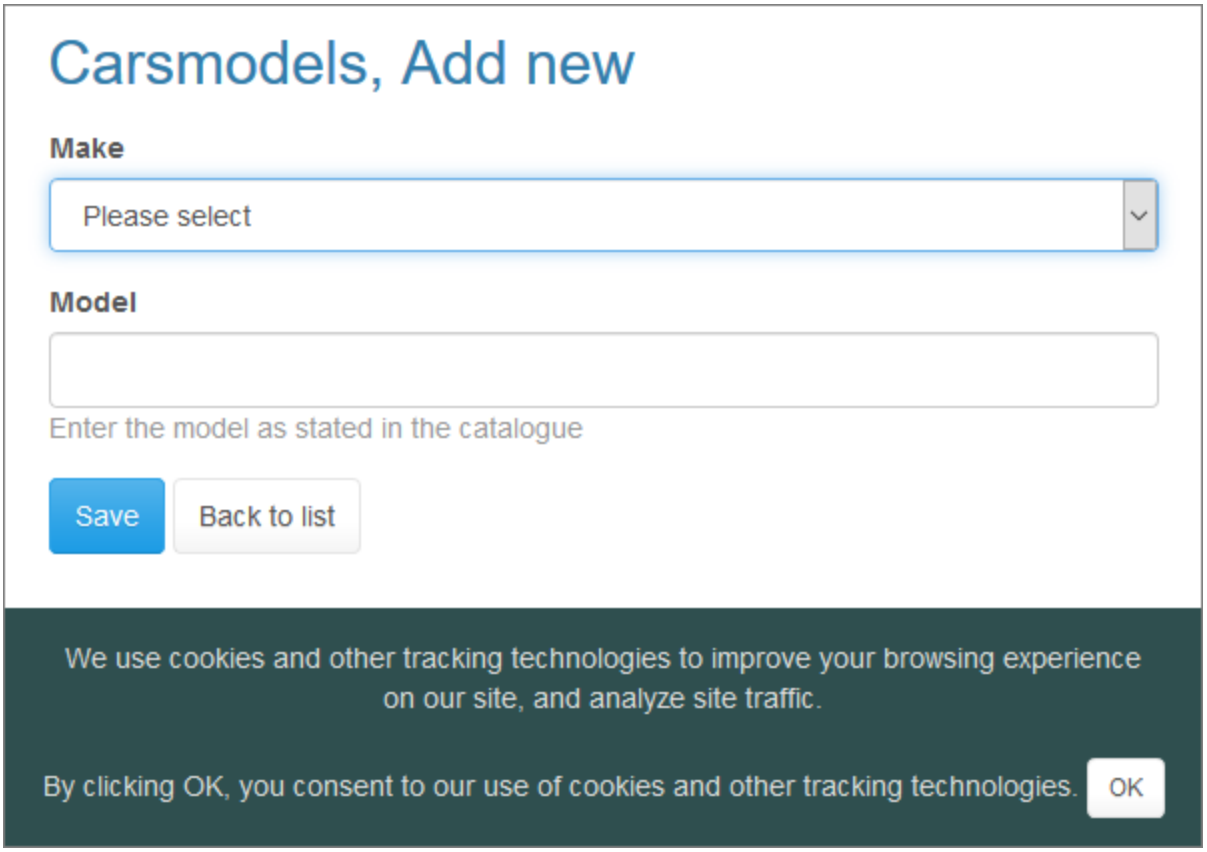
Add Chinese, Japanese, and Korean language support to PDF

Select this checkbox if you have selected any of the languages above as project languages and you use PDF-related options in your project.

EU cookie consent banner

Select the **EU cookie consent banner** checkbox to enable the customizable banner which appears at the bottom of the page. To edit the text, click the **Message** button, and enter the text into the EU cookie banner tab of the **Label Editor**.

Here is how the banner looks like in the browser:



Section 508 compatibility

Select this checkbox to make the project accessible to a wide range of people with disabilities. For more information, visit http://www.section508.gov/.

Additionally, there are keyboard hotkeys available as described below:

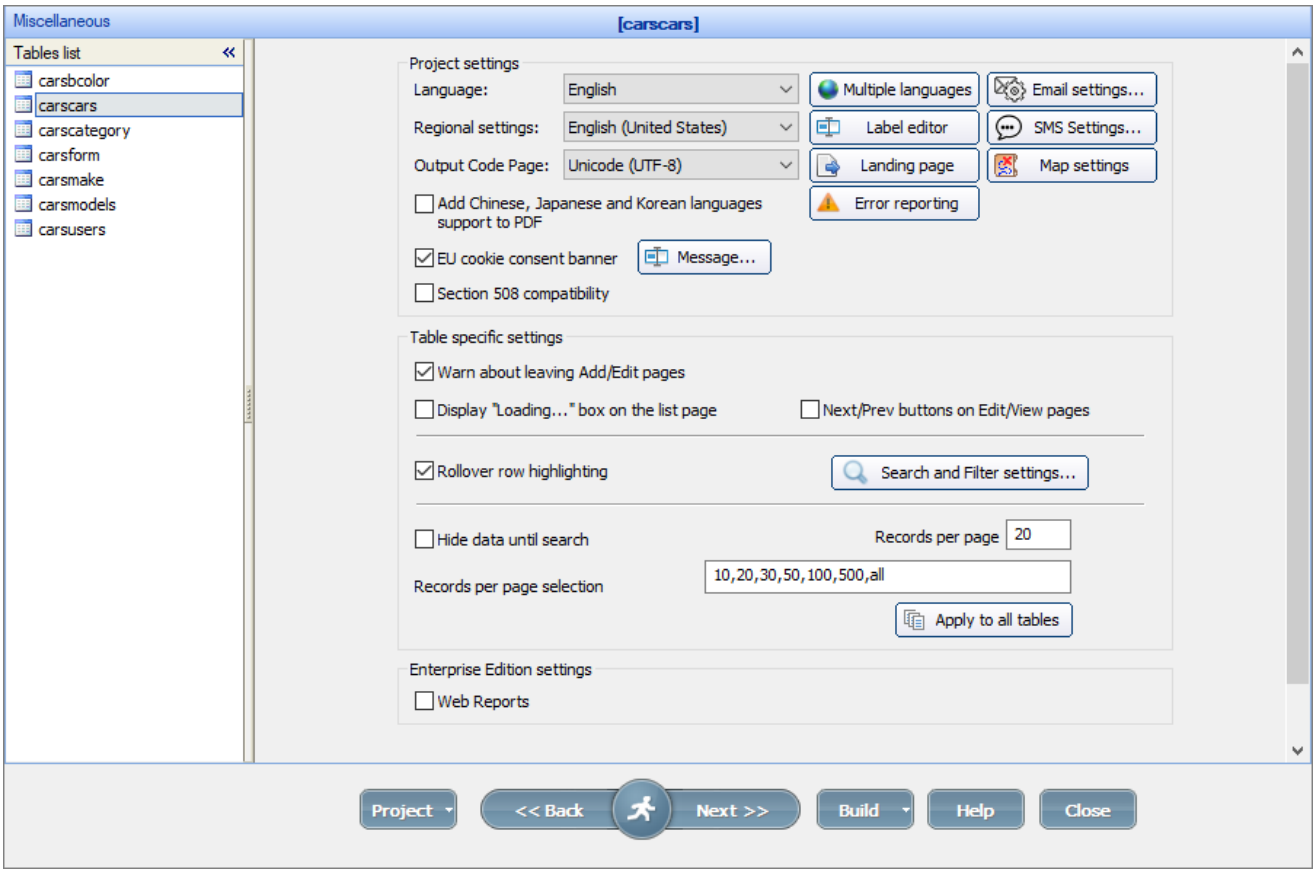
- **ALT+E** - inline edit current record;
- **ALT+S** - save current record;
- **ALT+C** - cancel inline edit;
- **ALT+F** - go to the advanced search;
- **ALT+M** - jump to the menu;
- **CTRL+left arrow** - previous page;
- **CTRL+right arrow** - next page.

You can also use the keyboard to navigate through the table links on the **List** page. Press **Arrow Up**, **Arrow Down**, **Tab**, **Shift+Tab** to switch between the elements.

When using special internet browser readers, the hotkeys reference link becomes available.

Table specific settings

The **Table specific settings** are miscellaneous settings that can vary from table to table. Use the **Tables list** on the left-side panel to switch between tables.



Warn about leaving Add/Edit pages

This checkbox allows displaying a warning message upon leaving the **Add**/**Edit** pages with unsaved changes.

Display "Loading..." box on the list page

This checkbox allows displaying a loading icon until the page loads completely. Use this option if you have large size images or other elements on the page.

Next/Prev buttons on Edit/View pages

This checkbox allows displaying the **Next**/**Prev** buttons on the **Edit**/**View** pages. These buttons allow switching between records without going back to the **List** page.

Rollover row highlighting

Select this checkbox to turn on the row highlighting feature. This feature makes the mouse pointer highlight the table row on the list page.

Search and Filter settings

This button adjusting the search and filter settings. For more information, see Choose fields: Search and Filter settings.

Hide data until search

This option hides all of the records on the page and displays a message: "Nothing to see. Run some search". The user needs to run a search to display the results.

Records per page

This option sets the initial number of records to be displayed when the users load the page for the first time.

> **Note**: to make this option work correctly, you must choose the key field for the selected table on the Choose pages screen.

Records per page selection

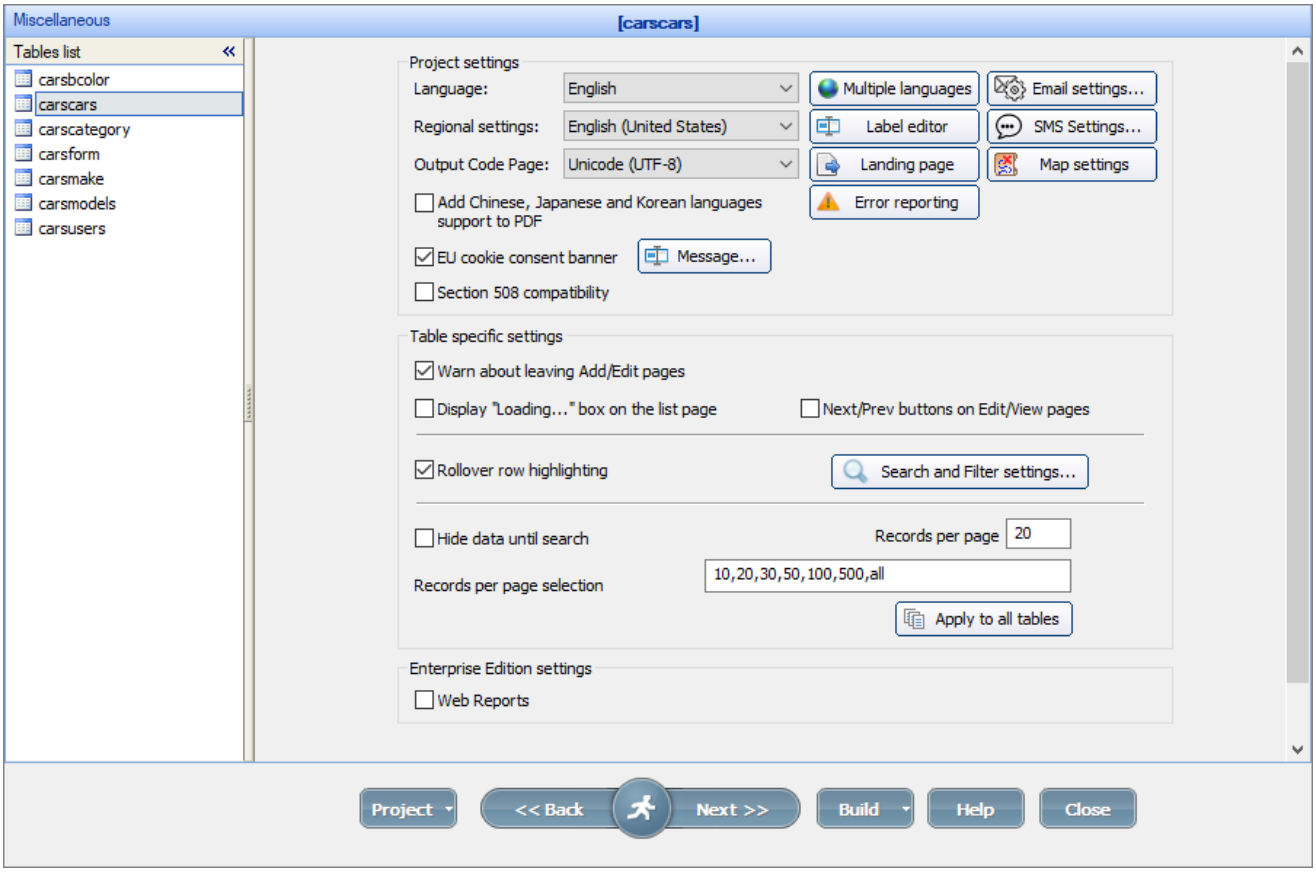
This field defines what options to show in pagination control. Type in *all* to enable the option to show all records.

Apply to all tables

This button applies the current **Hide data until search**, **Records per page**, and **Records per page selection** settings to every table in the project.

Enterprise Edition settings

These settings are available only in the Enterprise Edition of PHPRunner.



Web reports

Select the **Web reports** checkbox to enable the online report/chart builder. Click the **Administrator** button to specify the password to access the Web Reports and Charts admin area.

See also:

- Choose pages screen
- Choose fields screen
- Security screen

- Page Designer
- Two-factor authentication
- Insert map

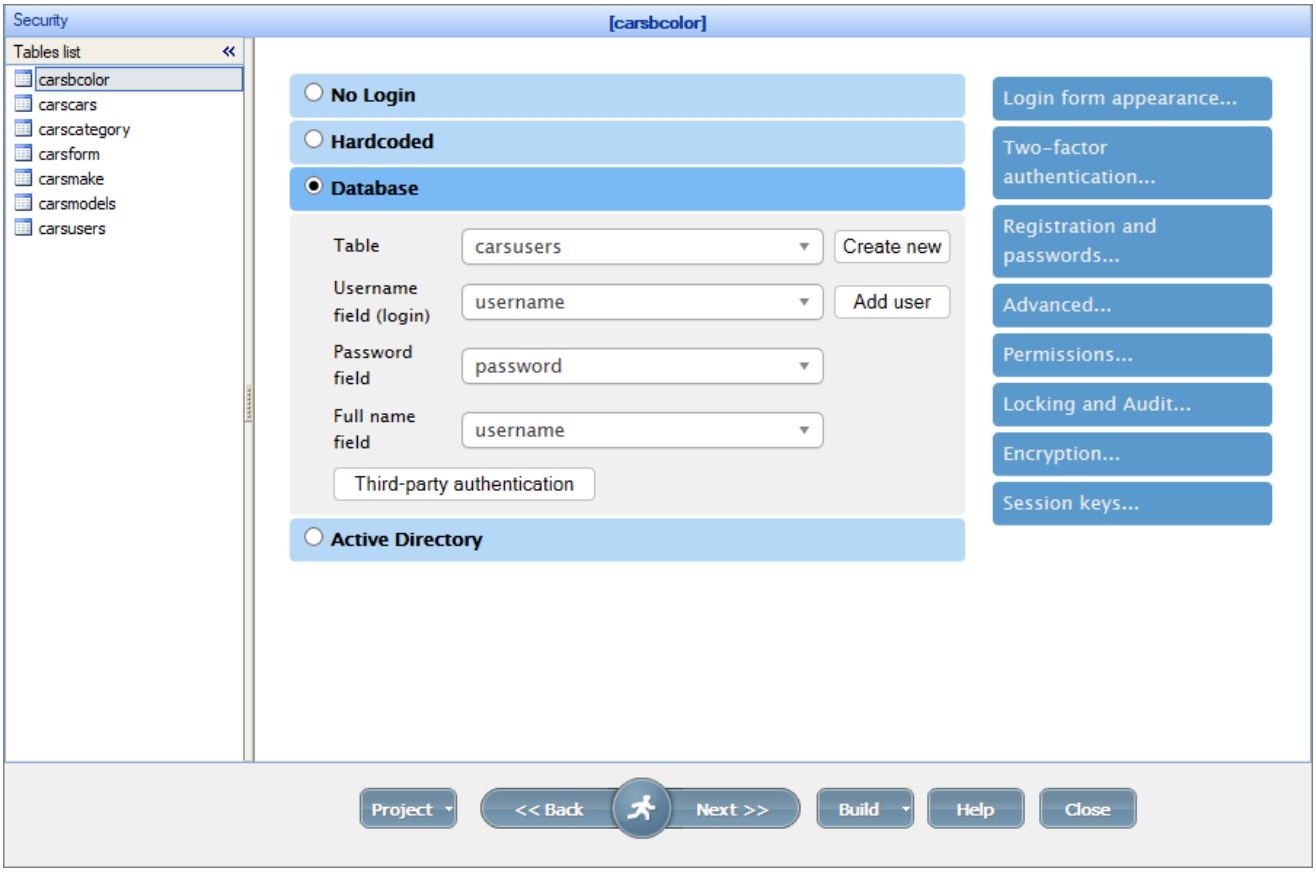
2.15 Security

2.15.1 Security screen

Quick jump

Security settings

The **Security screen** is a tool to help you restrict access to the database in the generated app.



You can choose one of the following options:

- **No Login** - there is no authentication. Everyone can access and edit the database.
- **Hardcoded** - set the only login/password combination which grants access to the database.
- **Database** - choose this method if you store username/password combinations in your database. In this case, you need to select the database table with the user data and choose the fields that store usernames, passwords, and full names. Full name content is displayed in the *Logged as* <> phrase after successful login.

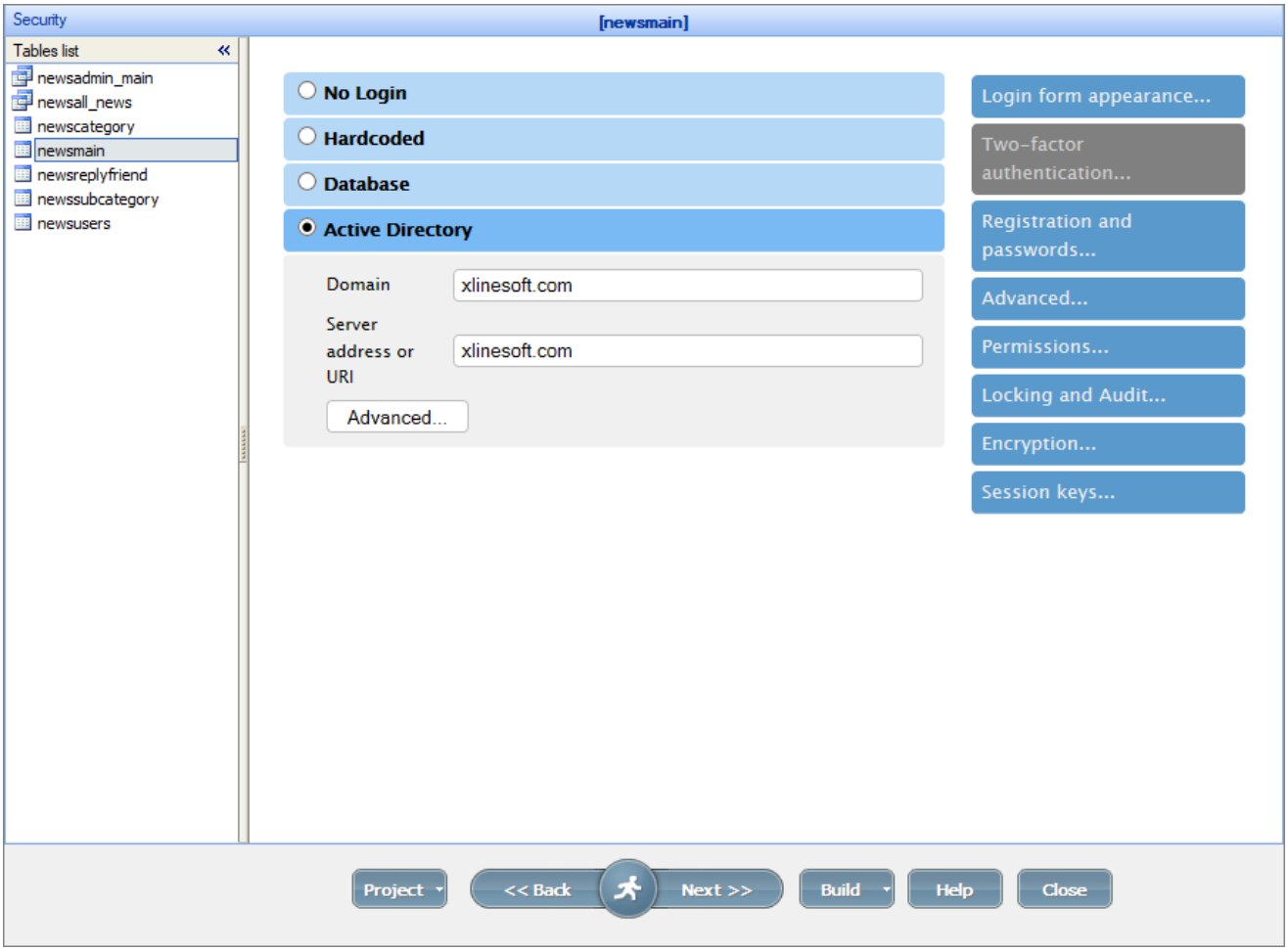
If you wish to display additional information in the *Logged as* <> phrase, use the AfterSuccessfulLogin event to add a custom code. For example:

```
$_SESSION["UserName"] = $data["FirstName"].' '.$data["LastName"]
```

You can create a new table to store user login info by clicking **Create new**, and add new users to the selected table by clicking **Add user**.

The **Database login** page has the **Third-party authentication** option. See Facebook connect and Sign in with Google to learn more.

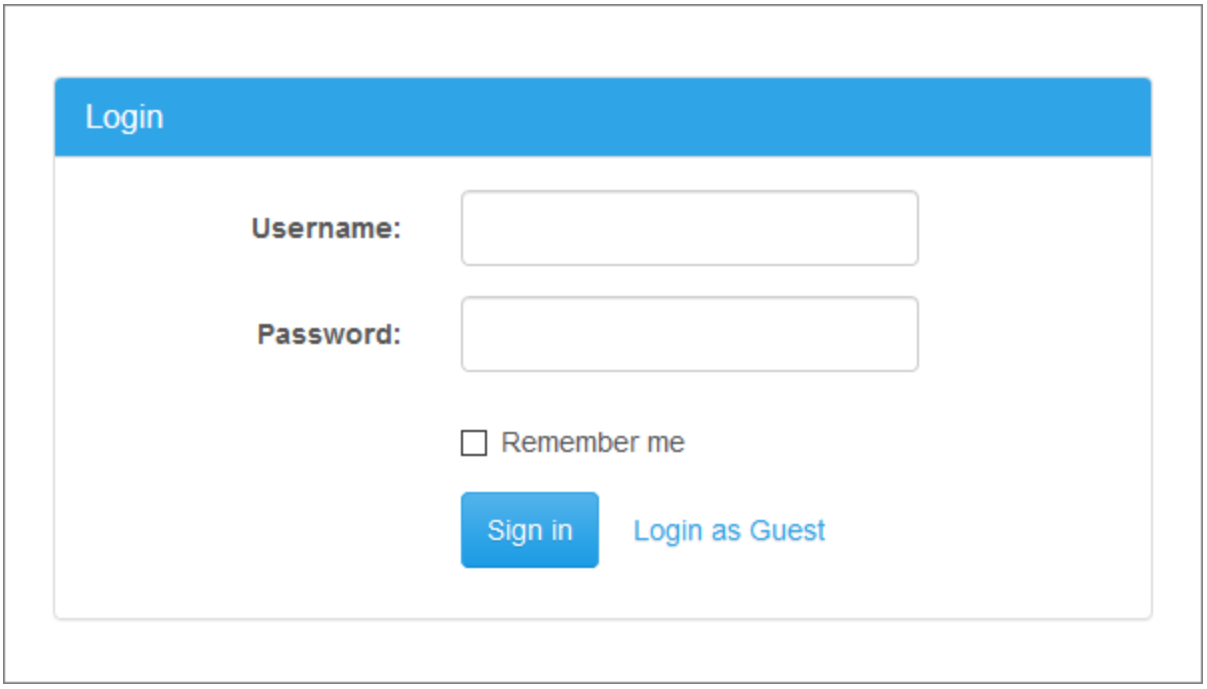
- **Active Directory** - this option allows working with the user data stored in the Active Directory. For more information, see Active Directory.



Note: the **Active Directory** option is only available in the **Enterprise edition**. See Editions comparison to learn more.

With active authentication, PHPRunner generates an additional PHP page called **Login**.

Here is how it looks like in the generated app:



Note: users can select the **Remember me** checkbox to store their session data in the cookies. This way, the user stays authenticated for as long as the cookies are relevant or active.

Additional options

The additional security options are located on the right:

- When using authentication, you can set the Login form appearance.
- Registration and passwords option allows you to create and set up the user registration, password reminder, and change password pages.

Note: if you need to customize the email templates that are sent when a new user is registered, use the Email templates option.

- Use Locking and Audit to set up record locking and user actions logging.
- Use Encryption to encrypt important data in the database.
- Use the Session keys option to enable a single logon for multiple projects.
- If the **Database** or **Active Directory** option is selected, you can set Advanced Security Settings and define Permissions.
- You can set up the Two-factor authentication with the **Database** option.

Scenarios for configuring security options

- "One or several persons with the same access rights have access to the site".

Use the **Hardcoded** option.

- "There is a single owner (administrator) with full access. Other users (guests) have read-only access to some pages/reports/charts".

Use the **Database** option, enable guest login in the Advanced security settings, configure the access for guests in the User group permissions.

- "There are many users with different access levels and administrators with full access".

Use the **Database** option, configure Advanced security settings if you need to restrict access for each table, configure User group permissions to assign table level permissions, configure **Admin group** for administrators.

- "All user account data is stored in the Active Directory".

Use the **Active Directory** option.

Security screen articles:

- Login form appearance
- Two-factor authentication
- Registration and passwords
- Advanced security settings
- User group permissions
- Dynamic permissions
- Audit and record locking

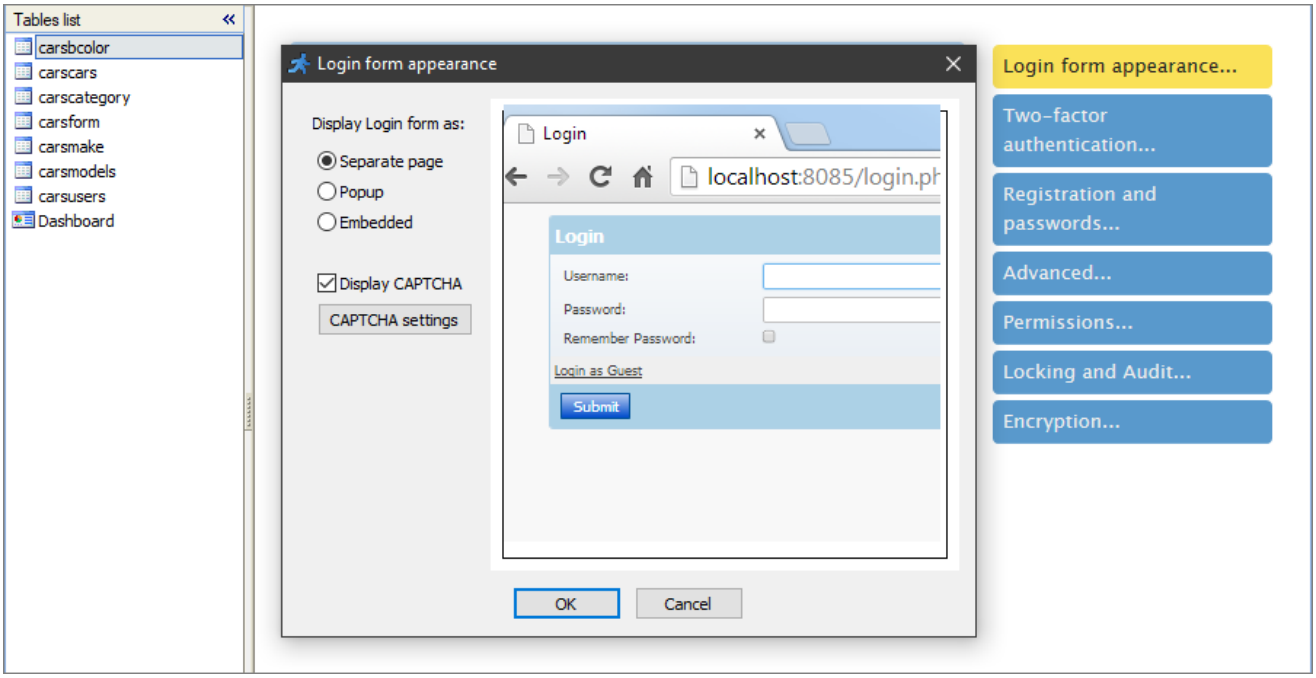
- Encryption
- Session keys
- Active Directory
- Facebook connect
- Sign in with Google
- CAPTCHA on authentication pages

See also:

- Security API
- Datasource tables screen
- Miscellaneous settings
- Page Designer
- Event editor

2.15.2 Login form appearance

Press the **Login form appearance** button on the **Security screen** to open a popup with the login appearance options.

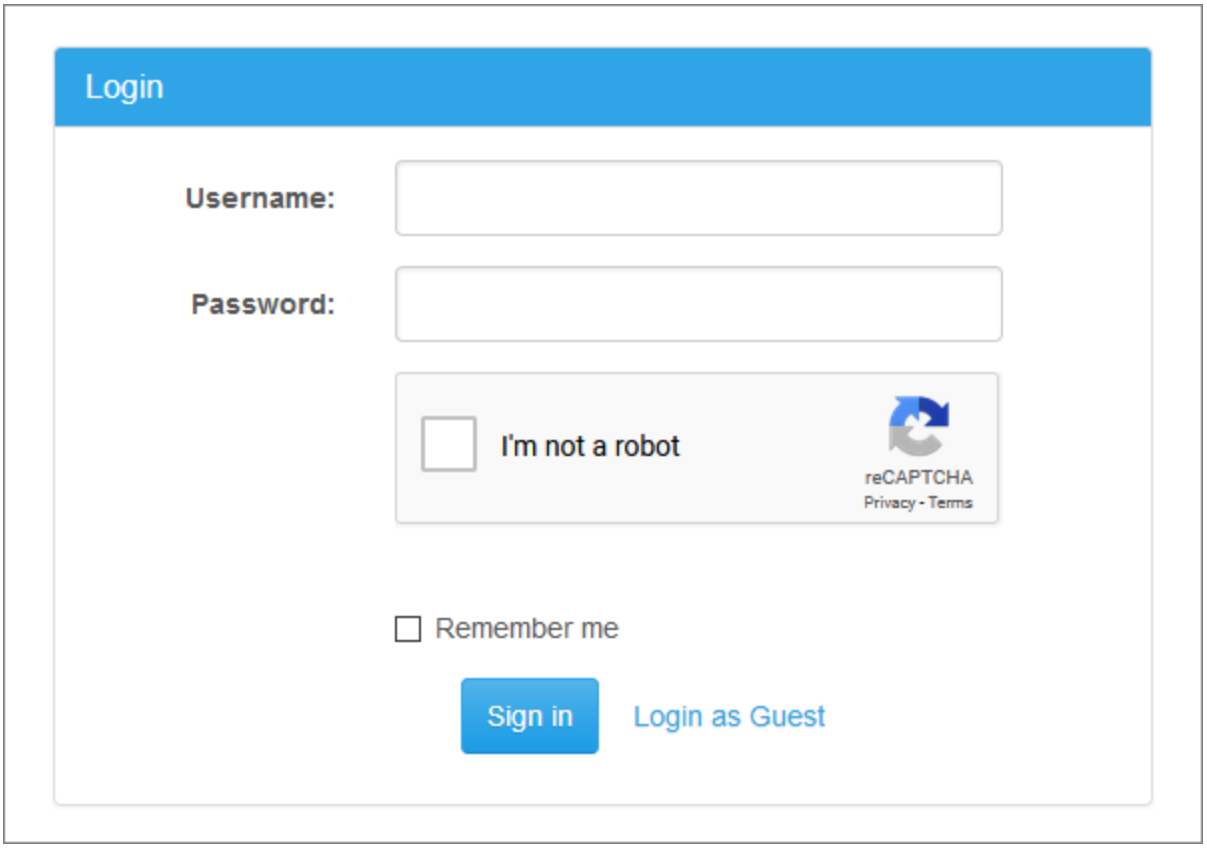


You can select between a **Separate page**, a P**opup** or an **Embedded** login form. **Separate page** and a **popup** form have the option to add CAPTCHA. For more information, see CAPTCHA on authentication pages.

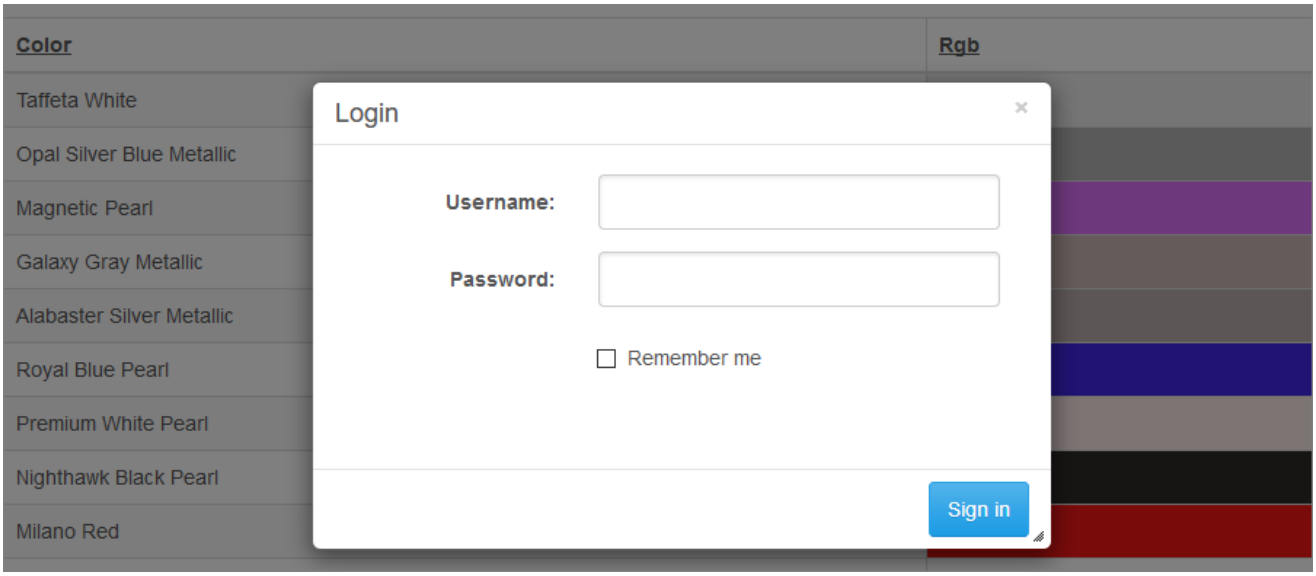
> **Note**: with the embedded form, users can enter their credentials right in the menu of the generated app. Make sure you have guest access enabled in the Permissions options.

Examples

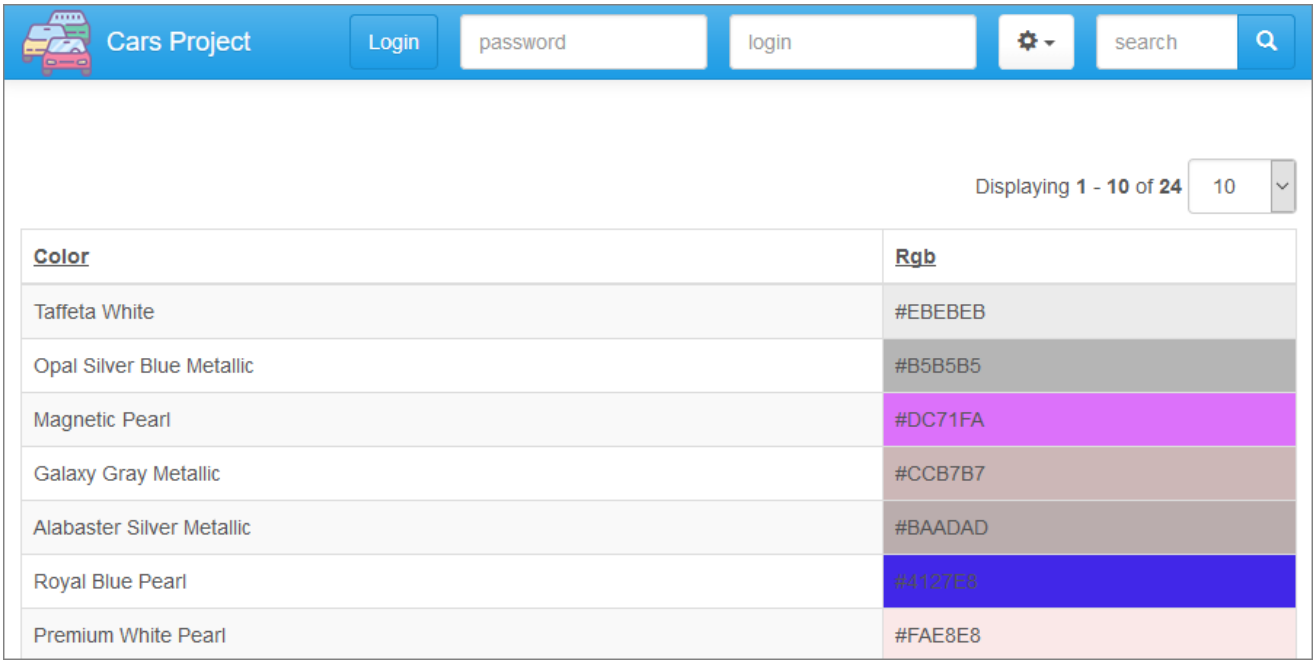
Separate login page with Google reCAPTCHA:



Login form in a popup:



Embedded login form:



Security screen articles:

- Security screen
- Two-factor authentication
- Registration and passwords
- Advanced security settings
- User group permissions
- Dynamic permissions
- Audit and record locking
- Encryption
- Session keys
- Active Directory
- Facebook connect
- Sign in with Google
- CAPTCHA on authentication pages

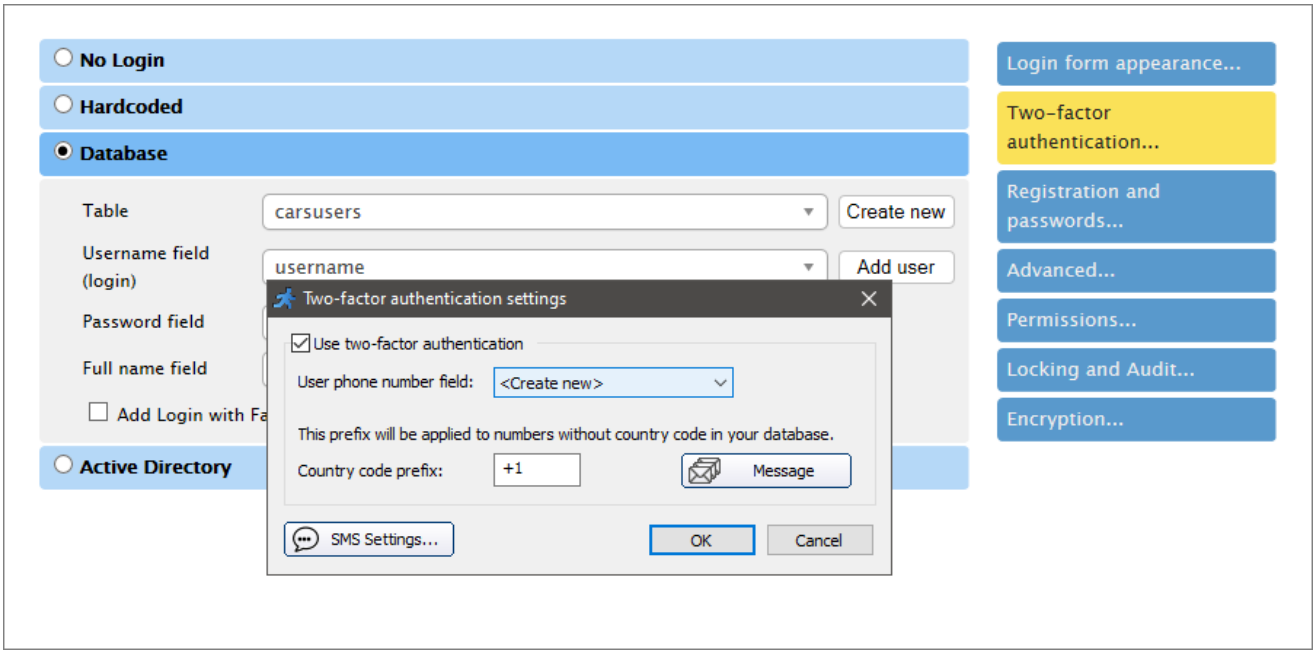
See also:

- Security API
- Datasource tables screen
- Miscellaneous settings
- Page Designer
- Event editor

2.15.3 Two-factor authentication

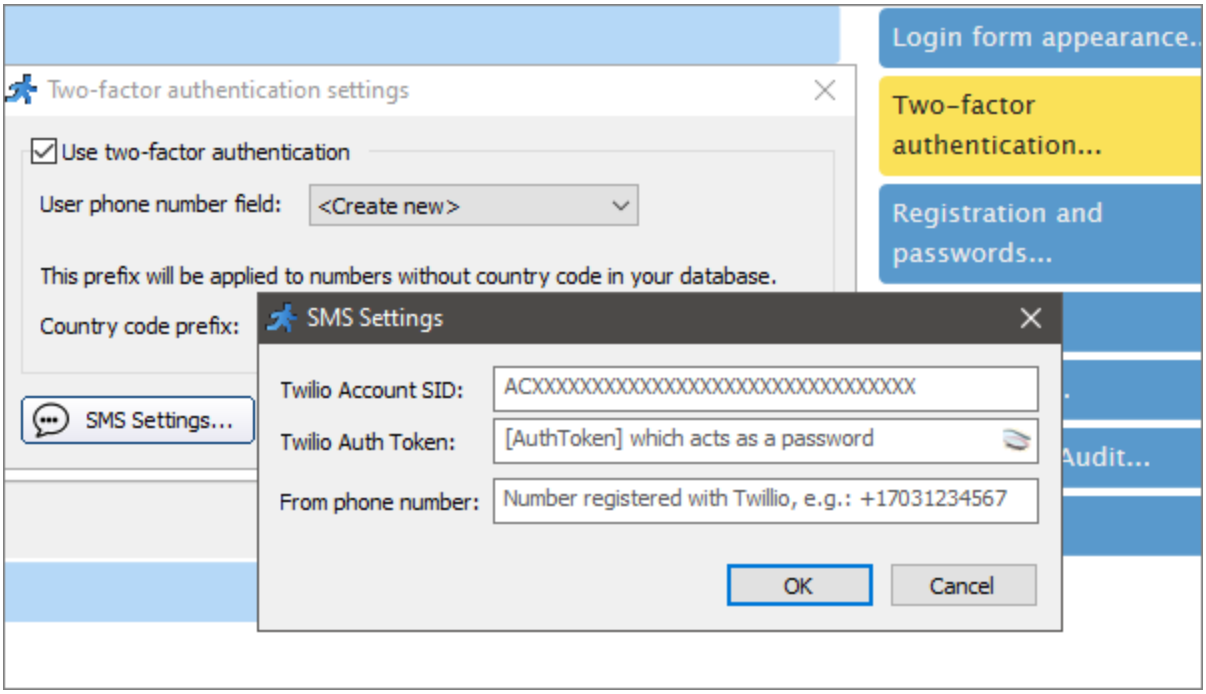
Two-factor authentication is a security mechanism that requires two types of credentials for authentication: login/password and an SMS verification code. It is designed to provide an additional layer of validation, minimizing security breaches.

Press the **Two-factor authentication** button on the **Security screen** to open a popup with the two-factor authentication options.



Enable the two-factor authentication and select the field containing the user's phone number. You can also set the country code prefix for the numbers without the country code in your database.

The next step is to set up the SMS Settings. In case you use the Twilio messaging platform, you need to fill the Twilio Account SID, Twilio Auth Token and the number registered with Twilio fields.



> **Note**: you may also set up the SMS settings at the Miscellaneous screen.

If you have chosen another one of the multiple SMS providers, you need to activate it first by making several changes in your source folder. See SMS settings to get instructions on how to do it.

After you finish configuring the SMS settings, you can customize the message by pressing the **Message** button.

Security screen articles:

- Security screen
- Login form appearance
- Registration and passwords

- Advanced security settings
- User group permissions
- Dynamic permissions
- Audit and record locking
- Encryption
- Session keys
- Active Directory
- Facebook connect
- Sign in with Google
- CAPTCHA on authentication pages

See also:

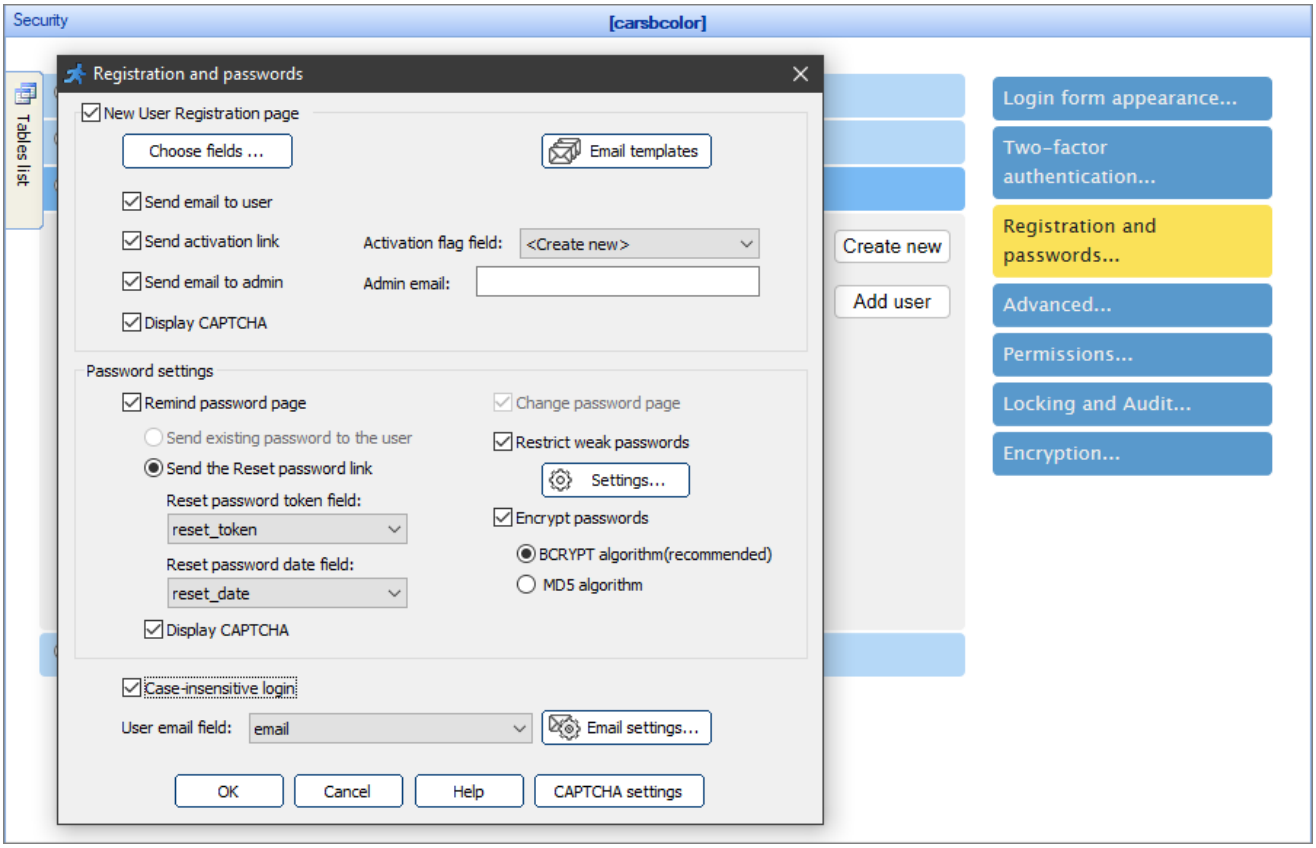
- Security API
- Datasource tables screen
- Miscellaneous settings
- Page Designer
- Event editor

2.15.4 Registration and passwords

Quick jump

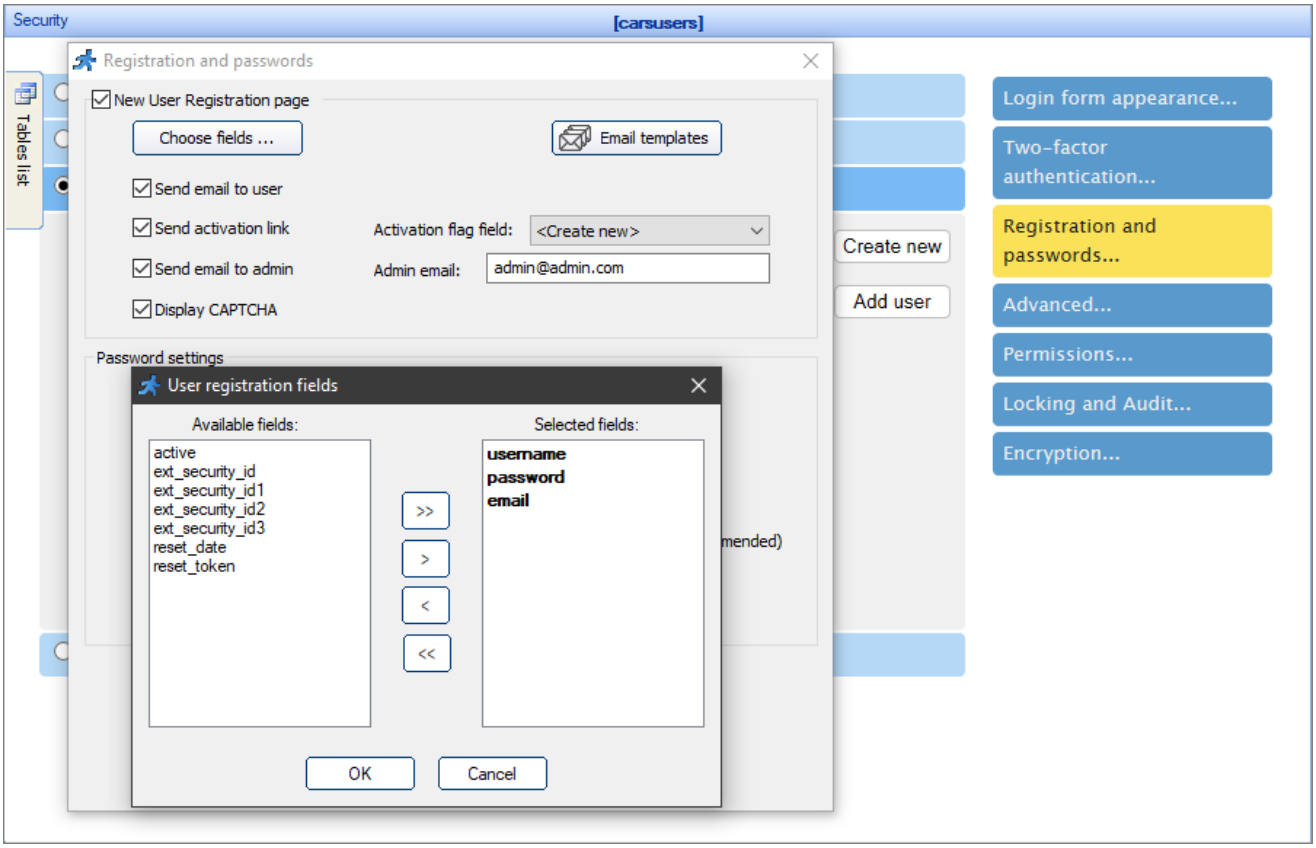
By default, the generated app doesn't have the user registration page. It uses either the hardcoded login/password or an already existing table with the login/password in the database. You can change that with the **Registration and passwords** option.

Press the **Registration and passwords** button on the **Security** screen to open a popup with the registration and passwords settings.



Registration page settings

To create a new user registration page, select the corresponding checkbox, and click **Choose fields** to select the fields that appear on the registration page.



Select the **Send email to user** checkbox to send an email to the users upon their registration. You can edit the user email template with the Email templates dialog.

> **Note**: do not forget to choose the *User email* field at the bottom of the popup.

With the **Send email to user** checkbox selected, you can also select the **Send activation link** checkbox to include the activation link into the registration email. The purpose of the activation link is to ensure that the user signs up with a real, active email address that they own. This helps in reducing the number of users with inactive or fake emails. The access is denied until the user opens the activation link in the browser.

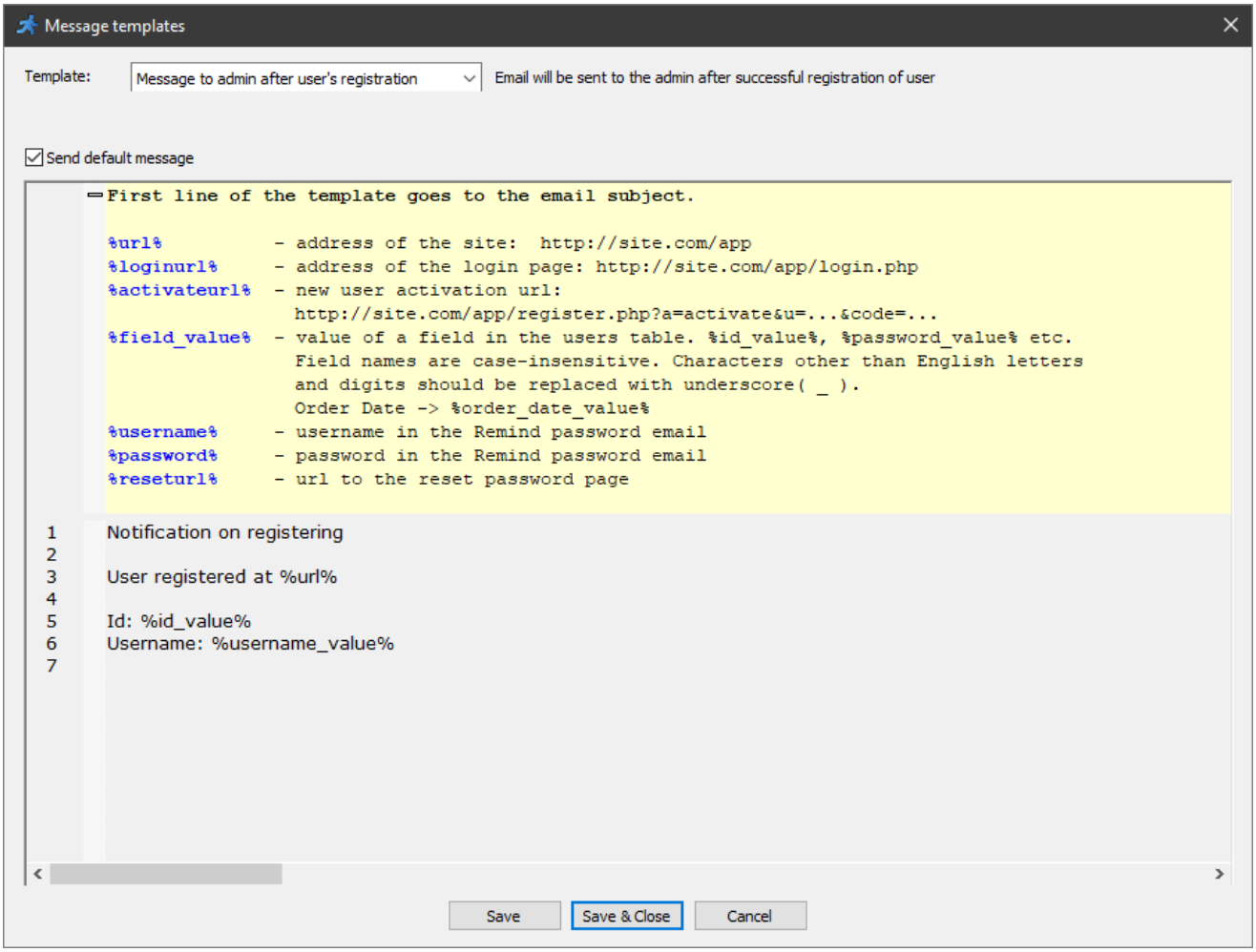
> **Note**: the activation link option requires having an activation flag field in the table. Select one of the existing fields or create a new one to store the activation flag. This field needs to be numeric (Number, INT or TINYINT).

You have the option to send an email to the admin when a new user registers on the site. Select the **Send email to admin** checkbox and fill the admin email field to do so. You can edit the admin email template with the Email templates dialog.

Select the **Display CAPTCHA** checkbox to display CAPTCHA on the **Registration** page. For more information, see CAPTCHA on authentication pages.

Email templates

You can customize the templates for the emails that are sent to the user/admin upon the new user registration with the **Email templates** dialog.



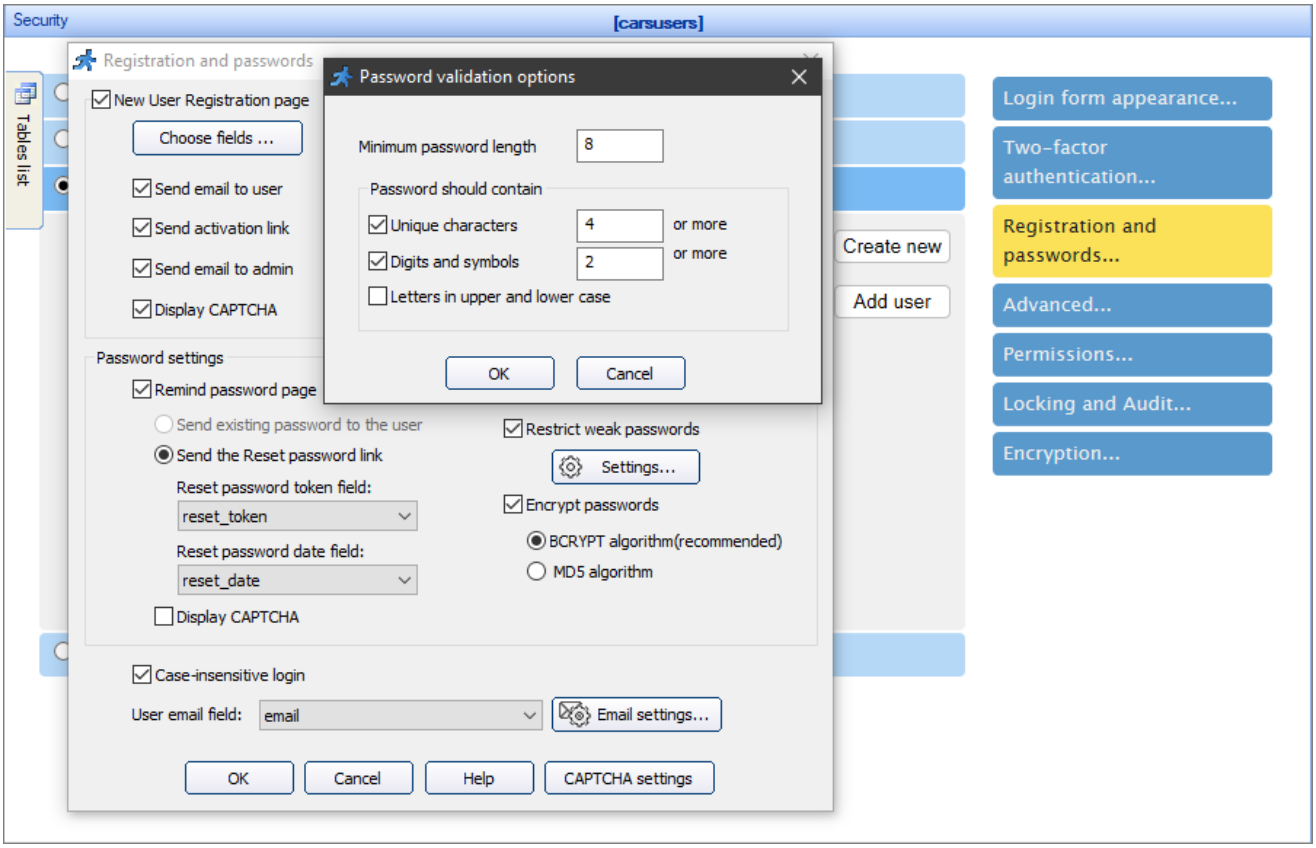
Select the template at the top of the window to view it. Deselect the **Send default message** checkbox to edit the template.

Note: you can use the text variables in the templates. The list of variables is available within the *Message templates* collapsible description.

Passwords settings

You can turn on the **Change password** and **Remind password** pages with the respective checkboxes.

You have the option to **Restrict weak passwords**. Click **Settings** to set the minimum password length, the number of unique characters, digits and symbols, or to accept passwords with both upper and lower case letters only.



Select the **Encrypt passwords** checkbox to protect the passwords in the database. Choose between the BCRYPT algorithm (the industry standard), or the MD5 algorithm.

> **Note**: BCRYPT requires PHP 5.5 or better.

You can also hash your passwords manually using Events.

For instance, you want to provide the admin with direct access to the *login* table. Add the following code to the BeforeAdd/BeforeEdit events of the *login* table:

For BCRYPT:

```
$values["password"] = getPasswordHash( $values["password"] );
```

For MD5:

```
$values["password"] = md5( $values["password"] );
```

The **Remind password** page has the option to send the existing password to the user or to send the **Reset password link**. With the **Send the Reset password link**, you need to choose or create a new *Reset password* token and *Reset password* date fields.

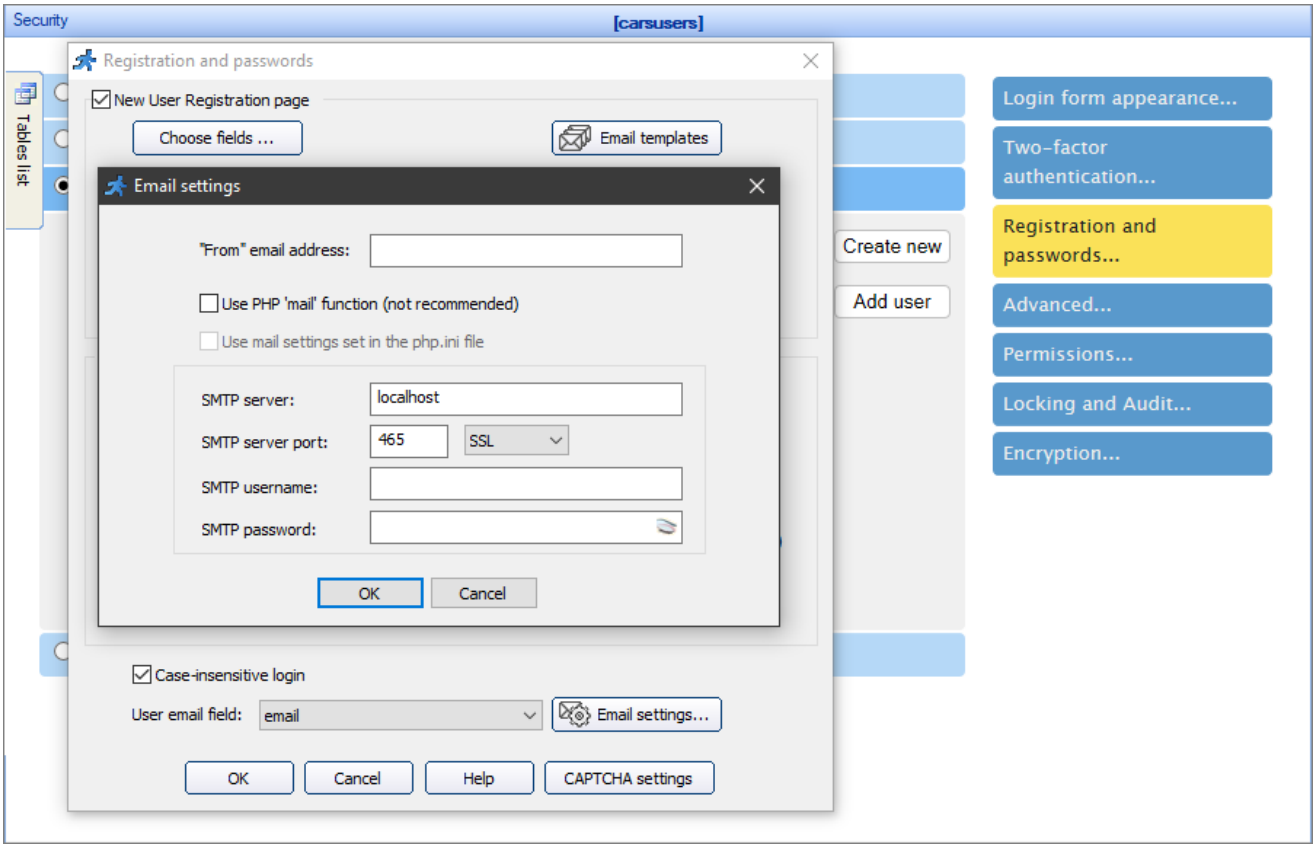
> **Note**: the **Send existing password to the user** option is unavailable when you have the **Encrypt passwords option** enabled.

Select the **Display CAPTCHA** checkbox to display CAPTCHA on the **Remind password** page. For more information, see CAPTCHA on authentication pages.

Additional settings

Select the **Case-insensitive login** checkbox to make the site, for instance, consider *'USER'* and *'user'* the same logins.

With **Email settings**, you can enter the email from which to send emails to the users, and define the settings of the custom mail server, if you do not use the built-in mail server. See Miscellaneous settings to learn more.



Security screen articles:

- Security screen
- Login form appearance
- Two-factor authentication
- Advanced security settings
- User group permissions
- Dynamic permissions
- Audit and record locking
- Encryption
- Session keys
- Active Directory
- Facebook connect
- Sign in with Google

- CAPTCHA on authentication pages

See also:

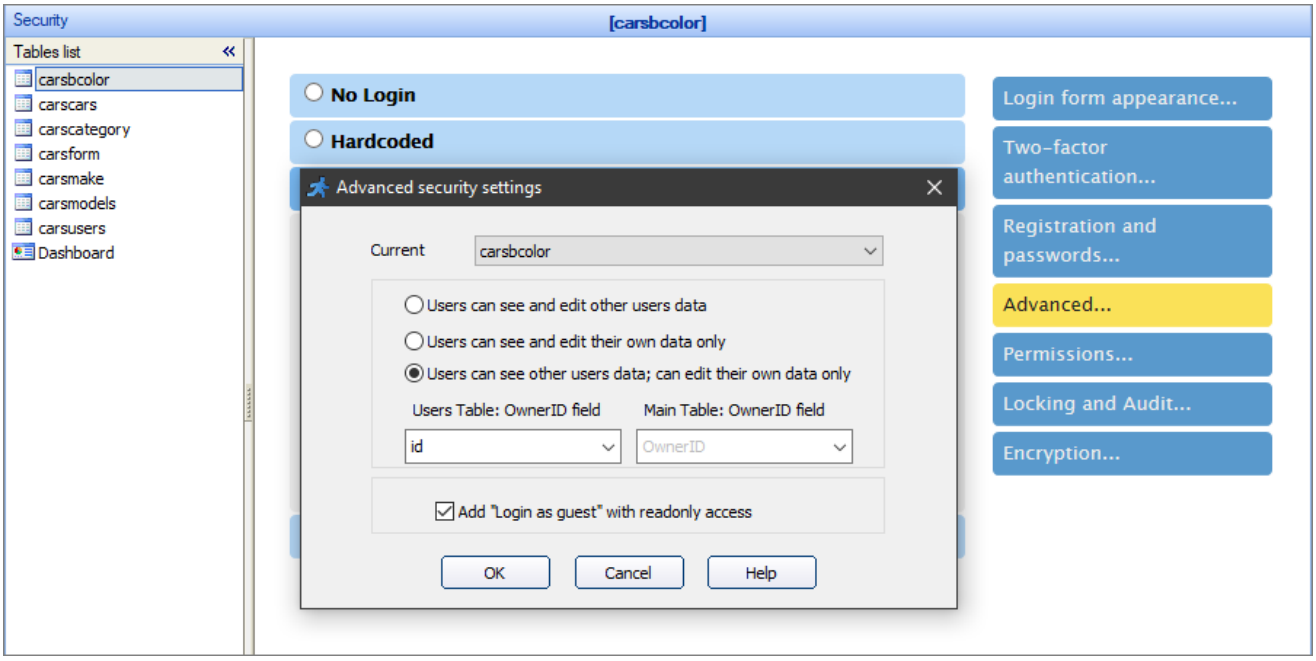
- Security API
- Datasource tables screen
- Miscellaneous settings
- Page Designer
- Event editor

2.15.5 Advanced security settings

Advanced options

Press the **Advanced** button on the **Security** screen to open the popup with advanced security settings.

The **Advanced security settings** popup gives you additional options that affect accessing, viewing, and editing the database.



> **Note**: each table in the database can be set up individually, select it in the dropdown at the top of the popup window.

The options are:

Users can see and edit other users data

This is the default option. Every authorized user can view, edit, and delete other users' data.

Users can see and edit their own data only

This option restricts authorized users so that they can view and edit only the records they created.

This option requires you to select the *OwnerID* fields in the *Users* table and the current table. The field in the current table stores the ID (when you store usernames/passwords in the Database) or the username (when you select the Active Directory authentication) of the user who created the record.

> **Note**: the field to store the ID needs to be an *Int* type.

If you don't have the appropriate field to store the ID/username, you can create it by modifying the table. See Datasource tables to learn more.

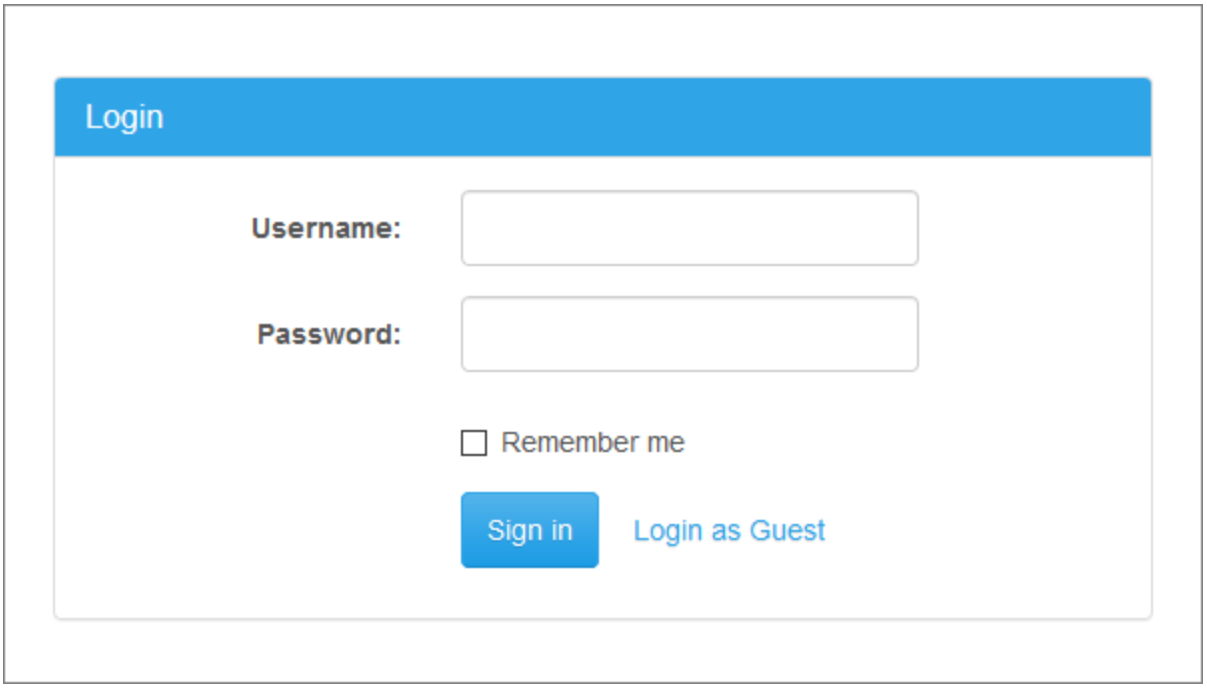
Users can see other users data, can edit their own data only

This option is similar to the previous one. However, the authorized users can view the other users' data, but cannot edit or delete it.

Login as guest

This option adds read-only guest access to the generated app. Guests cannot edit, delete or add new records to the database.

With this option selected, the **Login as Guest** link appears on the **Login** screen:



Note: See User group permission to learn more about guest permissions.

Security screen articles:

- Security screen
- Login form appearance
- Two-factor authentication
- Registration and passwords
- User group permissions
- Dynamic permissions
- Audit and record locking
- Encryption
- Session keys
- Active Directory
- Facebook connect
- Sign in with Google
- CAPTCHA on authentication pages

See also:

- Security API
- Datasource tables screen
- Miscellaneous settings
- Page Designer
- Event editor

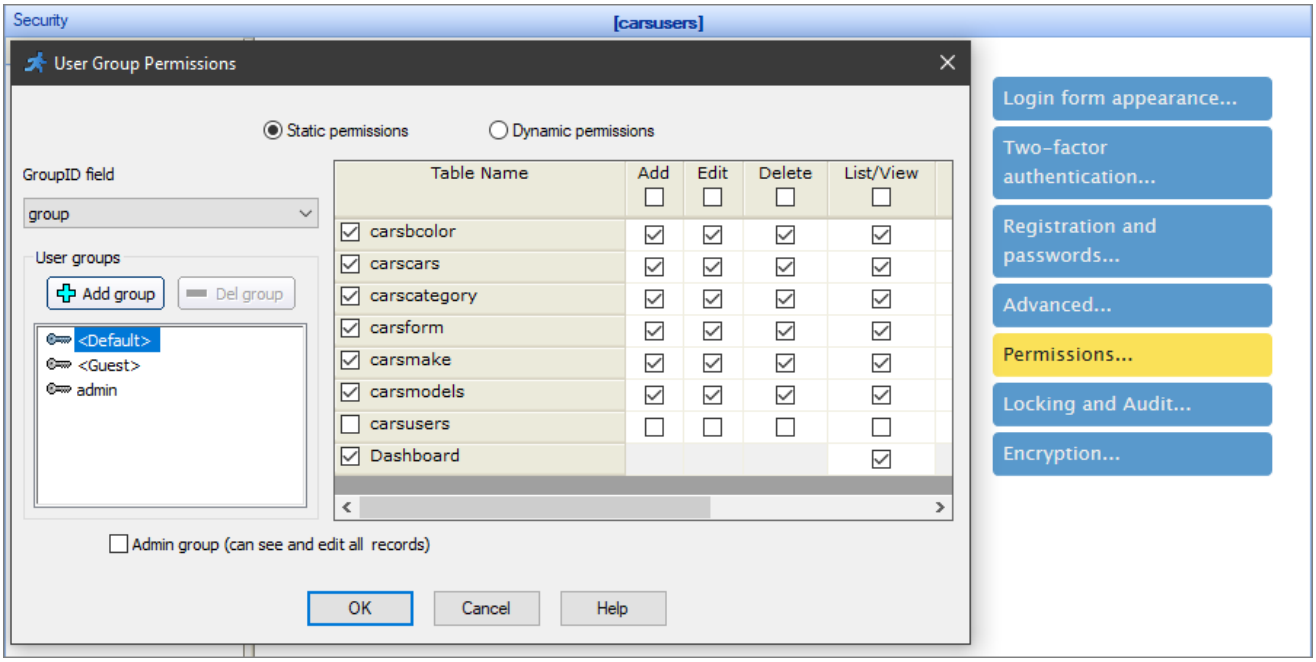
2.15.6 User group permissions

Quick jump

Permissions

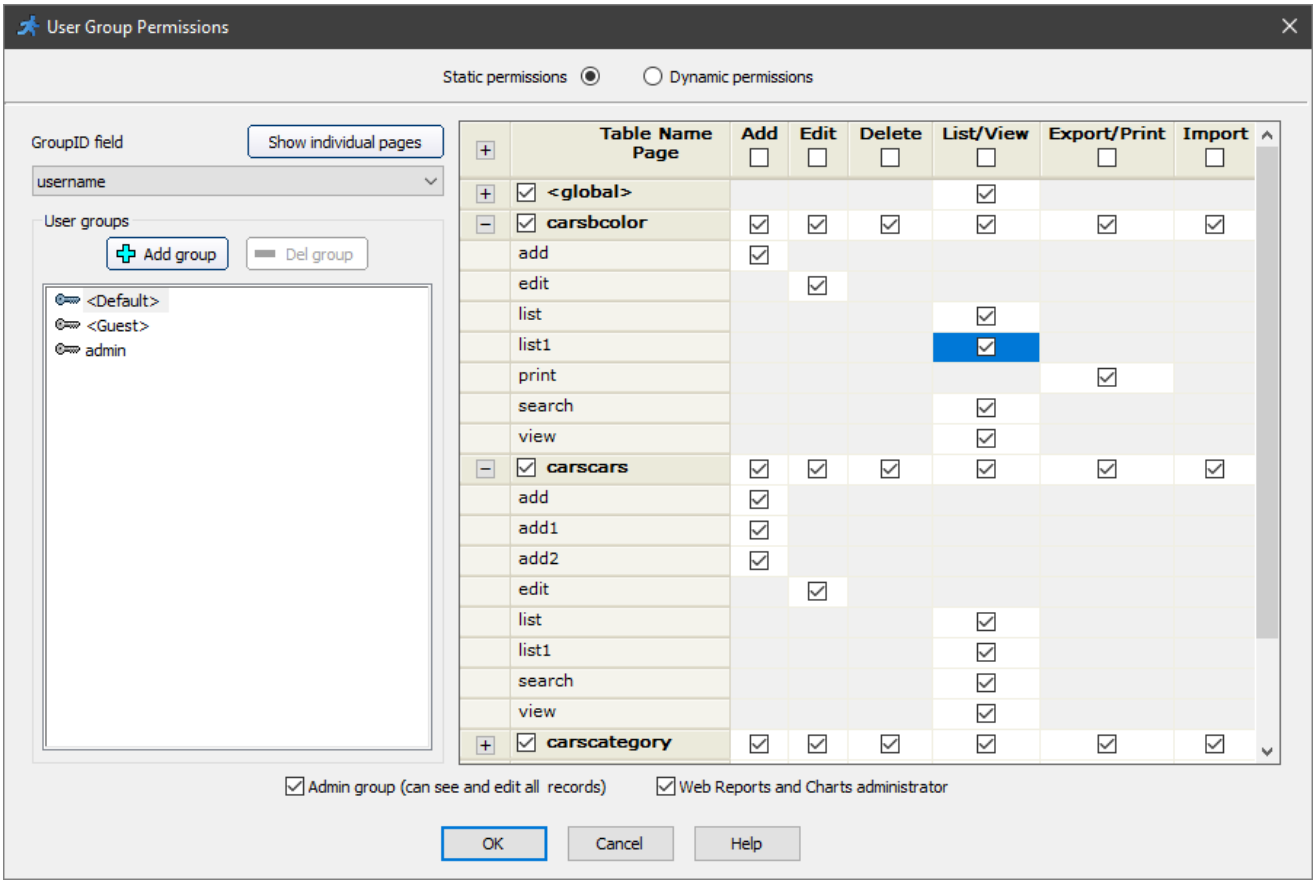
Press the **Permissions** button on the **Security** screen to open the popup with user group permissions.

The **User group permissions** popup allows you to assign table-level access to groups of users.



For example, you can set up the permissions, so that users by default cannot edit/delete the *Cars* table, guests cannot view any tables, other than *Cars*, and Admins have access to every page of the table.

Starting with PHPRunner version 10.3, you can set individual permissions for table pages and additional pages. This option works for both Static and Dynamic permissions.



There are two ways to define the user group permissions:

Static permissions

You can edit the permissions in PHPRunner directly before building the project.

This option works best when you are the only person who can edit the permissions. Keep in mind that you need to rebuild and publish the project each time you make the changes.

Dynamic permissions

Administrators can edit the permissions online, in the admin panel of the generated app.

This option works best when you have multiple administrators.

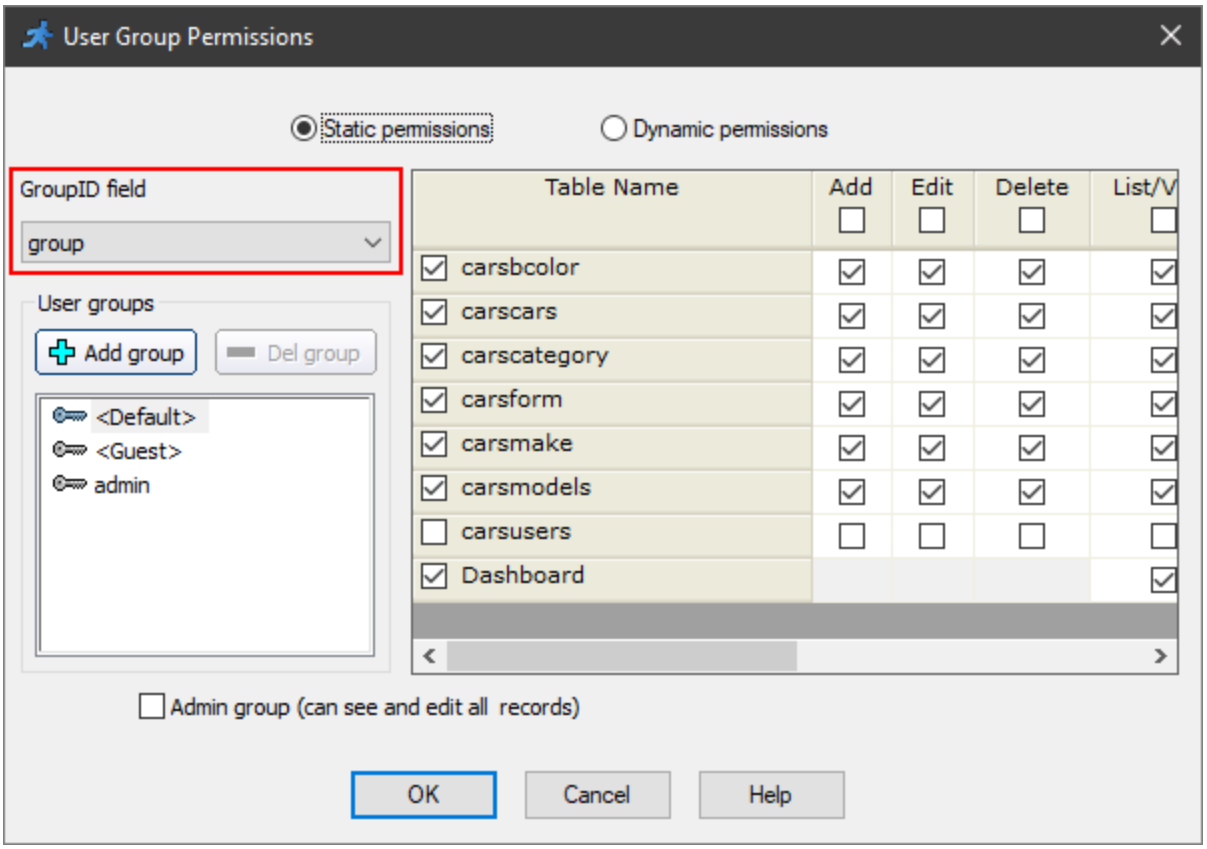
> **Note**: when you select **Dynamic permissions**, PHPRunner creates new tables in the database to store the user group permissions settings. You can set the prefix for the new names of the tables.

See Dynamic permissions to learn more.

Static permissions

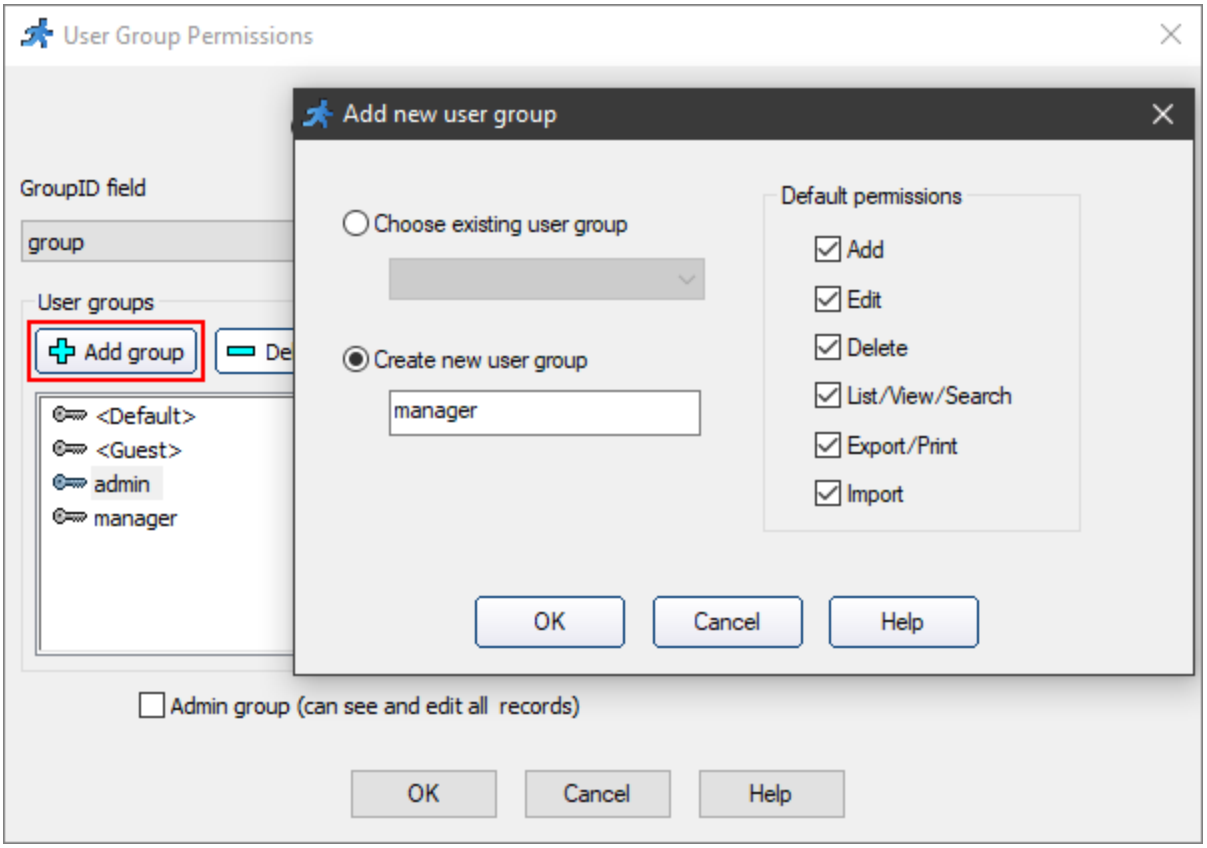
You can create user groups and assign the permissions for each group on this screen.

First, choose the *GroupID* field that stores the group name or ID.



Then you can add a new group. To do this, click the **Add group** button. A dialog window appears where you can choose the existing user group, based on the values in the GroupID field of the *Users* table, or create a new group to add the users later.

Define the initial permissions of the selected group with the checkboxes on the right side of the window.



To delete the selected group, click the **Del group** button. To overwrite an existing group, click the **Add group** button and choose the existing GroupID value.

You can also set up the default group - a default set of permissions for authorized users. Every user without a group is automatically assigned to the Default group.

If you enabled the guest access, the Guest group appears in the **Permissions** popup. You can set up the permissions for any user that logs in as a guest.

If you have selected any option, other than the default, in the Advanced security settings, a new checkbox called **Admin group** appears in the **Permissions** popup. Select it to make the current group an admin group.

Security screen articles:

- Security screen
- Login form appearance
- Two-factor authentication
- Registration and passwords
- Advanced security settings
- Dynamic permissions
- Audit and record locking
- Encryption
- Session keys
- Active Directory
- Facebook connect
- Sign in with Google
- CAPTCHA on authentication pages

See also:

- Security API
- Datasource tables screen
- Miscellaneous settings
- Page Designer
- Event editor

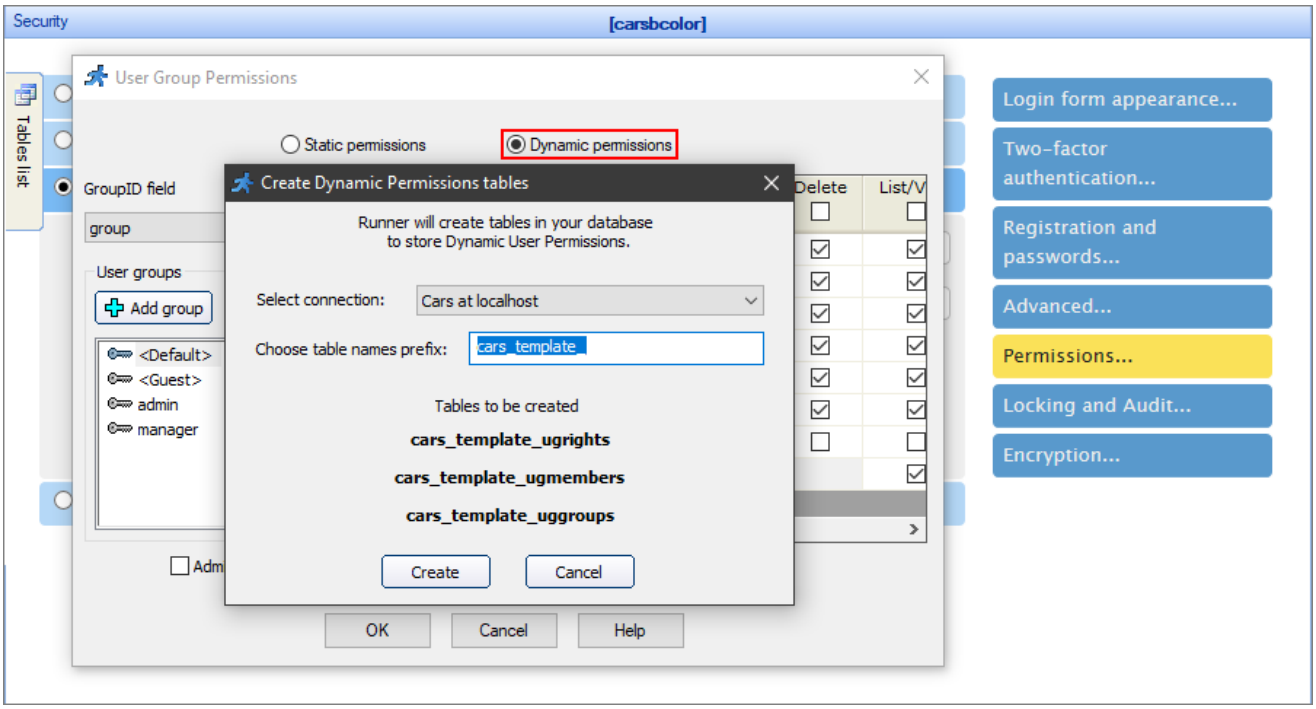
2.15.7 Dynamic permissions

Quick jump

What are dynamic permissions?

Dynamic permissions are a User group permissions mode, that allows administrators to update the group permissions online, in the admin area of the application.

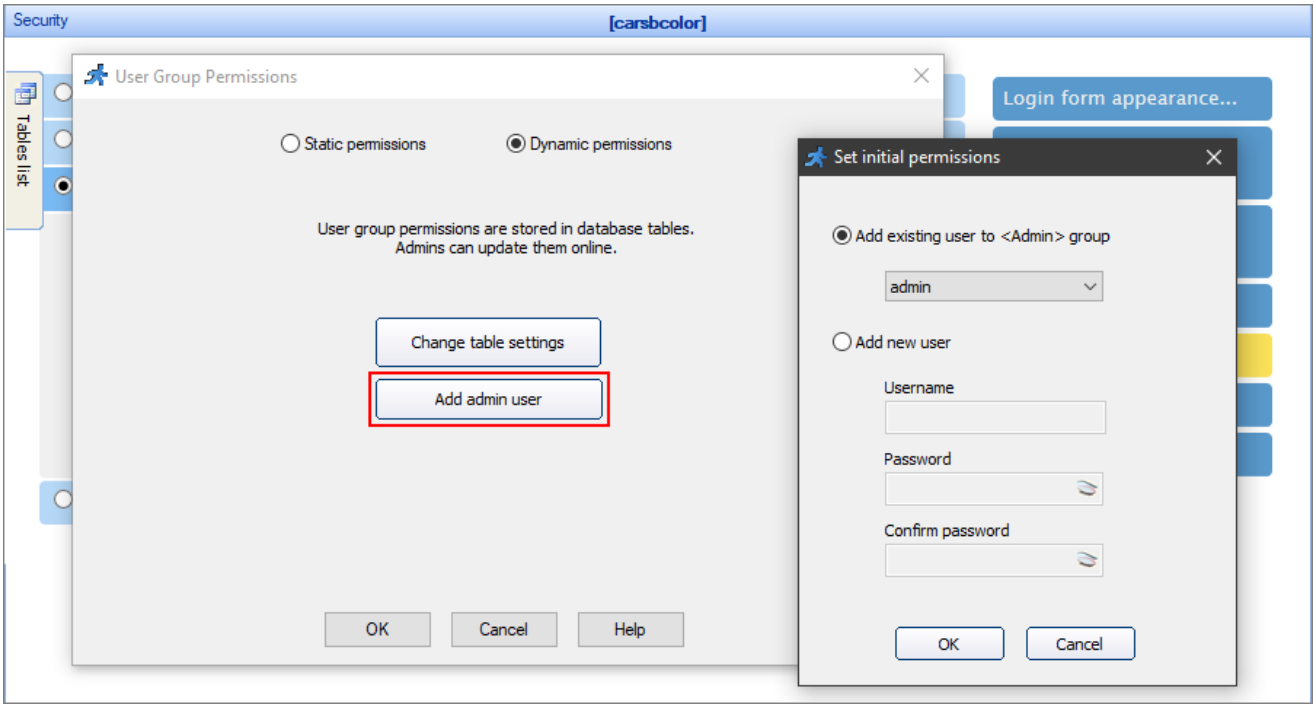
Press the **Permissions** button on the **Security** screen to open the **User group permissions** popup, then click **Dynamic permissions**.



Dynamic permissions require three database tables to store the groups, their permissions, and the group members. You can select the existing tables, or PHPRunner can create these tables automatically, you only have to choose the table names prefix.

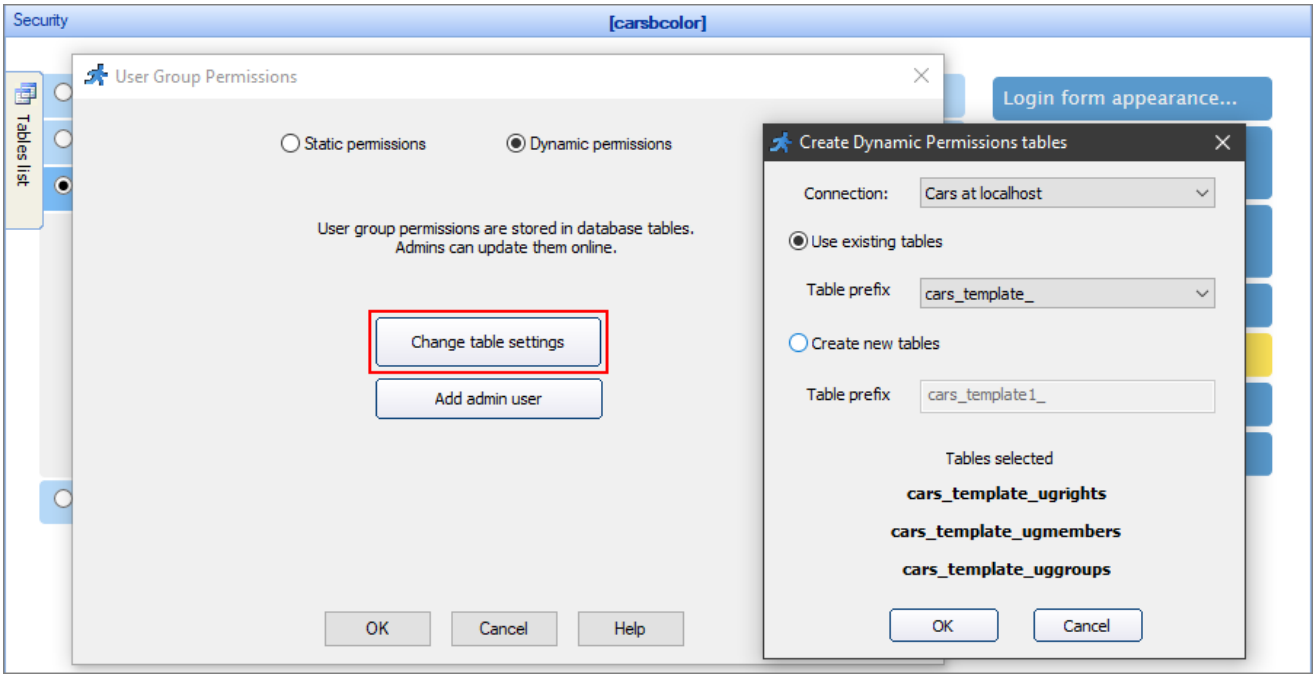
> **Note**: if you have several projects that share the same database, make sure to avoid permission conflicts by adding unique prefixes to the table names.

The next step is to select the administrators - the users with access to the admin area of the application. Click the **Add admin user** button to open the dialog. You can add an existing user or create a new one.



Note: you can have several administrators - add the users one by one using the **Add admin user** dialog.

Click **Change table settings** to change the database connection, select another existing table name prefix, or create a new set of tables.

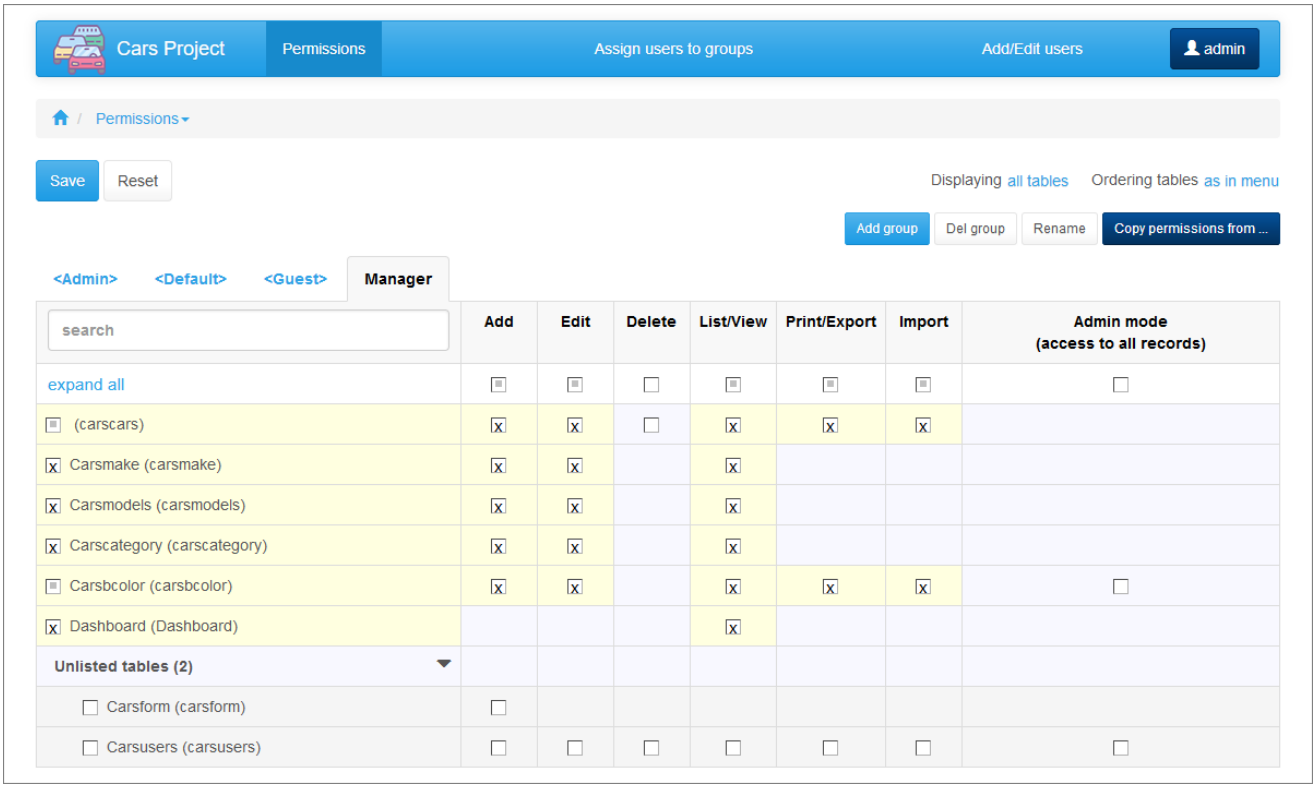


Admin area of the generated app

The admin area has several pages, where you can edit the group permissions and add new groups, assign users to groups, add or edit the users.

To access the admin area, log in as an administrator, click your profile, and select **Admin Area**.

Permissions



You can edit the permissions for the existing groups on the **Permissions** page. To switch between the groups, press the name tabs.

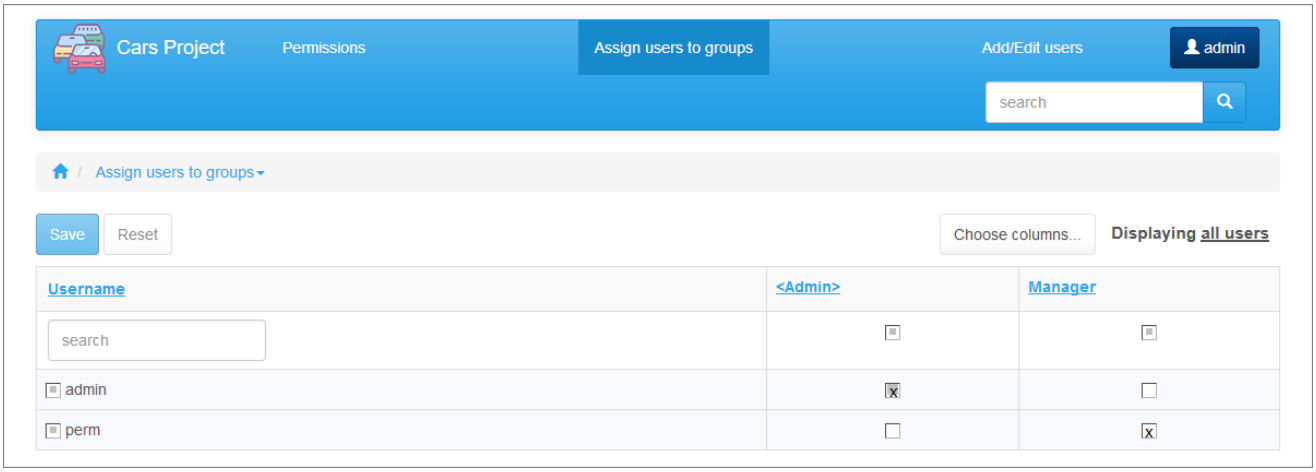
Add, **Delete**, or **Rename** the groups using the respective buttons. Click the **Copy permissions** from button to copy the permissions from another user group.

There are additional instruments to help you adjust the list of tables:

- Use the live search to narrow down the list of the tables.
- Display all tables or modified only.
- Order the tables alphabetically or as in the menu.

> **Note**: starting with PHPRunner version 10.3, you can set individual permissions for table pages and additional pages.

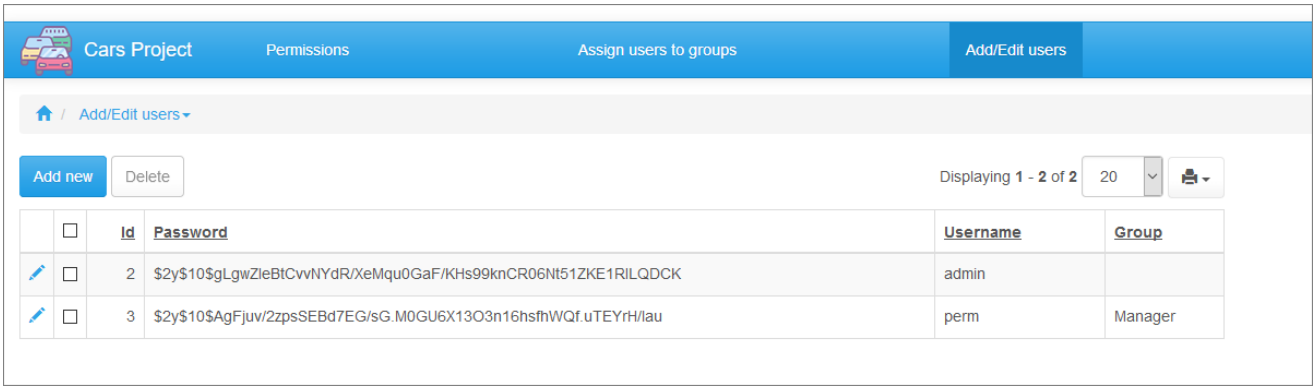
Assign users to groups



You can assign existing users to the added groups on the **Assign users to groups** page.

Choose the columns to show and select whether to display all users or the modified users. Use the live search to narrow down the list of the users.

Add/Edit users



You can add, edit, or delete the existing users on the **Add/Edit users** page.

> **Note**: user passwords are encrypted. See Encryption to learn more.

Security screen articles:

- Security screen
- Login form appearance
- Two-factor authentication
- Registration and passwords
- Advanced security settings
- User group permissions
- Audit and record locking
- Encryption
- Session keys
- Active Directory
- Facebook connect
- Sign in with Google
- CAPTCHA on authentication pages

See also:

- Security API
- Datasource tables screen
- Miscellaneous settings
- Page Designer
- Event editor

2.15.8 Audit and record locking

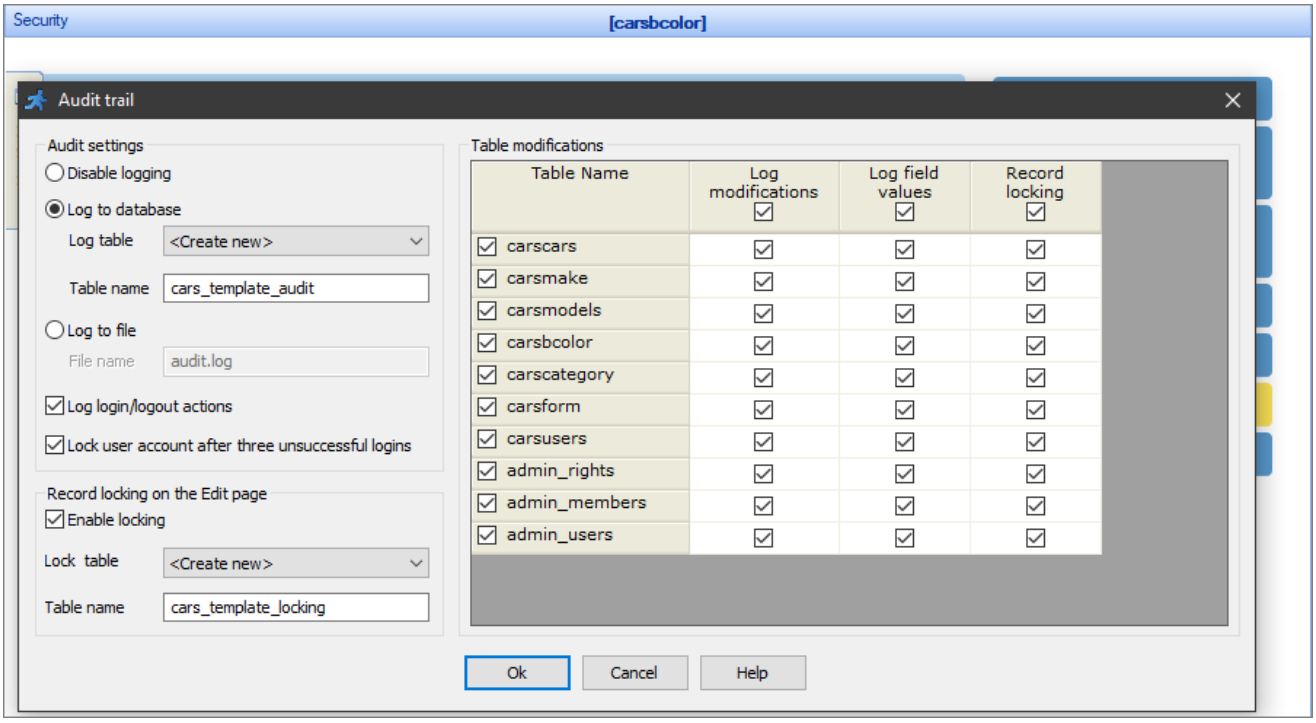
Quick jump

Auto unlocking

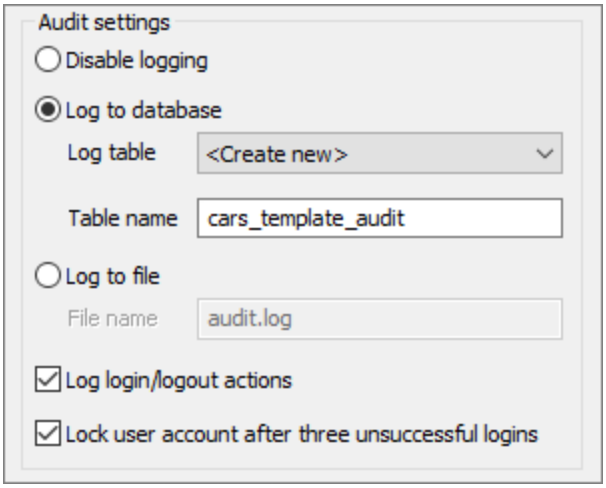
Locking and Audit

The **Audit trail** window allows you to configure logging options and record locking on the Edit page.

Press the **Locking and Audit** button on the **Security** screen to open the **Audit trail** popup.



Audit settings



The **Audit settings** is a section dedicated to logging.

A log is a record of user actions like login/logout/adding/editing/deleting records. The logging is disabled by default.

To enable logging, select one of the options:

- **Log to database**. Select a table to store the log or create a new one. You can set the name for the new table.
- **Log to file**. Enter the log *File name*, for example, *audit.log*. By default, the log files are saved to the root folder of the project. You can enter the *File name* as *<folder>/<file name>* to change the location of the log file. For example, if you set the *File name* to *log/audit.log*, the log files will be saved to the *%root folder of the project%/log*.

> **Note**: PHPRunner creates new log files each day, so the log file names look like this: *%file_name%_yyyymmdd.log*.

You can also select the **Log login/logout actions** and **Lock user account after three unsuccessful logins** checkboxes to enable the corresponding actions.

> **Note**: the **Lock user account after three unsuccessful logins** option is available only if you selected **Log to database**.

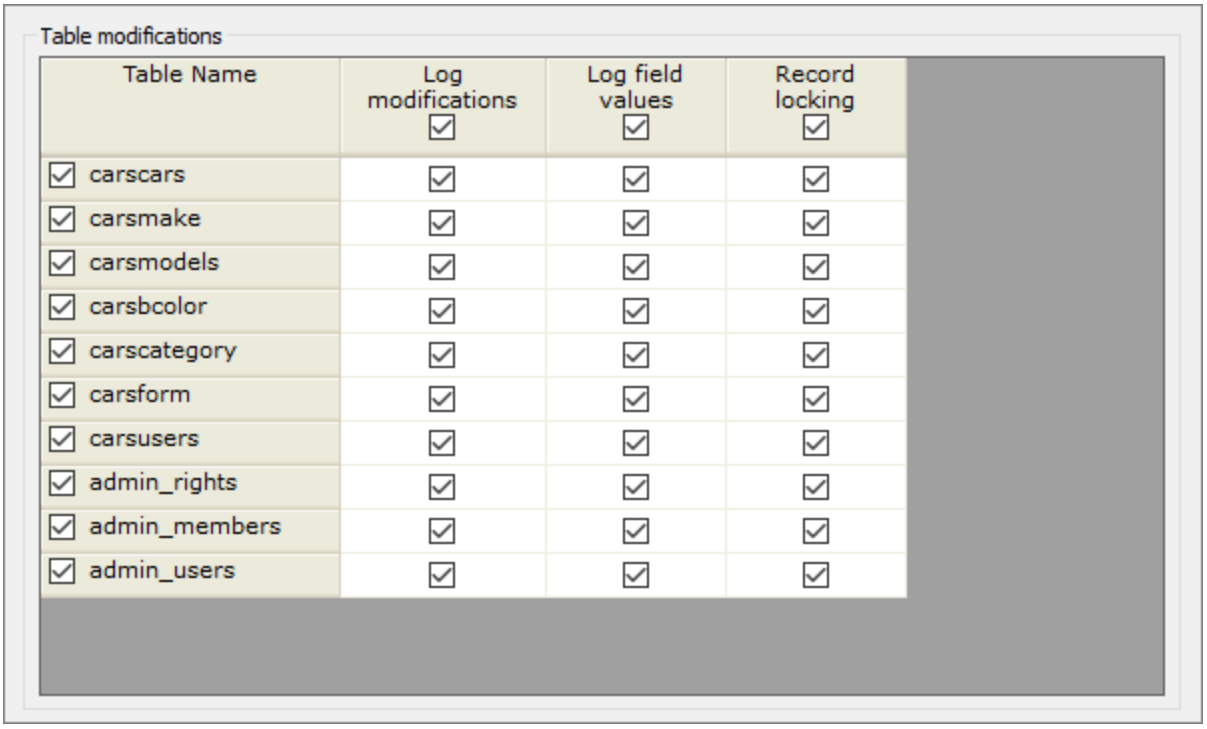
Select the tables to which to apply the modifications or field values logging in the Table modifications section.

Here is an example of a log file:

Date	Time	IP	User	Table	Action	Key	field	Field Old value	Field New value
Jul 23,2019	18:13:56	127.0.0.1	user	carsusers	login				
Jul 23,2019	18:14:20	127.0.0.1	user	carsbcolor	edit	25	color	Simple Black	Full Black
Jul 23,2019	18:14:44	127.0.0.1	user	carsbcolor	add	26	color		Basic White
Jul 23,2019	18:14:44	127.0.0.1	user	carsbcolor	add	26	rgb		#FFFFFF
Jul 23,2019	18:14:44	127.0.0.1	user	carsbcolor	add	26	OwnerID		3
Jul 23,2019	18:14:53	127.0.0.1	user	carsusers	logout				
Jul 23,2019	18:14:55	127.0.0.1	admin	carsusers	login				
Jul 23,2019	18:14:59	127.0.0.1	admin	carsusers	logout				

Note: to analyze information from the log table, you can add a report or chart to your project, or use the online report/chart builder.

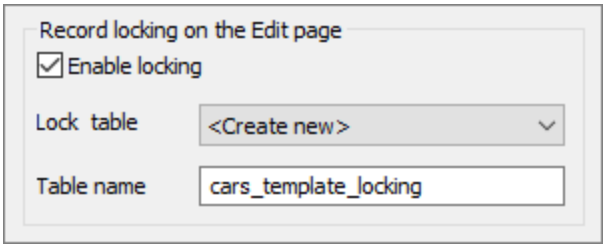
Table modifications



This is the section where you can fine-tune the logging and record locking.

Select the corresponding checkboxes to enable logging modifications, logging field values, or record locking for the tables of your choosing.

Record locking

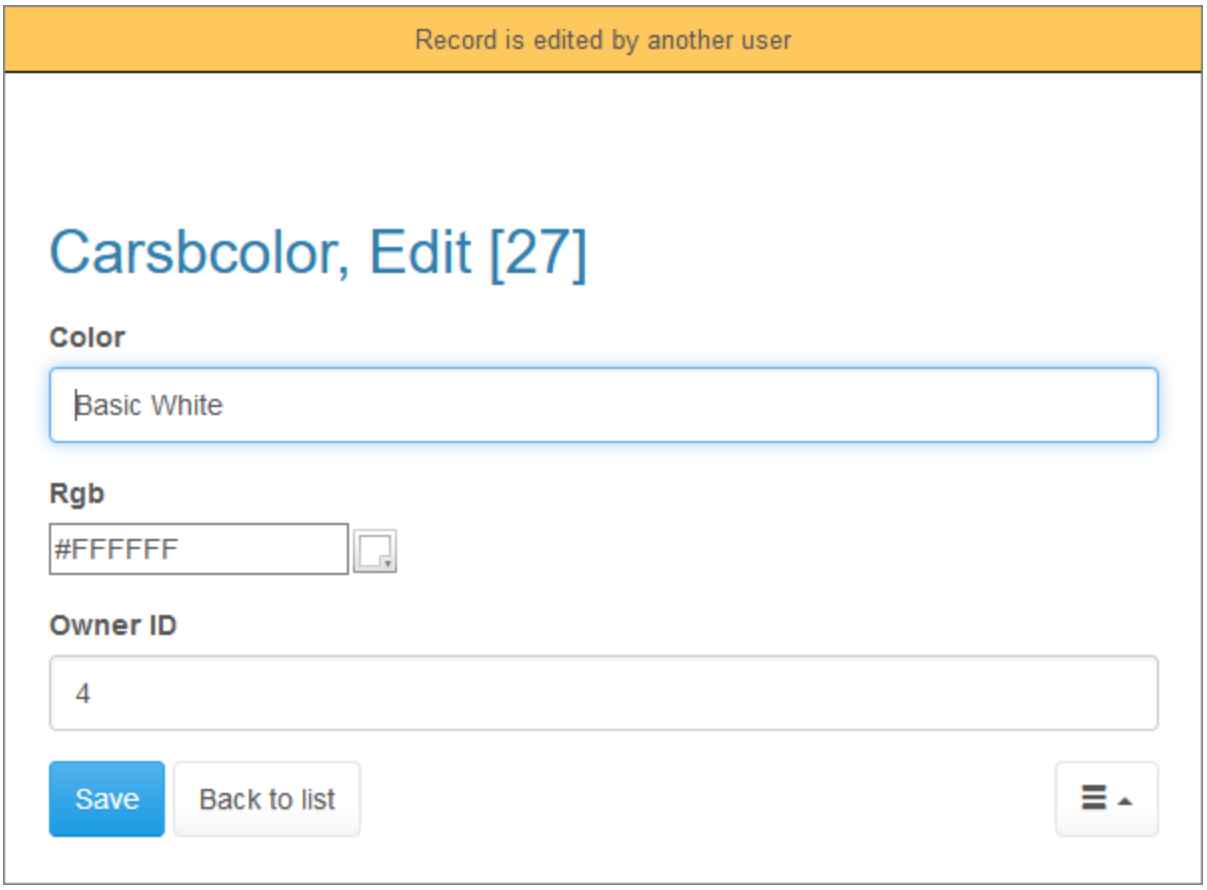


This section allows you to enable the record locking to prevent situations when users simultaneously edit the same record. While one user is editing the record, it becomes locked, so other users can't change it. The record locking is disabled by default.

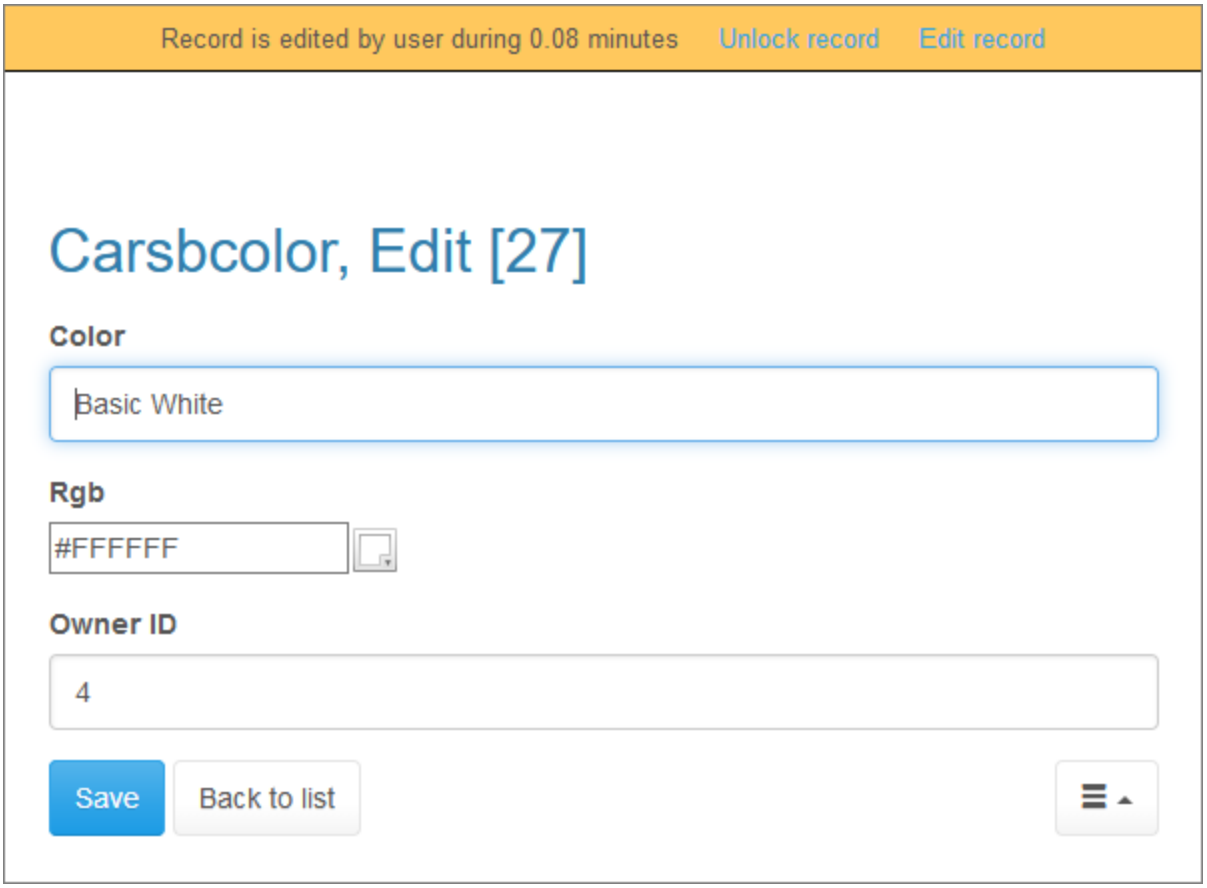
To enable record locking, select the **Enable locking** checkbox. You also need to choose an existing table to store the locking data or create a new one. You can set the name for the new table.

Select the tables to which to apply the record locking in the Table modifications section.

PHPRunner shows the following message to any user who tries to edit the locked record:



Note: administrators can unlock the record (the user receives a message, that admin aborted the edit session) or edit it (the user receives the standard message):



Auto unlocking

If a user goes to the **Edit** page and then clicks **Back** in the browser, the record remains blocked. To prevent this, PHPRunner utilizes an automatic unlocking method. By default, the record is unlocked after 550 seconds. You can change this value by editing two parameters in the file *include/locking.php*:

```
var $ConfirmTime = 250;
var $UnlockTime = 300;
```

The variable *ConfirmTime* determines the interval (in seconds) between requests for confirmation that the user hasn't left the **Edit** page (i.e., checking if the record is still locked).

The variable *UnlockTime* determines the time (in seconds) after a failed confirmation when the record is unlocked automatically.

Security screen articles:

- Security screen
- Login form appearance
- Two-factor authentication
- Registration and passwords
- Advanced security settings
- User group permissions
- Dynamic permissions
- Encryption
- Session keys
- Active Directory
- Facebook connect
- Sign in with Google
- CAPTCHA on authentication pages

See also:

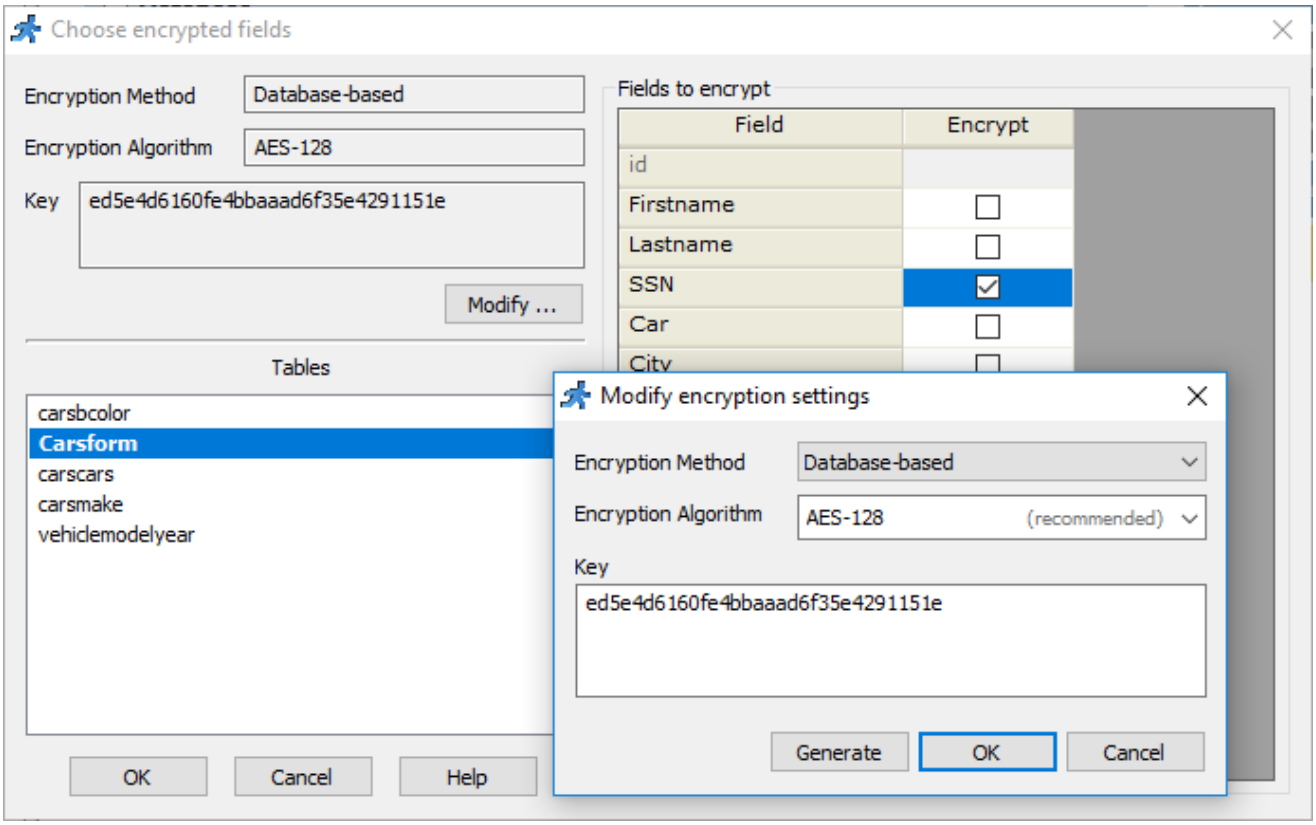
- Security API
- Datasource tables screen
- Miscellaneous settings
- Page Designer
- Event editor

2.15.9 Encryption

Encryption allows you to encrypt important data in the database, such as credit card numbers or Social Security Numbers. You need to select the encryption method, enter the encryption key, and choose the fields to be encrypted.

Note: the **Encryption** feature is available only in the **Enterprise Edition** of PHPRunner. See Editions Comparison.

Press the **Encryption** button on the **Security** screen to open the **Encryption** popup.



Encryption methods

You can select a **Database-based** or **Code-based** encryption method.

Note: the **Database-based encryption** method is available only for MySQL, Oracle, PostgreSQL, and MS SQL Server databases.

The **Database-based** method is preferable since it has more features than the **Code-based** method. With the **Database-based** encryption, for example, the encrypted fields can be sorted and grouped, the search offers suggestions and includes all operators (CONTAINS, EQUALS, MORE THAN, etc.).

Database-based encryption requirements

PostgreSQL:

- install the pgcrypto module.

Oracle:

- give users full rights to the SYS.DBMS_CRYPTO package;
- the Oracle version must be 10 or higher.

MySQL:

- enable SSL support.

Code-based encryption requirements

To use the PHP encryption, enable the mcrypt extension in php.ini (PHP 5.3 and higher includes this extension by default).

Encryption key

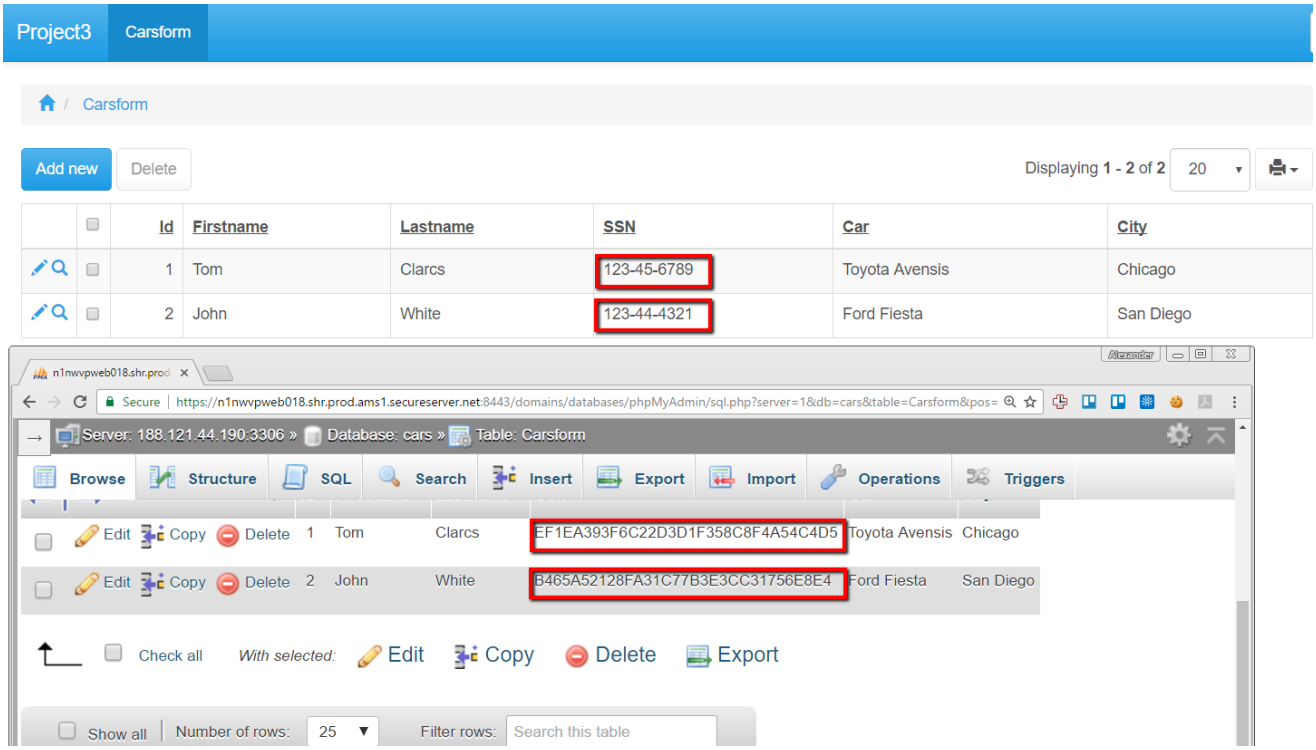
We recommend using the encryption key that is at least 10 characters long. You can also use the **Generate button** to generate a random key.

PHPRunner can encrypt only text fields. Since the encrypted value usually is at least 2-3 times longer than source value, you should choose the maximum length fields such as TEXT in MySQL or MEMO in MS Access.

> **Note**: PHPRunner does not encrypt the existing data. Encryption is applied to the record during the **Add**/**Edit** operations.

> **Note**: once the encrypted data is stored in the database, you should not change the encryption type and key, or deactivate the encryption, as the data will remain encrypted.

Here is an example of encrypted data in the application:



Functions used for database-based encryption

Oracle:

- Encryption: DBMS_CRYPTO.ENCRYPT()
- Decryption: DBMS_CRYPTO.DECRYPT()

MS SQL Server:

- Encryption: EncryptByPassPhrase(), EncryptByKey()
- Decryption: DecryptByPassPhrase(), DecryptByKey()

MySQL:

- Encryption: DES_ENCRYPT(), AES_ENCRYPT()
- Decryption: DES_DECRYPT(), AES_DECRYPT()

PostgreSQL:

- Encryption: pgp_sym_encrypt()
- Decryption: pgp_sym_decrypt()

Functions used for code-based encryption

PHPRunner uses DES or AES-128 encryption algorithms.

Encrypt existing values in the database

> **Note**: before starting this procedure, make a backup of the database!

You may encrypt the existing values only once. We do not recommend double encryption as it causes problems with decryption.

> **Note**: it is not possible after the encryption to determine whether the data had been encrypted or not.

To encrypt the existing values in the database, add the following code to the List page: Before process event of your table:

```
include_once("ciphcoding.php");
```

Then run the **List** page that contains the encrypted fields with the ciphcoding=1 parameter, e.g.:

```
mytable_list.php?ciphcoding=1
```

Once the data has been encrypted, it is necessary to delete the file *ciphcoding.php* in the output directory, remove the code from the **List page: Before process** event and re-upload the application.

We recommend performing this procedure on the development machine or a server without public access.

Decrypt custom query results

MySQL, AES encryption

The key variable should contain the encryption key specified in PHPRunner on the **Encryption** screen.

```
//define encryption key
$key='09a308862fbe462095dd6eba33ab9dd21b8fd35b0d884b48819a34ce8636983b';

$sql = "SELECT cast(AES_DECRYPT(unhex(customer_name), '".$key."') as char) AS
custname FROM customers_table WHERE id = 'CUST123'";
$rs = CustomQuery($sql);
$data = db_fetch_array($rs);

echo $data["custname"];
```

Security screen articles:

- Security screen
- Login form appearance
- Two-factor authentication
- Registration and passwords
- Advanced security settings
- User group permissions
- Dynamic permissions
- Audit and record locking
- Session keys
- Active Directory
- Facebook connect
- Sign in with Google
- CAPTCHA on authentication pages

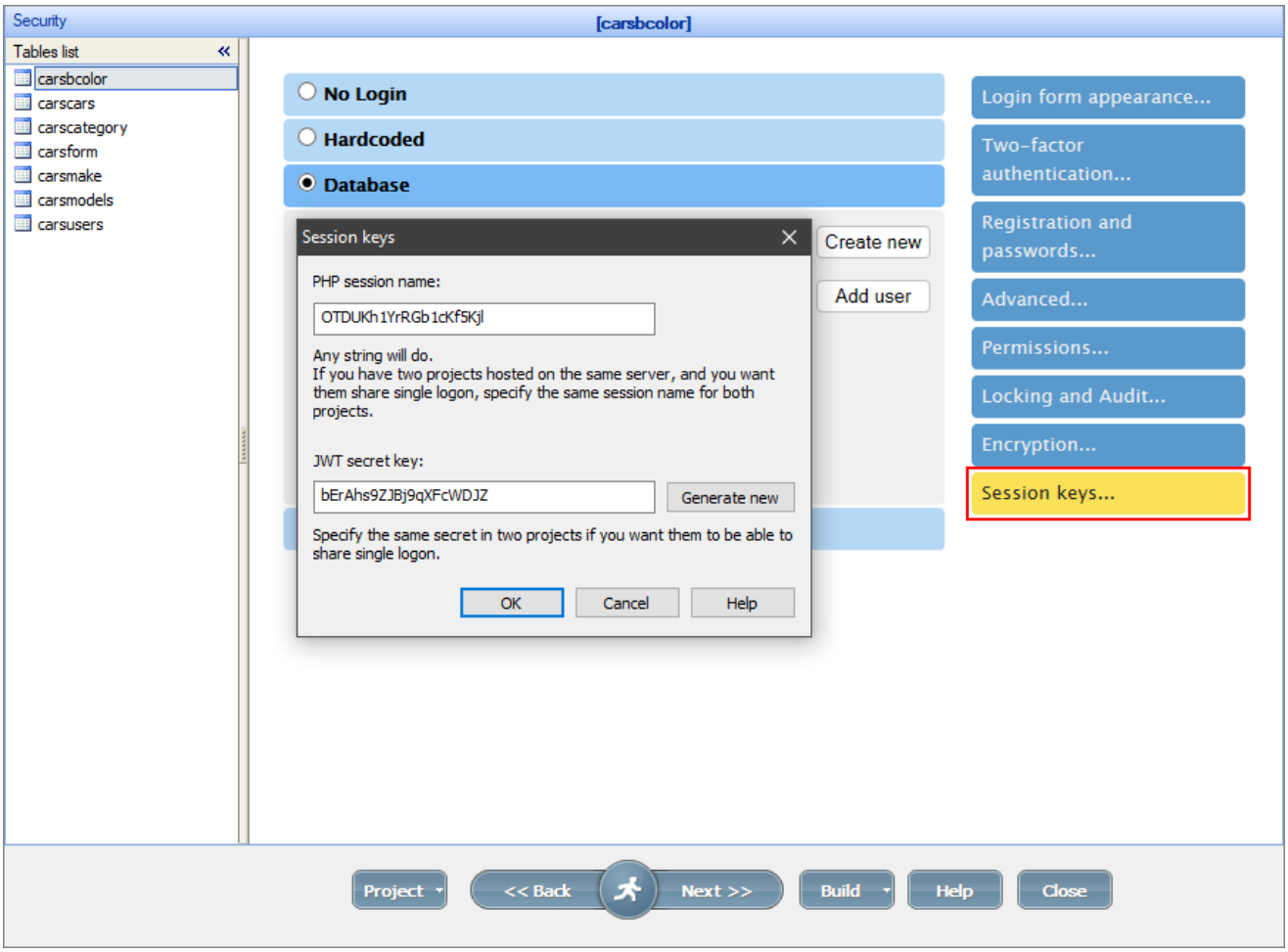
See also:

- Security API

- Datasource tables screen
- Miscellaneous settings
- Page Designer
- Event editor

2.15.10 Session keys

Starting with PHPRunner version 10.3, you can find a **Session keys** button on the Security screen. Click this button to open a popup containing PHP session name and JWT (JSON Web Token) secret key options.



PHP session name

This option allows you to enter any string as a **PHP session name**.

If you have two PHP projects on the same server and you want them to share a single logon, enter the same session name for both projects.

JSON Web Token secret key

JSON Web Tokens are a secure, cryptographically protected way of exchanging data over the network.

In PHPRunner, **JSON Web Tokens** are encrypted with a secret key specific to your application. If you want the users to log in only once to access all your applications, make all your projects share the same secret key.

You can enter the same **JWT secret key** for all your projects on the **Security** screen -> **Session keys** dialog. This allows different projects to verify each other's tokens.

Visit https://jwt.io/ to learn more about **JSON Web Tokens**.

Security screen articles:

- Security screen
- Login form appearance
- Two-factor authentication
- Registration and passwords
- Advanced security settings
- User group permissions
- Dynamic permissions
- Audit and record locking
- Encryption
- Active Directory

- Facebook connect
- Sign in with Google
- CAPTCHA on authentication pages

See also:

- Security API
- Datasource tables screen
- Miscellaneous settings
- Page Designer
- Event editor

2.15.11 Active Directory

Quick jump

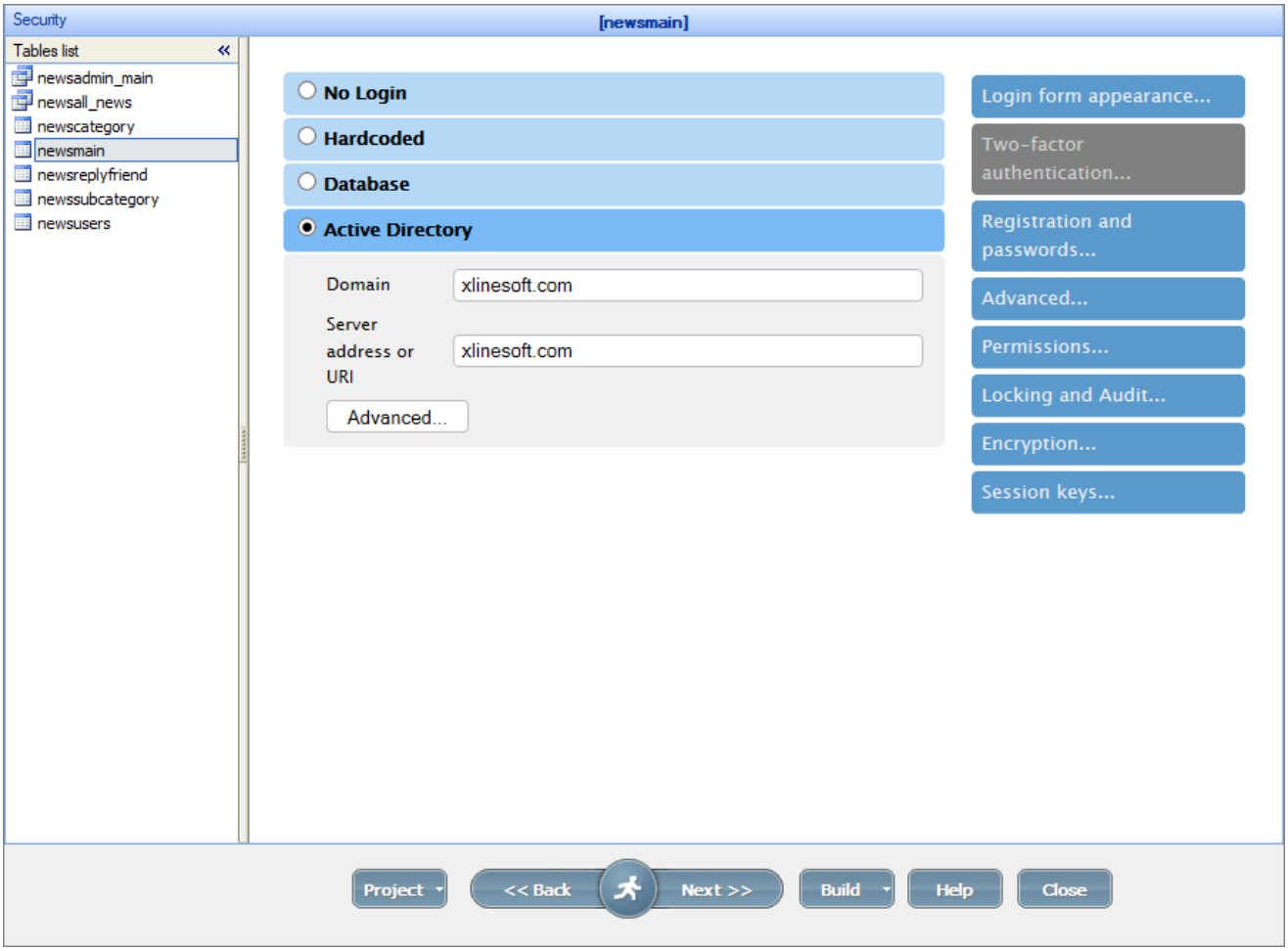
How to enable Windows Authentication

Active Directory authentication and Permissions

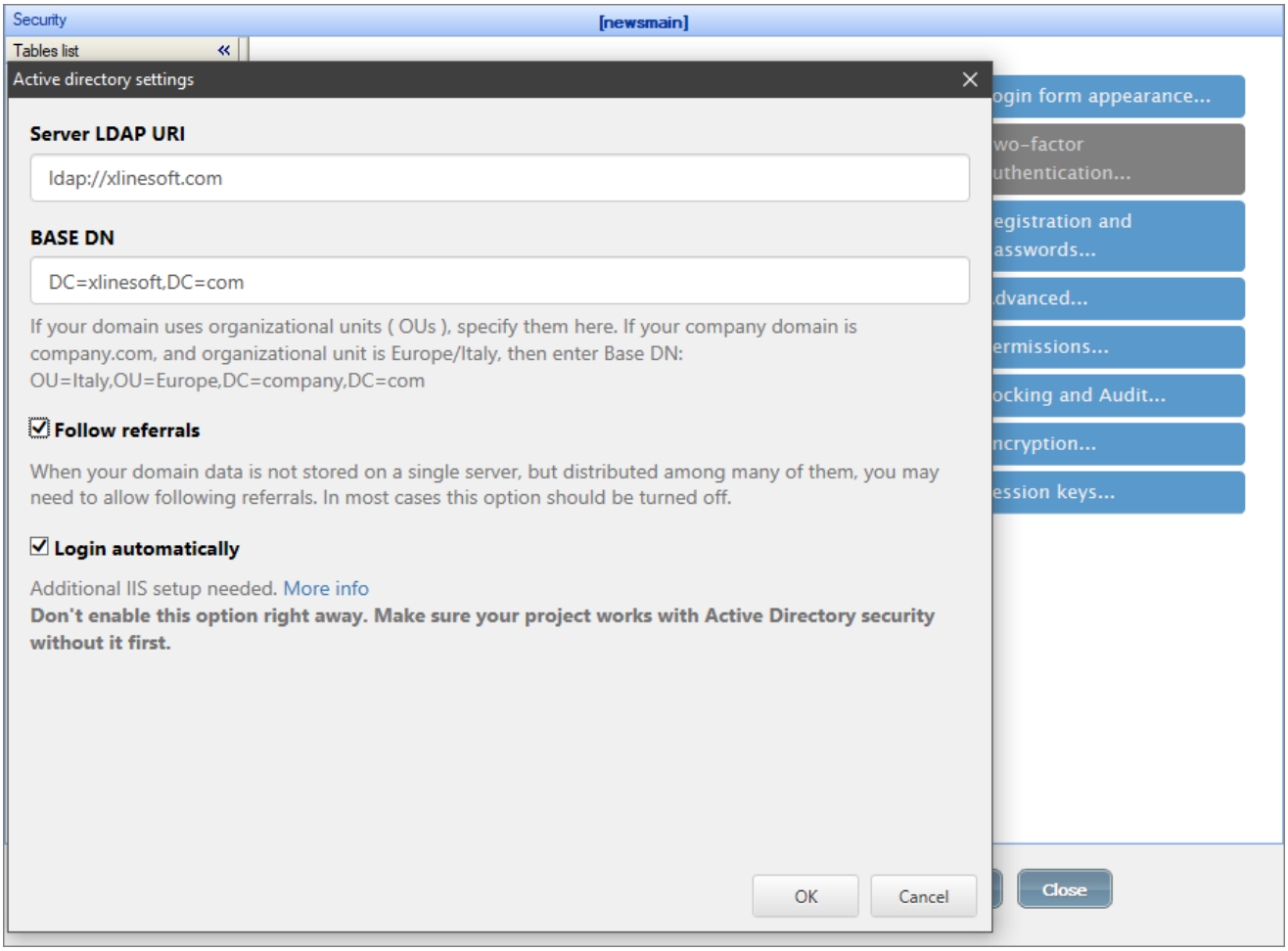
Active Directory authentication allows users to log in to the generated by PHPRunner applications if they have an account in an Active Directory domain. When logging in, the login and password are checked against Active Directory.

> **Note**: the **Active Directory authentication** feature is available only in the **Enterprise Edition** of PHPRunner. See Editions Comparison to learn more.

To use this type of authentication, you need to fill the *Active Directory Domain* and *Server*. In the most straightforward use case, no additional configuration is needed.



Starting with PHPRunner version 10.3, you can access **Active Directory settings** by clicking the **Advanced** button:



- **Server LDAP URI**. With this option, you can specify *LDAPS protocol* and *port number*.
- **Base DN**. With this option, you can specify *Organizational Units (OU)* in the wizard software. If your domain uses organizational units (OUs), specify them in this field. If your company domain is *company.com*, and the organizational unit is *Europe/Italy*, then enter the *Base DN* as following: *OU=Italy,OU=Europe,DC=company,DC=com*.
- **Follow referrals**. When your domain data is not stored on a single server, but distributed among many of them, you may need to allow following referrals. In most cases, this option should be turned off.
- **Login automatically**. This checkbox enables the **Autologin** functionality: if a person is already logged into Windows, they are automatically logged into the generated application. To use this feature, make sure Windows **Authentication** is enabled in **Internet Information Services (IIS)**.

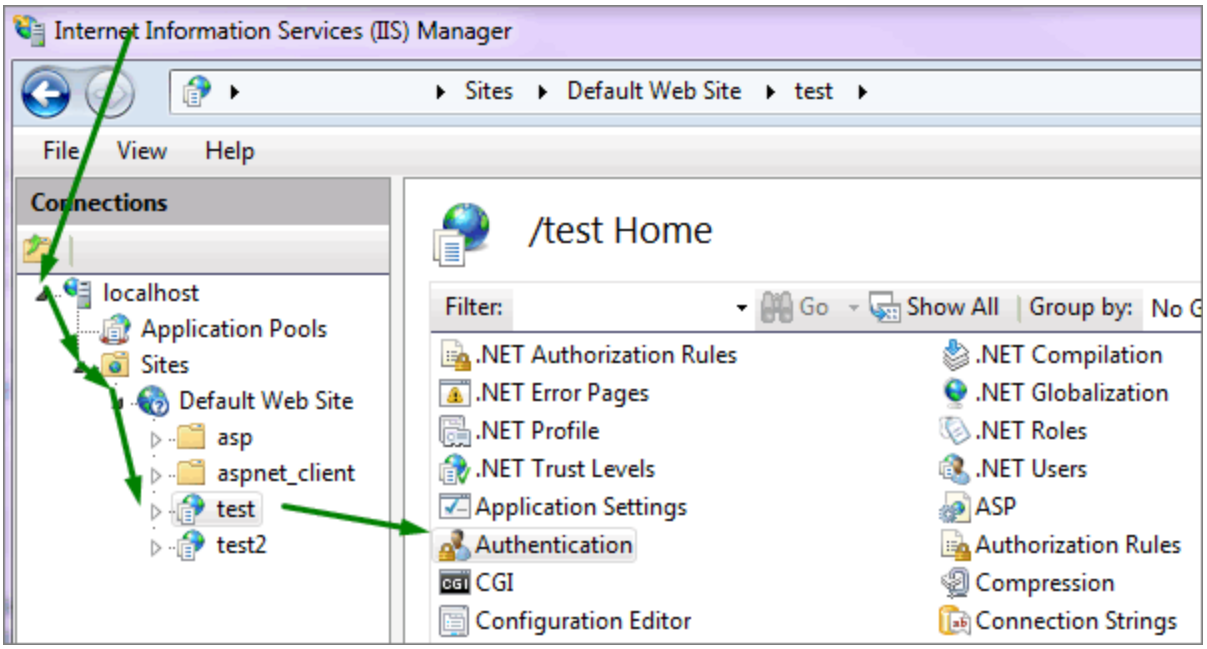
How to enable Windows Authentication

1. Make sure you have IIS installed. Go to **Control panel** -> **Programs** -> **Turn Windows features on or off** and select the **Internet Information Services**. After that, select the **Windows Authentication** under **IIS** -> **World Wide Web Services** -> **Security**. Click **OK** and wait for everything to install.

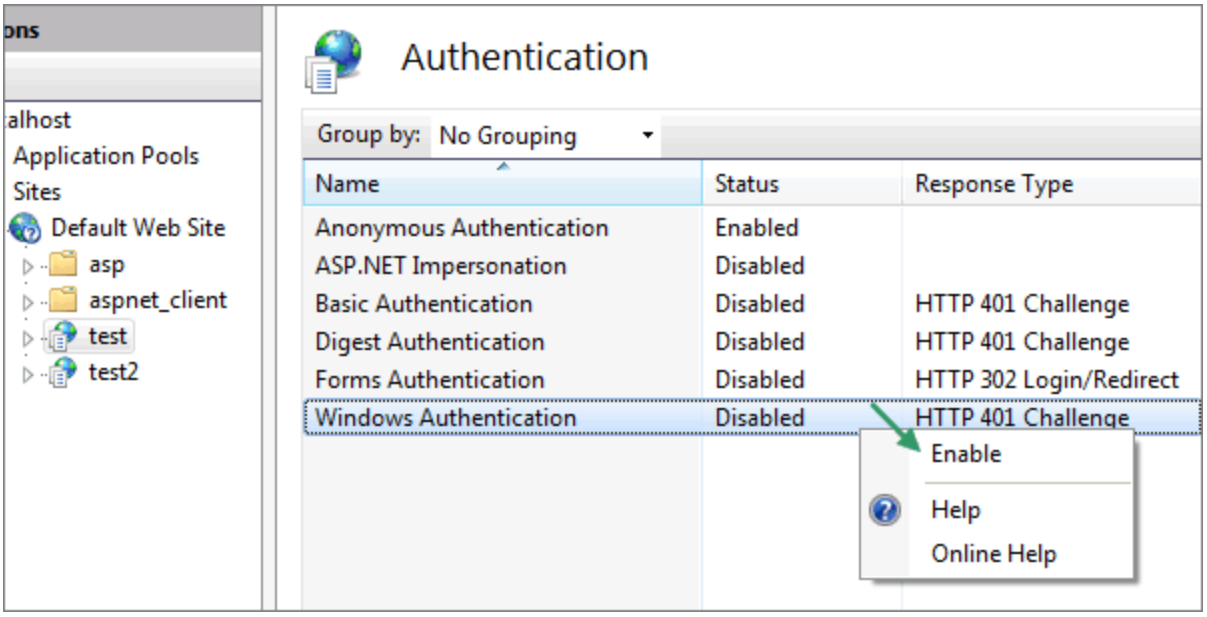
2. Run IIS manager as the administrator: Go to **Control Panel** -> **Administrative Tools** -> **Internet Information Services (IIS) Manager**.

3. Expand the server in the **Connections** frame and choose the site, or click on the server if you wish to apply settings for all sites.

4. Double-click the **Authentication** icon in the main window.



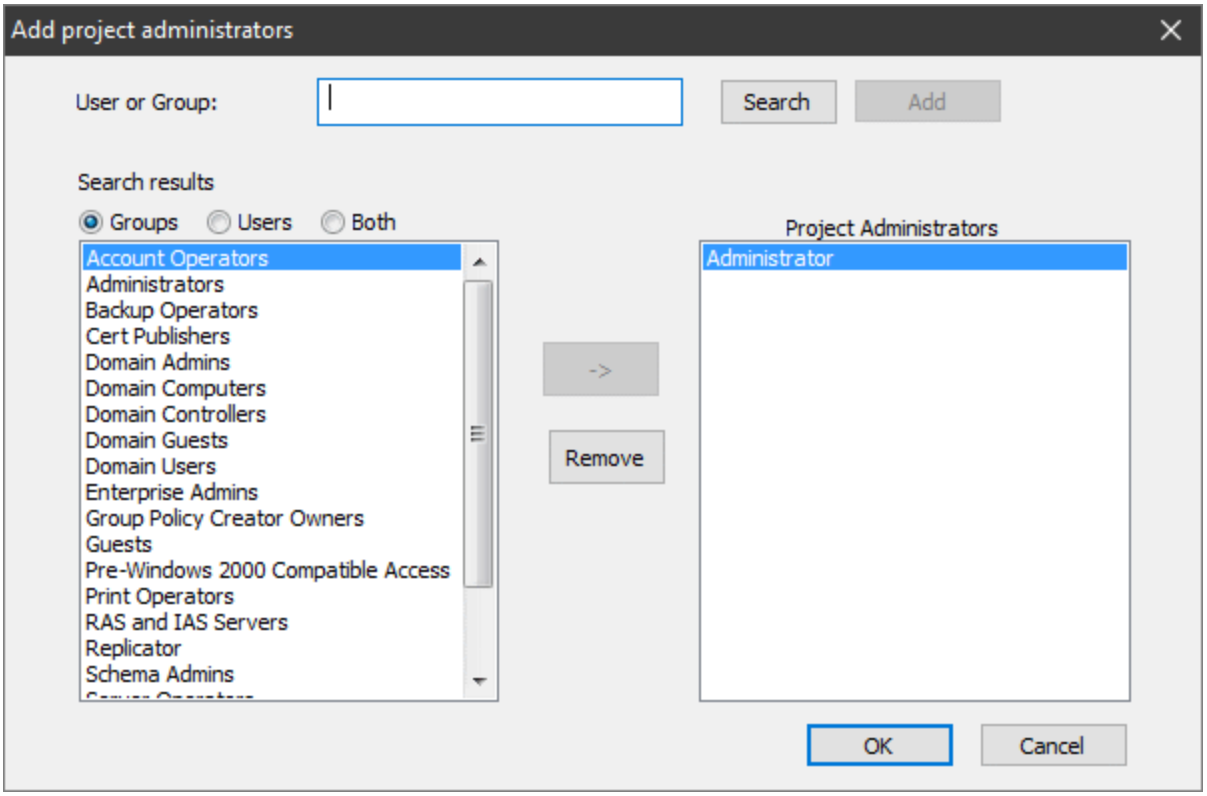
5. Right-click **Windows Authentication** and choose **Enable**.



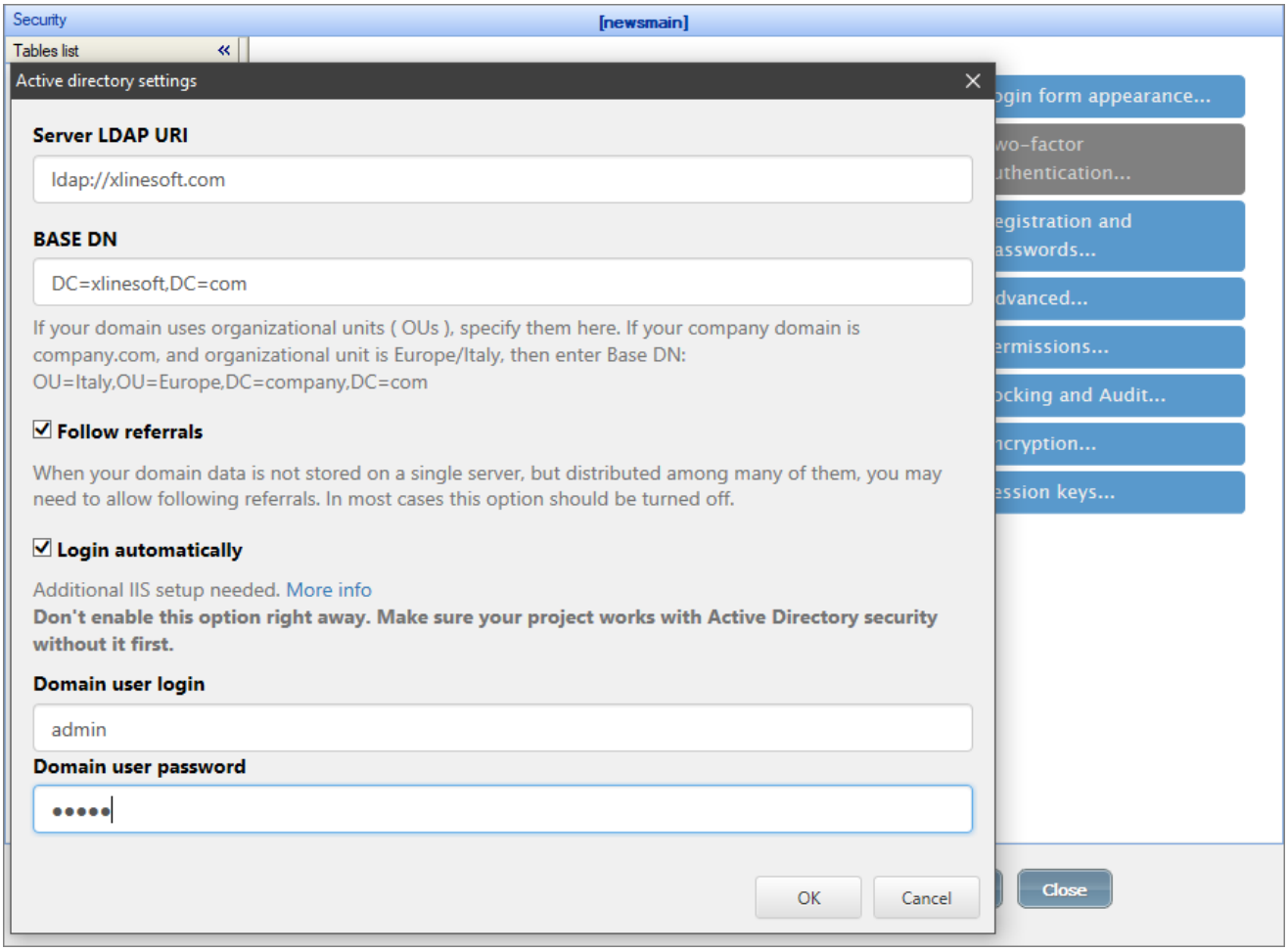
Active Directory authentication and Permissions

You can use the Permissions feature along with the **Active Directory authentication**. Click on **Permissions** and enable the Use **Dynamic Permissions** checkbox. You need to choose tables to store the permissions and create an admin user.

To add an admin user, click **Add admin user** and then **Search**. You need to fill the username and password to connect to **Active Directory**. Then you can to select a group or groups to have the admin access.

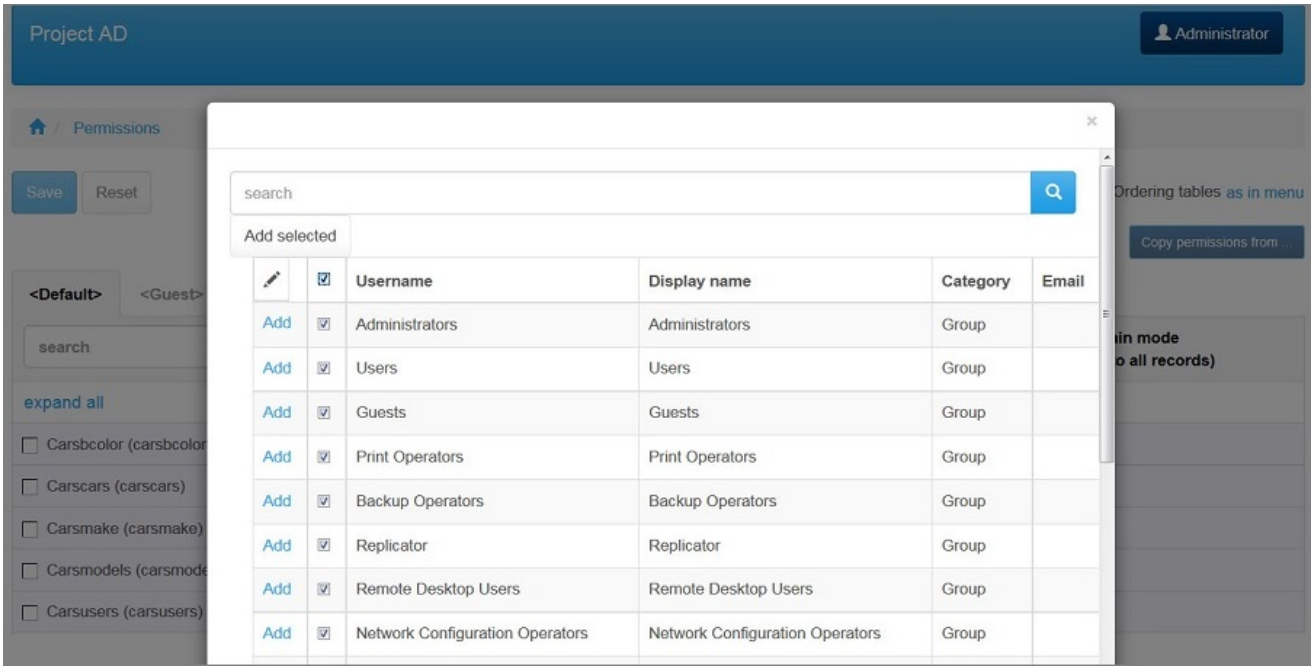


If your project utilizes **Dynamic permissions** and you have enabled the **Login automatically** checkbox, you also need to specify the *Domain user login* and *password*.



Build your project and login as admin to the generated application. In the **Admin Area** on the **Admin Rights** page, you can add groups via **Add Group** and assign permissions to them.

> **Note**: you can not create groups manually since they are stored on the Active Directory server and should be modified there.



Security screen articles:

- Security screen
- Login form appearance
- Two-factor authentication
- Registration and passwords
- Advanced security settings
- User group permissions
- Dynamic permissions
- Audit and record locking
- Encryption
- Session keys
- Facebook connect
- Sign in with Google
- CAPTCHA on authentication pages

See also:

- Security API
- Datasource tables screen
- Miscellaneous settings
- Page Designer
- Event editor

2.15.12 Facebook connect

With PHPRunner you can integrate **Facebook Connect** into the generated application.

Facebook Connect is a set of APIs that provide a simple way for people to sign in to web sites, applications, and mobile devices using their Facebook ID. In addition to simplifying the registration process, **Facebook Connect** can make it easier to share the content through Facebook.

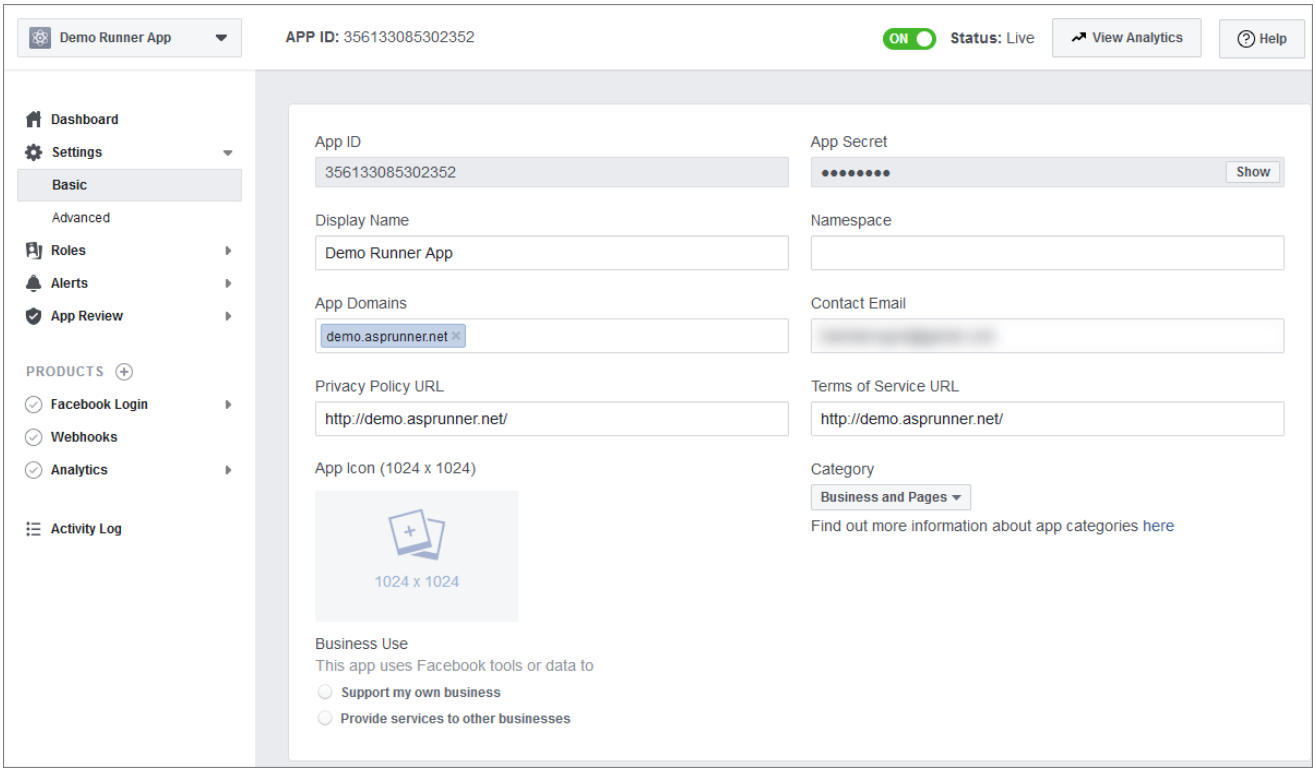
To use the **Facebook authentication** option, you need to configure **Facebook Connect** first.

How to configure Facebook connect

Register your FB profile as a developer at http://developers.facebook.com/setup/. After verifying the phone number and picking a role, click **Create first app**. Set the display name and contact email, then click **Create app ID** and select the **Integrate Facebook login** scenario.

Proceed to **Settings** -> **Basic** on the left-side panel, and fill in the *App Domains*, *Privacy Policy URL*, *Terms of Service URL* fields, and click **Save changes** at the bottom of the page.

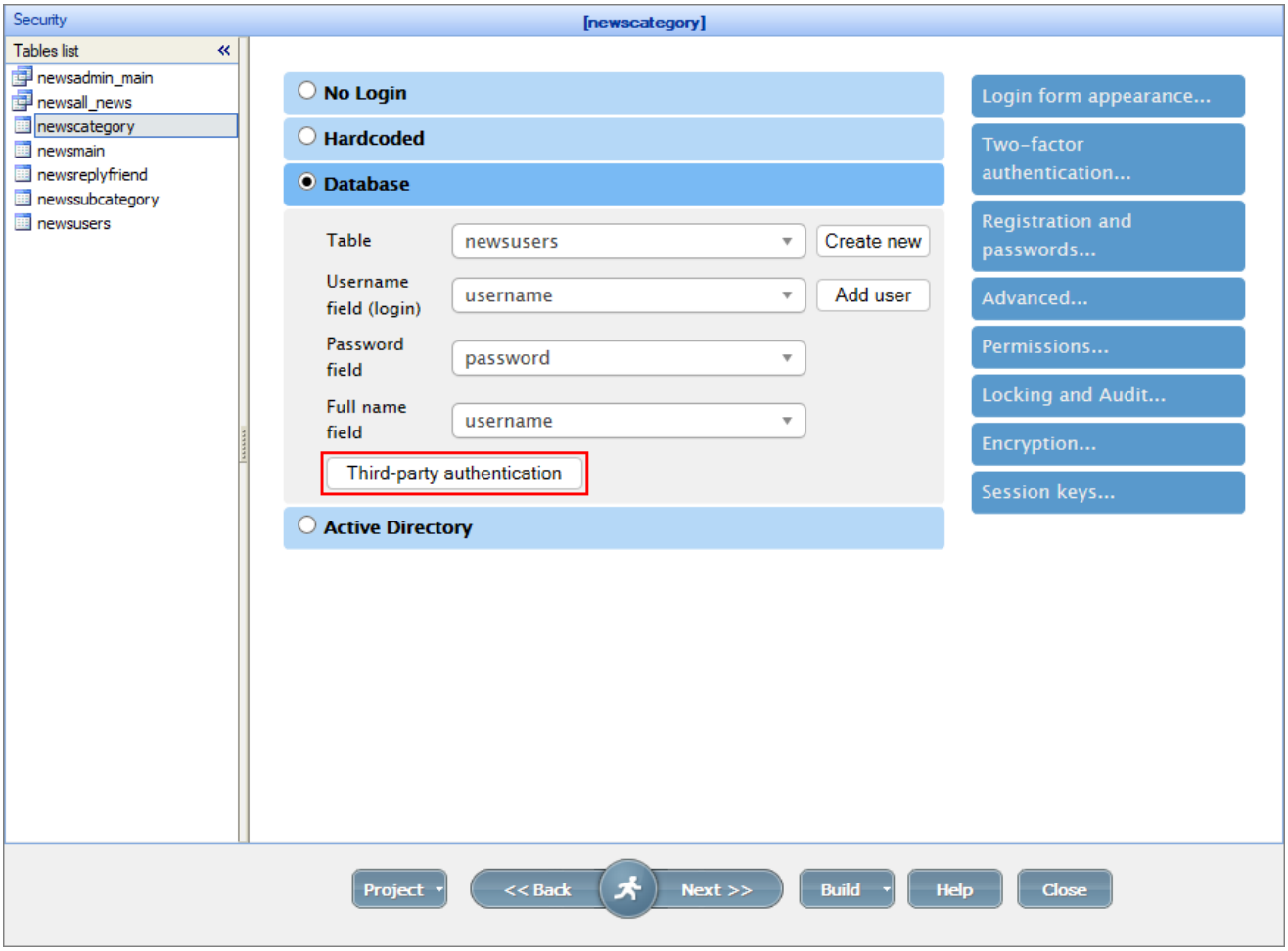
That's all you need to start using **Facebook Connect**. Now you can copy the **App ID** and **App secret**.



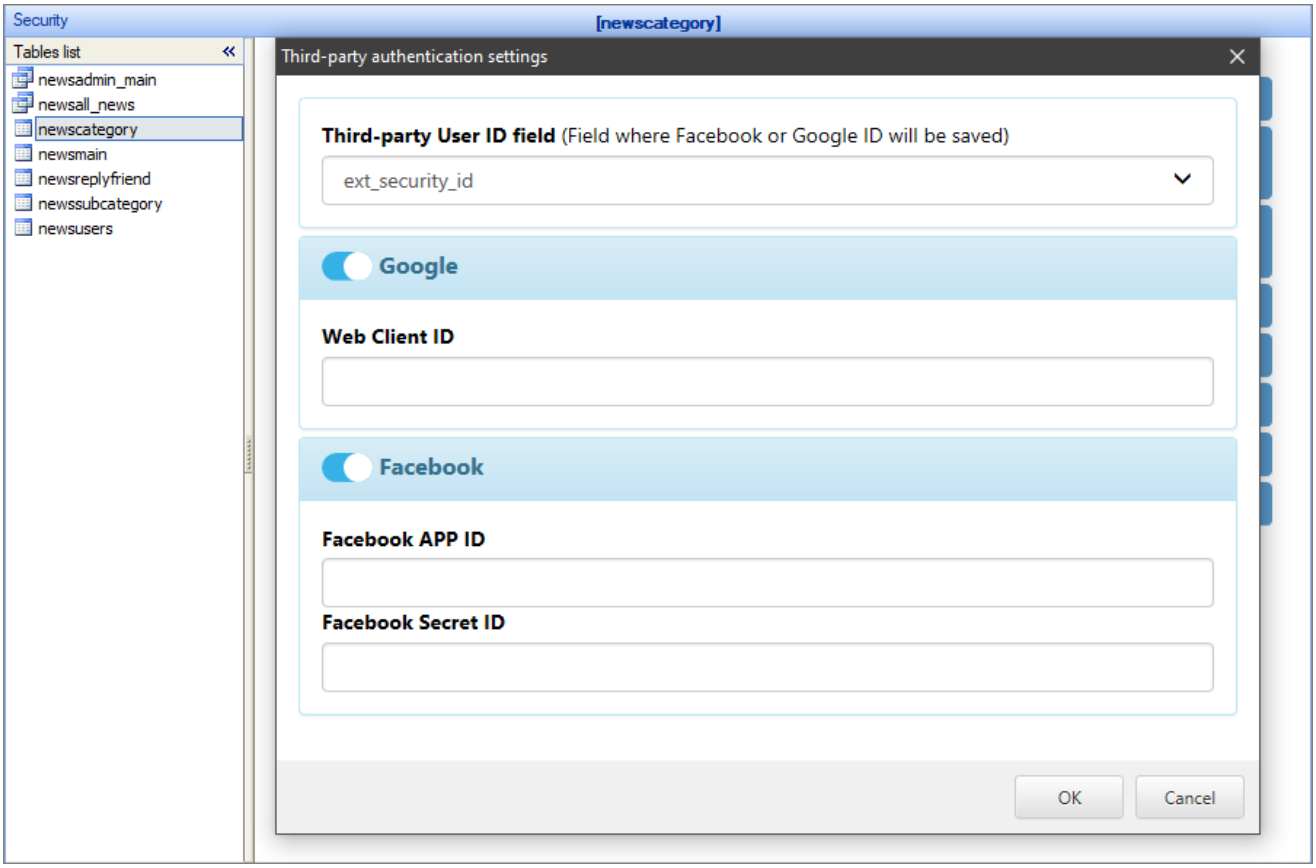
> **Note**: you can find your created Facebook apps at http://www.facebook.com/developers/apps.

Setting up the Facebook authentication

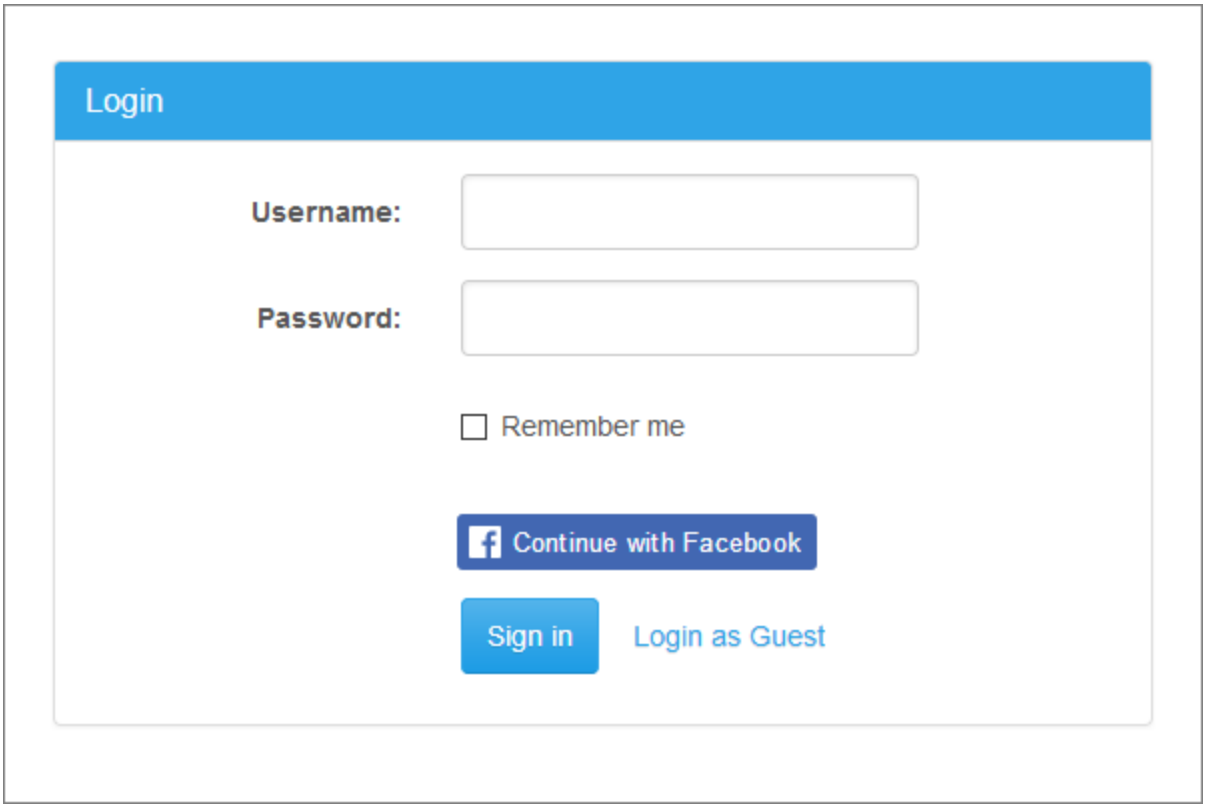
Click the **Third-party authentication** option on the Security screen.



Toggle the **Facebook option** and fill in the *Facebook App ID*, *Facebook Secret ID*. You also need to select a field to store the *Third-party User ID*.



After you Build the app, you can see the **Login with Facebook** button on the **Login** page of your application.



If you are interested in additional integration capabilities, check the article at http://xlinesoft.com/blog/...facebook-connect/.

If you have enabled Dynamic permissions, the Facebook users are assigned the rights of the Default group: *Facebook users -> Default users*.

Security screen articles:

- Security screen
- Login form appearance
- Two-factor authentication
- Registration and passwords
- Advanced security settings
- User group permissions
- Dynamic permissions
- Audit and record locking

- Encryption
- Session keys
- Active Directory
- Sign in with Google
- CAPTCHA on authentication pages

See also:

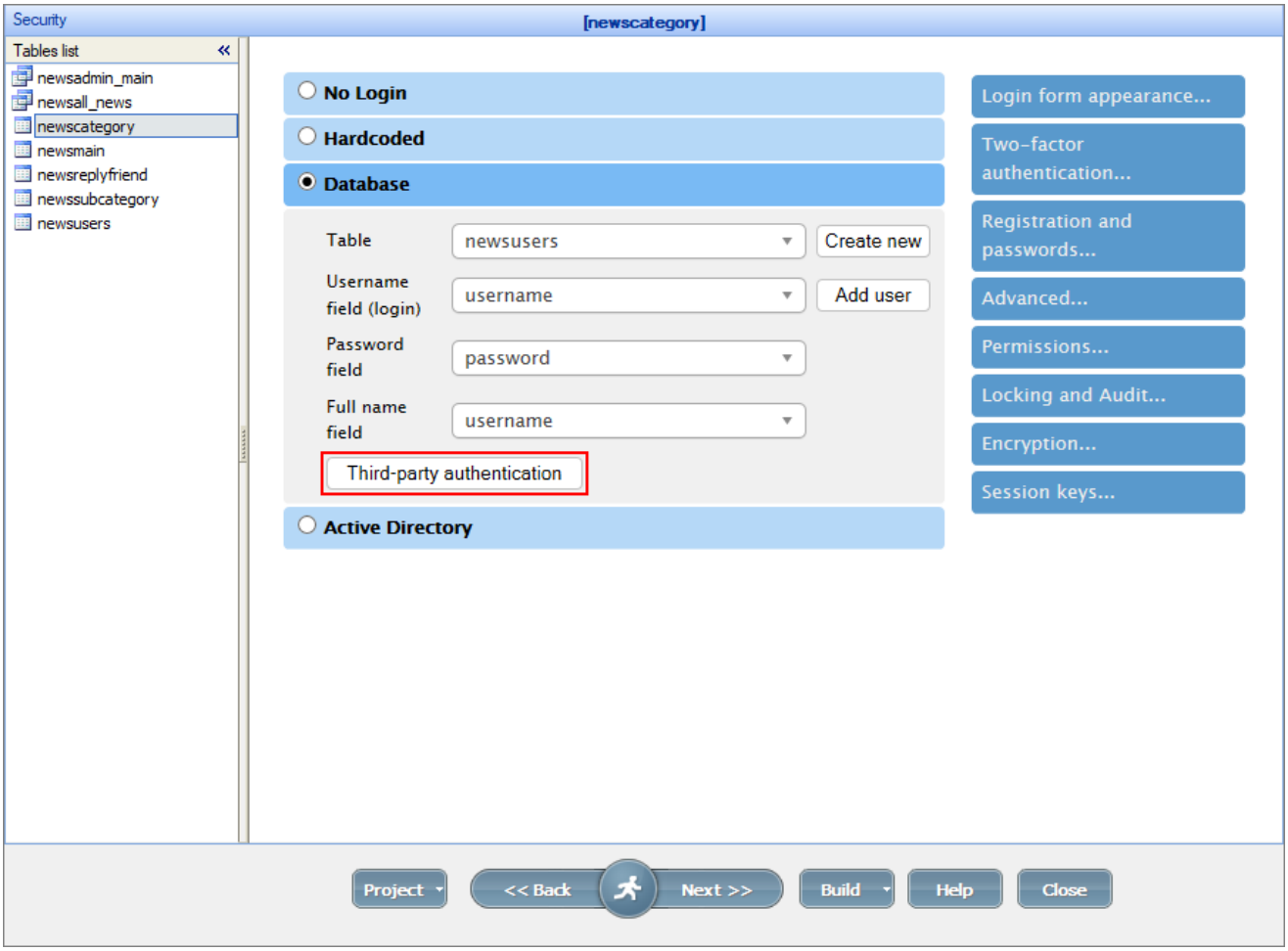
- Security API
- Datasource tables screen
- Miscellaneous settings
- Page Designer
- Event editor

2.15.13 Sign in with Google

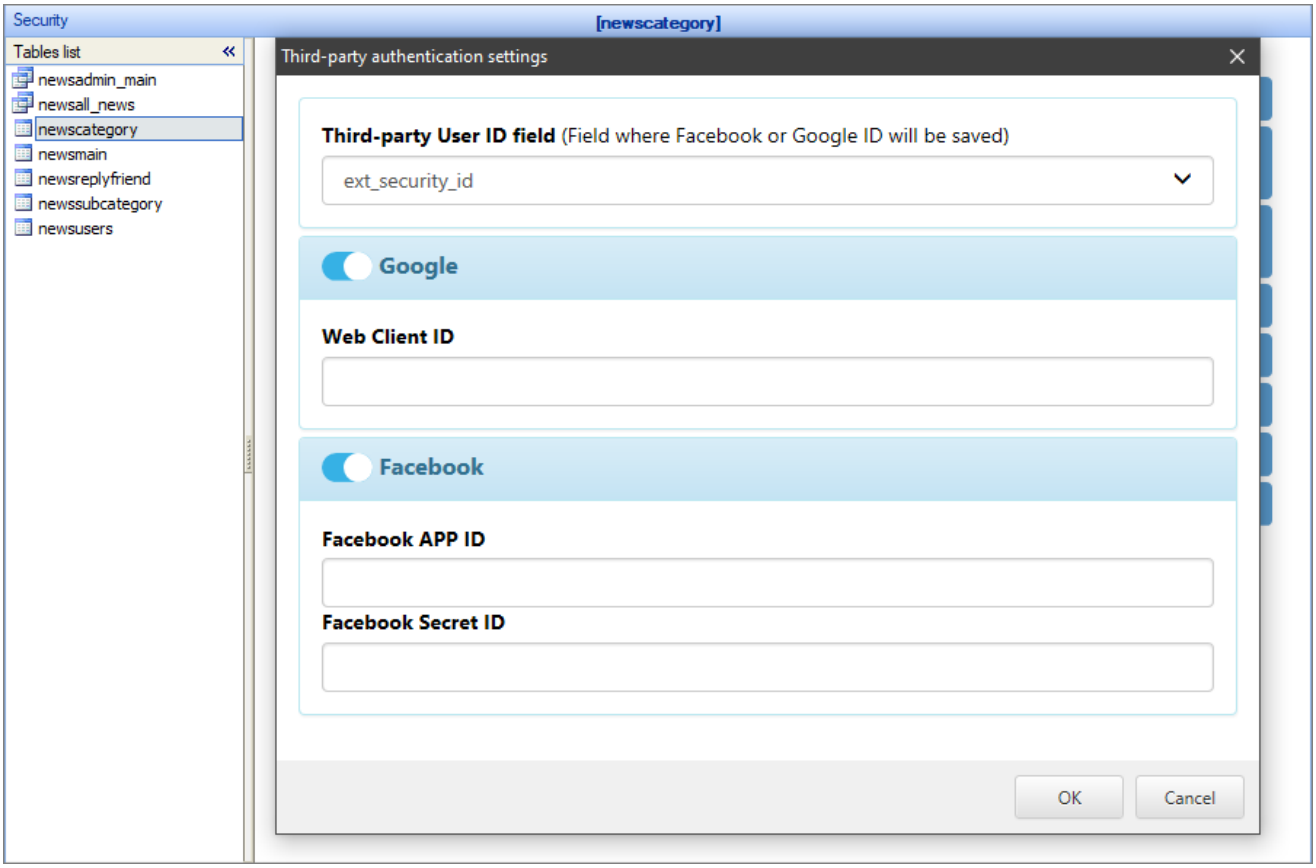
Starting with PHPRunner version 10.3, you can let the users sign in with Google into the generated app.

This option provides a simple way to authorize users, as anyone with a Google account can easily log into your app.

You can access these options by clicking the **Third-party authentication** option on the Security screen.



This button opens the window with Google and Facebook authentication options. Use the toggles to show or hide the fields under these options.



The **Third-party User ID** field is a dropdown where you can select the field to store the Facebook or Google IDs. Choose an existing field or create a new one.

To use the **Sign in with Google** option, you need to get the **Web client ID**. Here are the necessary steps:

1. Open the Google Developers Console.

> **Note**: you need a Google account to access the Developers Console. If you don't have it, you can create a new account.

2. Proceed to **Credentials** -> **Create credentials** -> **Oauth Client ID** -> **Web application**.

> **Note**: if the **Web application** item is not available, read the instructions on the page on how to enable it.

3. Add the **Authorized JavaScript origins**. You can modify this list later.

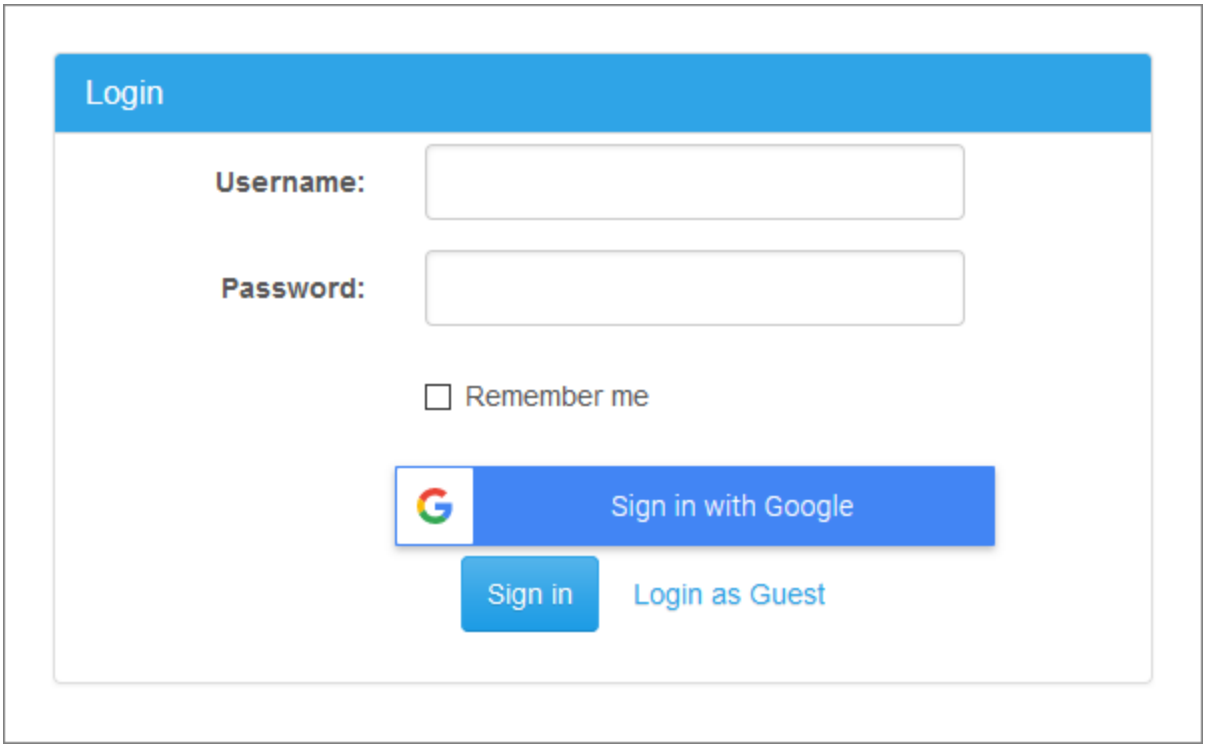
For local testing, add the following:

- *http://localhost:8085*
- *http://localhost:8086*
- *http://localhost:8087*
- *http://localhost*

For testing using the Demo account, add *http://demo.asprunner.net*.

4. Click **Save** to obtain the **Web client ID**. Copy and paste it into the respective field in the **Third-party authentication** window.

5. After you Build the app, you can see the **Sign in with Google** button on the **Login** page of your application.



If you have enabled Dynamic permissions, the Google users are assigned the rights of the Default group: *Google users -> Default users*.

Security screen articles:

- Security screen
- Login form appearance
- Two-factor authentication
- Registration and passwords
- Advanced security settings
- User group permissions
- Dynamic permissions
- Audit and record locking
- Encryption
- Session keys
- Active Directory

- Facebook connect
- CAPTCHA on authentication pages

See also:

- Security API
- Datasource tables screen
- Miscellaneous settings
- Page Designer
- Event editor

2.15.14 CAPTCHA on authentication pages

CAPTCHA is a simple test to determine whether the user is a human or a bot. It is used to prevent spamming and other automated abusive behavior on websites.

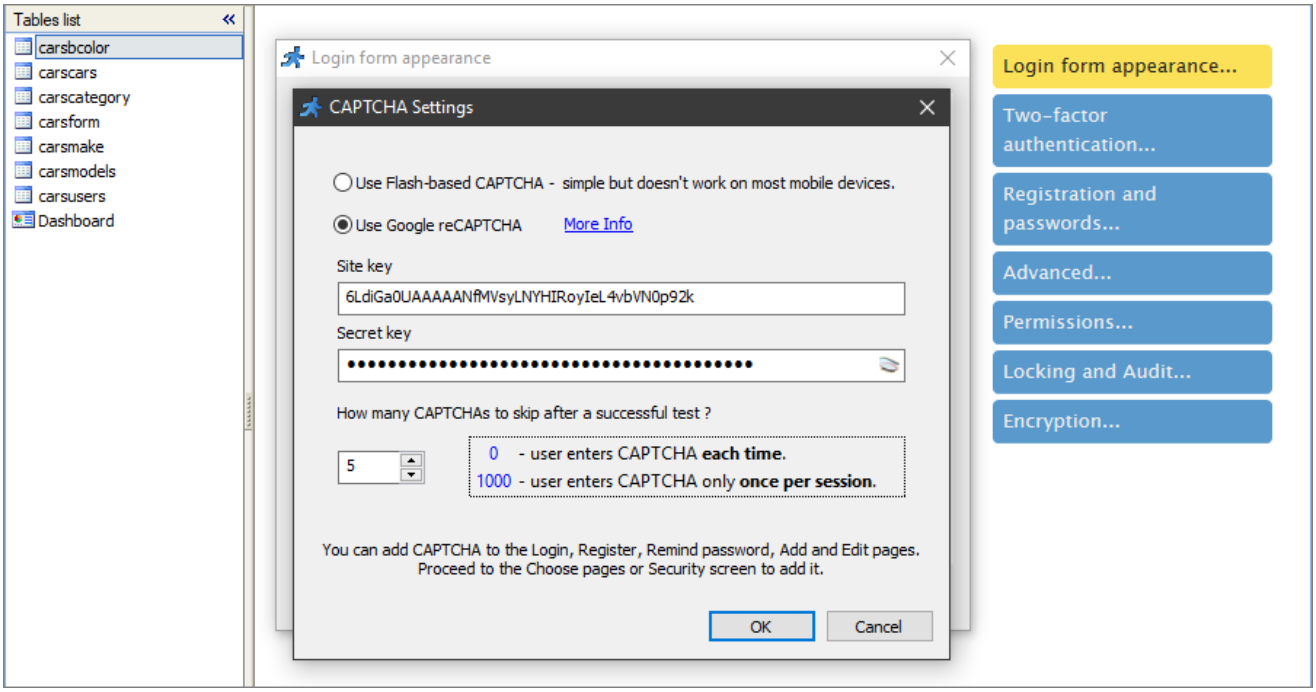
You can add CAPTCHA to the Add/Edit or **Login**/**Register**/**Remind password** pages:

- To add CAPTCHA to the **Login** page, click Login form appearance on the **Security** screen, and select the **Display CAPTCHA** checkbox.
- To add CAPTCHA to the **Register** and **Remind password** pages, click Registration and passwords on the **Security** screen, and select the **Display CAPTCHA** checkbox.

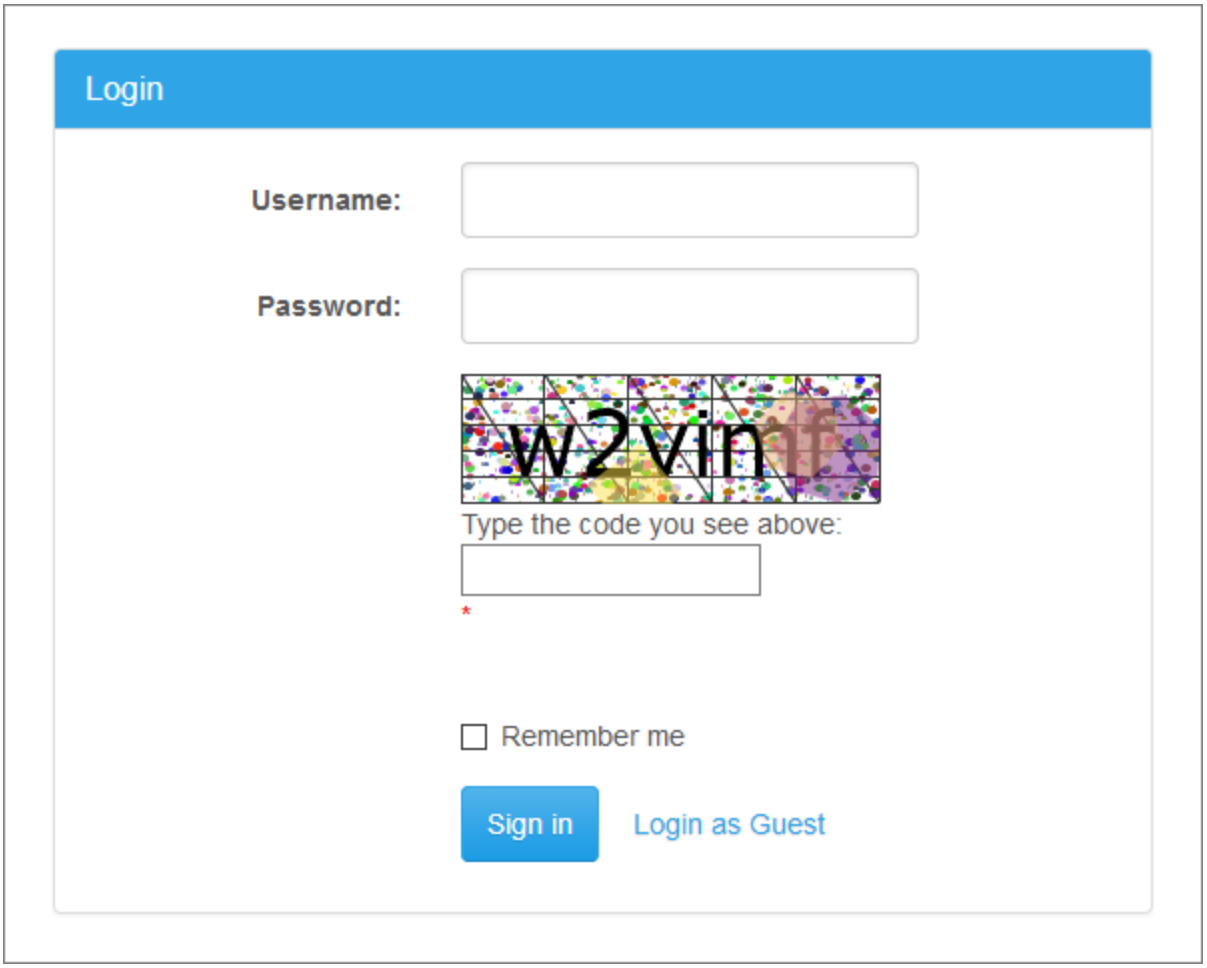
Then click **CAPTCHA settings** and choose what type of CAPTCHA to use:

- Flash-based CAPTCHA;
- Google reCAPTCHA.

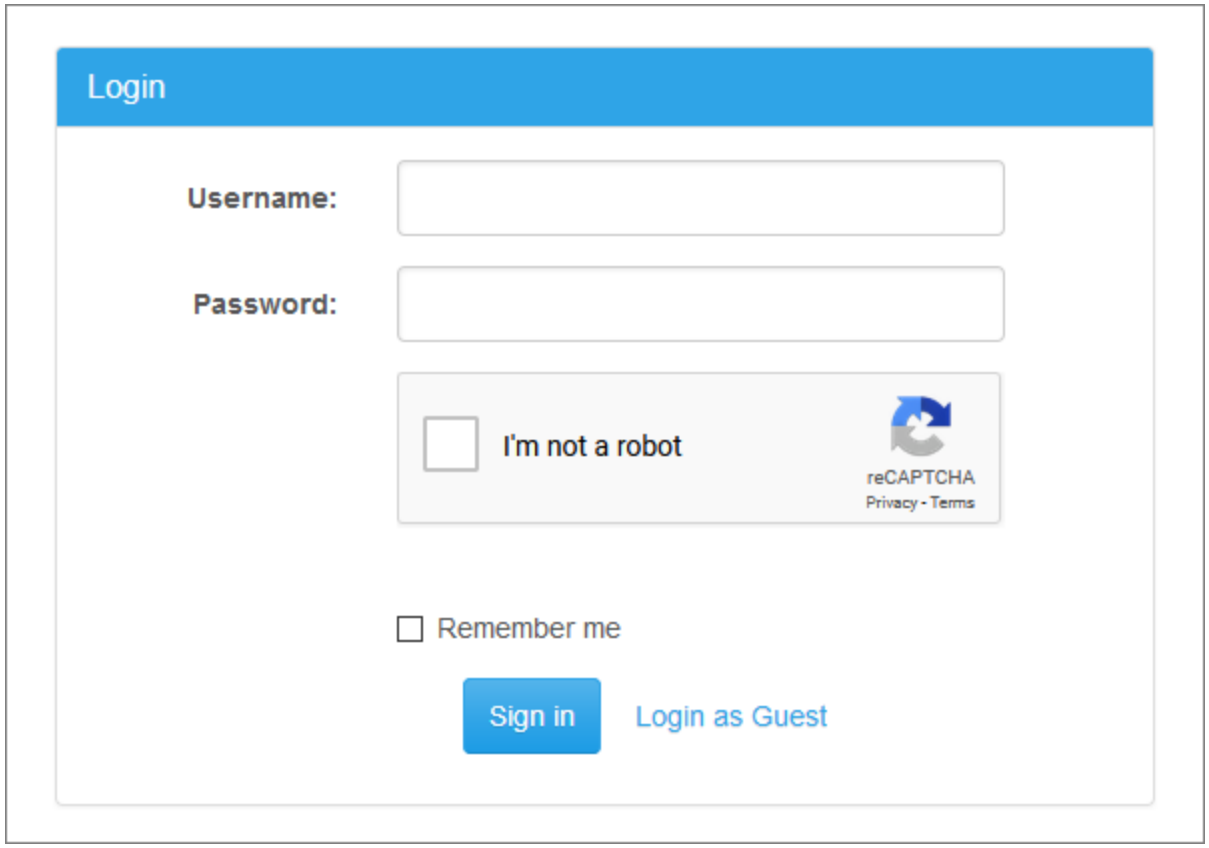
To use Google reCAPTCHA, register your web site at https://www.google.com/recaptcha/intro/index.html, copy the *site key* and *secret key*, and paste them into the respective fields in the settings popup.



Flash-based CAPTCHA on the Login page



Google reCAPTCHA on the Login page



Security screen articles:

- Security screen
- Login form appearance
- Two-factor authentication
- Registration and passwords
- Advanced security settings
- User group permissions
- Dynamic permissions
- Audit and record locking
- Encryption
- Session keys
- Active Directory
- Facebook connect

- Sign in with Google
- CAPTCHA on authentication pages

See also:

- CAPTCHA on Add/Edit page
- Security API
- Datasource tables screen
- Miscellaneous settings
- Page Designer
- Event editor

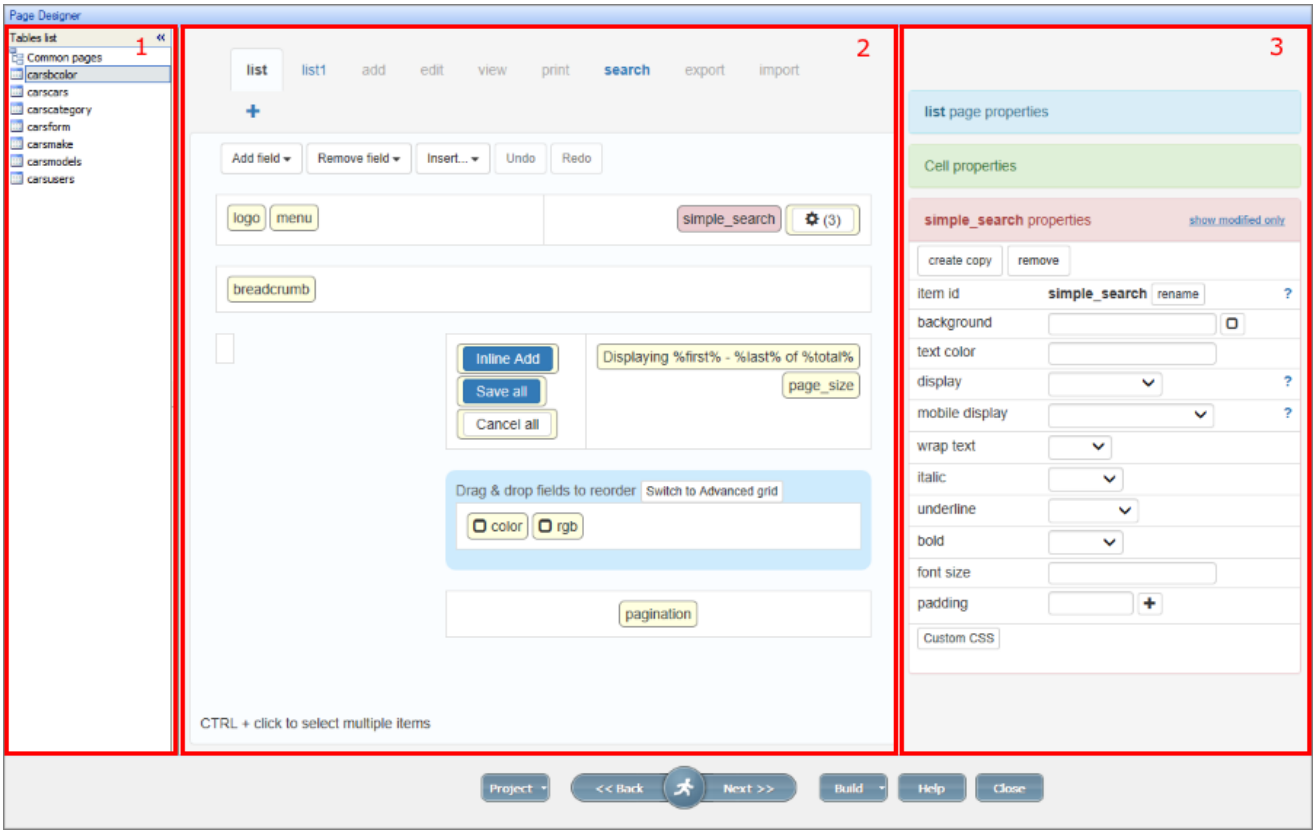
2.16 Page Designer

2.16.1 About Page Designer

What is Page Designer?

Page Designer is a grid-based WYSIWYG (*What You See Is What You Get*) editor to design pages in your app. You can modify the appearance of every page, insert elements such as buttons, code snippets, fields, tabs, and sections. **Page Designer** supports drag-n-drop feature, so you can move and rearrange the elements using the mouse.

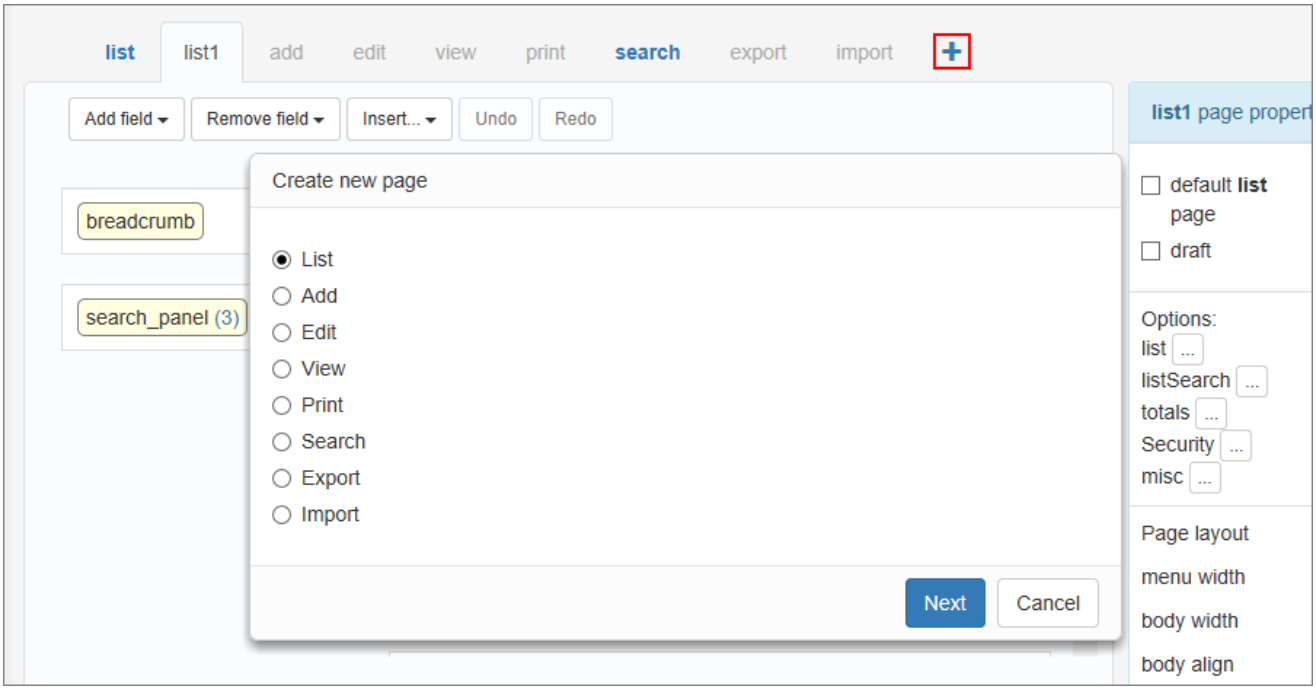
The **Page Designer** interface is divided into three parts:



1) **Tables list**. It shows all active tables and common pages. Select one to show a group of pages available in the app.

2) **Table pages**. It shows the visual editor for the group of pages associated with the selected element in the **Tables list**. You can select and design different types of pages: **List**, **Add**, **Edit**, **View**, **Print**, **Search**, **Export**, **Import**.

To add new pages click the '**+**' button next to the tabs on the top:



There is a group of buttons and controls available for the selected page under the **pages tabs**: **Add menu link**, **Add menu group**, **Insert**, **Undo** and **Redo**. You can read more about these elements in the articles Working with common pages, Working with table pages, and Tabs and sections.

> **Note**: you can learn more about the **Insert** element in the articles Insert custom button, Insert code snippet, Insert map, Insert Text / Image, Insert standard button.

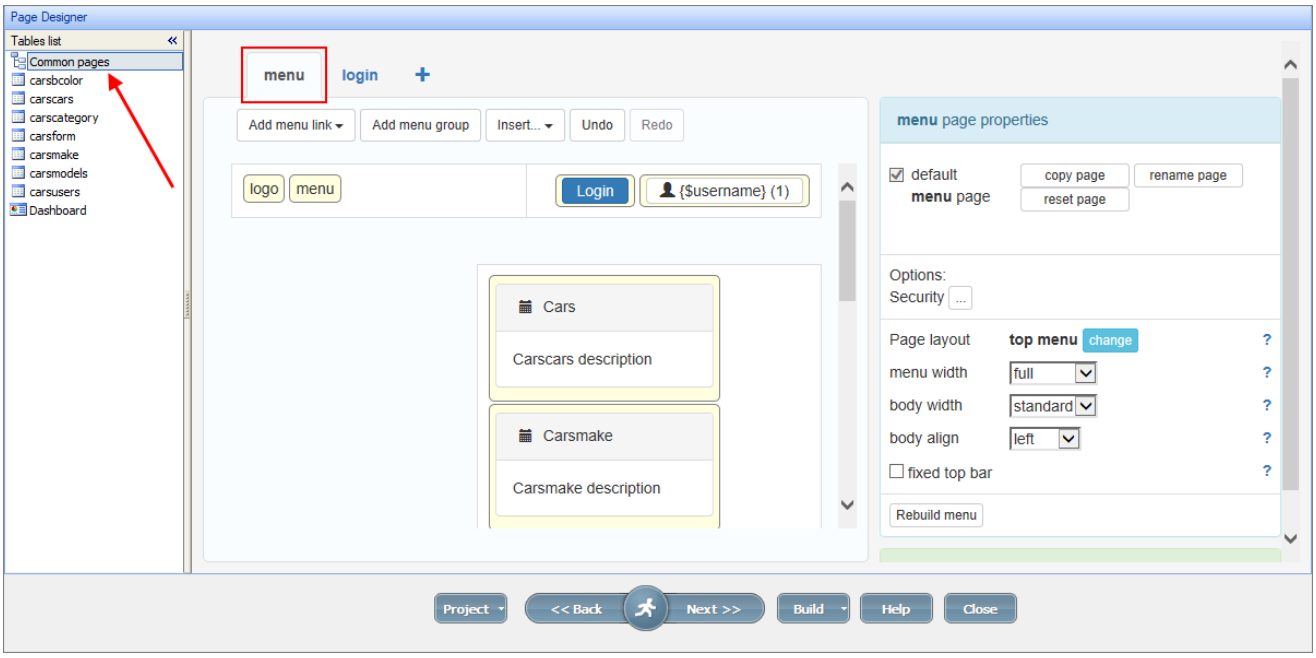
The central part of the screen is a **preview area**. It represents the layout of the page in the app. You can add new elements, click existing ones to select them and view their **properties**, or drag-n-drop elements between cells.

3) **Properties**. The properties of the selected elements are always shown on the right-side panel. You can edit the properties of pages, cells, and individual elements. Each type has a different set of options available.

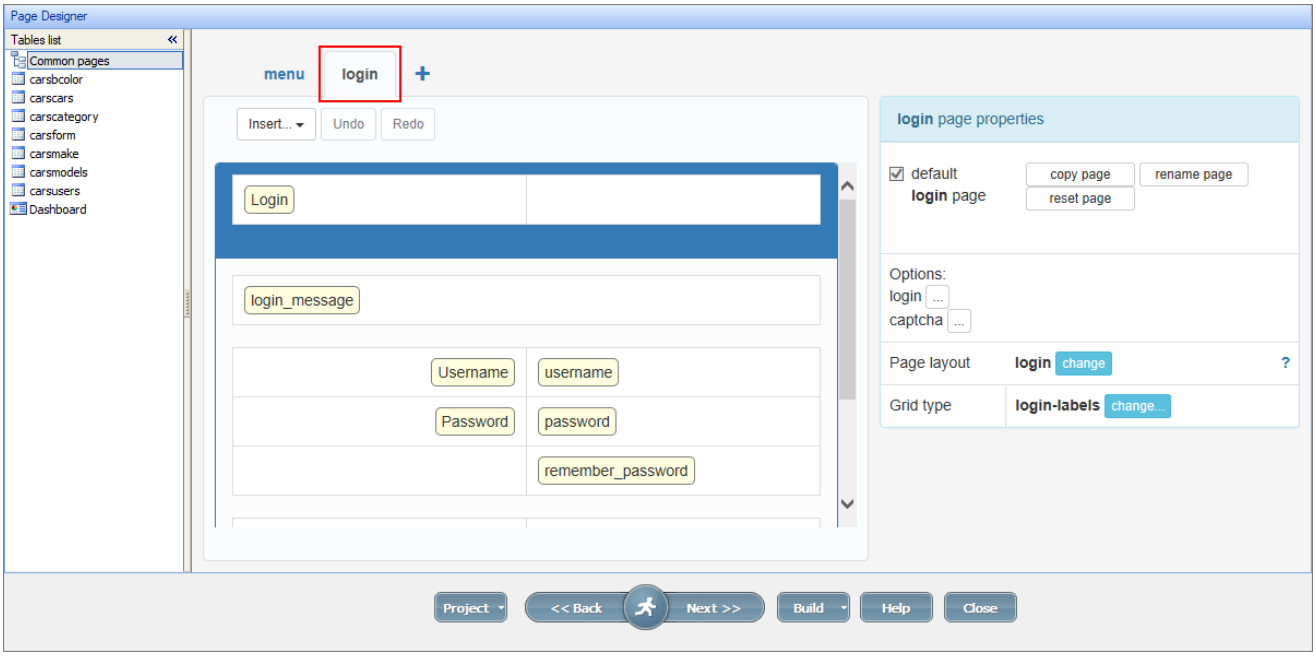
Using Page Designer

For example, let's see the **Page Designer** interface for the **menu** page. First, click on the *Common pages* in the **Tables list**. The menu tab opens, and the page preview appears in the central part of the screen.

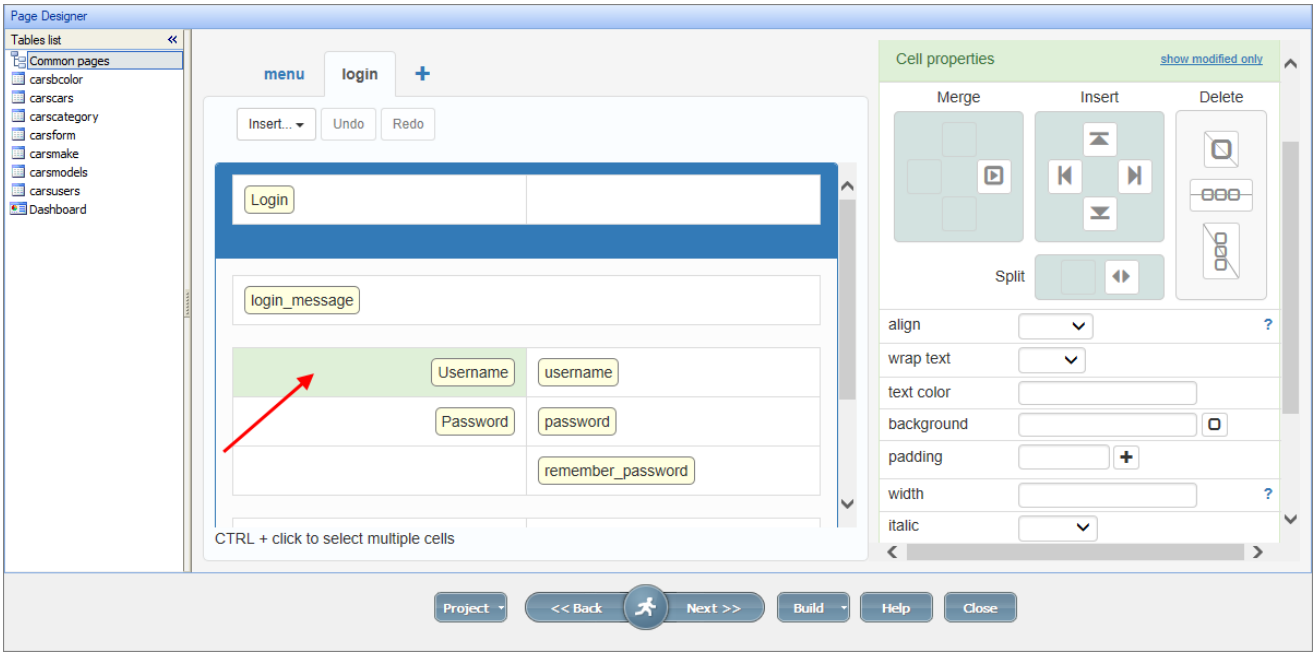
You'll see **menu page properties** and **Options** on the right side. You can edit these options for the **menu** page.



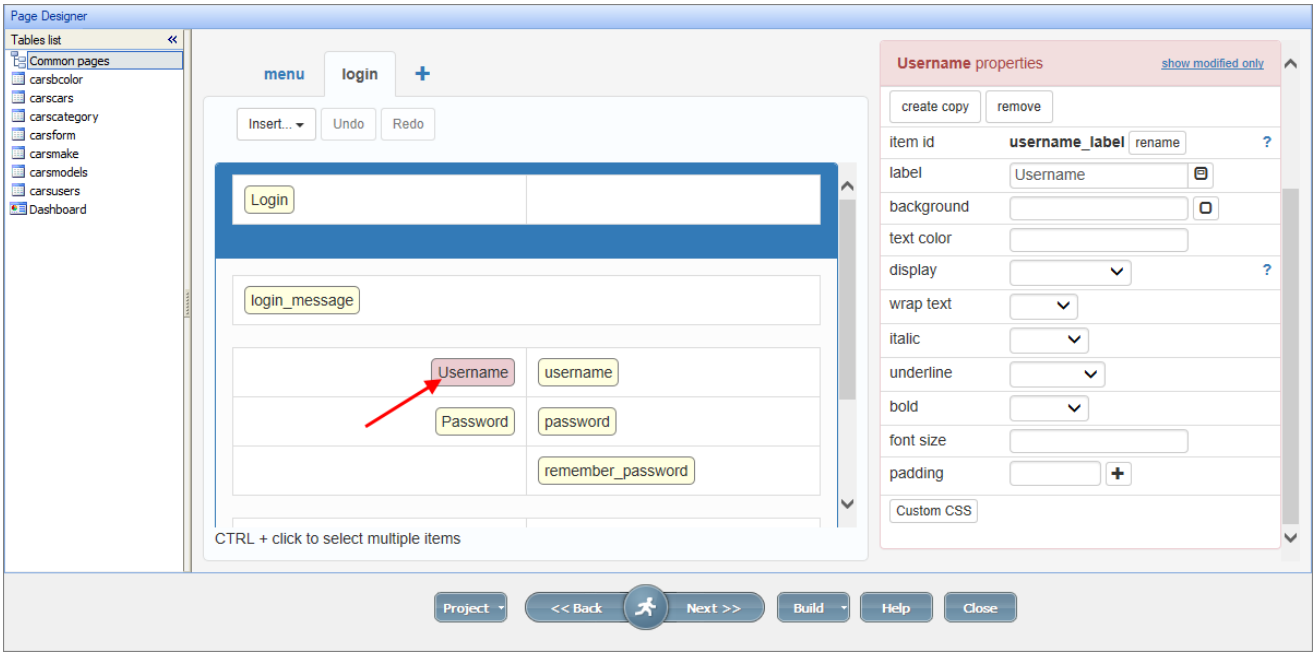
Click on the **login** tab to see the preview, properties, and options available for the **Login** page:



Click on the cell containing "*Username*". The selected cell becomes highlighted with light green fill, and you'll see **Cell properties** and options available for this cell on the right-side panel.



Now click on the "*Username*" element to see to the **properties** and options available for it. The selected element becomes highlighted with red fill.



Page Designer articles:

- Working with common pages
- Working with table pages
- Working with cells
- Working with page elements
- Working with additional pages
- Page layout and grid type
- Tabs and sections
- Insert custom button
- Insert code snippet
- Insert map
- Insert Text / Image
- Insert standard button
- "View as" settings
- "Edit as" settings
- "Filter as" settings

See also:

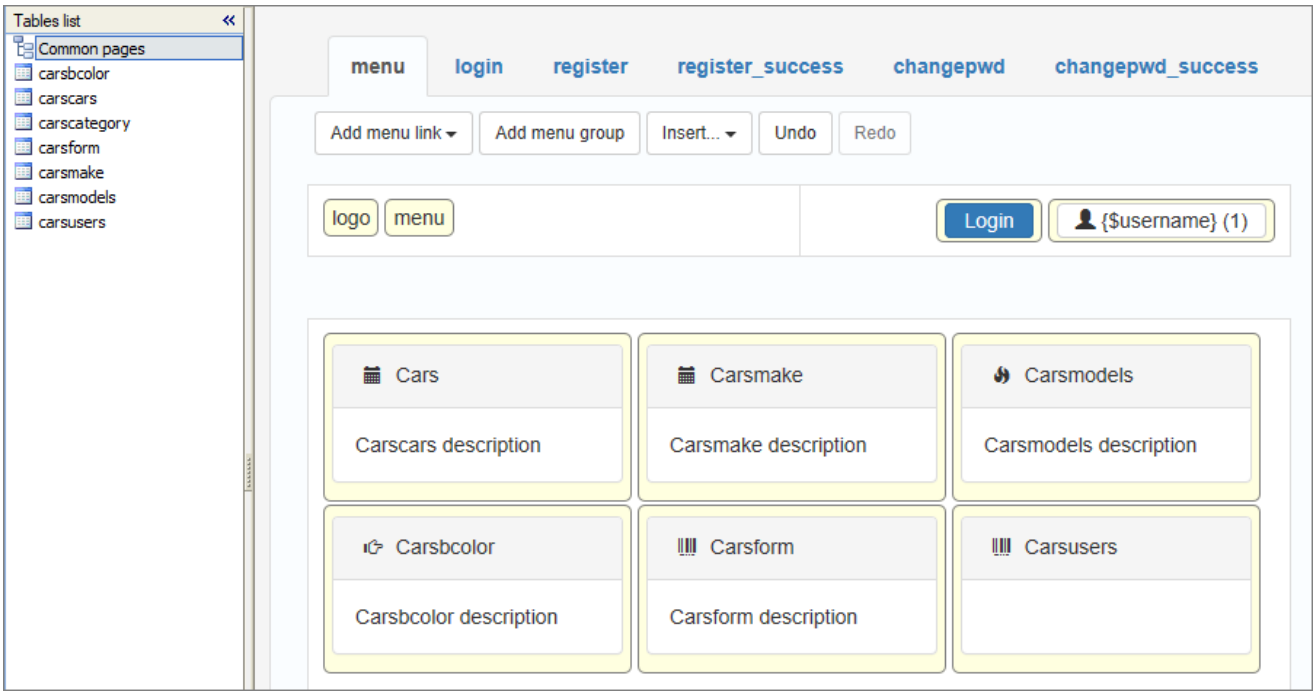
- Choose pages screen
- Choose fields screen
- Miscellaneous settings
- Editor screen
- Event editor

2.16.2 Working with common pages

Quick jump

Common pages

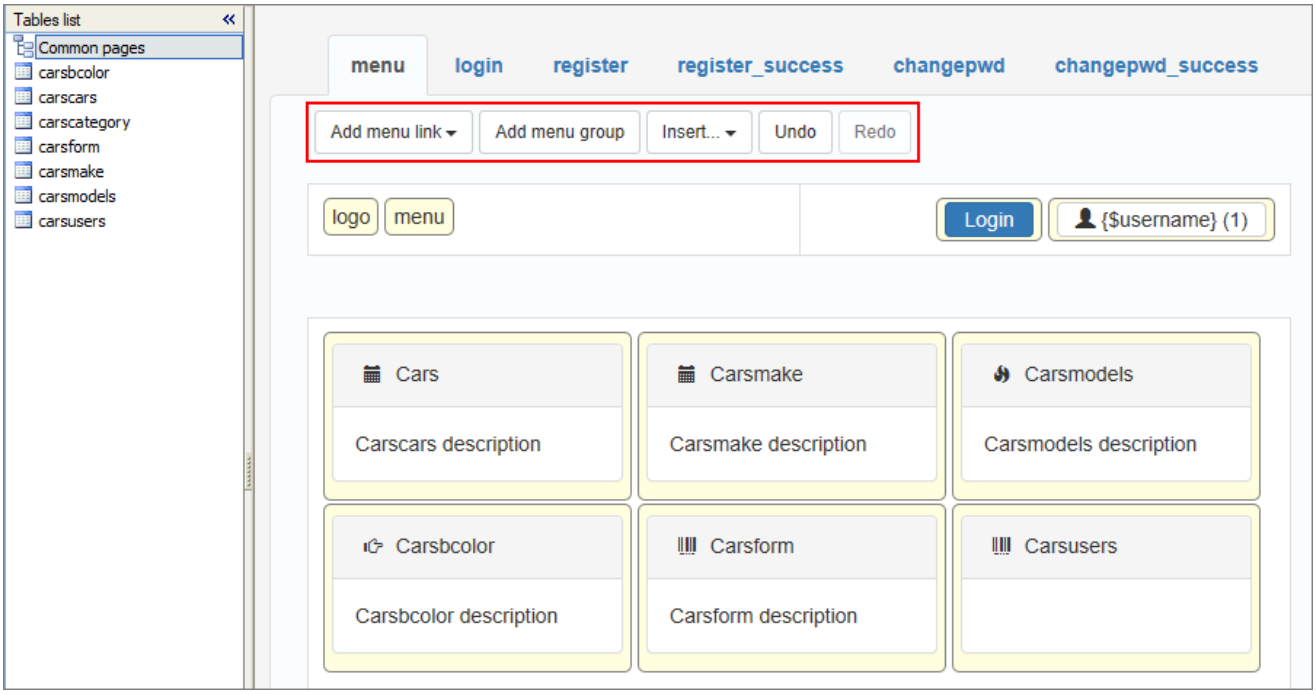
Common pages include **Menu**, **Login**, **Register**, **Change/Remind password** pages, and others. You can edit the layout of these pages with the **Page Designer**.



The pages, other than the **menu**, depend on the settings of the Security screen. You can also find that the **Security** screen contains most of the options for these pages. See Registration and passwords to learn more.

Working with the menu page

Use the **Add menu link**, **Add menu group**, **Insert** and **Undo/Redo** buttons to add or edit elements on the **menu** page:



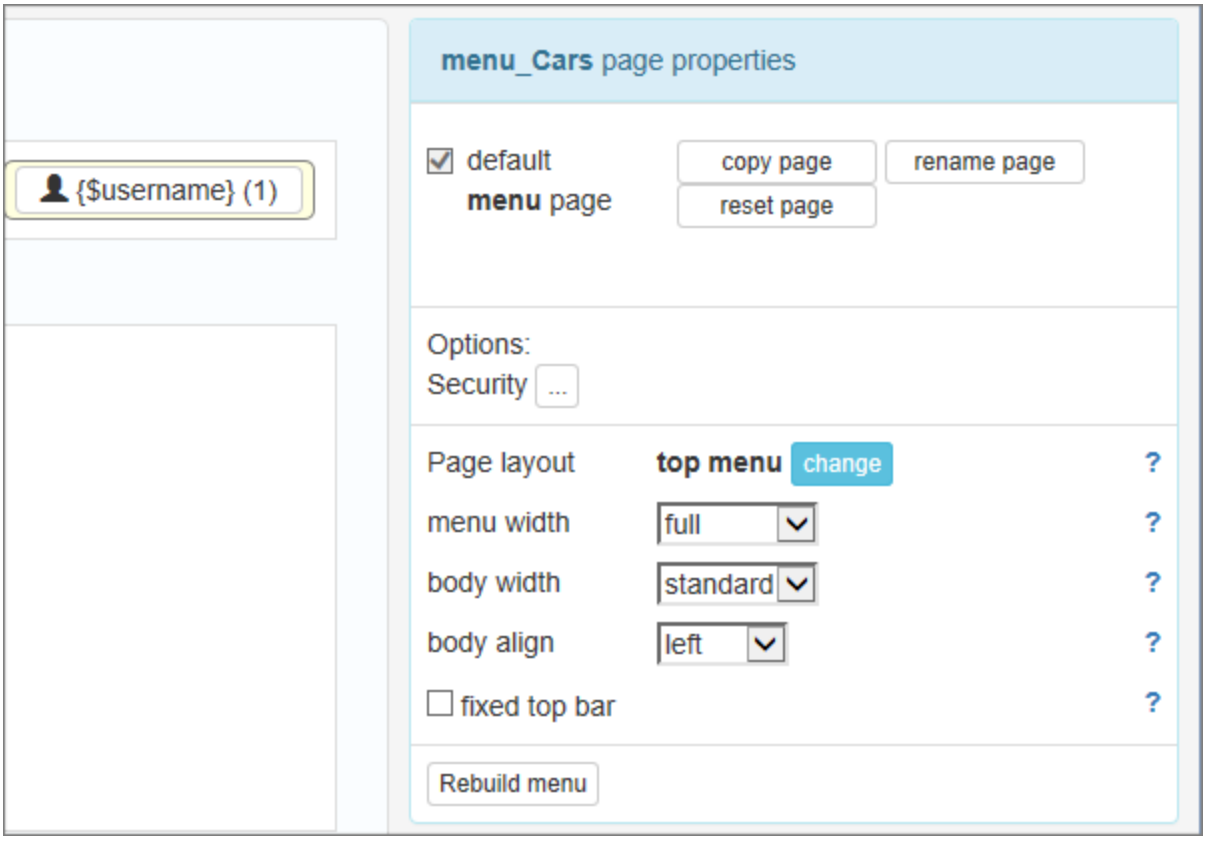
- Click the **Add menu link** button to add links to the selected project table. The added links appear on the generated **menu** page.
- Click the **Add menu group** button to add groups to the **menu** page. You can then add **menu links** into different groups. To do so, add the new menu group and then drag-n-drop the table link to the new group.

> **Note**: all **menu links** appear in the currently highlighted group in the **Page Designer** screen. If you need to move the link to another group, drag it to another group or cell.

- Click the **Insert** button to add buttons, snippets, maps or HTML texts and images to the page. See Insert button, Insert code snippet, Insert map, Insert Text / Image, Insert standard button to learn more.
- Use **Undo** and **Redo** buttons to undo/redo the changes on the page.

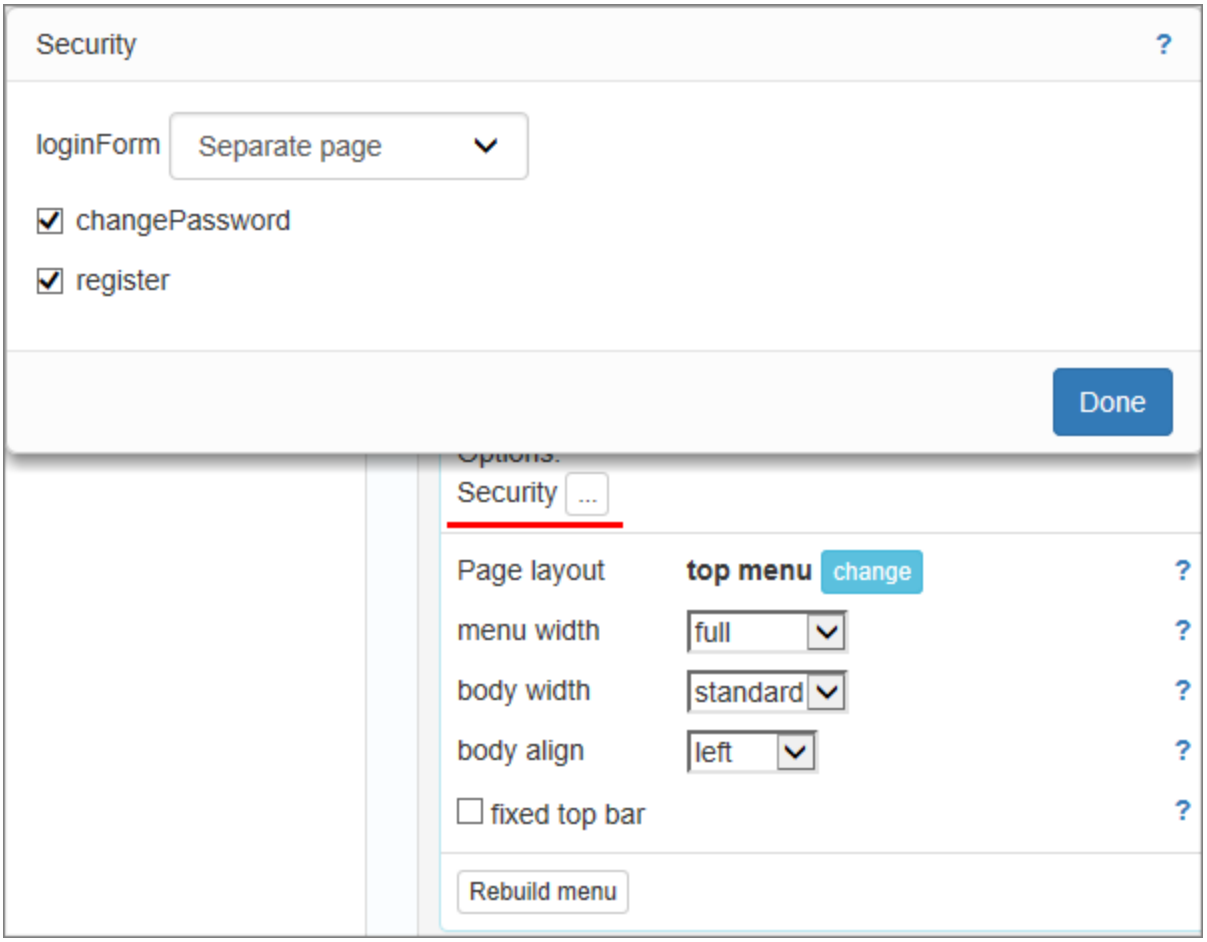
Menu Page Properties

You can see the **menu page properties** and **Options** on the right-side panel:



You can learn more about **page properties** in the article Working with table pages.

The **menu** page has only one option - **security**:



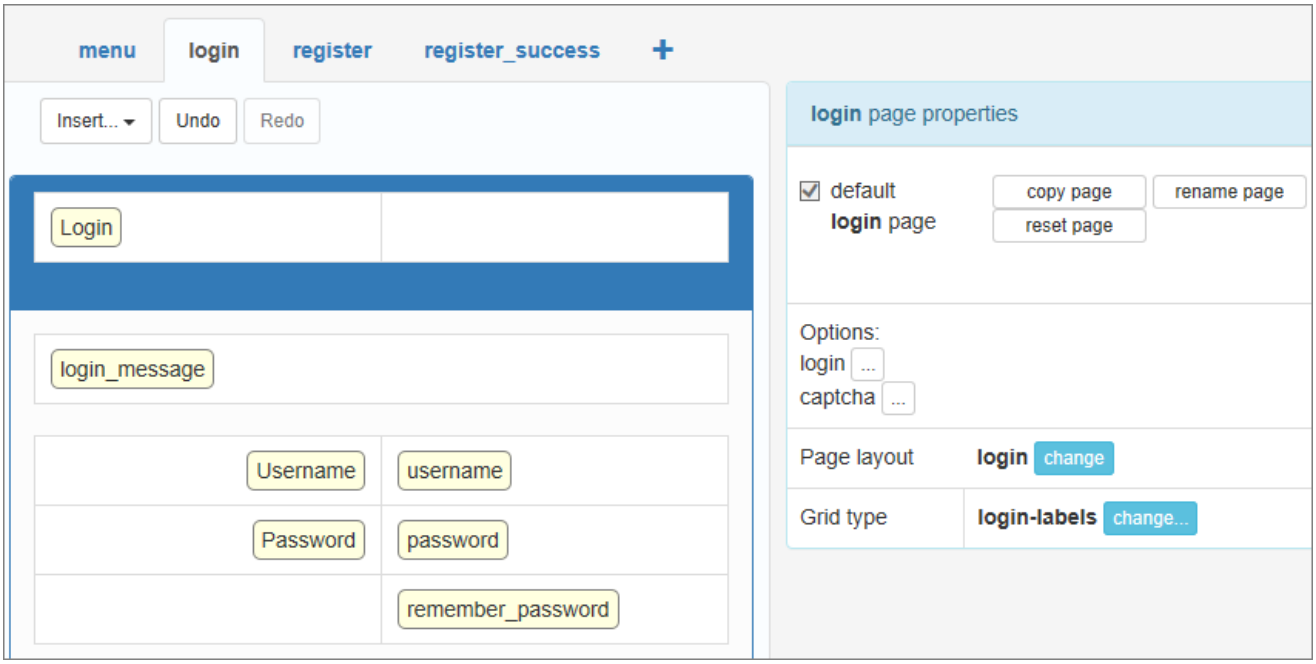
Press the **Security** button to choose the way to display the login form.

If you have enabled the **change password** and **register** page in the Security screen, you can also turn on/off the **change password** and **register** pages within the **Security** option popup. These options are also available in the user login settings.

Use the **Rebuild menu** button to reorder **menu groups** and **links** so that the groups and the links are neatly organized in separate cells.

Login and Register

Login page

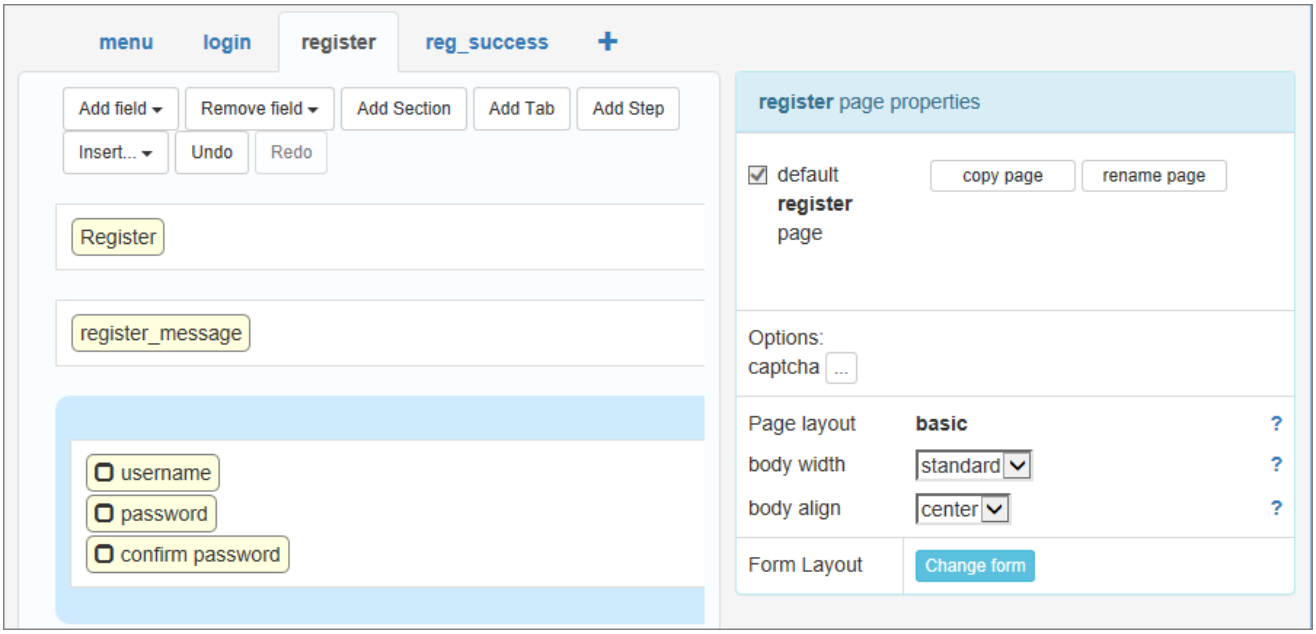


The **login** page has two options: you can choose to enable the Two-factor authentication with the *login* option and add Captcha with the *captcha* option.

You can change Page layout and Grid type in the **login page properties**. You can also add a new field to the **login** page via the **Add field** menu.

> **Note**: you can change the appearance of the login page on the **Security** screen.You can choose between standalone login page, embedded login form and login page in popup.

Register page



The **Register** page only has the captcha option.

You can also select the preferred body width and body align, and change the Form layout.

> **Note**: you can find more settings for the **Register** page on the Registration and passwords page of the Security screen.

Auxiliary pages

Auxiliary pages include **Registration success**, **Change password**, **Change password success**, **Remind password**, and **Remind password success** pages.

You can edit the cells, fields, buttons, and other elements, or even insert code snippets, text and images, standard and custom buttons.

You can find more settings for the auxiliary pages on the Registration and passwords page of the **Security** screen.

Page Designer articles:

- About Page Designer
- Working with table pages
- Working with cells
- Working with page elements
- Working with additional pages
- Page layout and grid type
- Tabs and sections
- Insert custom button
- Insert code snippet
- Insert map
- Insert Text / Image
- Insert standard button
- "View as" settings
- "Edit as" settings
- "Filter as" settings

See also:

- Choose pages screen
- Choose fields screen
- Miscellaneous settings
- Menu builder
- Editor screen
- Event editor

2.16.3 Working with table pages

Quick jump

Designing table pages

Page properties

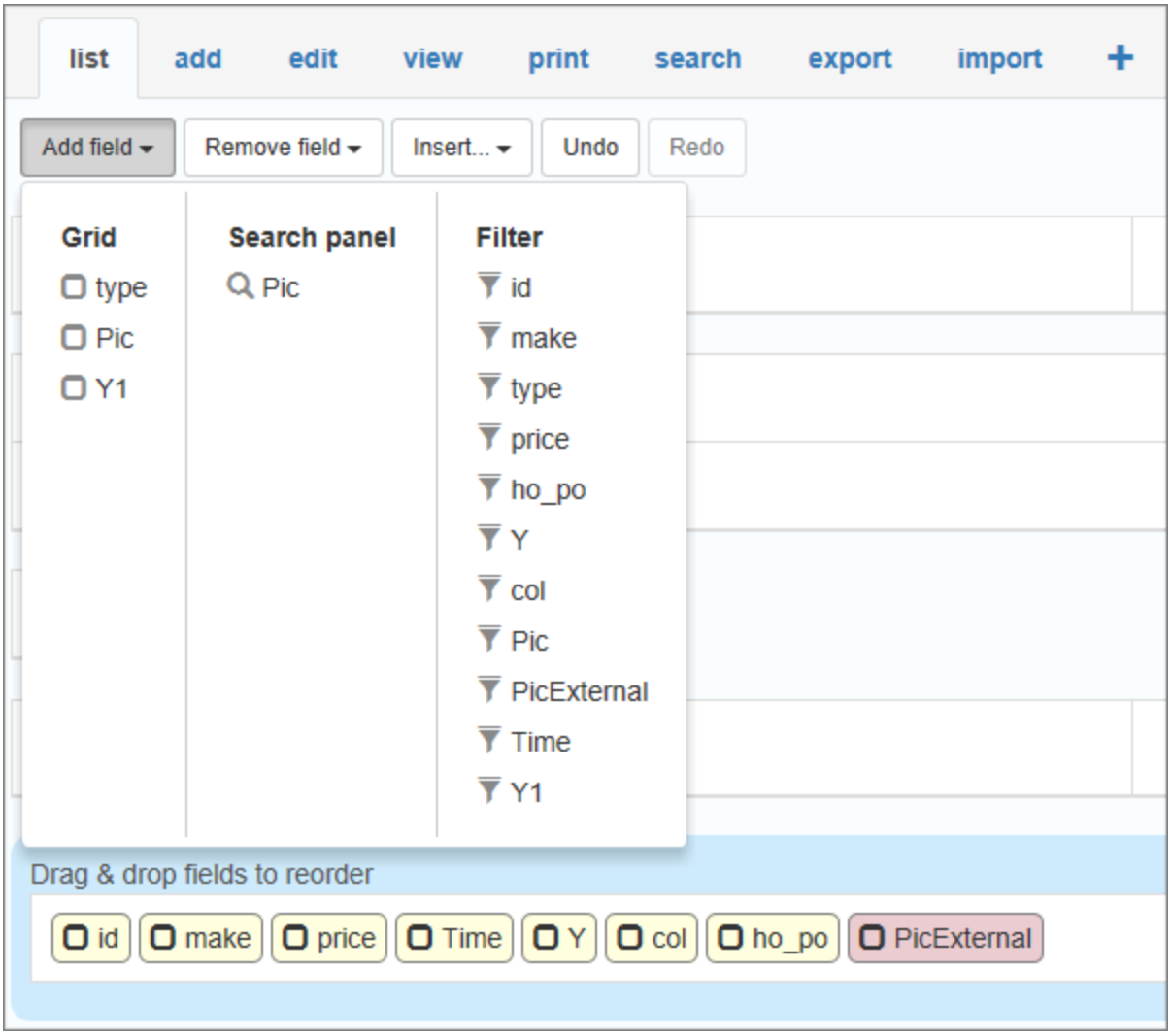
Page options

Page layout

Designing table pages

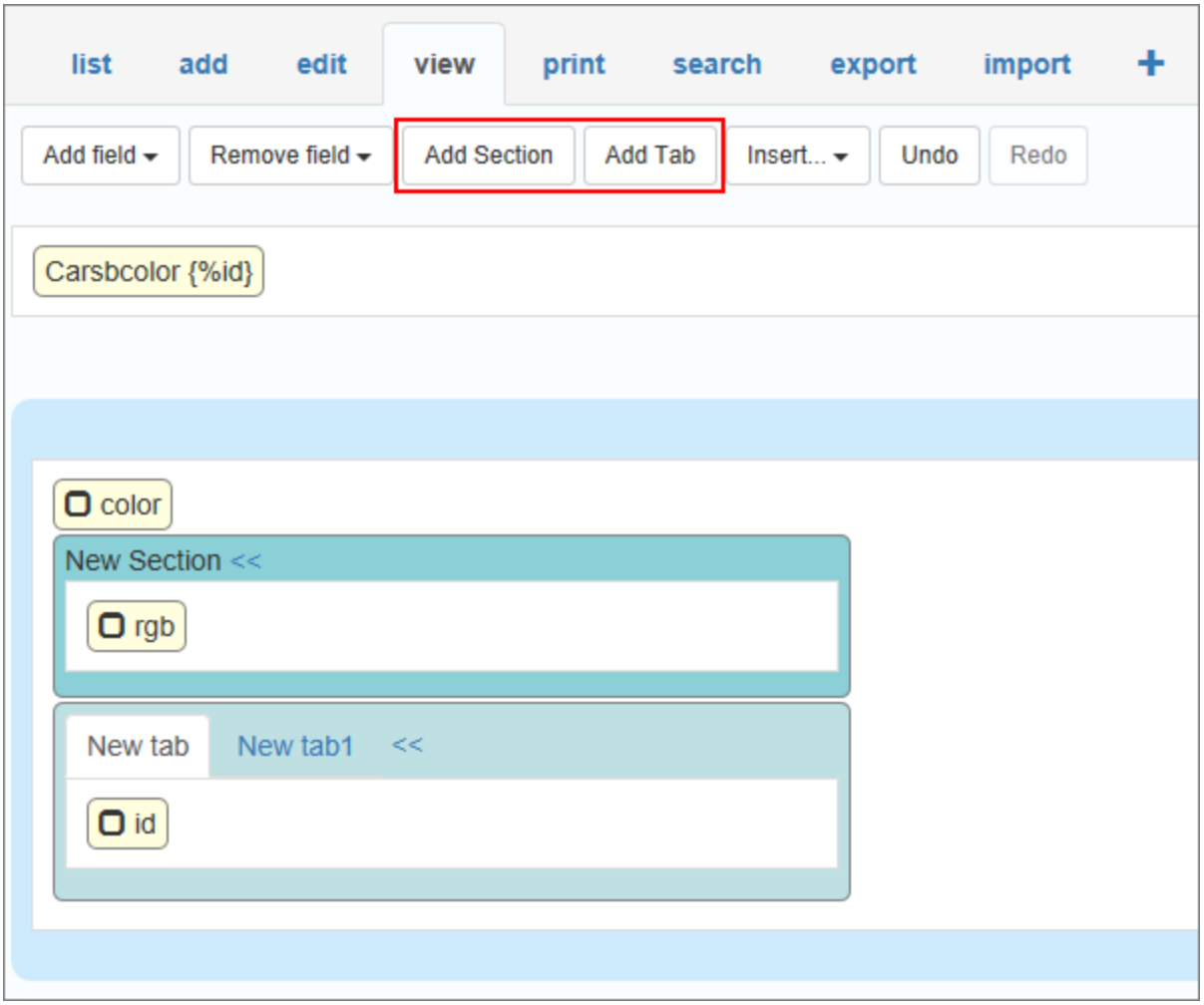
Table pages are the pages for working with tables. **List**, **Add**, **Edit**, and **View** are examples of **Table pages**. To learn more about the table pages, see Choose pages screen.

Add field, **Remove field** buttons are available for all types of pages. Use these buttons to add table fields to the page grid, filter, or search panels. Click on the field name to add or remove it.

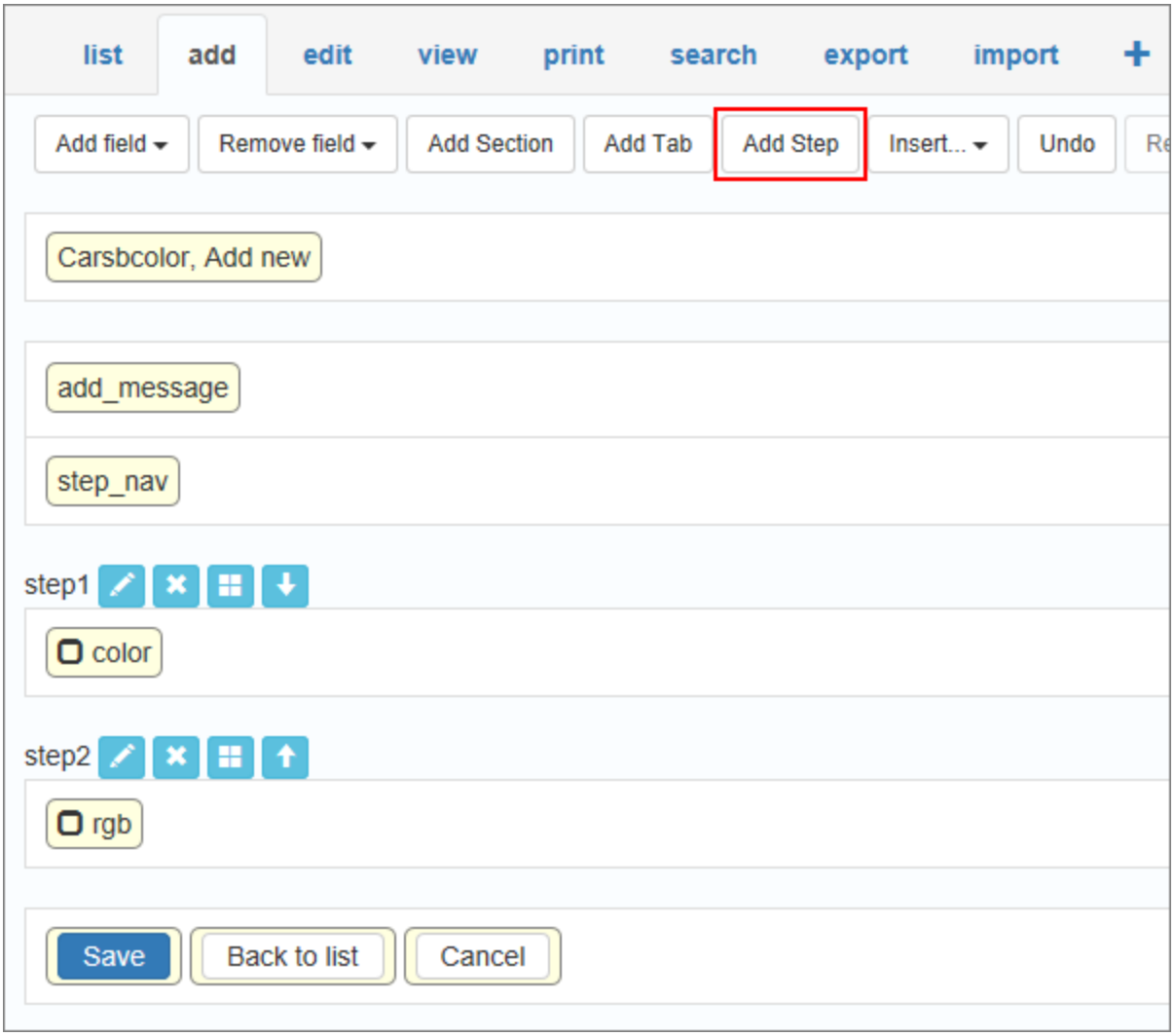


Add Section/**Add Tab** buttons are available for the **Add**, **Edit**, and **View** pages. Click these buttons to add a section or a tab to the preview area. Drag-n-drop fields to the new **section**/**tab**.

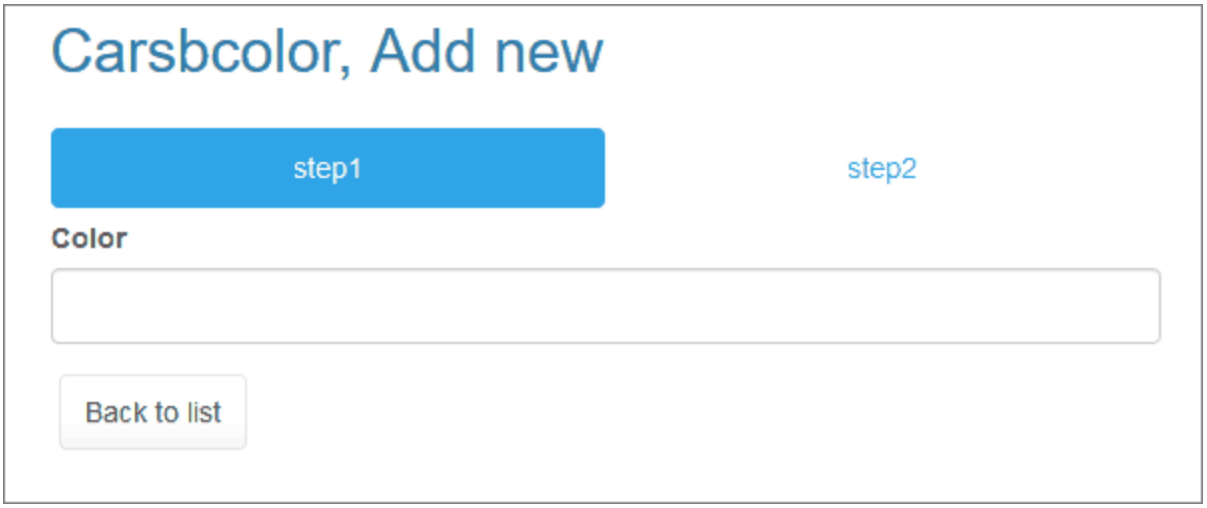
To rename or format the **section**/**tab**, click on it, and modify its **properties**. See Working with elements for details.



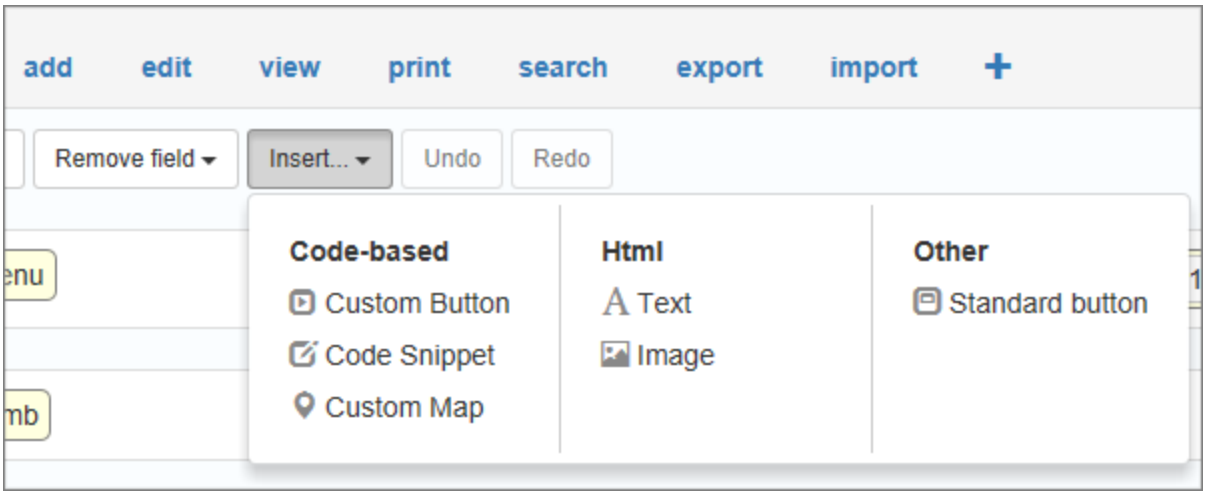
Add step button is available for the **Add**/**Edit** pages. It allows dividing the **Add**/**Edit** process into several steps.



Here is how a two-step **Add** page looks like:



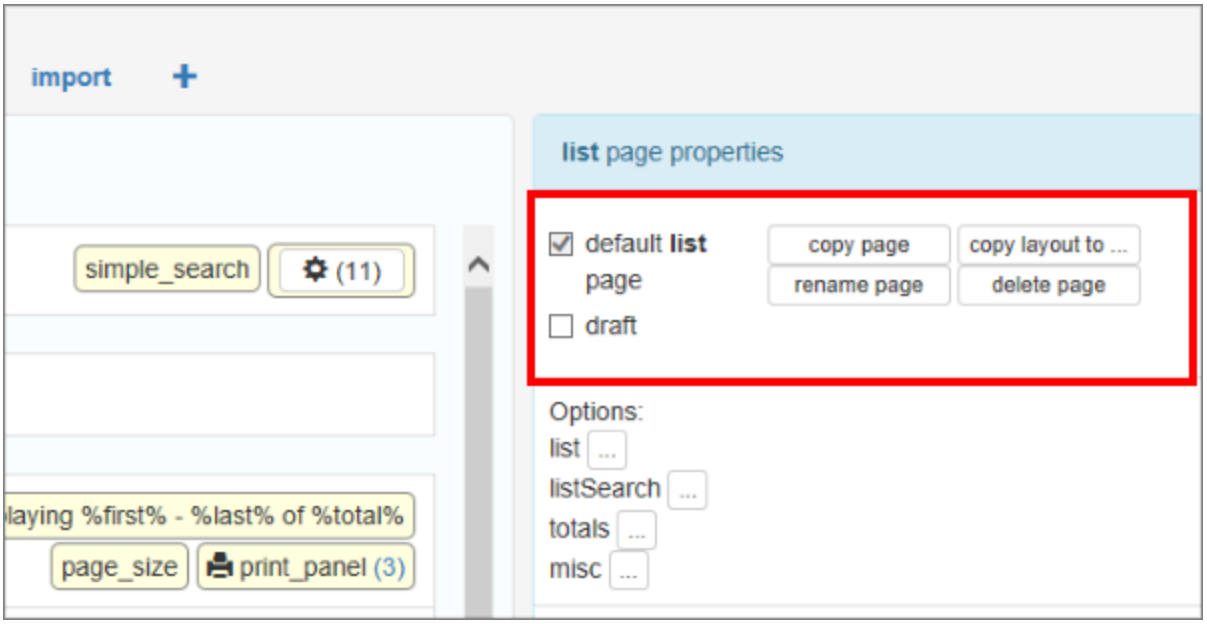
Insert button is used to add buttons, snippets, maps or HTML texts and images to the page. See Insert custom button, Insert code snippet, Insert map, Insert Text / Image, Insert standard button to learn more.



Use **Undo** and **Redo** buttons to undo/redo last change on the page.

Page properties

You can find **page properties** on the right-side panel. They are the same for all types of pages.



Default/Draft page checkbox.

Page Designer lets you create multiple pages of the same type. These **Default/Draft** options let you choose whether the selected page is the default one or a draft.

You can only have one **default** page of each type: **List**, **Add**, **Edit**, etc. The elements that link to the page type send the user to the default page unless you change it in the **page** option in the properties of these elements. For example, if you create an 'Edit1' page and make it default, the **Edit** button on the list page will lead the user to the 'Edit1' page instead of 'Edit'. See Working with additional pages to learn more.

The **draft** option lets you hide the selected page in the generated application.

Page management buttons:

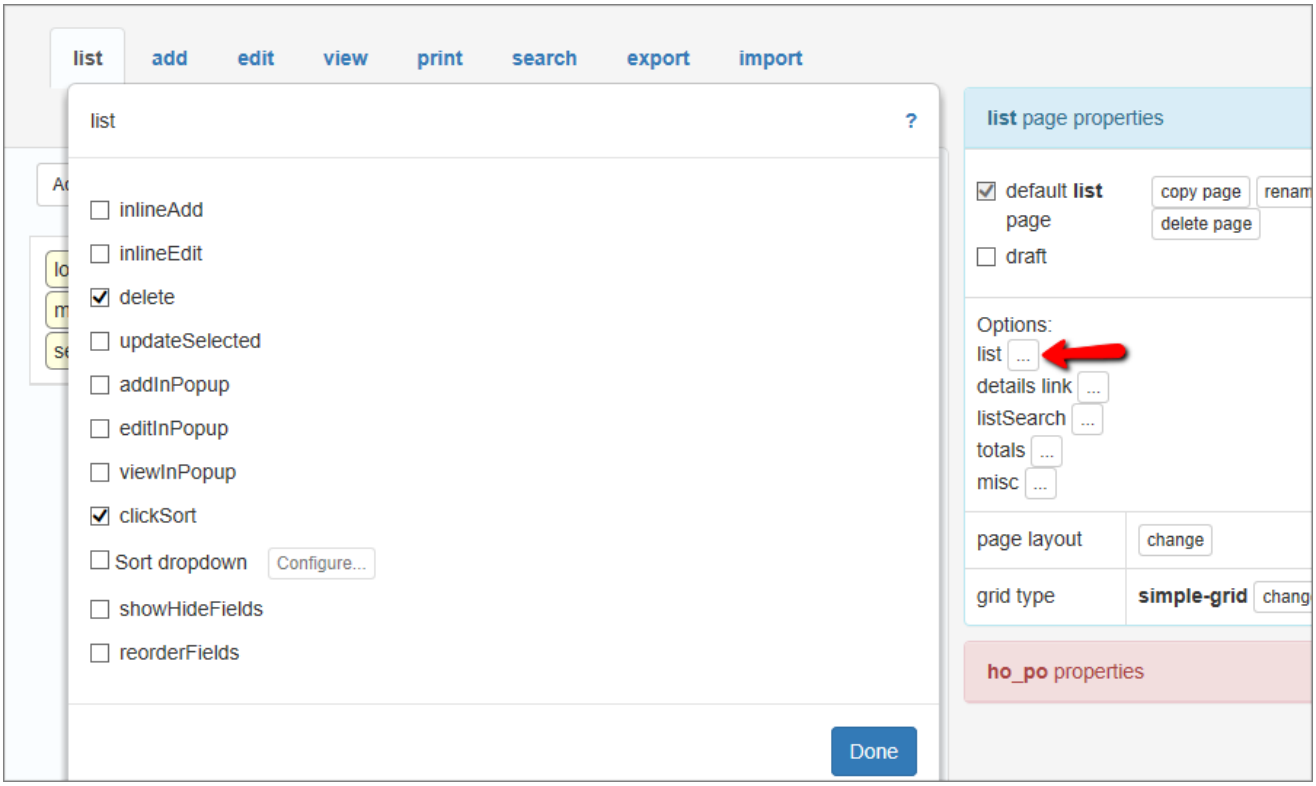
- **Copy page** button copies the current page. The auto-incremented number is added to the new page name.
- **Copy layout to ...** button copies the current page layout to the selected page.
- **Rename page** button renames the current page.
- **Delete page** button deletes the current page.

Page Options

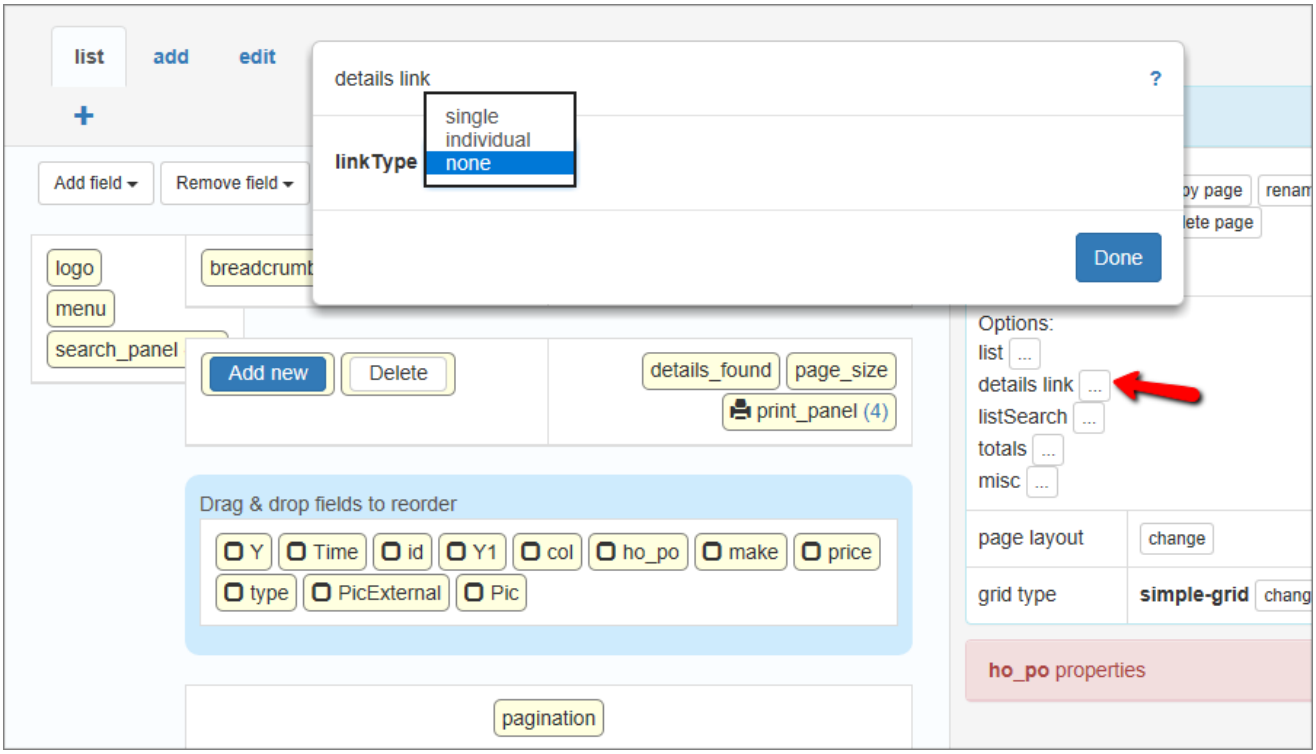
Under the **page properties**, you can see **options** available for the page. For example, the **List** page has options to control **Inline Add/Edit**, details link, listSearch, or Totals.

Add and **Edit** pages have the button to enable captcha on the generated page.

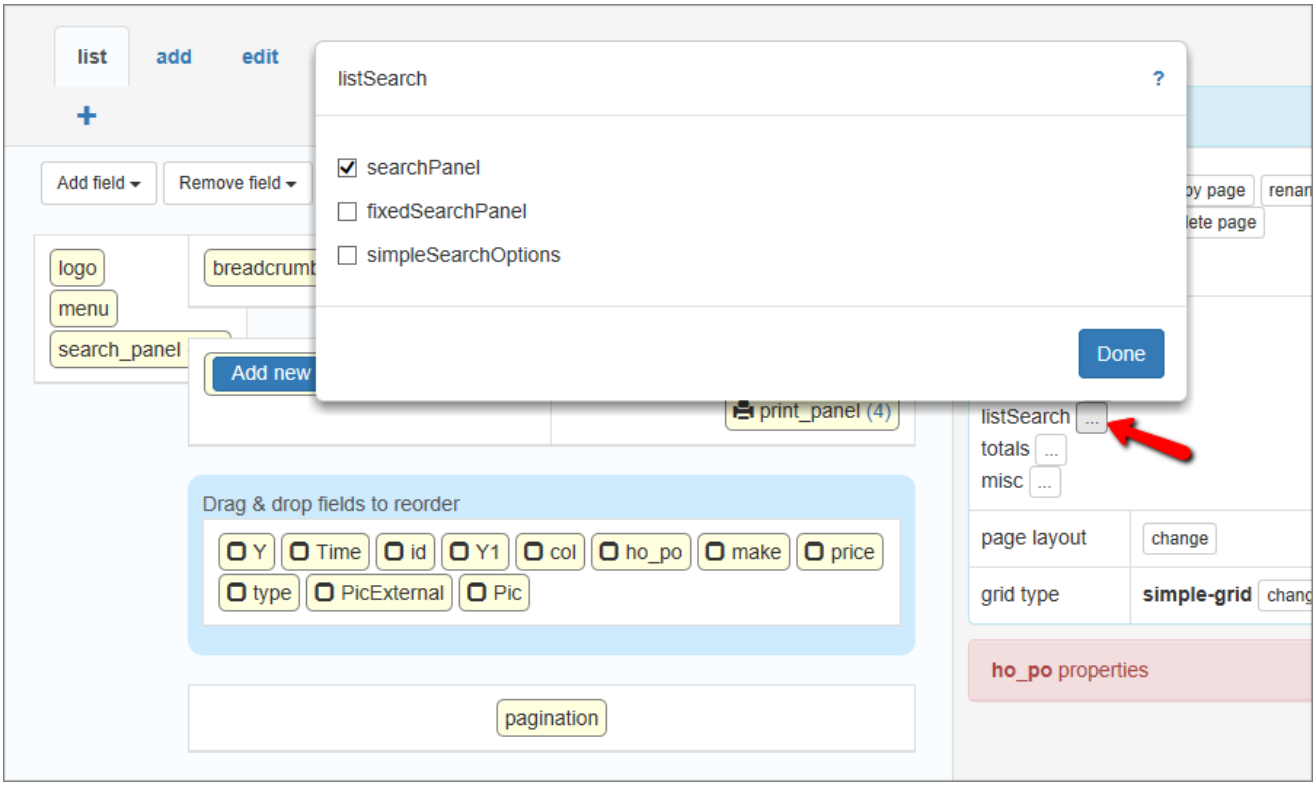
List options



Details link options

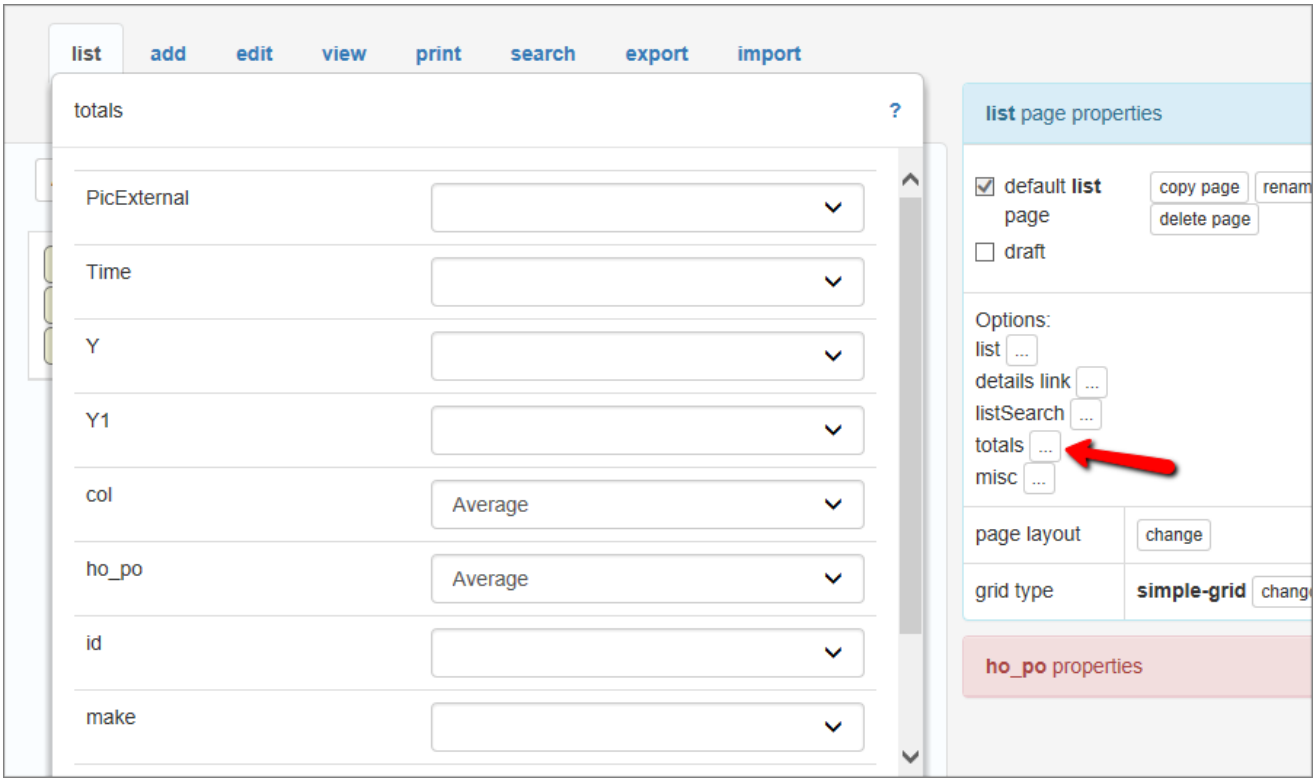


List search options

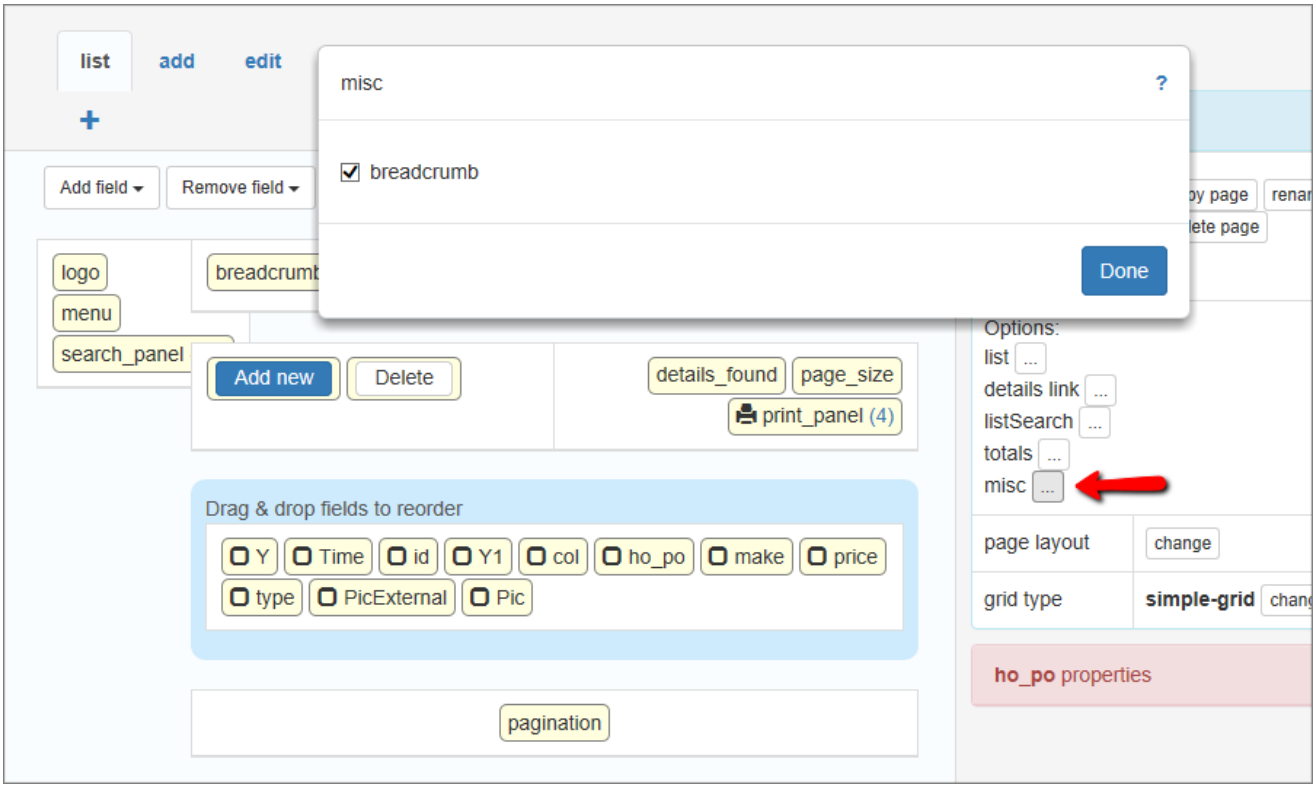


Totals options

Click on this control when you want to add totals functions to the fields. You can also add the totals for each field in element's properties - Totals.

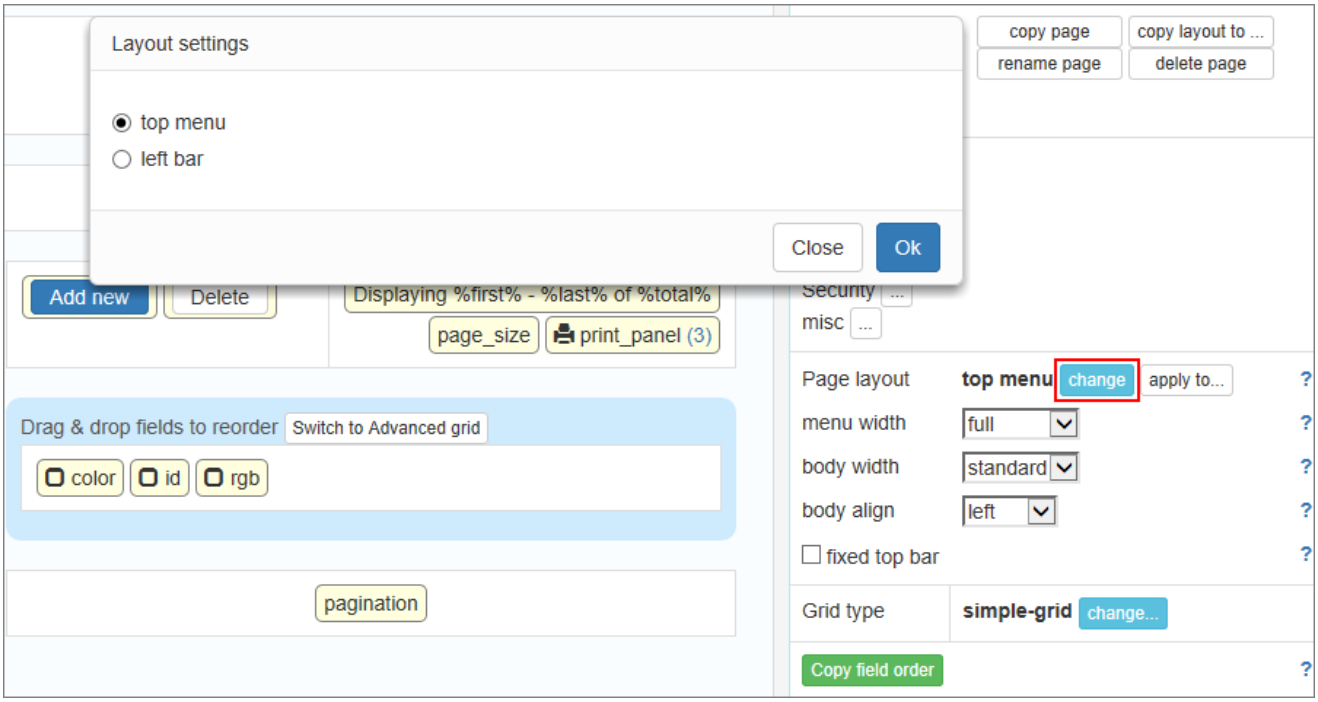


Misc options



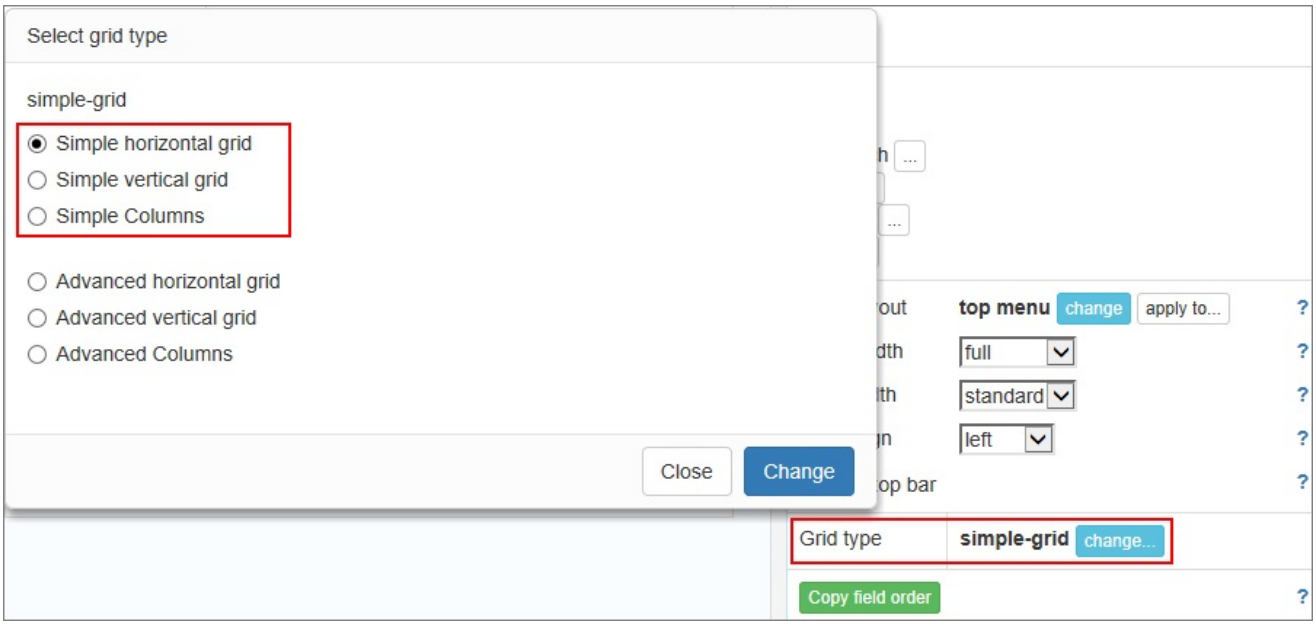
Page layout

Press this button to open dialog with Layout settings available for the current page.



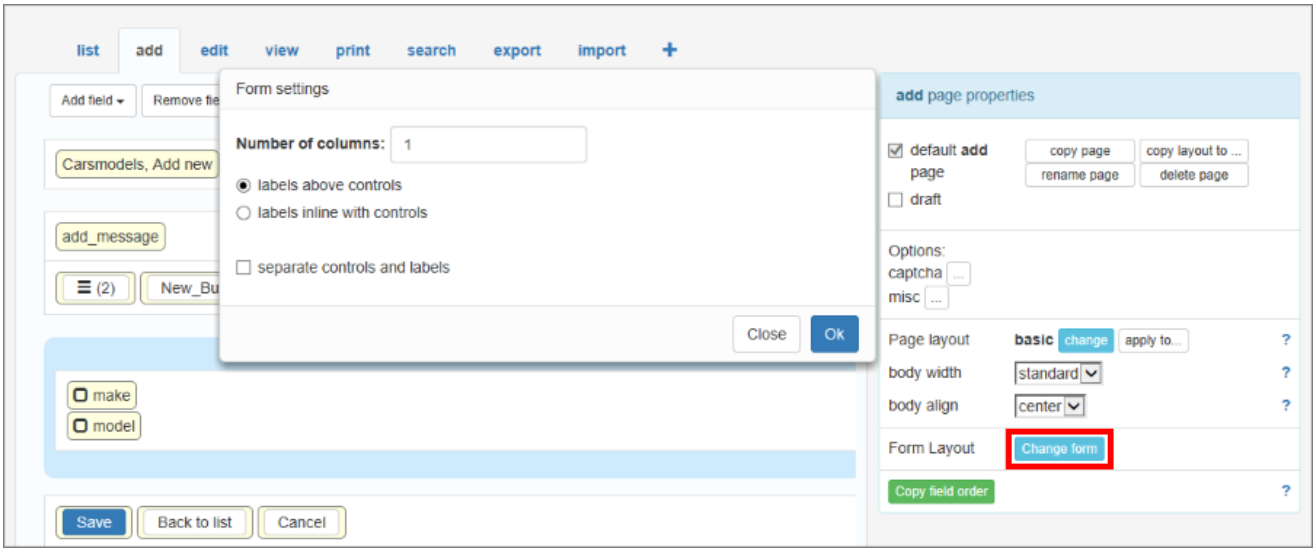
Page grid type

Press this button to open dialog with Grid types available for the current page.



Form layout

You can click the **Change form** button on the **Add/Edit** page, to set the numbers of columns and the position of labels. You can also choose the **separate controls and labels** layout.



Page Designer articles:

- About Page Designer
- Working with common pages
- Working with cells
- Working with page elements
- Working with additional pages
- Page layout and grid type
- Tabs and sections
- Insert custom button
- Insert code snippet
- Insert map
- Insert Text / Image
- Insert standard button
- "View as" settings

- "Edit as" settings
- "Filter as" settings

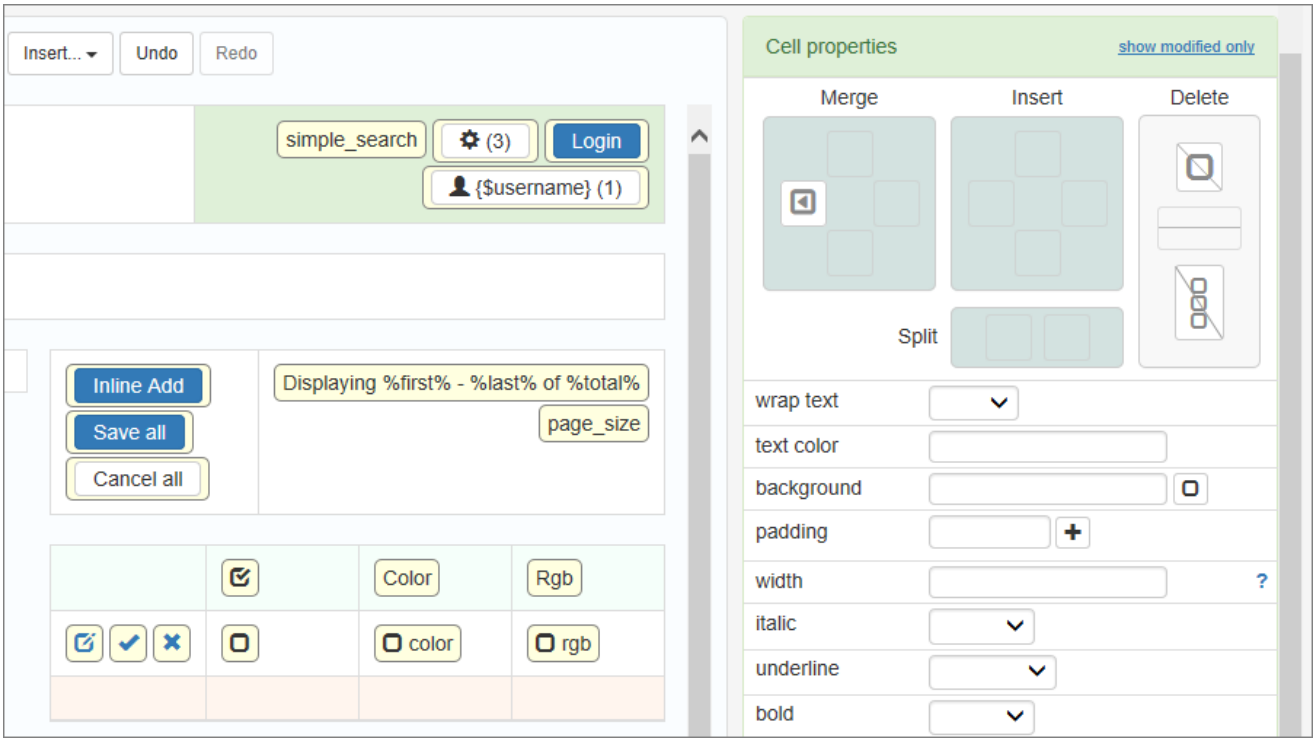
See also:

- Master-details relationship between tables
- Choose pages screen
- Choose fields screen
- Miscellaneous settings
- Editor screen
- Event editor

2.16.4 Working with cells

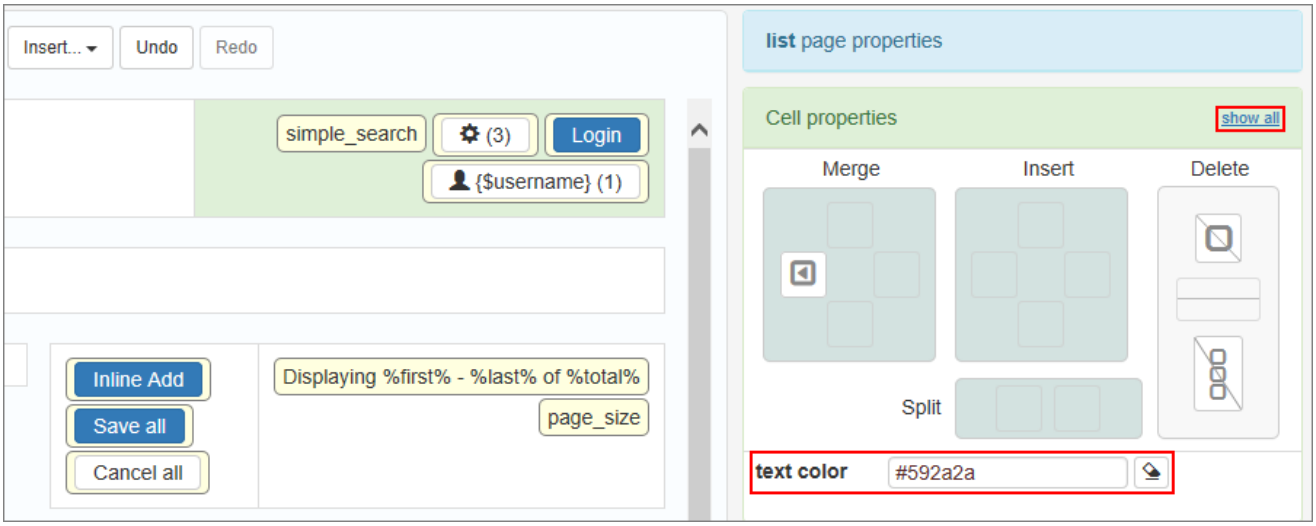
Cell properties

When you click on one of the cells in the **Page Designer**, you can see the **Cell properties** on the right-side panel. You can manage settings for the selected cell here.



Note: you can click on two or more cells while holding the CTRL button. In that case, the right-side panel shows only the common properties of all the selected cells. Click again on a single cell to disable multiple selection.

Click the **'Show modified only'** link to see only the modified cell properties. Click **'Show all'** to switch back to the default view.



Merging cells

If you need to merge cells, click on one of the cells to highlight it and make the **Cell properties** appear on the right panel. Now press one of the **Merge direction** buttons. Items within the merging elements are put into the resulting single cell.

Inserting cells

Click on the cell to show its **properties**. Press one of the **Insert direction** buttons to insert a new cell in the selected direction relative to the original.

Deleting cells

Click on the cell to show its **properties**. Press one of the **Delete** buttons to delete a single cell, the whole row or the whole column of cells.

Splitting cells

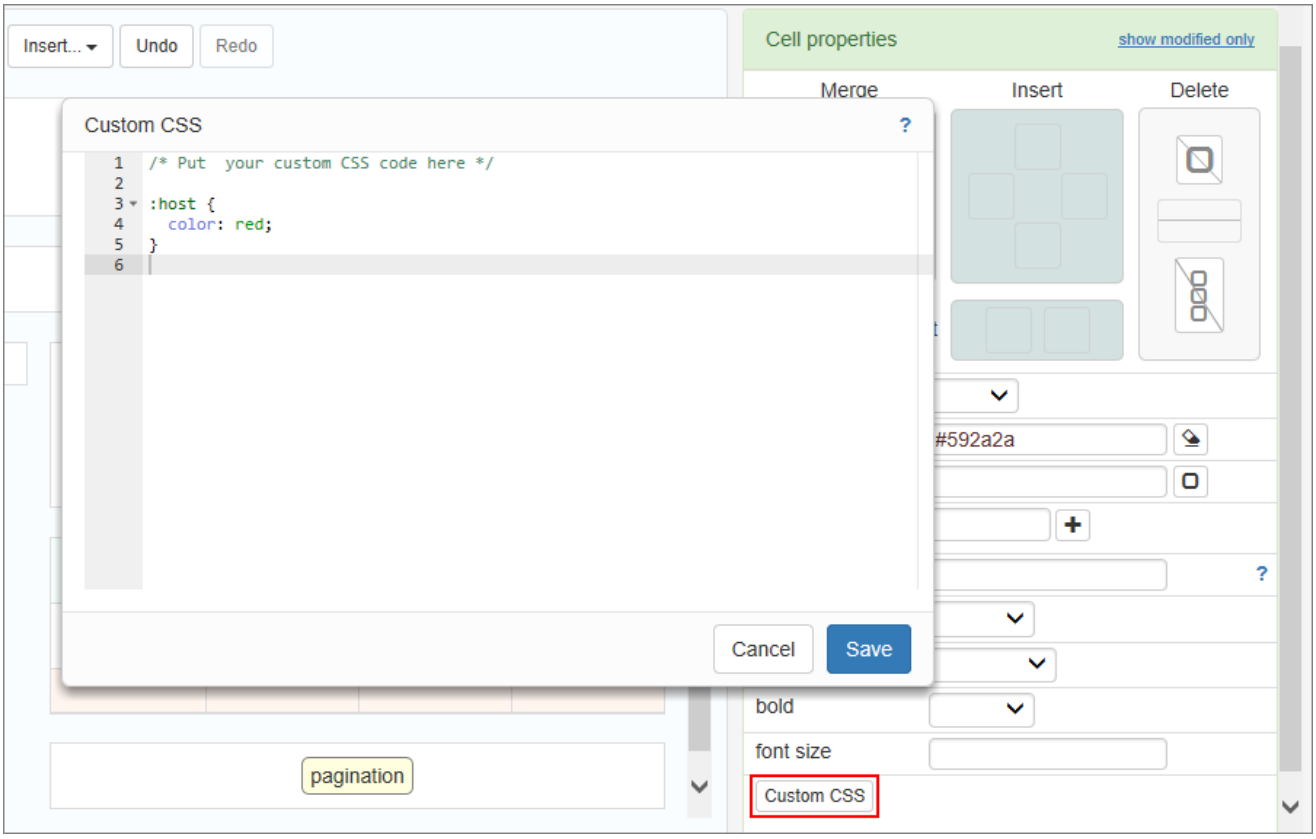
Click on the cell to show its **properties**. Press one of the **Split** buttons. You can split a cell vertically or horizontally.

Formatting cells

Click on the cell to show its **properties**. Use the formatting controls in the right-side panel to change the settings of the selected cell.

Custom CSS

Press the **Custom CSS** button to open the **Custom CSS** window. You can add the CSS code to the cell.



Note: you can add custom CSS to multiple cells at once. Press CTRL and select multiple cells, then click the **Custom CSS** button to add the custom CSS code to the selected cells.

Learn more about Customizing CSS.

Page Designer articles:

- About Page Designer
- Working with common pages
- Working with table pages

- Working with page elements
- Working with additional pages
- Page layout and grid type
- Tabs and sections
- Insert custom button
- Insert code snippet
- Insert map
- Insert Text / Image
- Insert standard button
- "View as" settings
- "Edit as" settings
- "Filter as" settings

See also:

- Customizing CSS examples
- Choose pages screen
- Choose fields screen
- Miscellaneous settings
- Editor screen
- Event editor

2.16.5 Working with page elements

Quick jump

Totals type

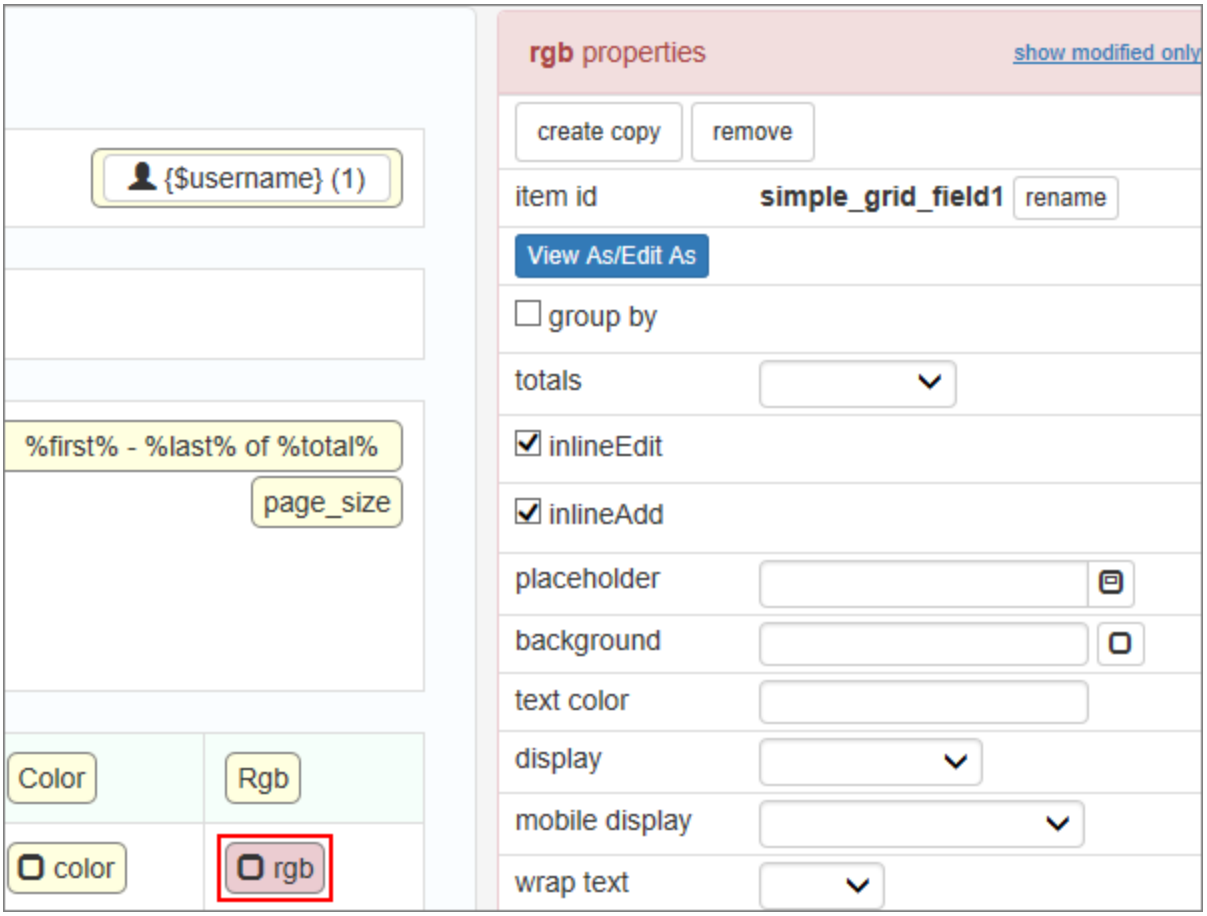
Custom CSS

Page elements

After clicking on an element, you can see that it becomes highlighted in red and the right-side panel shows its properties with settings that can be changed. You can also create a copy of an element or remove it. Additionally, you can change the View As / Edit As / Filter As settings for the element. To do this, press the **View As**/**Edit As** button.

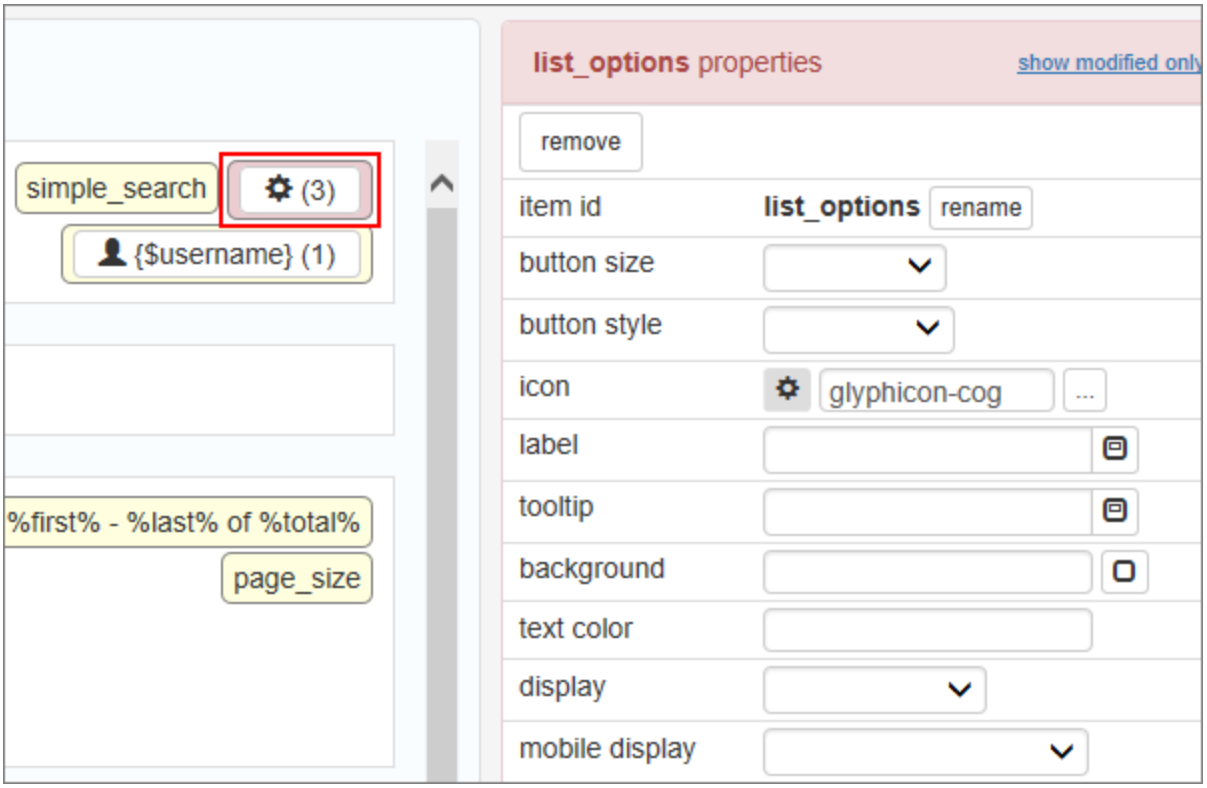
> **Note**: to learn more about View As / Edit As / Filter As settings, see "View as" settings, "Edit as" settings, "Filter as" settings.

Press **Show modified** only link in the header of the **properties** to only see the modified element properties. Press **Show all** to switch back to the default view. You can drag and drop an element from one cell to another when you want to change its default position. Alternatively, move the elements within the grid to re-order them.

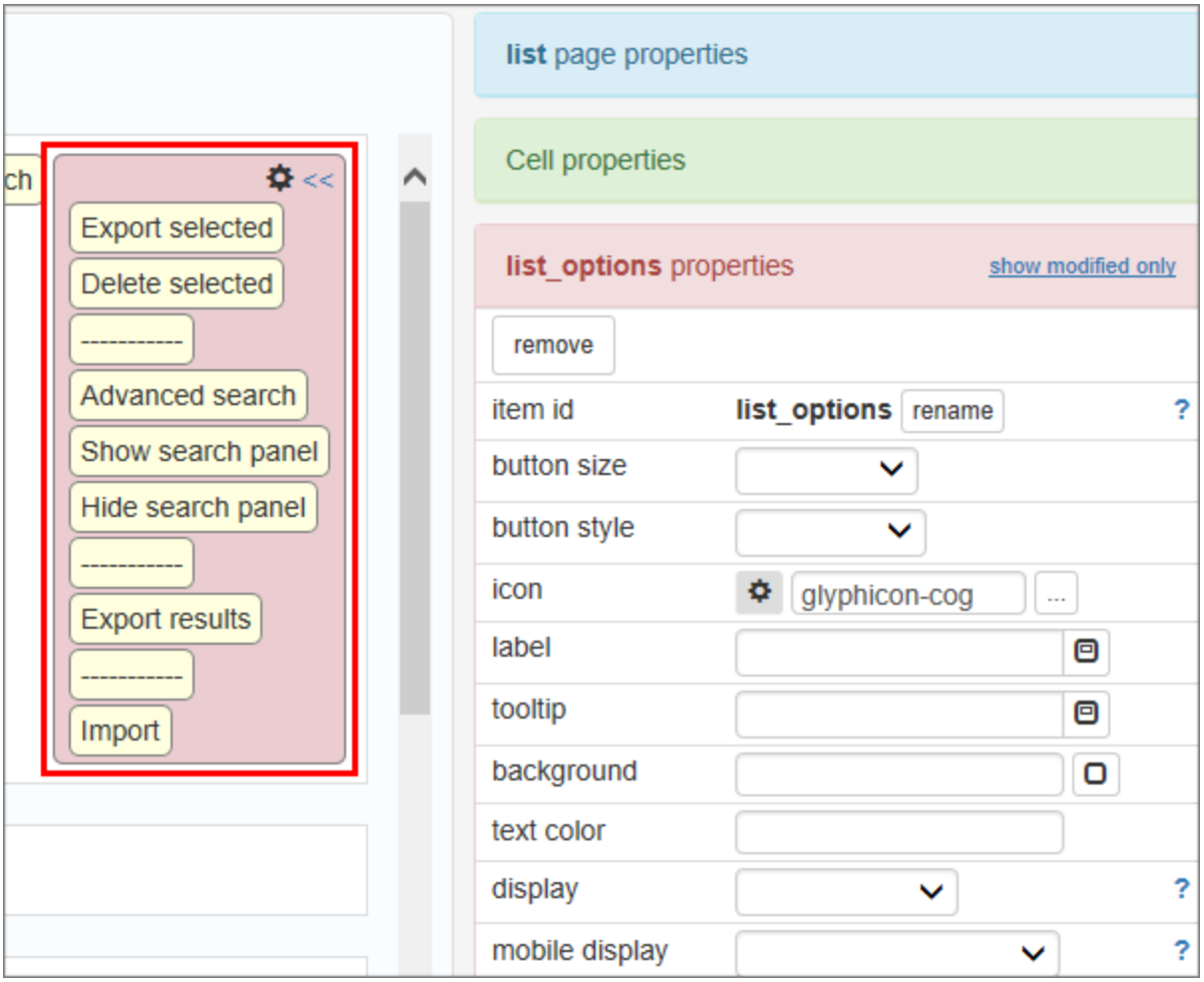


Different types of elements on the page have different properties. Simple page elements like **logo**, **menu**, and **simple search** only have their ID, formatting options, and Custom CSS.

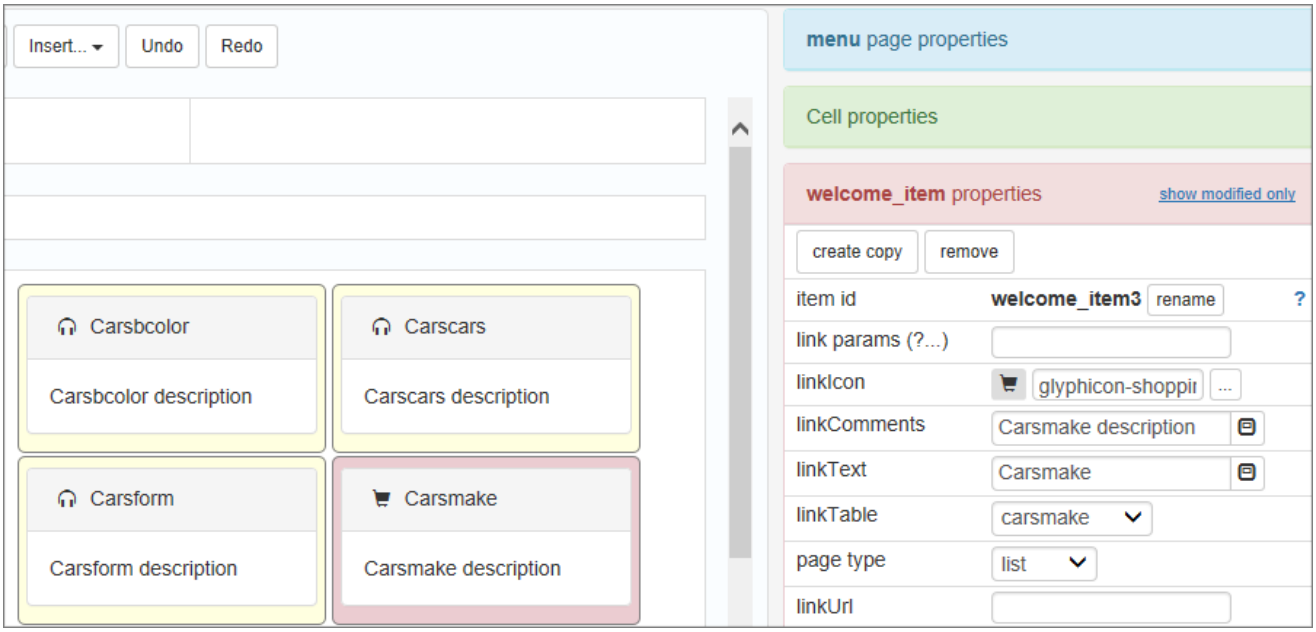
Buttons have some additional properties like label, icon, button style, and size.



Some of the page elements, like the one on the picture above, are **groups of predefined simple elements**. They have values in brackets, like "search_panel(2)" or "print_panel(4)". These numbers show how many sub-elements there are inside of the group. You can change settings for the whole group or for one of the elements inside that group.



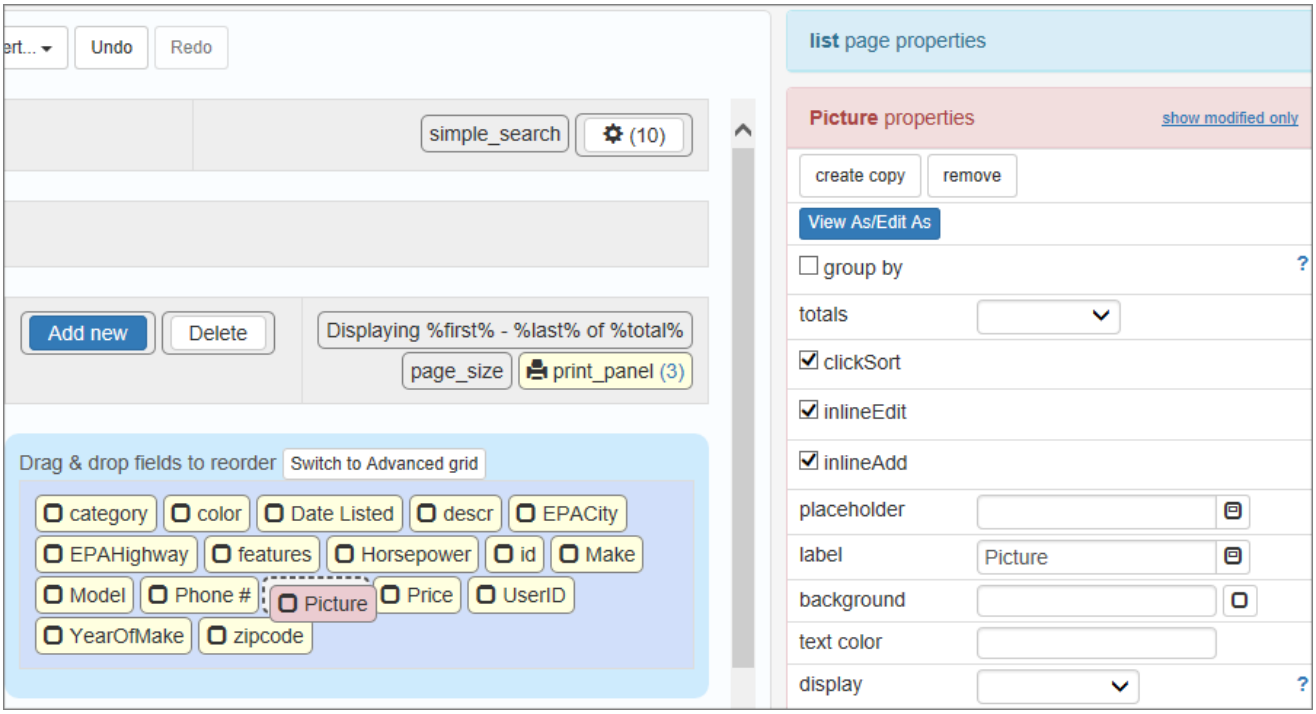
Elements on the menu page have some specific properties.



You can group/ungroup menu items if you need, change the link text, comments, and other options.

Working with database fields

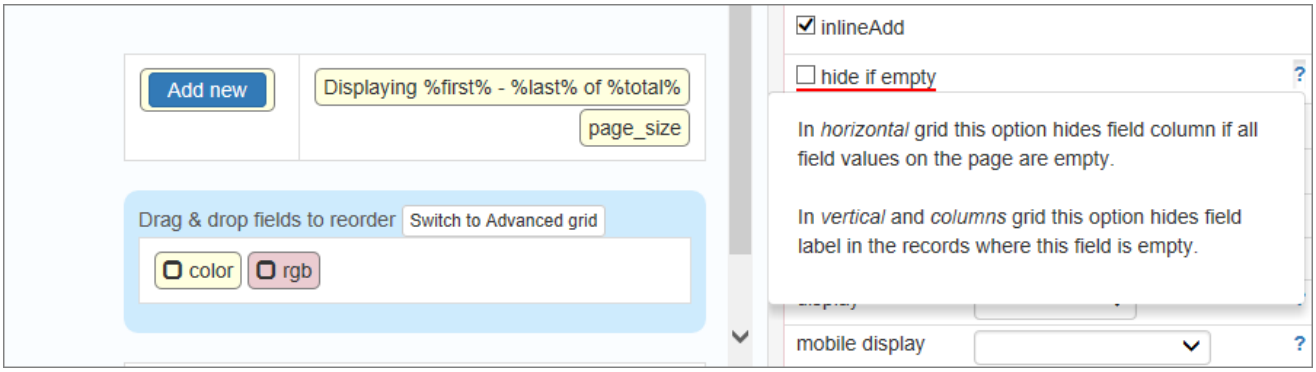
Database fields are also elements. To change the fields order on a specific page, drag the field element to the new place.



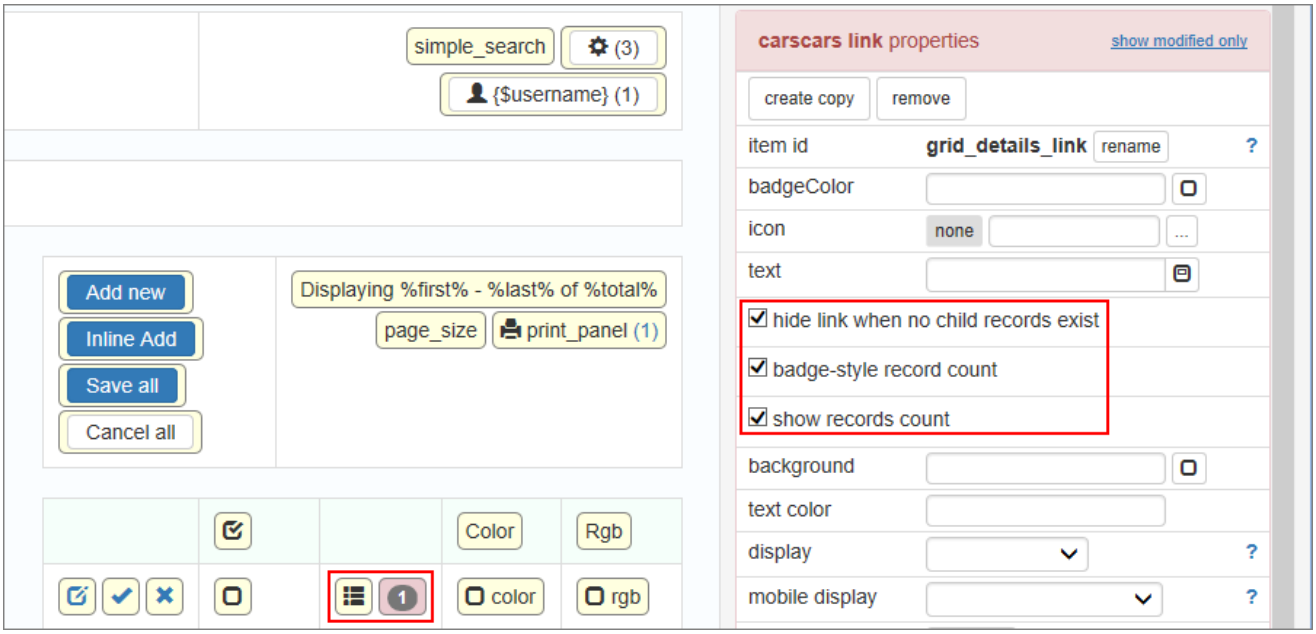
Database fields have special settings like a tooltip, placeholder, **Inline Add**/**Edit** checkboxes. Changes in **Inline Add**/**Edit** made in the **Page Designer** automatically apply to the Choose fields screen.

To change the **Inline Add**/**Edit** options, click on the element in the grid, and then use the controls that appear to the right. You can also access View As/Edit As settings and choose one of the calculated Totals from the dropdown on the right panel of a **Database field** element.

The **Database fields** also have the **hide if empty** option. You can select this checkbox to hide a field column or field labels if the respective field is empty. Press the **?** button to learn more.



Details icons in the Advanced grid mode have additional options: "**show records count**", "**badge-style record count**" and "**hide link when no child record exists**".

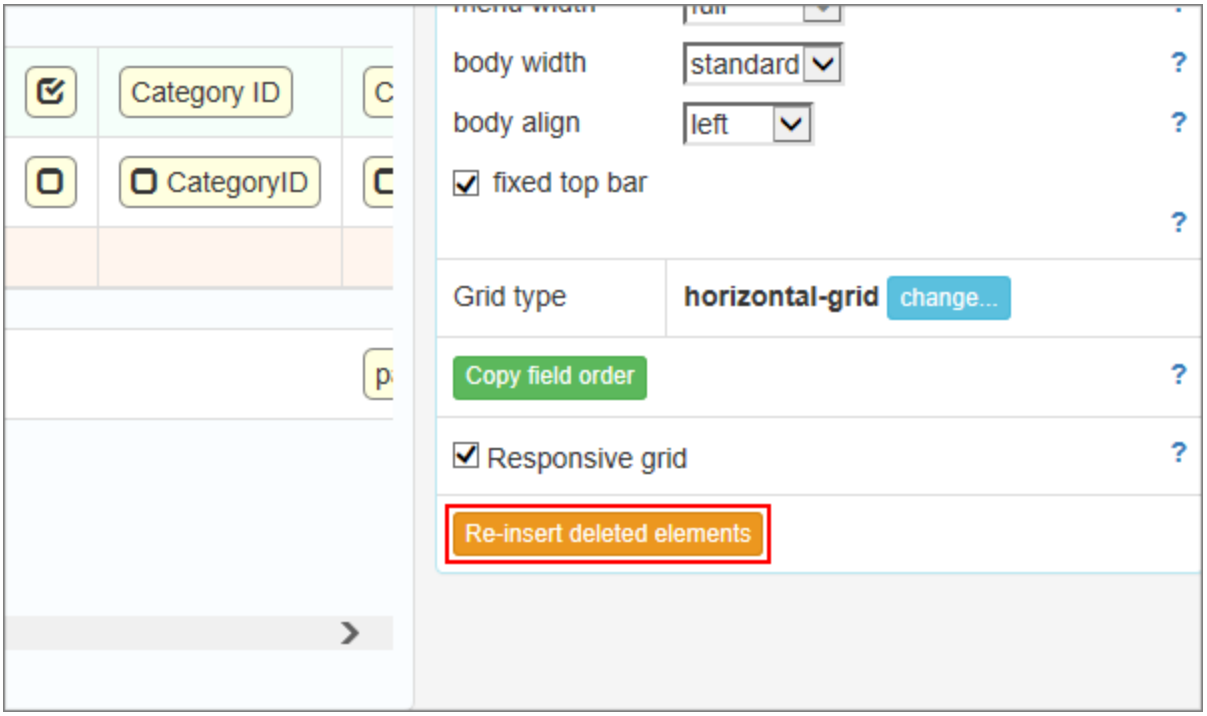


Restoring deleted page elements

If you accidentally removed the menu or some other "essential" element from the page, you have two ways to restore it.

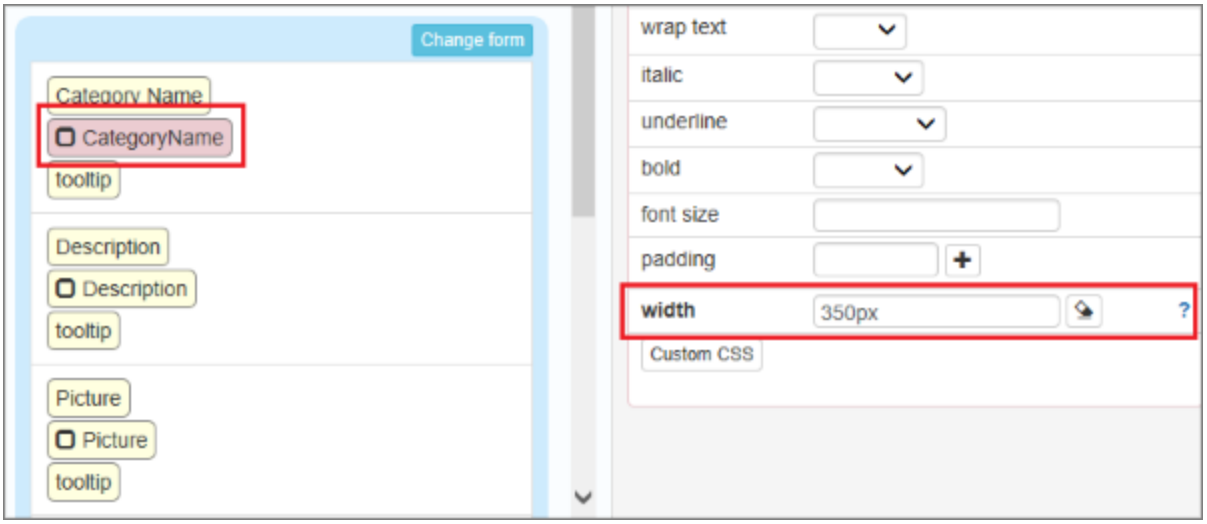
1. Create a new page of the same type and make it a default page. It works similar to a **'Reset'** function in older versions and gives you a brand new page. This option is not ideal if you have already applied a bunch of customizations to the page.

2. You can check the **Page properties** (or click outside of any page elements) and see the **Re-insert deleted elements** button appear. By clicking this button, you restore "essential" page elements like menu or breadcrumbs, while keeping your customization intact.

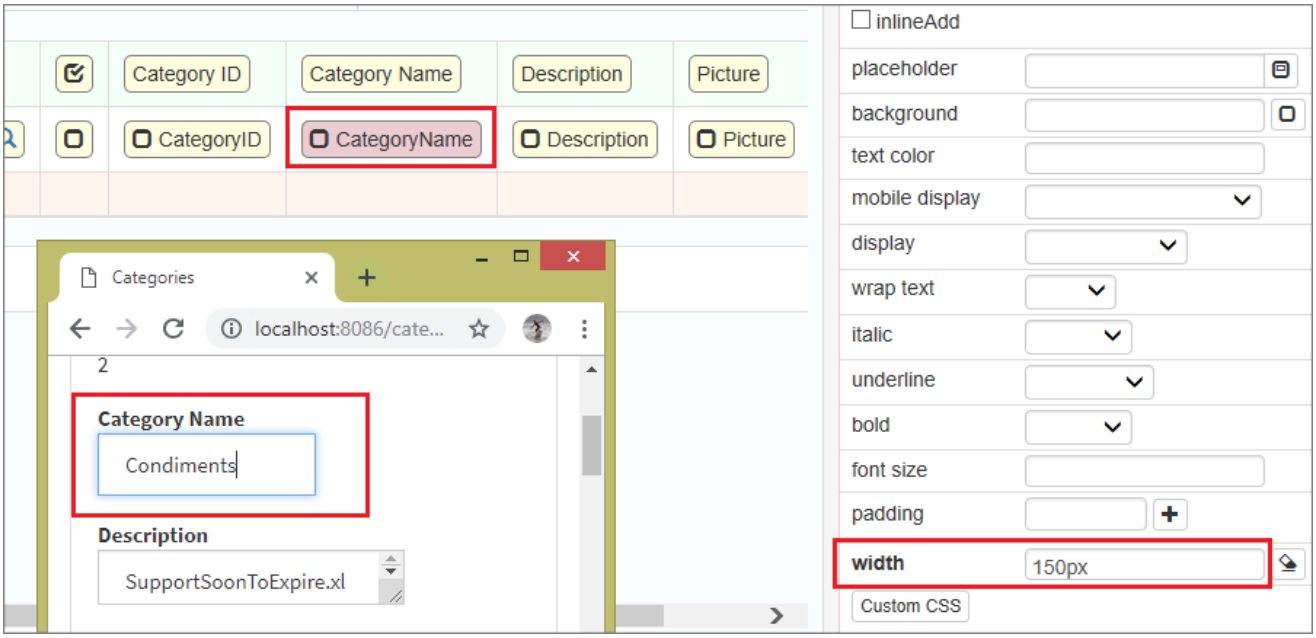


Changing Edit control and Column width

To change **Edit control** width on **Add/Edit** page, proceed to that page in **Page Designer**, and under **Form Layout** choose "**separate labels and controls**". Then you can modify settings of control and its label separately. Make edit control selected in **Page Designer** and set its width in pixels on the right side.



To change **Inline edit** control width, go to the chosen **List** page and switch the grid to the **Advanced grid** mode. Select the field itself and set its width in pixels on the right. The field width applies to the **List** mode and the Inline **Edit** control.

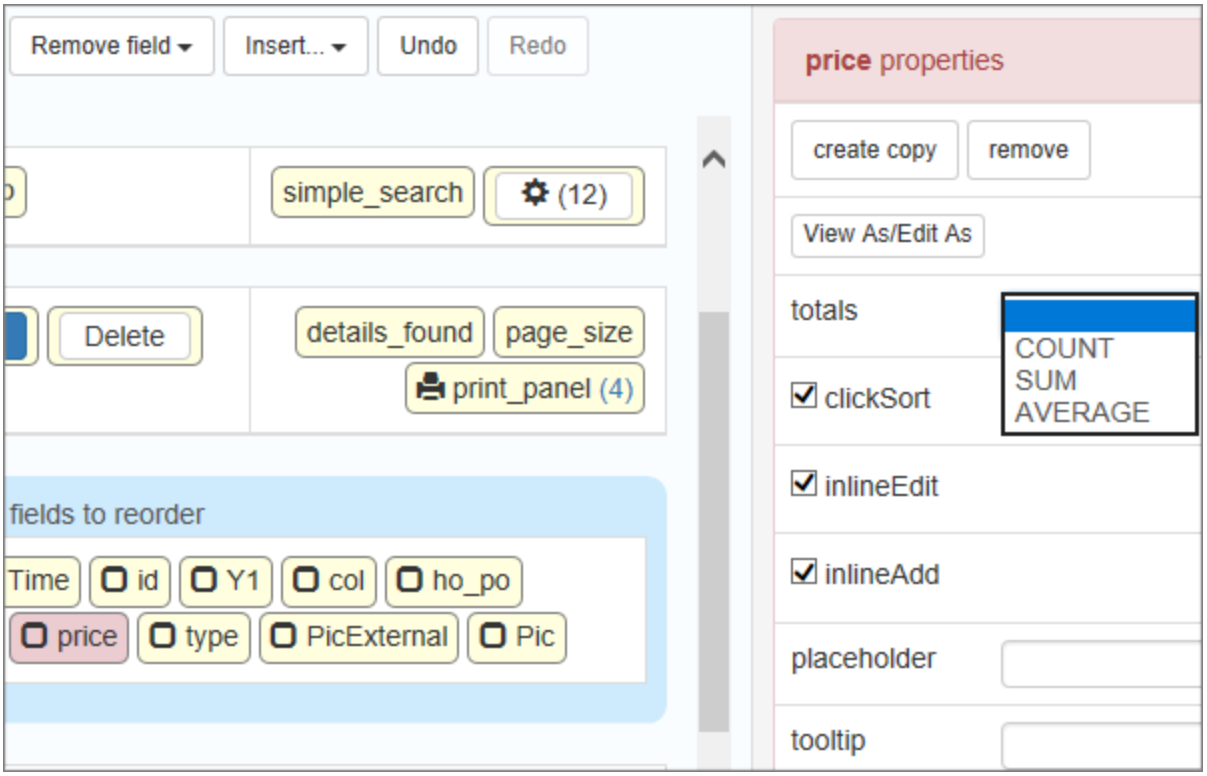


You can change the **Column width** in **Advanced grid** mode as well. To do so, select a column and set the width in pixels using width setting on the right.

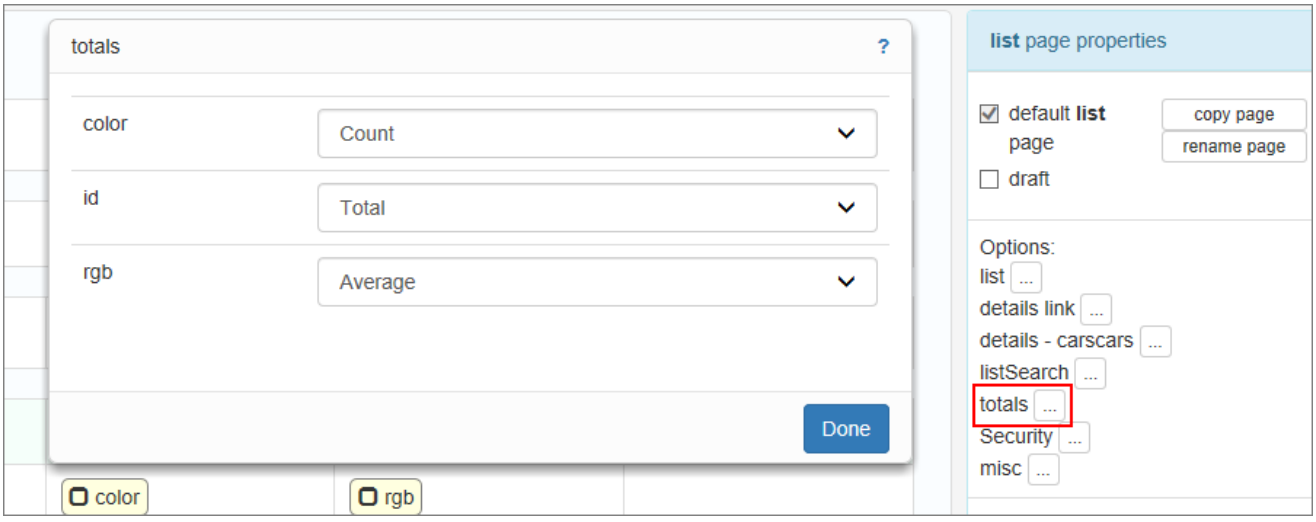
Totals type

You can add totals calculation to some fields on the **List**/**Print** pages. Totals calculation is available with **Simple horizontal** and **Advanced horizontal** Grid type.

You can choose between TOTAL, AVERAGE, and COUNT aggregate functions. Select the field you want to add totals calculation to and in the properties on the right, choose the function type from the totals dropdown box.

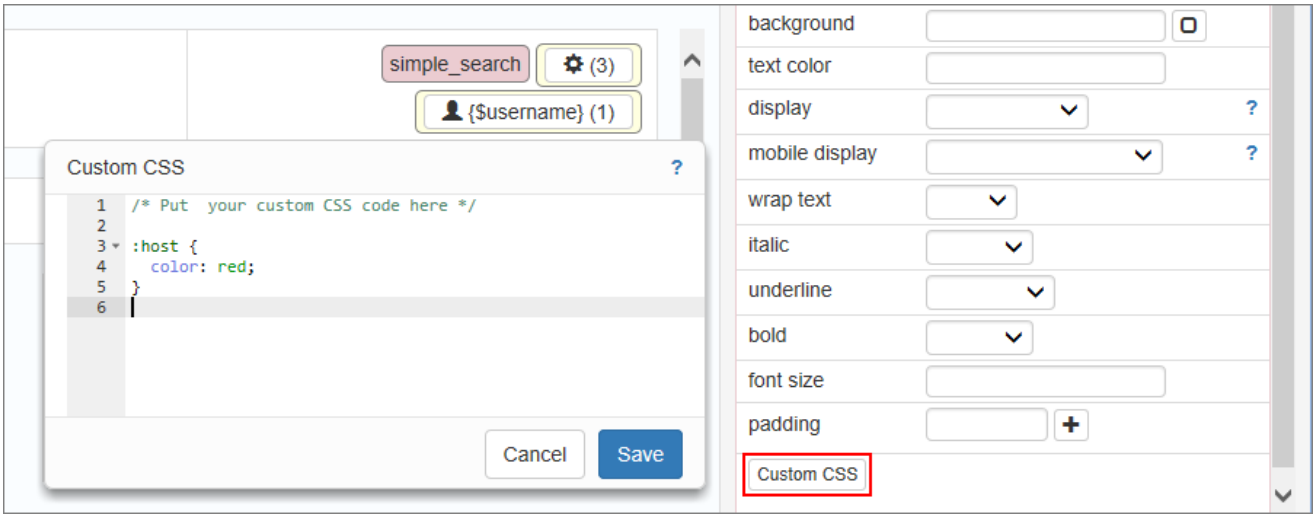


Another way of adding totals for the fields is to click on the totals button in the **list** or **print** page properties. You can choose totals functions for several fields at once there.



Custom CSS

Press the **Custom CSS** button to open the popup window. You can add the CSS code to the element here. See more about Customizing CSS.



Page Designer articles:

- About Page Designer
- Working with common pages
- Working with table pages
- Working with cells
- Working with additional pages
- Page layout and grid type
- Tabs and sections
- Insert custom button
- Insert code snippet
- Insert map
- Insert Text / Image
- Insert standard button
- "View as" settings
- "Edit as" settings
- "Filter as" settings

See also:

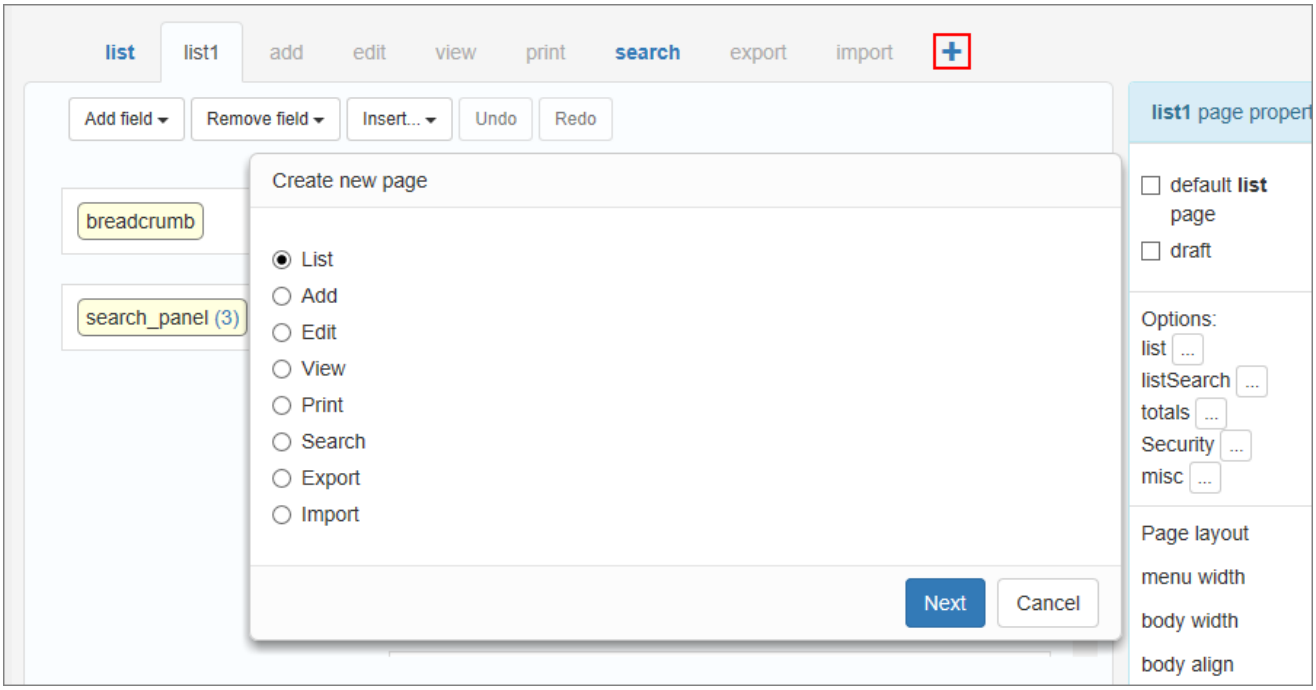
- Choose pages screen
- Choose fields screen
- Miscellaneous settings
- Editor screen
- Event editor

2.16.6 Working with additional pages

Creating additional pages

With the **Page Designer**, you can add extra pages of the same type. For instance, besides the main edit page, you can have extra edit pages named 'edit1', 'edit2', etc. Each of these pages can have its own set of elements, layout, and design.

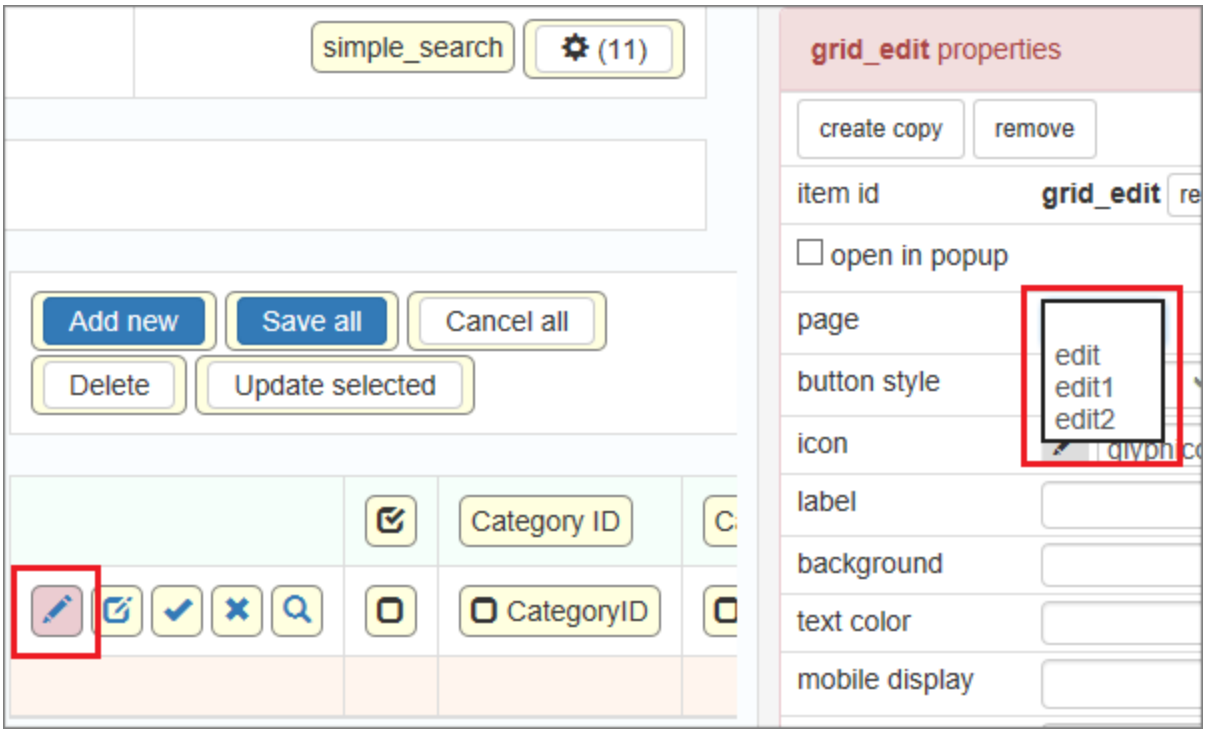
To create additional pages, click the '**+**' button to the right of the pages list and select the type of page you want to create.



Using additional pages

You can set, which page to open with the **page** dropdown option in the button properties.

For example, there is an **edit** button on a **List** page. By default, it takes you to the **Edit** page. If you have several edit pages, click the button and proceed to the **page** option on the right-side panel to change the page this button opens.



Using additional pages in events

You can address the additional pages manually, in the event code.

1) To access the **List** page named *'list1'*, use the following URL:

```
cars_list.php?page=list1
```

> **Note**: you can use any table name and/or page type instead of *'cars'* and *'list'*.

2) To redirect users to the additional **List** page *'list1'* for table *'cars'*, use this code:

```
header("Location: cars_list.php?page=list1");
exit();
```

Page Designer articles:

- About Page Designer
- Working with common pages
- Working with table pages
- Working with cells
- Working with page elements
- Page layout and grid type
- Tabs and sections
- Insert custom button
- Insert code snippet
- Insert map
- Insert Text / Image
- Insert standard button
- "View as" settings
- "Edit as" settings
- "Filter as" settings

See also:

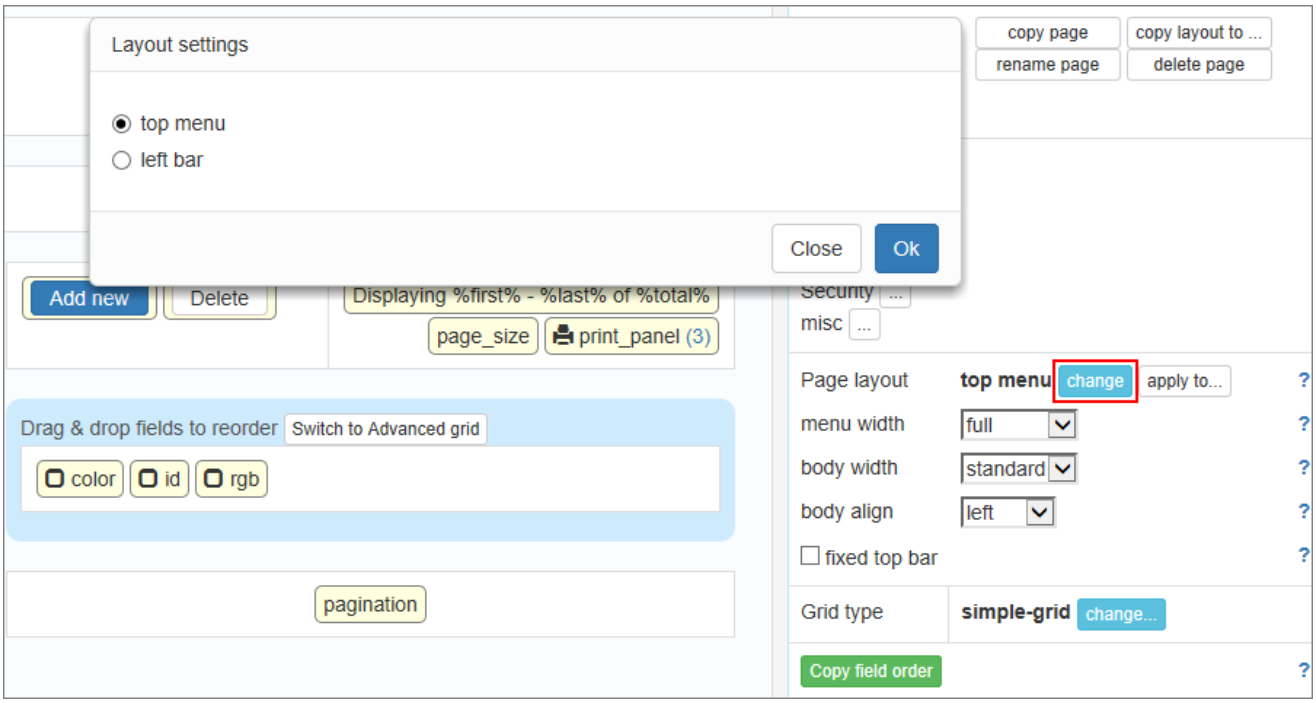
- Choose pages screen
- Choose fields screen
- Miscellaneous settings
- Editor screen
- Event editor

2.16.7 Page layout and grid type

Quick jump

Page layout

You can select two types of page layout for the regular pages: top menu and left bar. Click **change** on the **List** page properties to select the page layout:



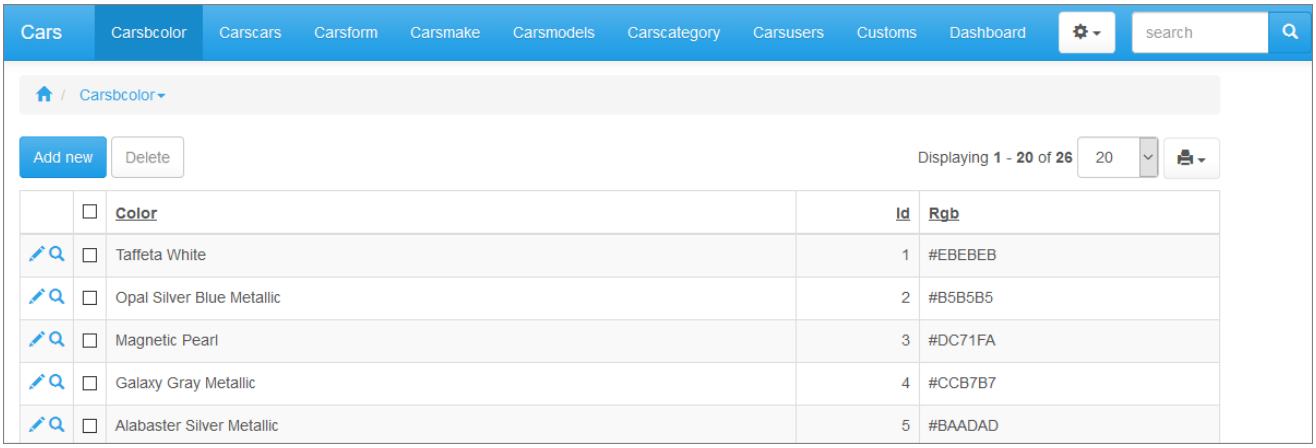
Layout options

Each layout has its own set of options.

Top menu layout options

- **Menu width**. You can choose the width of the top menu bar. **Full** - full window width. **Standard** - about 2/3 of a page width at its maximum.
- **Body width**. Body width sets how the page content uses the browser window space. **Full** - the page always stretches to the entire browser width. **Compact** - the content uses as little space as possible. **Standard** - the content stretches to about 2/3 of a page.
- **Body align**. This option lets you choose whether the page body aligns to the left or the center.
- **Fixed top menu**. With this checkbox on, the top menu stays in place as you scroll the page.

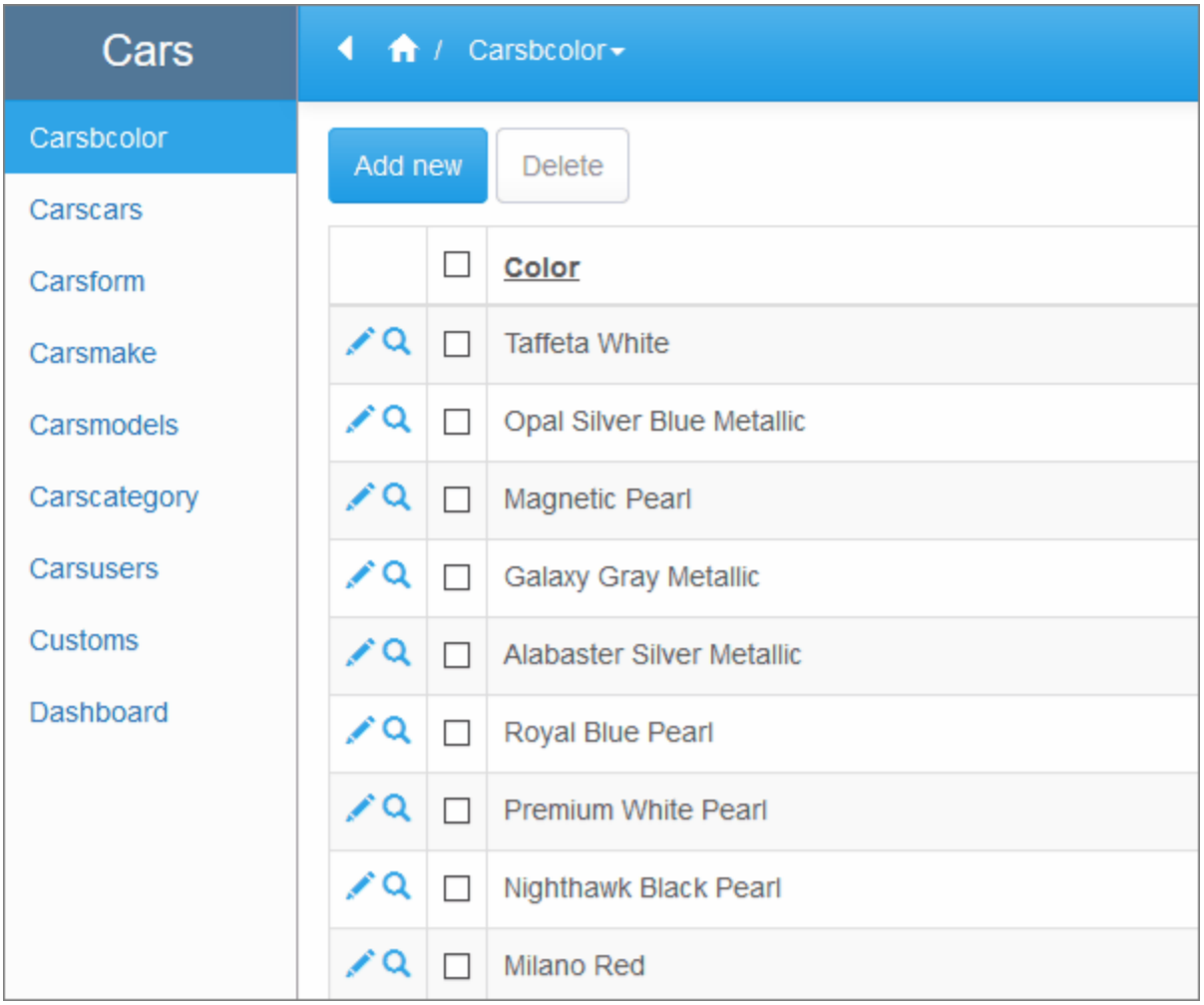
An example of the **top menu** page layout. Full menu, standard body width, left alignment:



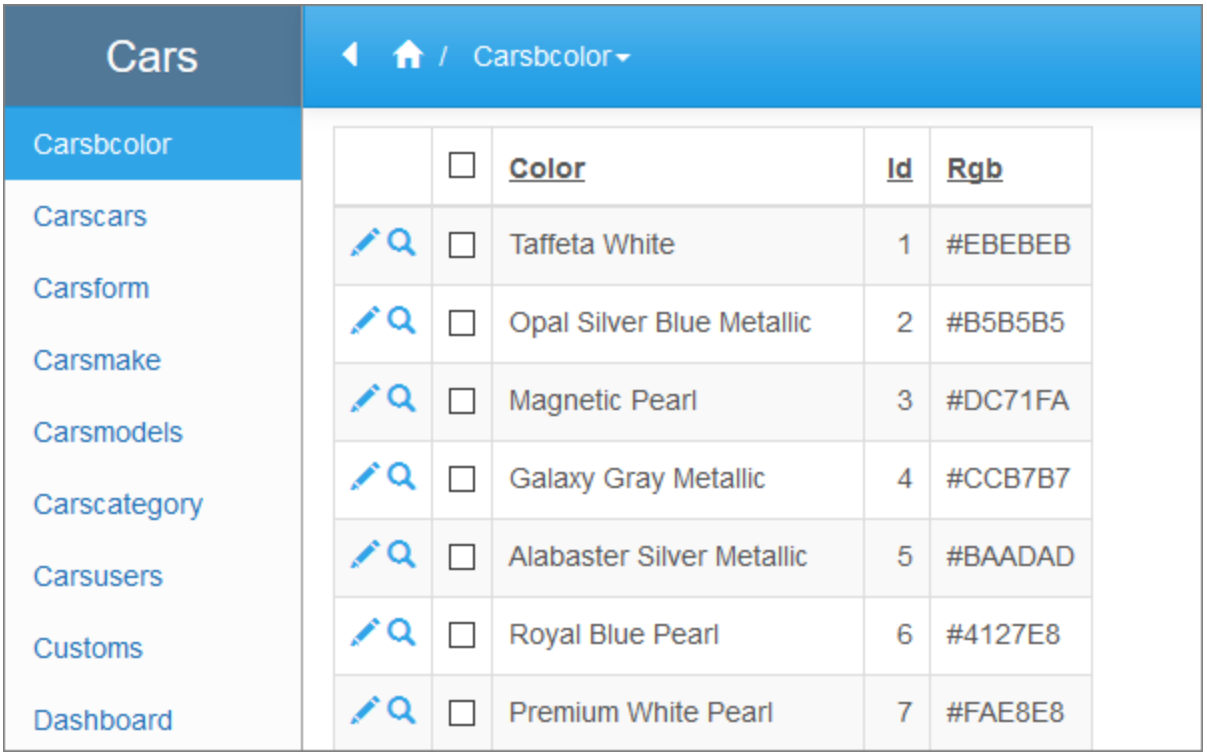
Left bar layout options:

- **Body width**. Body width sets how the page content uses the browser window space. **Full** - the page always stretches to the entire browser width. **Compact** - the content uses as little space as possible. **Standard** - the content stretches to about 2/3 of a page.
- **Left bar width**. Specify the width of the left menu bar. Clear the box to set the default value - *300px*. You can enter any units available in CSS, for example, *400px* or *20%*.
- **Fixed top bar**. With this checkbox on, the top bar stays in place as you scroll the page.
- **Fixed side bar**. With this checkbox on, the side bar stays in place as you scroll the page.
- **Collapsible left bar**. This option adds a button to collapse/expand the left bar.

Here is how the collapsible left bar works:



An example of the **left bar** layout. Compact body width, left bar width - 150 px, fixed top/side bar, collapsible left bar:

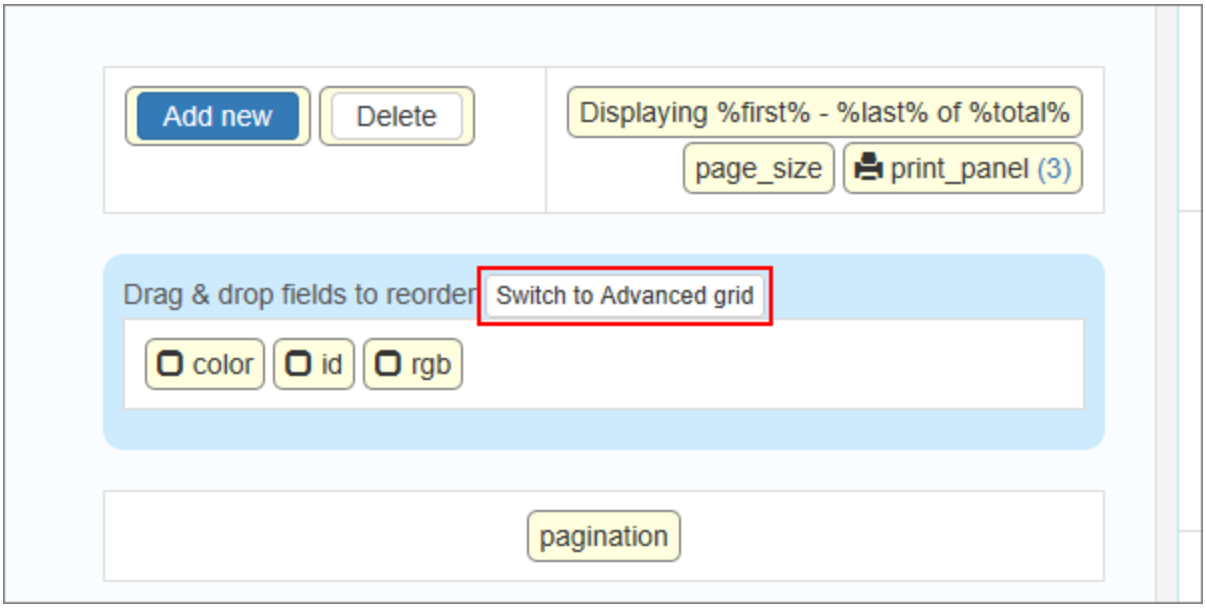


Grid type

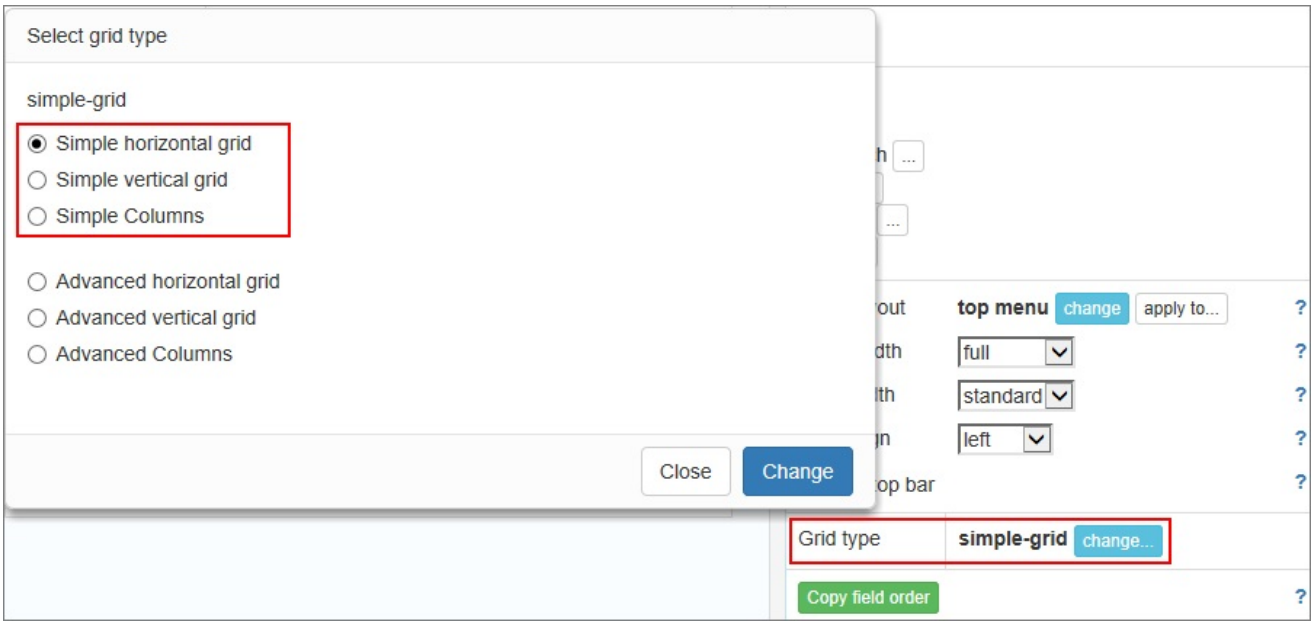
Page Designer can change the appearance of the grid type.

You can choose one of three grid types: horizontal, vertical or columns in **Simple** or **Advanced grid** mode. In the **Advanced grid** mode, you can easily change the properties of labels, cells, and columns when you need to set specific formatting. If you don't change anything in advanced mode, the generated pages look the same as in the simple mode.

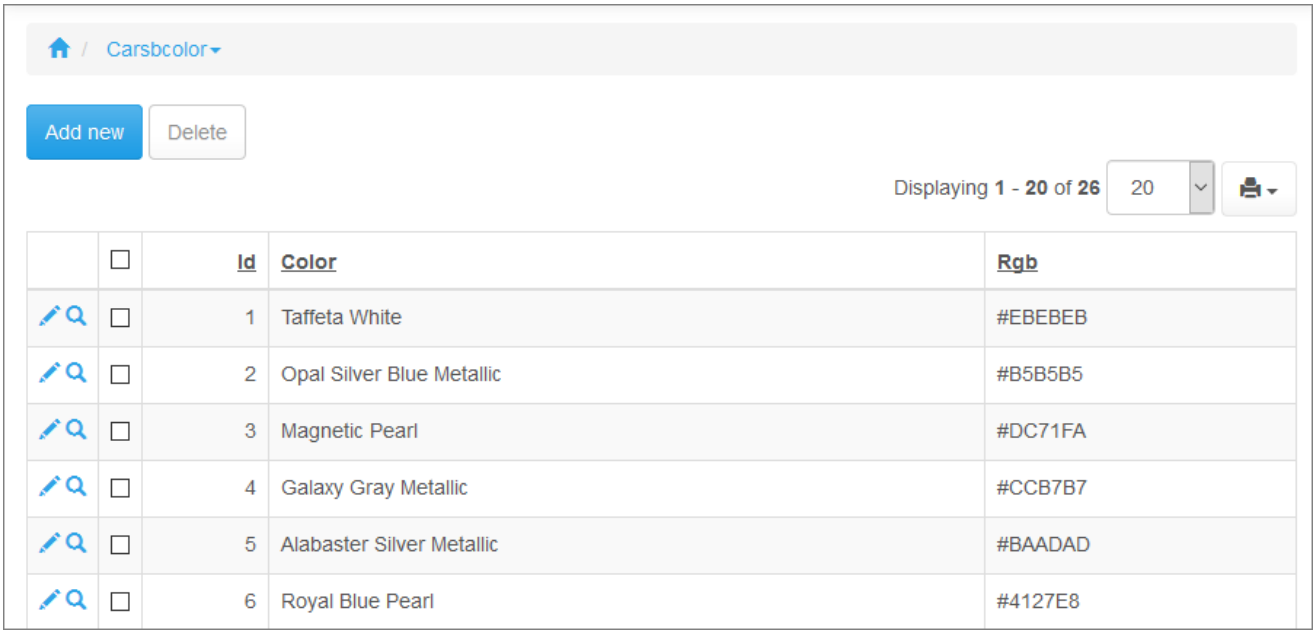
To switch to **Advanced grid** mode, press the button above the grid:



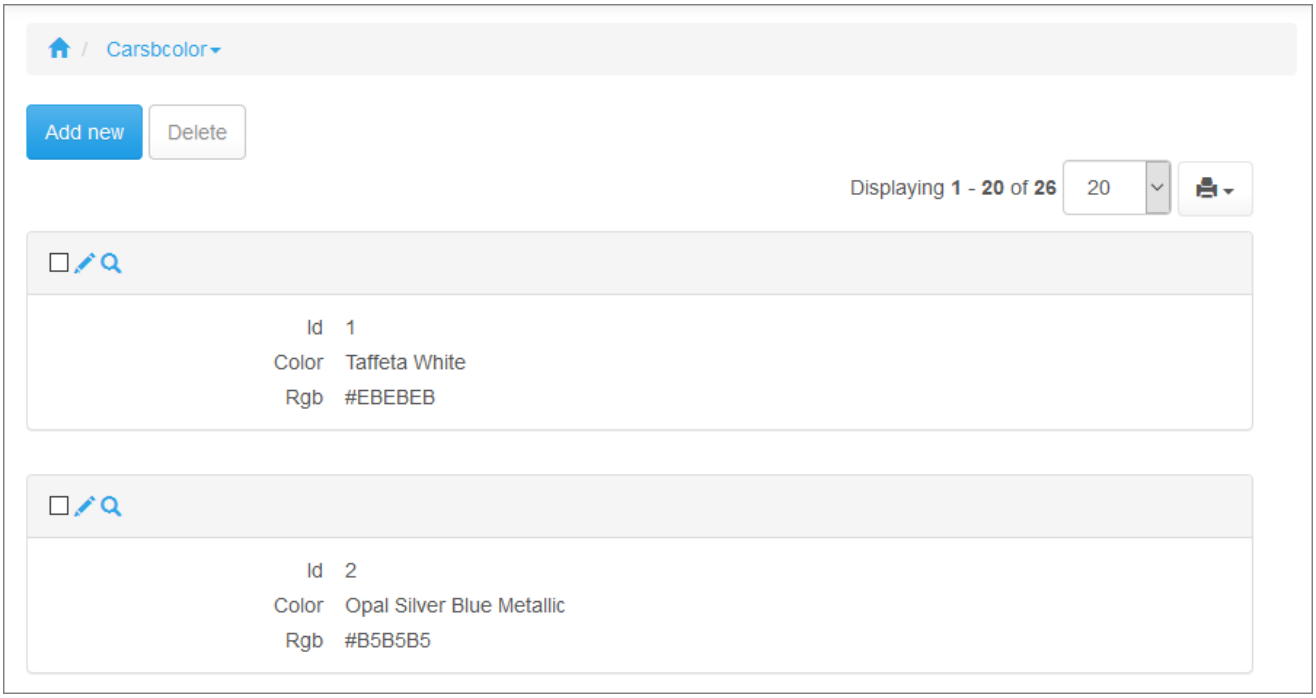
To switch back to a **Simple grid** type, click **Change** under the page **properties** on the right side, and choose one of the simple grid types.



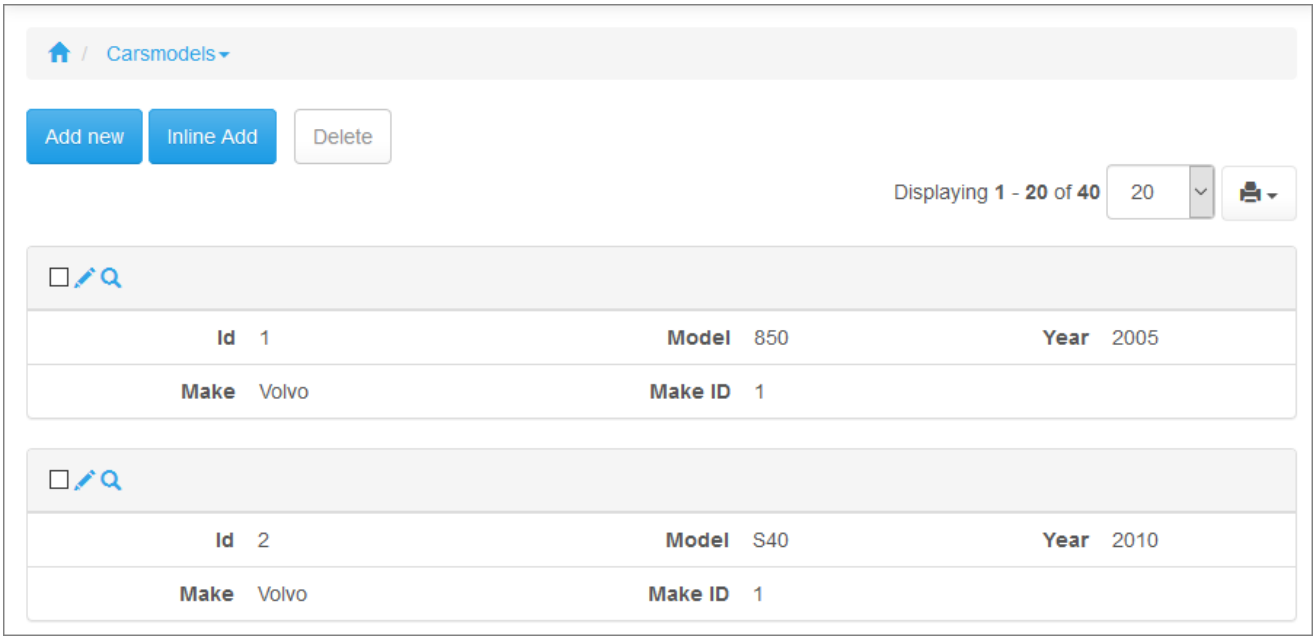
Example of a horizontal grid:



Example of a vertical grid:



Example of a columns grid with three columns:



Page Designer articles:

- About Page Designer
- Working with common pages
- Working with table pages
- Working with cells
- Working with page elements
- Working with additional pages
- Tabs and sections
- Insert custom button
- Insert code snippet
- Insert map
- Insert Text / Image
- Insert standard button
- "View as" settings
- "Edit as" settings
- "Filter as" settings

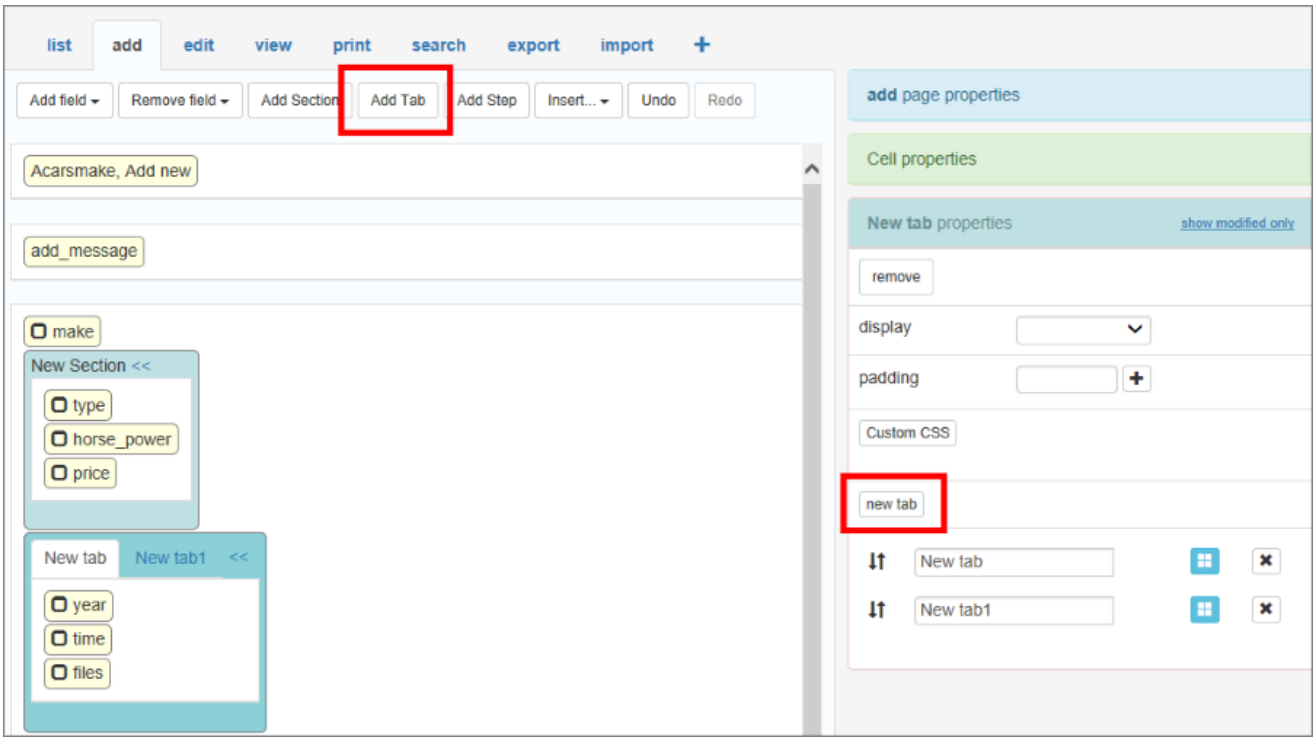
See also:

- Choose pages screen
- Choose fields screen
- Miscellaneous settings
- Editor screen
- Event editor

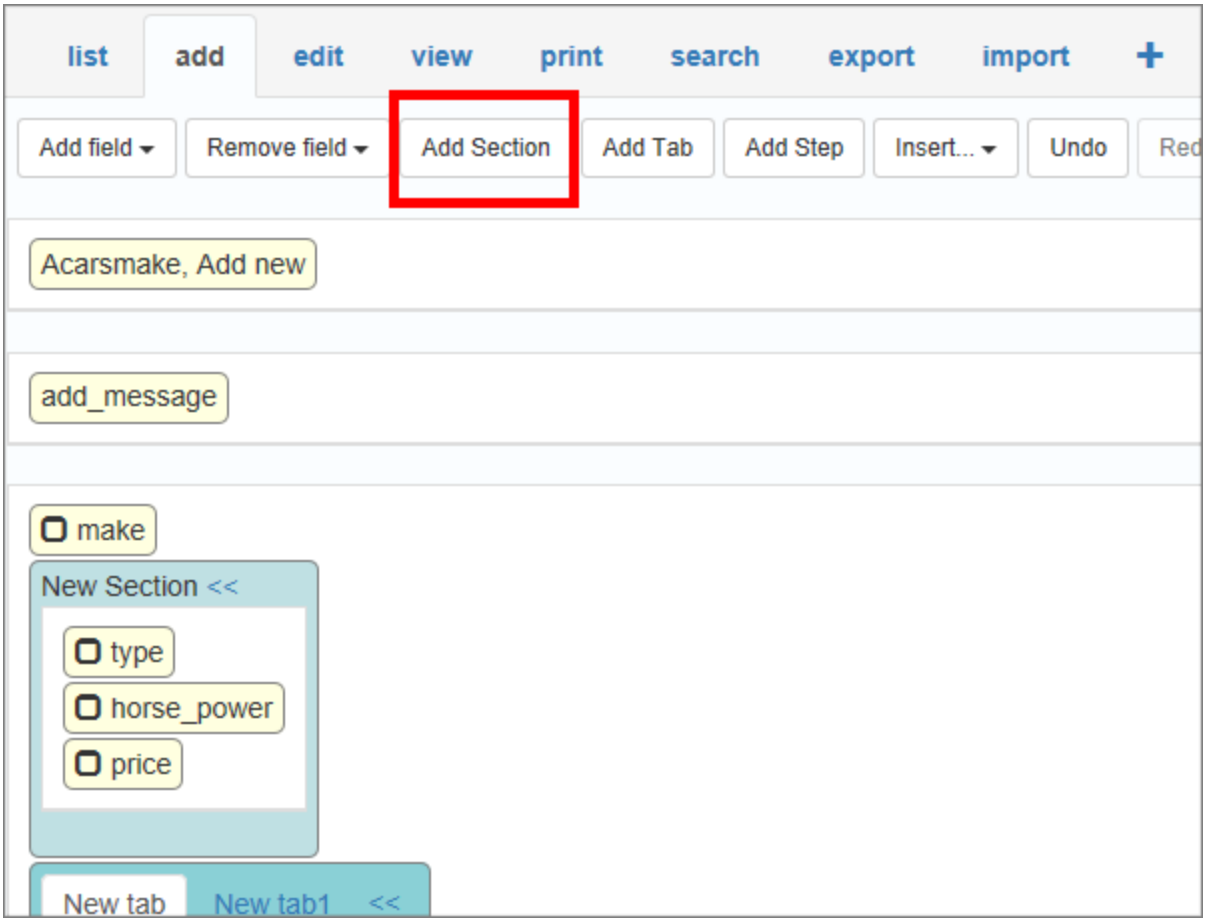
2.16.8 Tabs and sections

Tabs/Folding sections

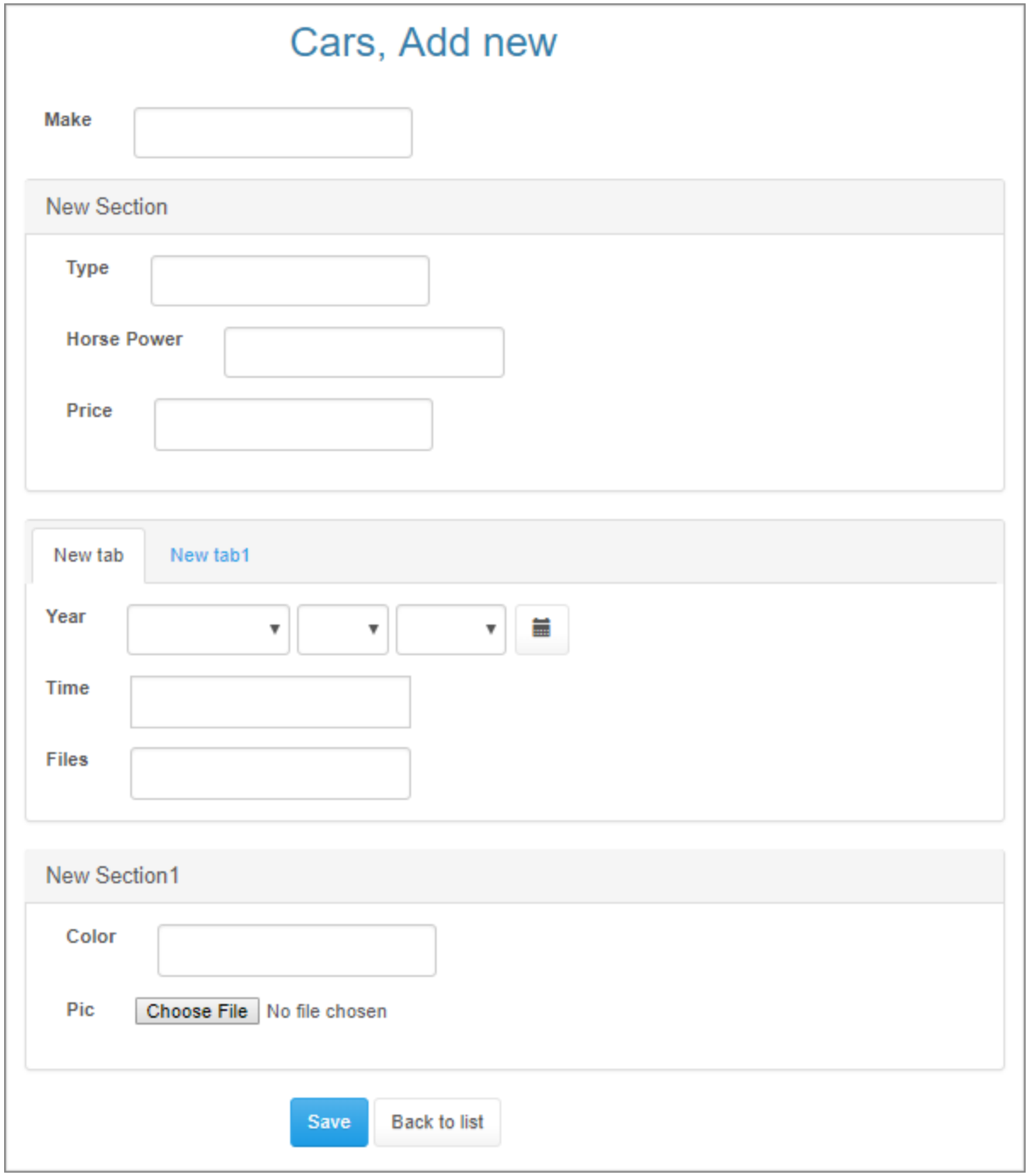
You can use tabs and folding sections on the **Add**/**Edit**/**View** pages. To create a new tab, press the **Add tab** button on the top. You can drag-n-drop fields into the tab. Press the **new tab** button in the **tab properties** to add the new tab to the group.



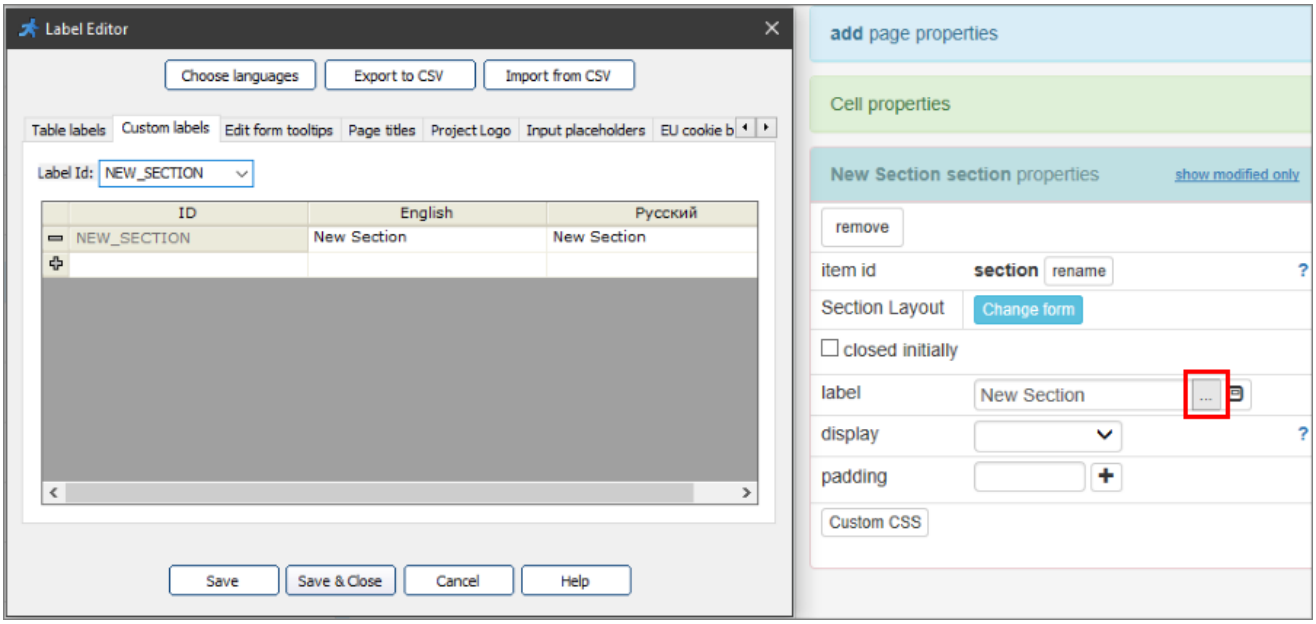
Click the **Add Section** button to create a new section.



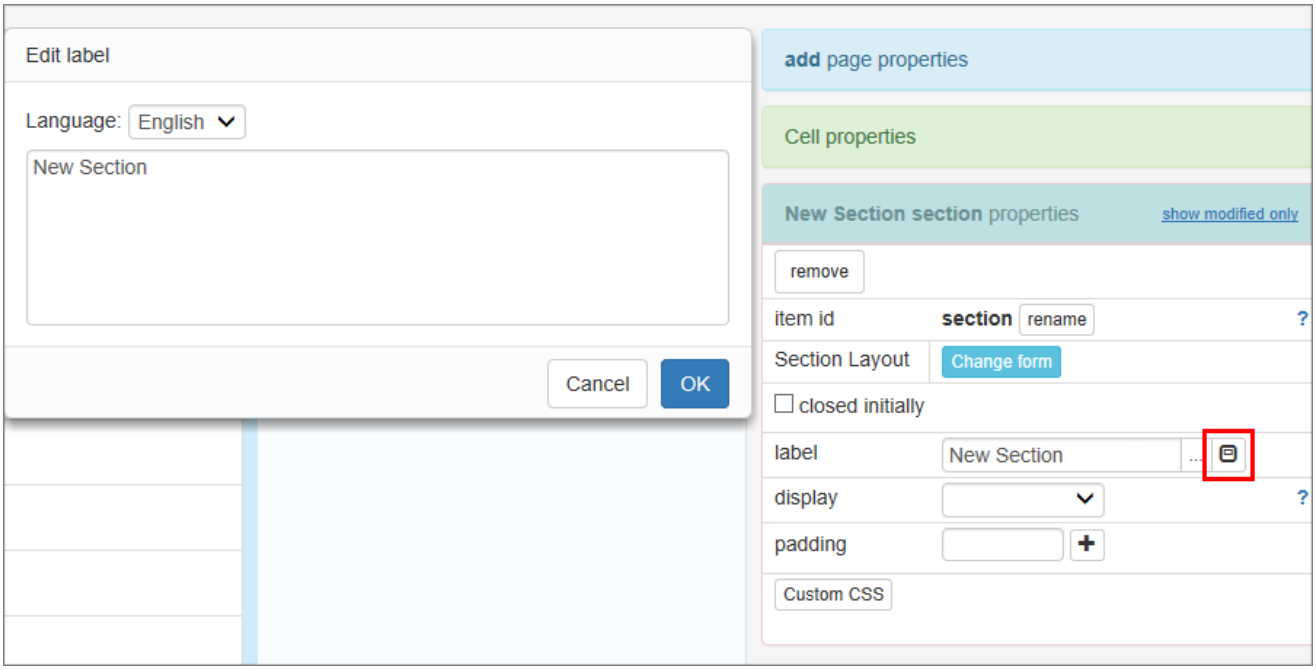
Here is how the result looks in the generated "Add" page:



You can translate the names of tabs and sections if you have several languages selected in the Miscellaneous settings. Click the **...** button in the **properties** to open the **Label Editor** and translate the names.

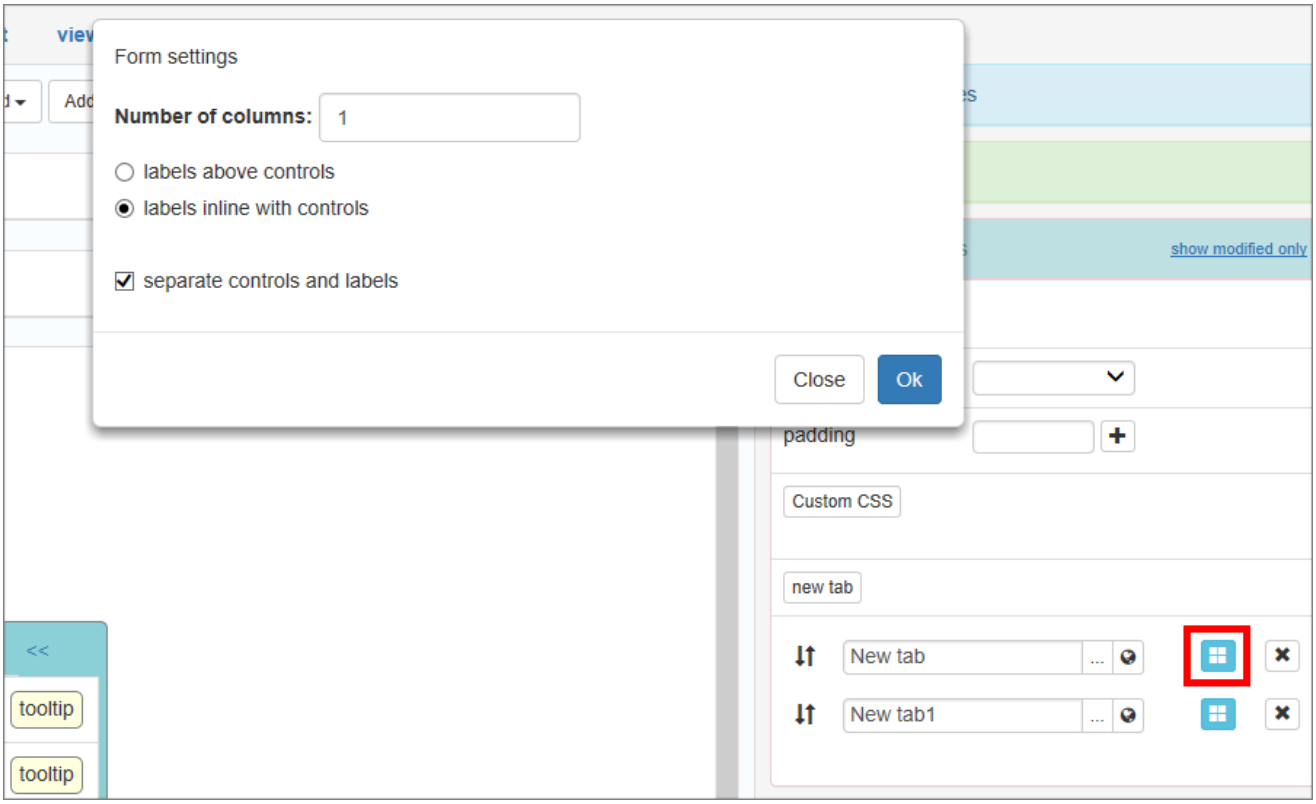


Alternatively, you may also change it using the dialog for the current tab/section.



You can also change the way the labels and field values appear: in **columns**, **above the controls** or **inline** with them. You can select **separate controls and labels** to apply some custom formatting. This can be done for each tab or section separately.

Press the **Form Layout** button to do so, then choose the desired settings for the Tab or Section.



Page Designer articles:

- About Page Designer
- Working with common pages
- Working with table pages
- Working with cells
- Working with page elements
- Working with additional pages
- Page layout and grid type
- Insert custom button
- Insert code snippet
- Insert map
- Insert Text / Image

- Insert standard button
- "View as" settings
- "Edit as" settings
- "Filter as" settings

See also:

- Choose pages screen
- Choose fields screen
- Miscellaneous settings
- Editor screen
- Event editor

2.16.9 Insert custom button

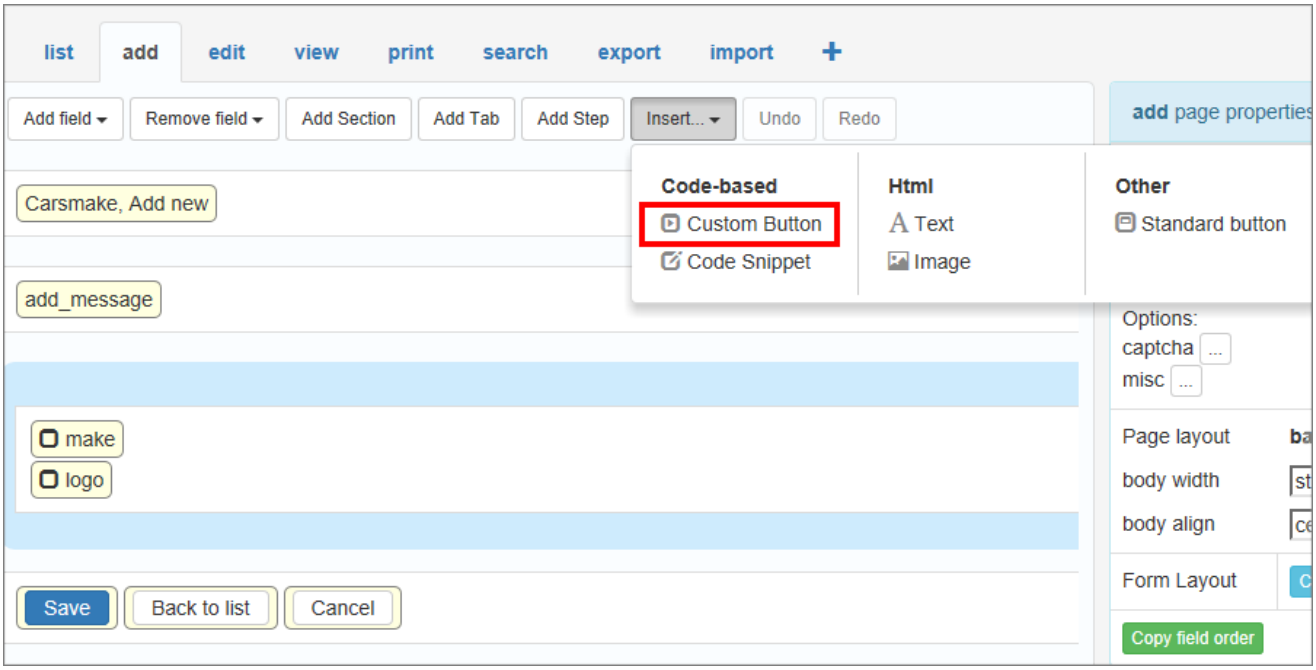
Quick jump

Adding buttons

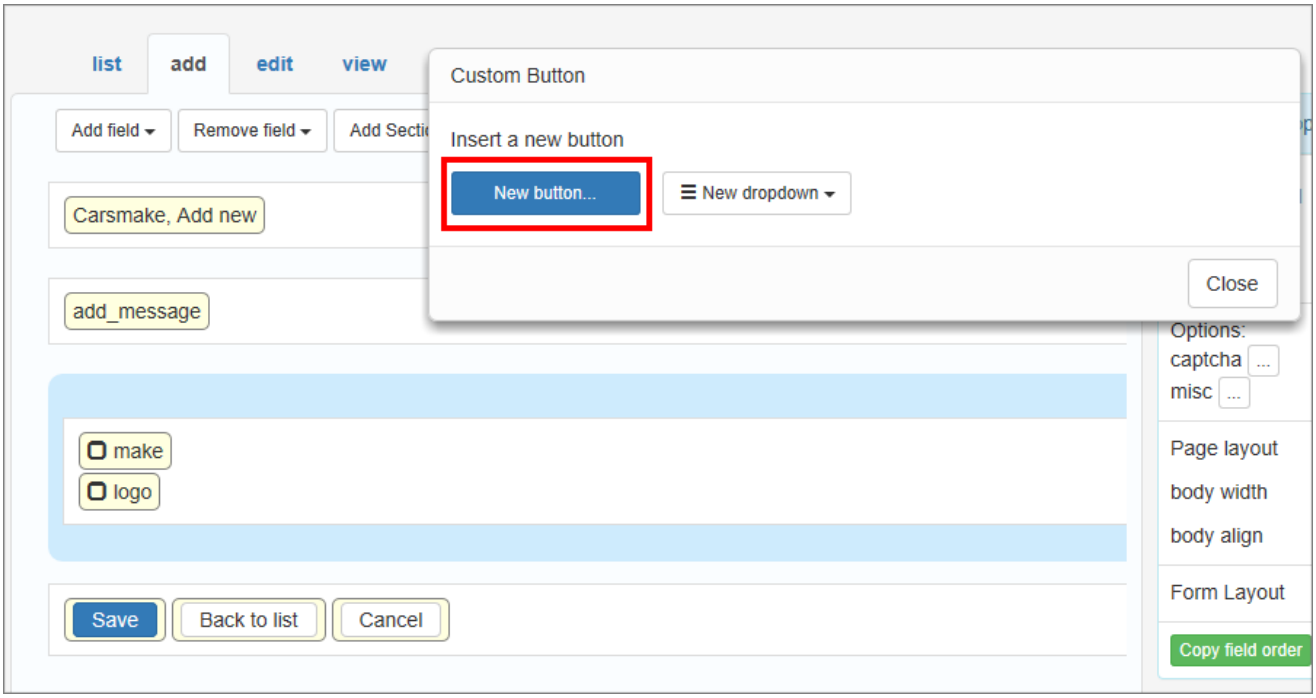
To add a button to a web page:

1. Proceed to the **Page Designer** and select a page you wish to modify.

2. Click **Insert** - **Custom button** on the main toolbar.

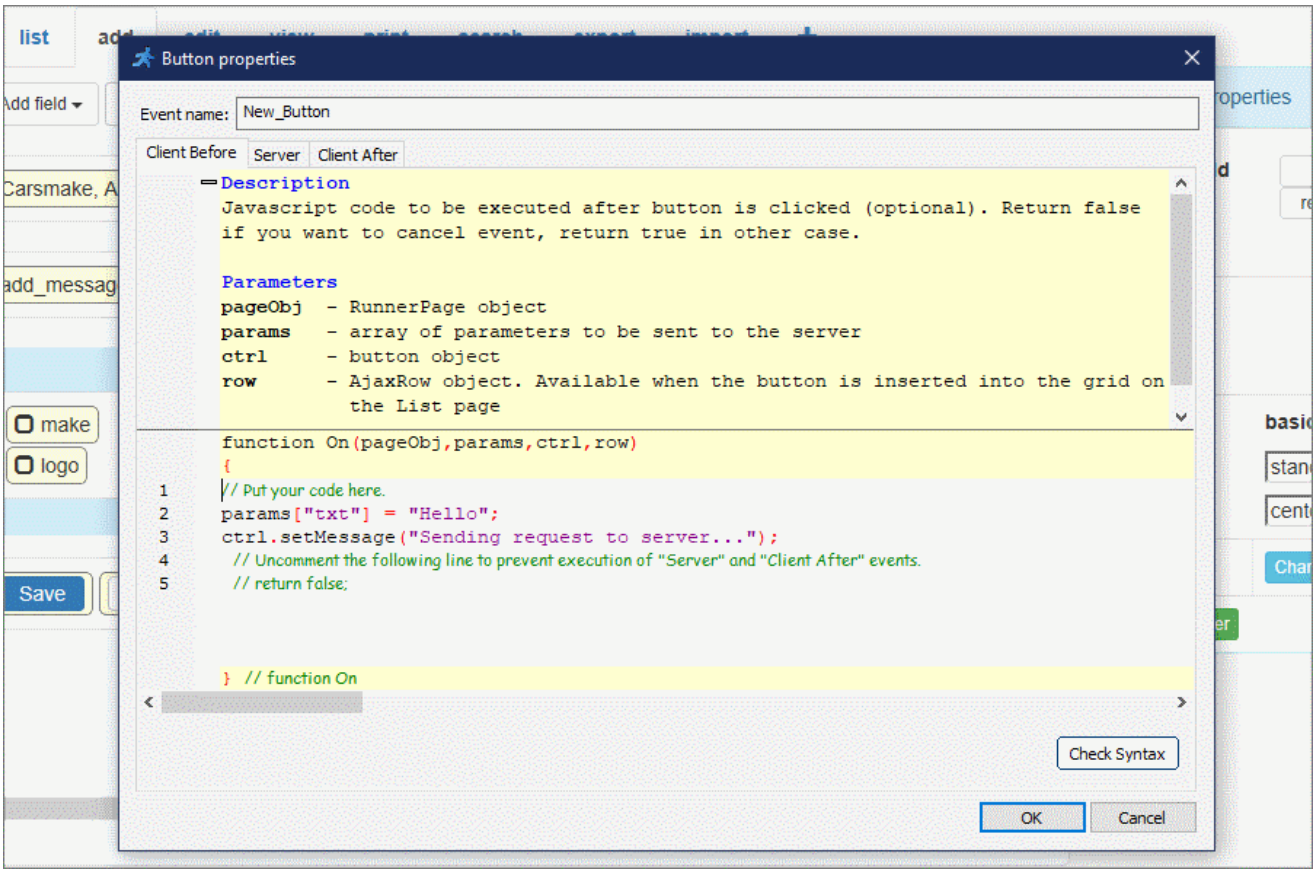


3. Choose **New button**.



Note: The already added custom buttons appear in the **Insert** -> **Custom button** window, so that you can re-use them. Any changes in the button code are made in the copies as well.

4. Type in the button caption and insert the code to be executed after the button is pressed.

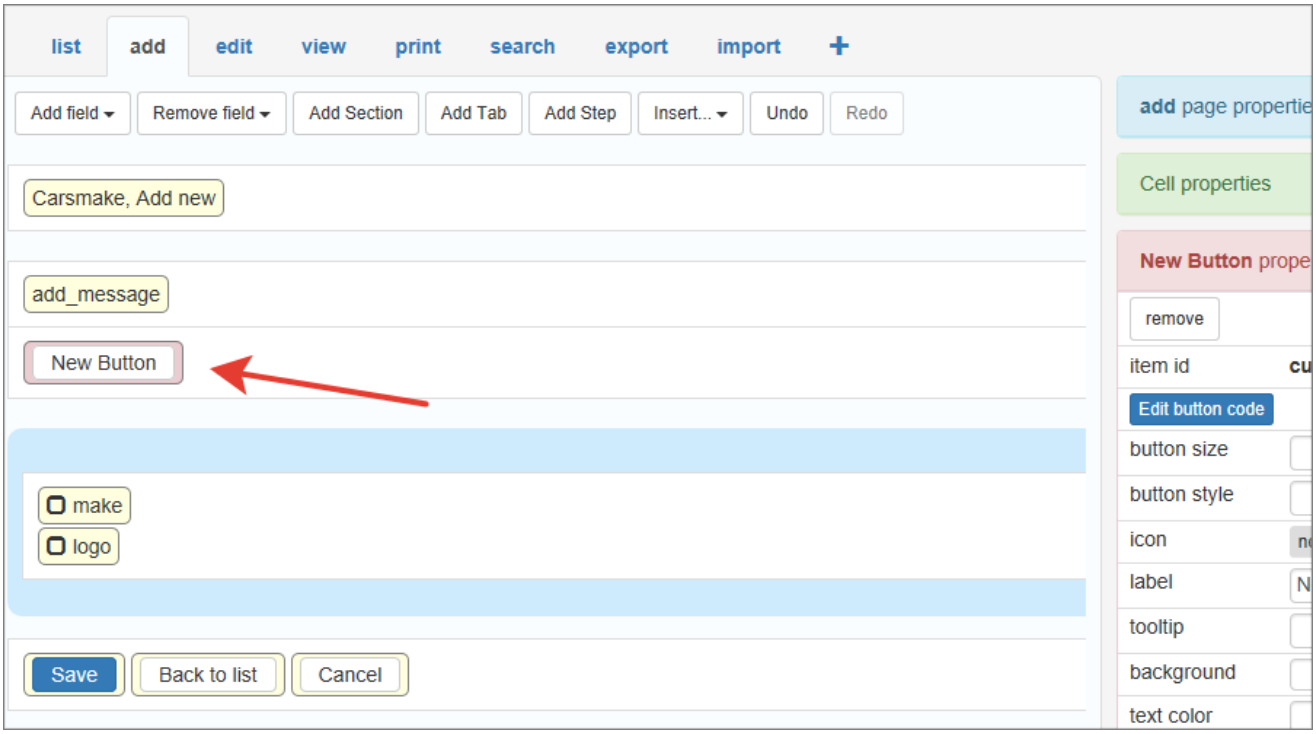


Note: **Custom buttons** utilize the Tri-part event system, that consists of three parts: Client Before, Server, Client After.

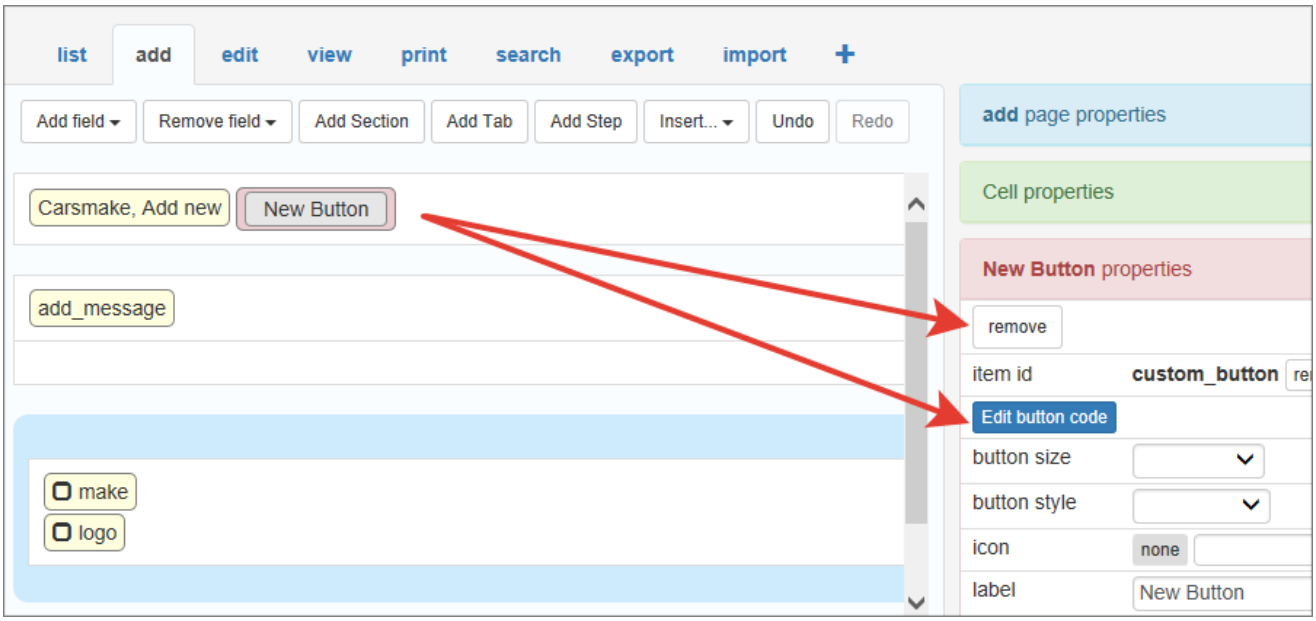
The **Client Before** part runs JavaScript code in the browser, then passes parameters to the **Server** part that runs PHP code, and then back to the browser to run the JavaScript code of the **Client After** part.

To learn more about these types of events, see Tri-part events.

5. Now the button is added to your web page. You can drag the button to change its location.



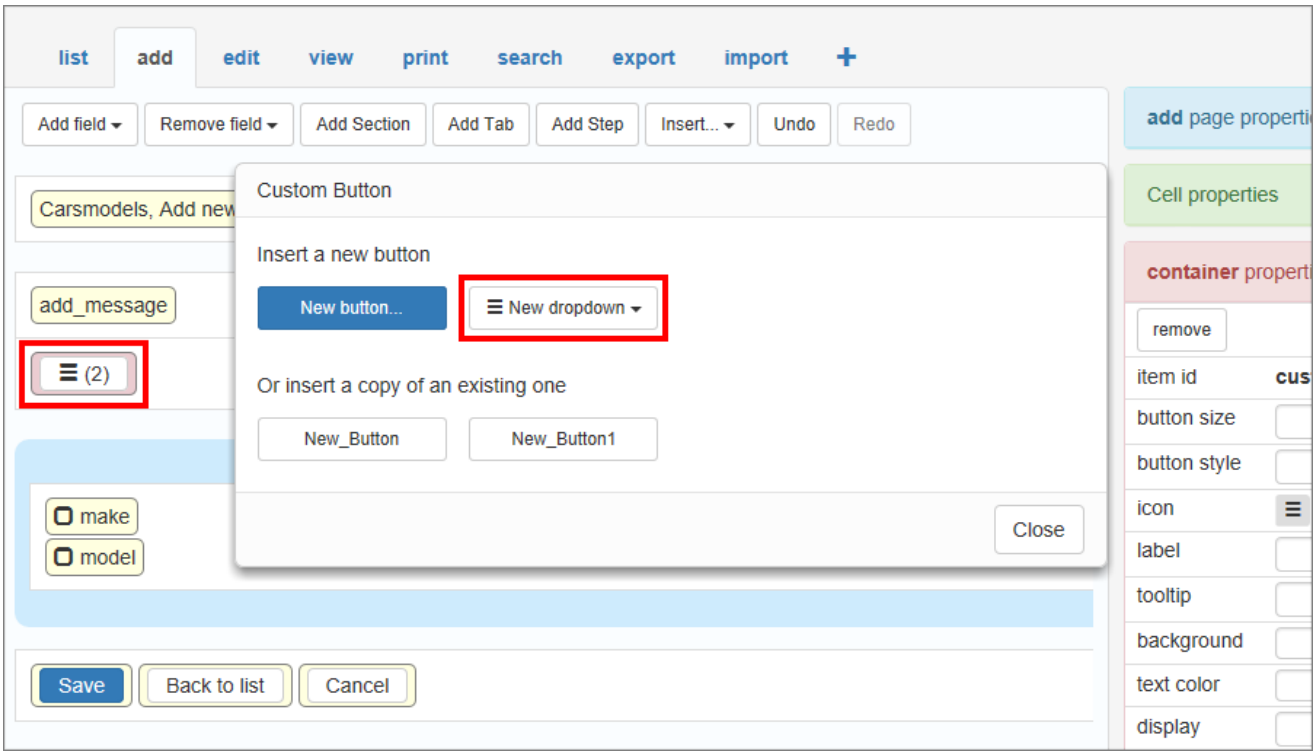
To edit the code, click **Edit button code** in **Properties**. You may also edit the code on the Events page. To delete the button, click **Remove** in **Properties**.



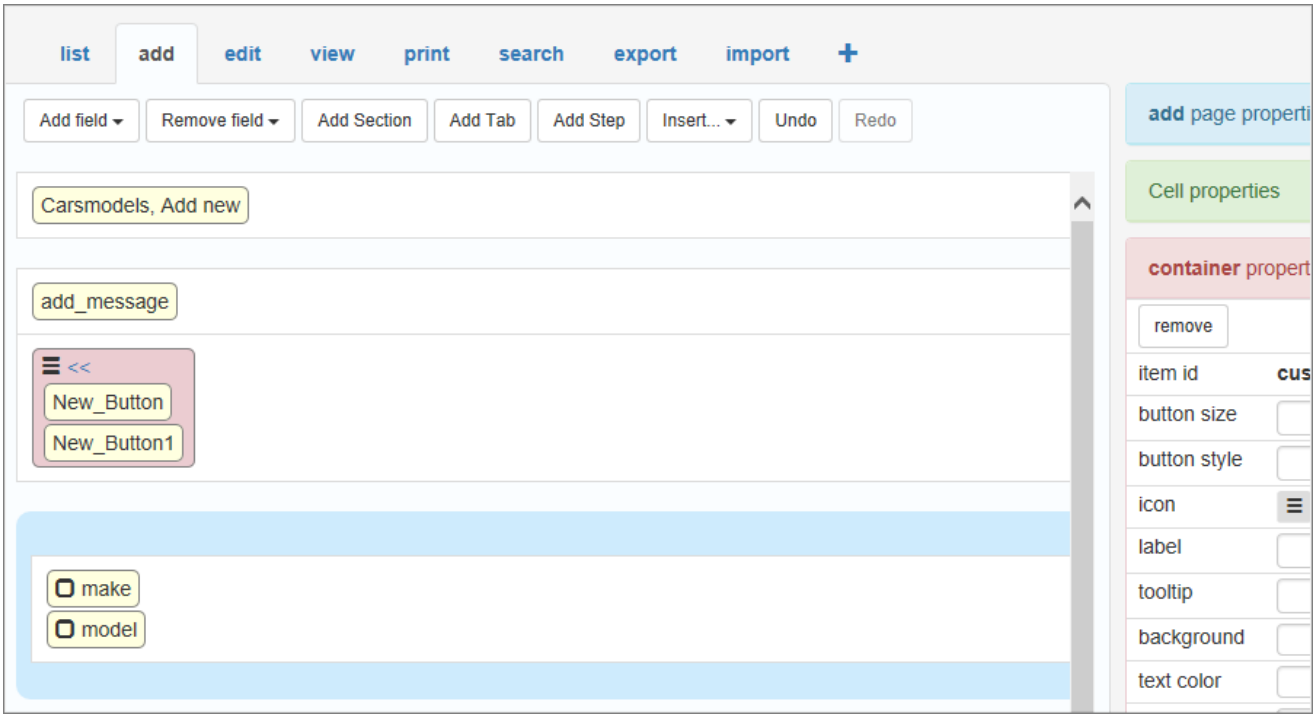
Inserting containers

The **container** is a specific dropdown element in which you can insert other elements using drag-n-drop.

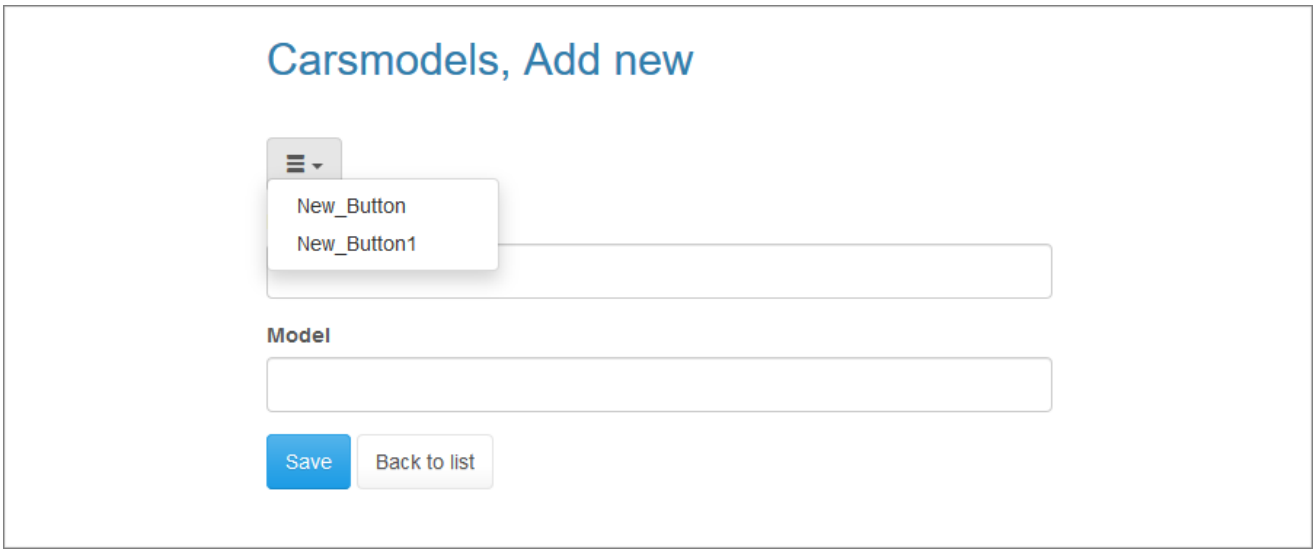
To insert a new **container**, do **Insert** -> **Custom button** -> **New dropdown**, then drag-n-drop it to any place you want.



You can see the number of elements within the **container** in the brackets next to it. Click this number to view the elements.



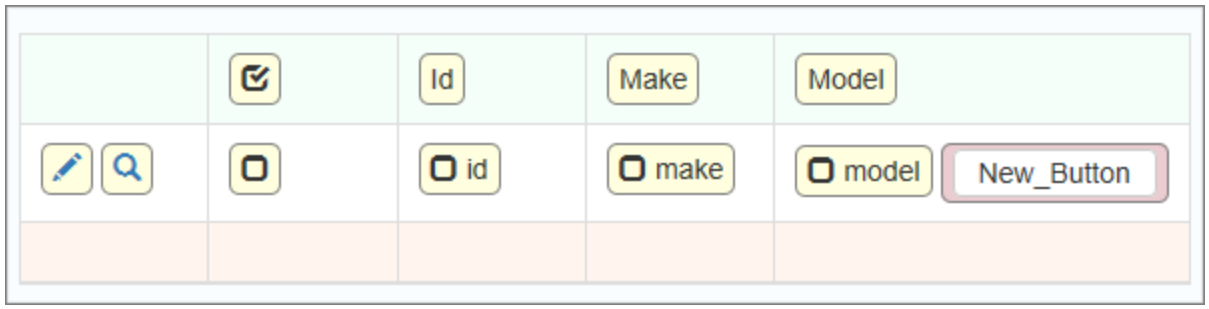
The result looks like this:

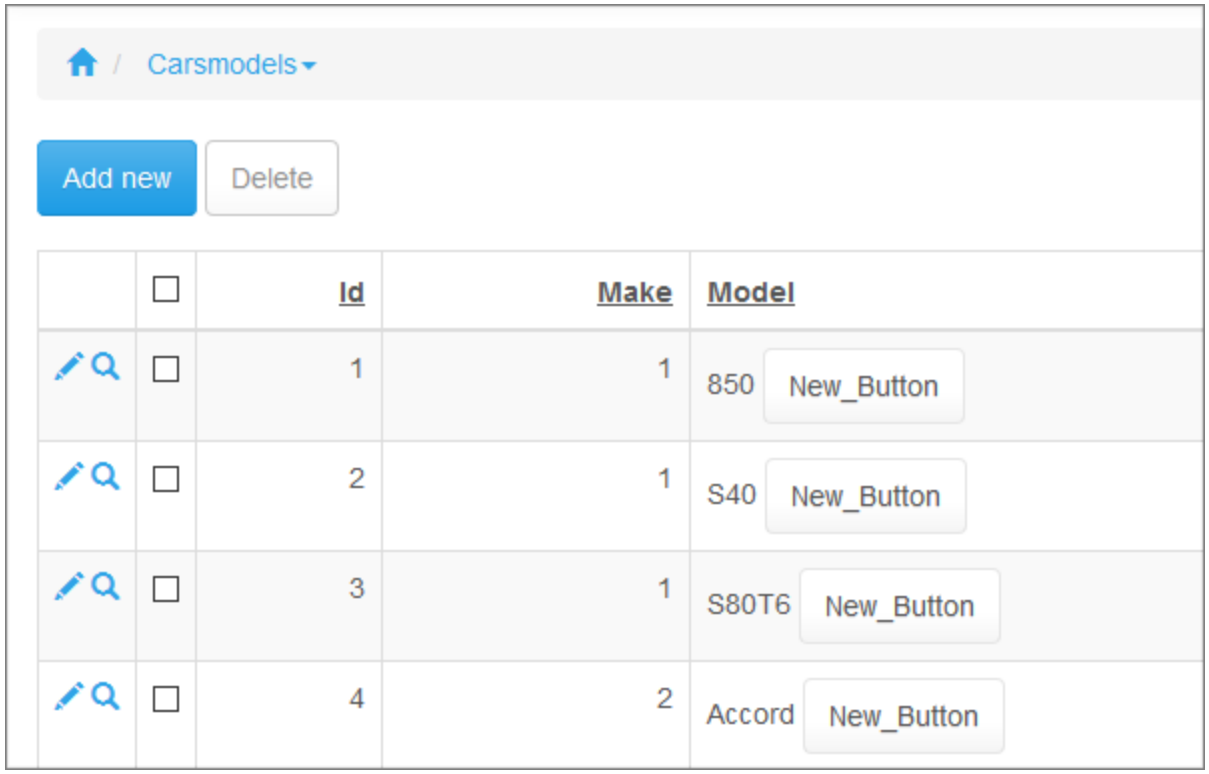


Insert a button into datagrid

You can insert a button into a data table to make it apply to each data record on a web page.

Switch to Advanced Grid first. Then do **Insert** -> **Custom Button** -> **New Button**. Then drag-n-drop the button to the grid or anywhere where it needs to appear.



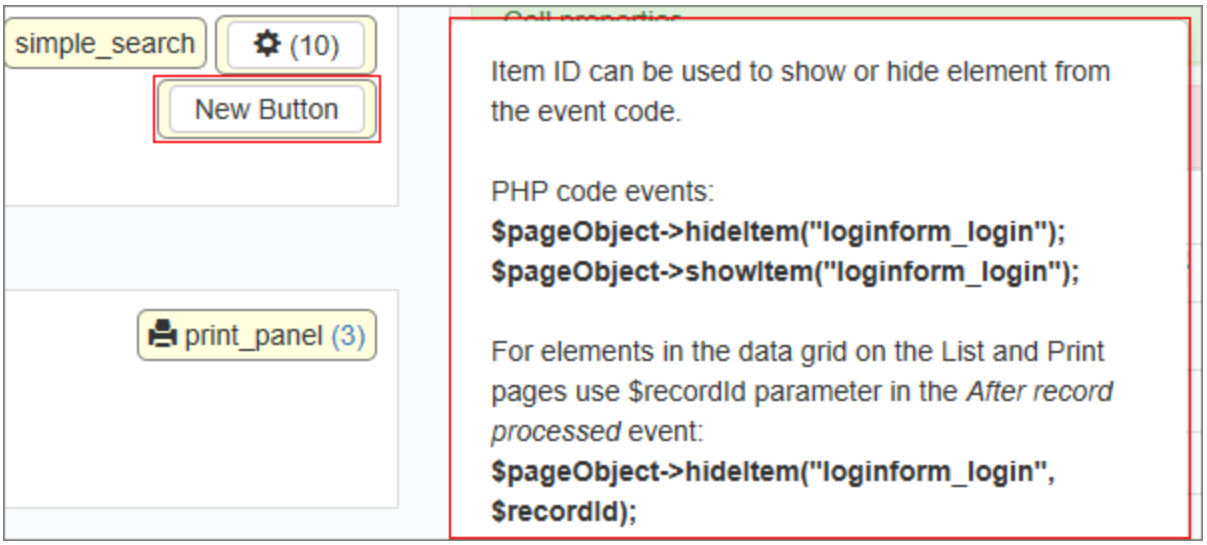


Note: a button inserted into a datagrid works only if you select a key field for the table on the Choose pages screen.

You can use the rowData and $button objects to program buttons inserted into a grid.

Hiding custom buttons

Select the button and click the **'?'** within the **Item ID** section on the right. It shows code examples of how to hide/show the button conditionally both on the server side and client side in PHP and JavaScript.



> **Note**: code examples in PHPRunner already use correct **Item ID**, so you can copy and paste them to your code.

Use the following code to show or hide the items:

1) To hide the item insert this code into the **Server** event:

```
$pageObject->hideItem("loginform_login");
```

2) To show the item insert this code into the **Server** event:

```
$pageObject->showItem("loginform_login");
```

3) For elements in the data grid on the **List** and **Print** pages, use the *$recordId* parameter in the After record processed event:

```
$pageObject->hideItem("loginform_login", $recordId);
```

In JavaScript events, use the *toggleItem* function to show or hide elements:

```
pageObj.toggleItem("loginform_login", true );
//or
pageObj.toggleItem("loginform_login", false );
```

Use the following code for elements in the grid:

```
pageObj.toggleItem("loginform_login", true, recordId );
```

The *recordId* can be obtained:

- with **row.id()** in Click actions;
- in the Custom button code when the button is inserted into the grid;
- with **pageObj.id** in Field events;
- with JS: Onload events from **Inline Add** and **Edit**.

Passing data between events

It's essential that a functional **Tri-part event** is able to pass data between its parts. For example, the **Client Before** part receives input from the user, passes it to the **Server** event. The latter runs some database queries using the data and passes the results to **Client after**, which shows the results to the user.

Two objects serve as links between the three parts of the event:

- **params** object passes data from the **Client Before** to the **Server** event.

- **result** object passes data from the **Server** to the **Client After** event.

Mind the syntax difference between the **Client** and the **Server** events. The **Client** code is JavaScript, and the **Server** code is PHP.

The key names, users, and variables are up to you. You can choose any names here.

You can also pass arrays and objects this way:

ClientBefore:

```
params["data"] = {
    firstname: 'Luke',
    lastname: 'Skywalker'
};
```

Server:

```
do_something( $params["data"]["firstname"] );
```

Examples

Example 1. Inserting an "Add this product to shopping cart" button

For the **Edit**/**View** pages, the **$keys** parameter in the **Server** event contains the information about the current record. You may use *$keys["KeyFieldName"]* to access a specific key column.

For example, there is a *Products* table. To insert an **Add this product to shopping cart** button on the **View** page, add the following code to the **Server** event:

```
global $dal;
$record = $button->getCurrentRecord();
if ($record["ProductID"])
{
```

```
  //add new records to the ShoppingCart table
  //save current username in the UserID field
  $ShoppingCart = $dal->Table("ShoppingCart");
  $ShoppingCart->Product = $record["ProductID"];
  $ShoppingCart->Quantity = 1;
  $ShoppingCart->UserID = $_SESSION["UserID"];
  $ShoppingCart->Add();
}
$result["txt"] = "Product was added";
```

and this code to the **Client After** event:

```
var message = result["txt"] + ".";
```

For more information about using the **Data Access Layer (DAL)**, see Data Access Layer.

Example 2. Send records selected on the List page via email

To send records selected on the **List** page via email, add the following code to the **Server** event:

```
$email_msg = "";
$email_msg.= "List of records";
$i=1;
while($data = $button->getNextSelectedRecord())
{
  $email_msg.= "Record: ".($i++)."\r\n";
  $email_msg.= "Make: ".$data["make"]."\r\n";
  $email_msg.= "Qwerty: ".$data["qwerty"]."\r\n";
  $email_msg.= "\r\n";
}
//send email
$email = "test@test.com";
$subject = "Sample subject";
runner_mail(array('to' => $email, 'subject' => $subject, 'body' => $email_msg));
```

For more information, see getNextSelectedRecord().

Example 3. Modifying the value of a field for all selected records on the List page

To modify the value of a field for all selected records on the **List** page, add the following code to the **Server** event:

```
while($record = $button->getNextSelectedRecord())
{
  $sql = "update Invoices set Status='Paid' where InvoiceID="
  .$record["InvoiceID"];
  DB::Exec($sql);
}
```

Example 4. Inserting a button redirecting to another page

To make a button redirect to another page, add the following code to the **Client Before** event or **Client After** event:

```
location.href="http://cnn.com";
```

Example 5. Showing a message to the customer for a limited time

Let's say you have a button that retrieves a piece of sensitive info from the server that needs to be shown to the customer for a short time. After that, the message needs to be hidden. To do this, add the following code to the **Client After** event:

```
// this message includes the SSN retrieved from the server
var message = result["txt"];
ctrl.setMessage(message);

// clear the message after 5 seconds
setTimeout(function(ctrl)
{
  ctrl.setMessage("");
}
, 5000);
```

Example 6

Let's say you want to add a button to the **View** page that redirects to one of the application pages, depending on the value of the *Category* field. To do so, you may add this code to the **Server** event:

```
// get the value of the current record and pass it to the ClientAfter event
$result["record"] = $button->getCurrentRecord();
```

and this code to the **Client After** event:

```
if (result.record["Category"]=='Schools')
  location.href="Schools_list.php";
else if (result.record["Category"]=='Organisations')
  location.href="Organisations_list.php";
else if (result.record["Category"]=='Careers')
  location.href="Careers_list.php";
```

For more information, see getCurrentRecord().

Example 7

To refresh the page, add the following code to the **Client After** event:

```
location.reload();
```

Page Designer articles:

- About Page Designer
- Working with common pages
- Working with table pages
- Working with cells
- Working with page elements
- Working with additional pages
- Page layout and grid type
- Tabs and sections
- Insert code snippet
- Insert map
- Insert Text / Image
- Insert standard button
- "View as" settings
- "Edit as" settings
- "Filter as" settings

See also:

- RunnerPage class: hideItem()
- RunnerPage class: showItem()
- JavaScript API: RunnerPage object > toggleItem()
- Button object: getCurrentRecord()
- DAL method: Add()
- Function runner_mail
- Button object: getNextSelectedRecord()
- Database API: Exec()
- JavaScript API: AJAX helper object > submit()
- Tri-part events
- About Dialog API
- Choose pages screen
- Choose fields screen
- Miscellaneous settings
- Editor screen
- Event editor

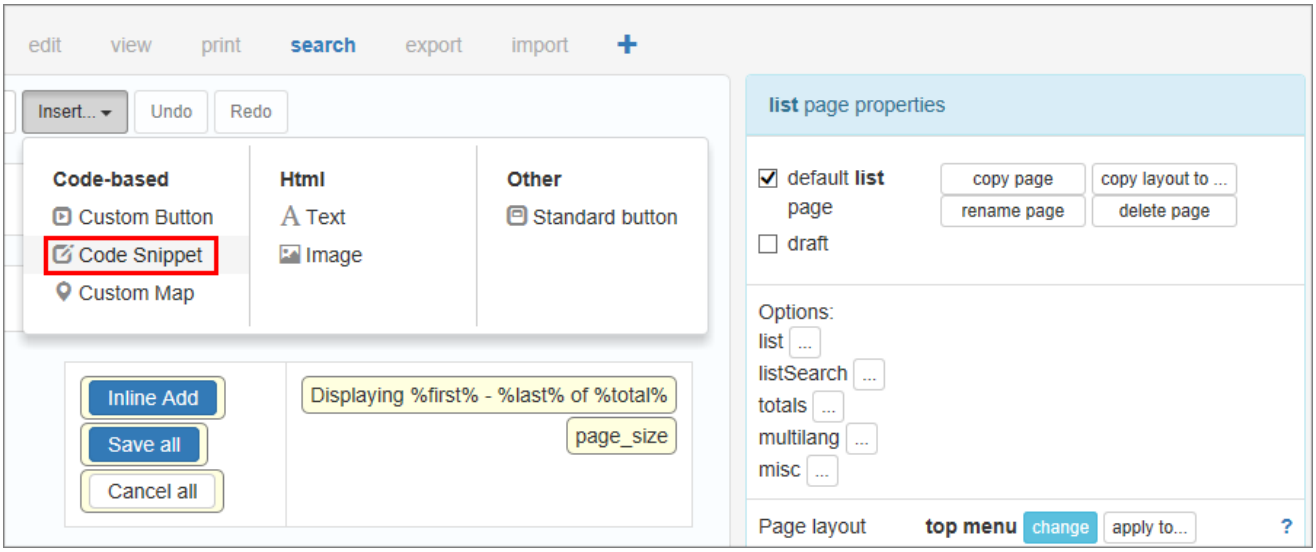
2.16.10 Insert code snippet

You can add PHP **code snippets** to modify the appearance and functionality of the generated web pages. For example, you can add additional control elements to a web page or display some information.

To add a PHP **code snippet**, follow the instructions below.

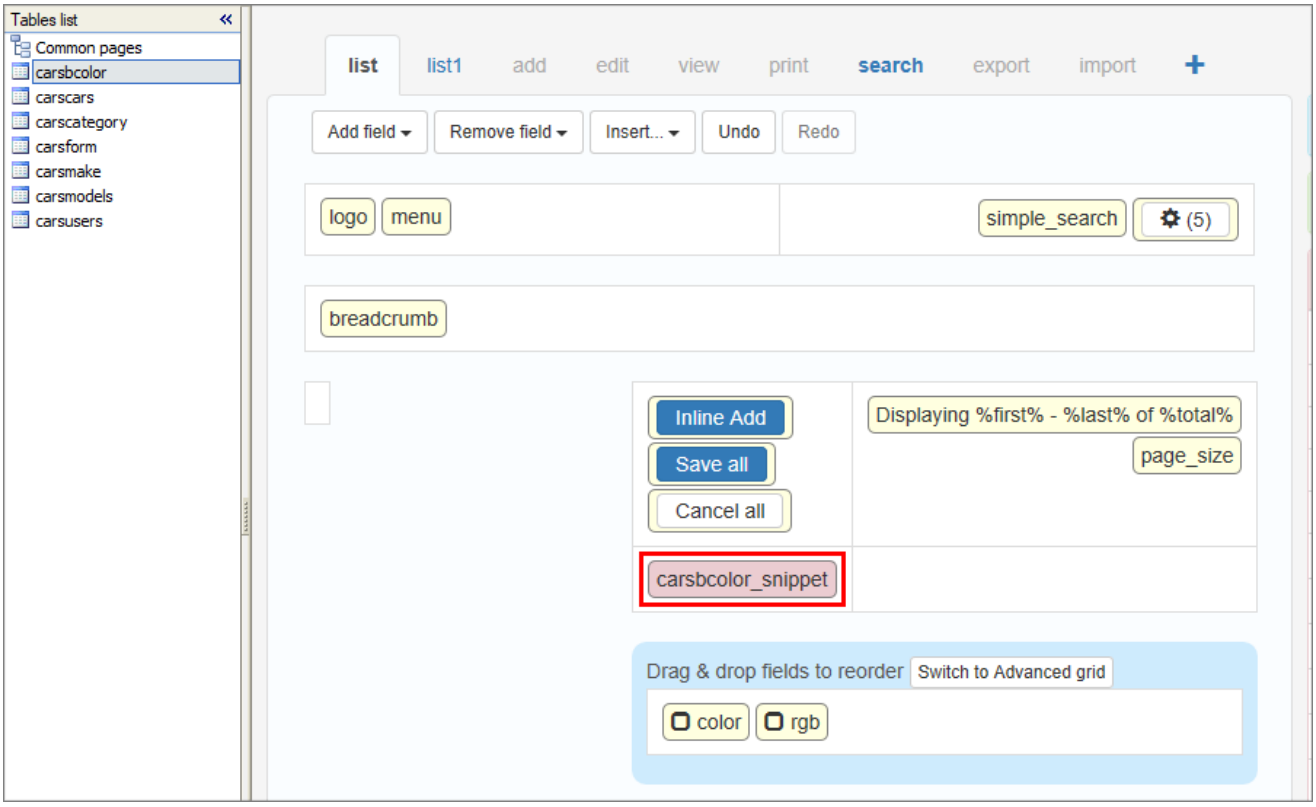
1. Proceed to the **Page Designer** and select a page you wish to modify.

2. Click **Insert** - **Code snippet** on the main toolbar.

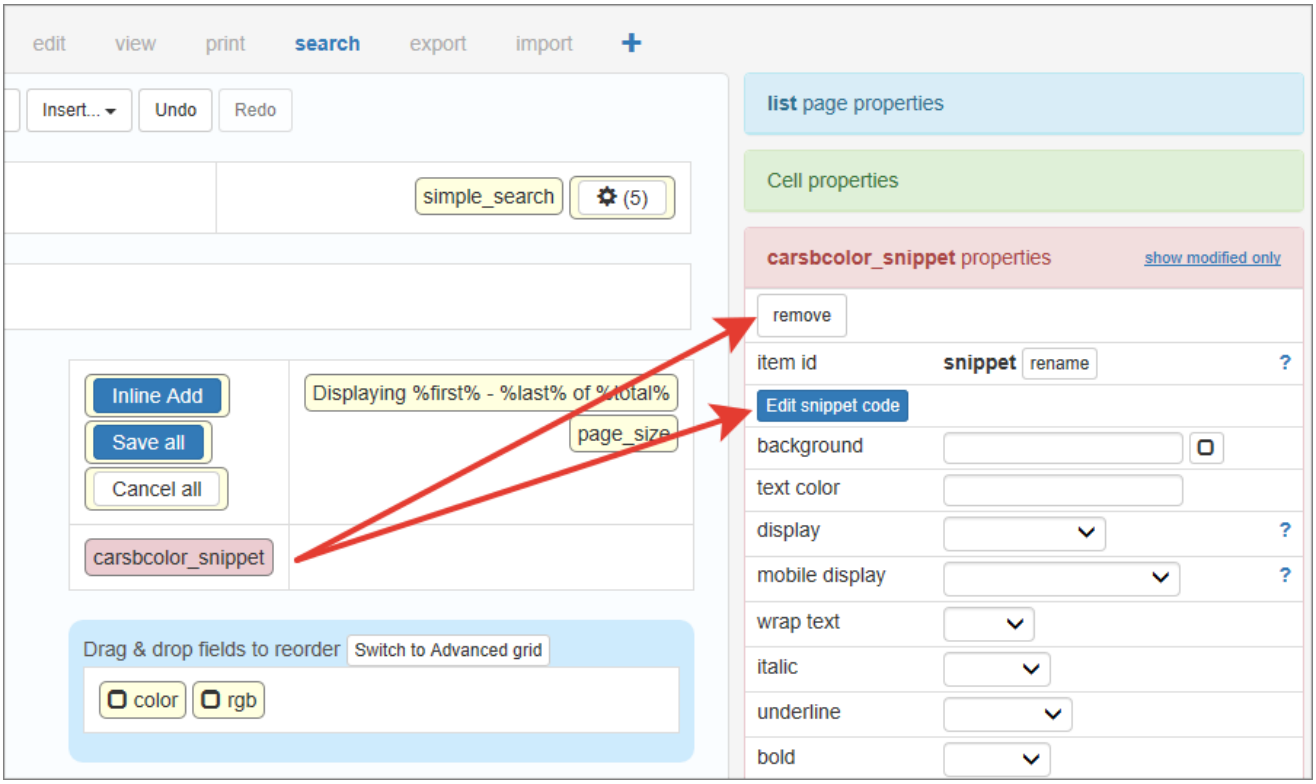


3. Add the PHP code into the opened window and click **OK**.

4. The new page element labeled *<Table name>_snippet* appears on the page. If your code snippet displays some data on a web page, move the "snippet" element to where you wish to see that data.



Use the **Edit snippet code** button to edit a code snippet. You can edit a code snippet on the Events page as well. To delete a code snippet, click the **Remove** button.



Examples

1. Display the current time:

```
echo now();
```

2. You can get access to fields of the current record with the code snippet added to the **Edit**/**View** page. For example, you can print the value of the field "*Make*" using the following code:

```
global $pageObject;
$record = $pageObject->getCurrentRecord();
echo $record["Make"];
```

3. Add a dropdown list box with specific values.

4. Show a list of customer orders.

5. Show data from a master table on the details view/edit/add page.

Page Designer articles:

- About Page Designer
- Working with common pages
- Working with table pages
- Working with cells
- Working with page elements
- Working with additional pages
- Page layout and grid type
- Tabs and sections
- Insert custom button
- Insert map
- Insert Text / Image
- Insert standard button
- "View as" settings
- "Edit as" settings
- "Filter as" settings

See also:

- Choose pages screen
- Choose fields screen
- Miscellaneous settings
- Editor screen
- Event editor

2.16.11 Insert map

Quick jump

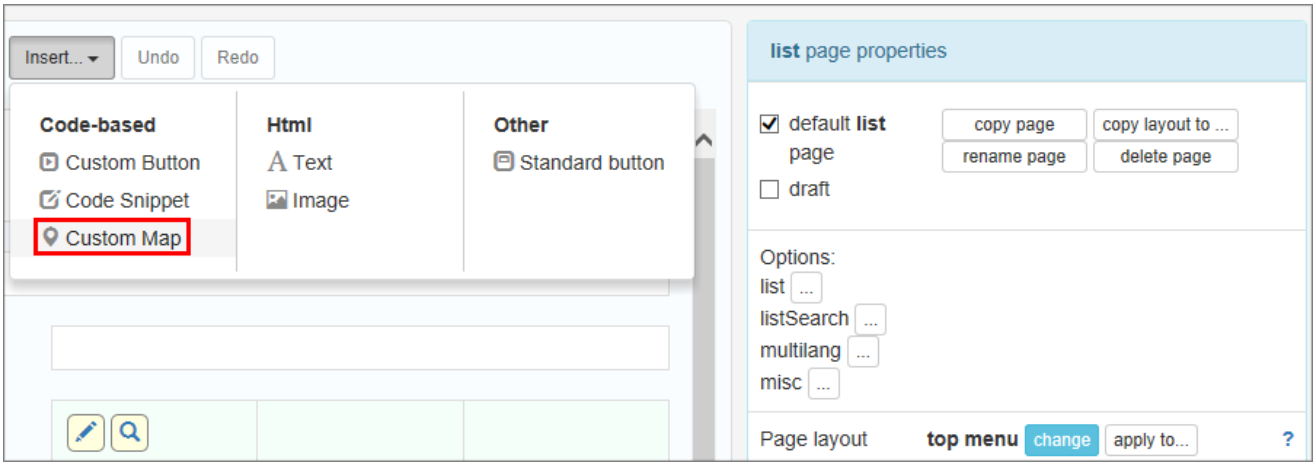
Inserting custom map

You can display the location data as a separate map for each table record or insert a large map to the List page. PHPRunner supports several map providers, so that you can select the suitable one on the **Miscellaneous** screen -> Map settings.

To insert a large map into the **List** page, follow the instructions below:

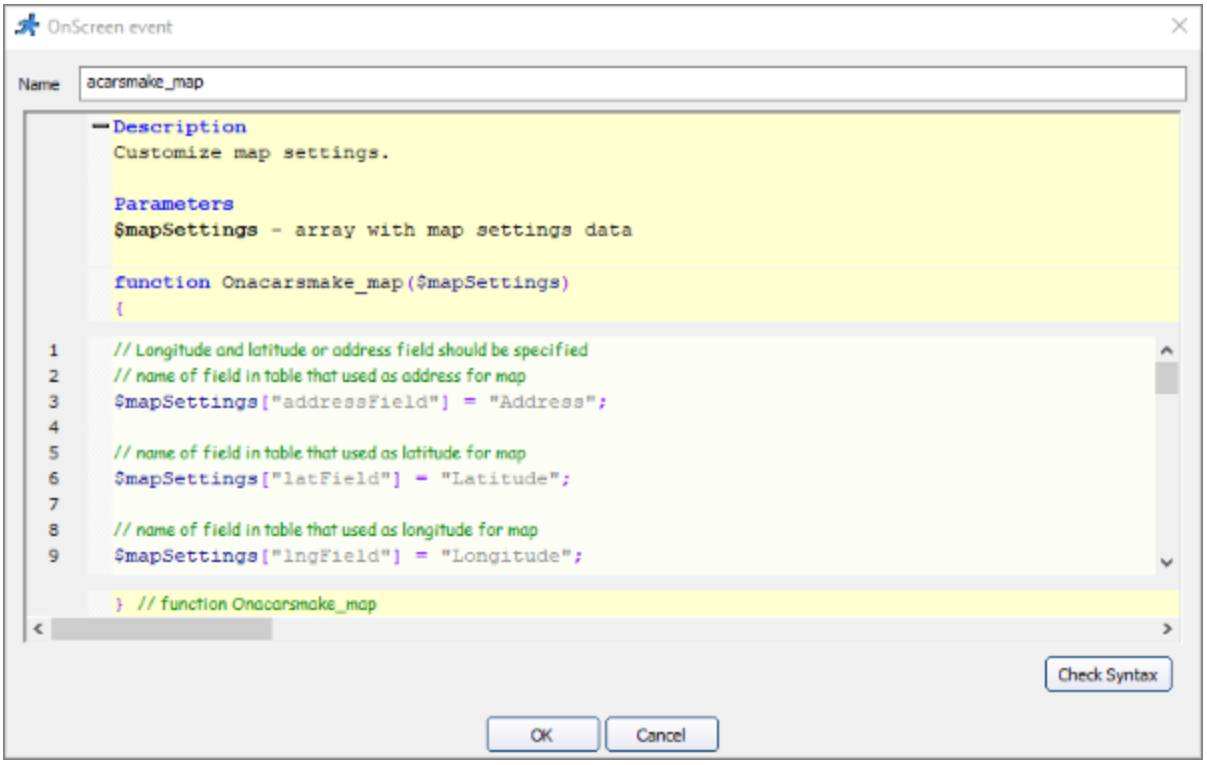
1. Proceed to the **Page Designer** and select a page you wish to modify.

2. Click **Insert** - **Custom Map** on the main toolbar.



3. Enter the address, latitude, and longitude as text fields. If you do not have a field with address or if you want only to use the Latitude/Longitude fields, comment out the following line:

```
//$mapSettings["addressField"] = "Address";
```



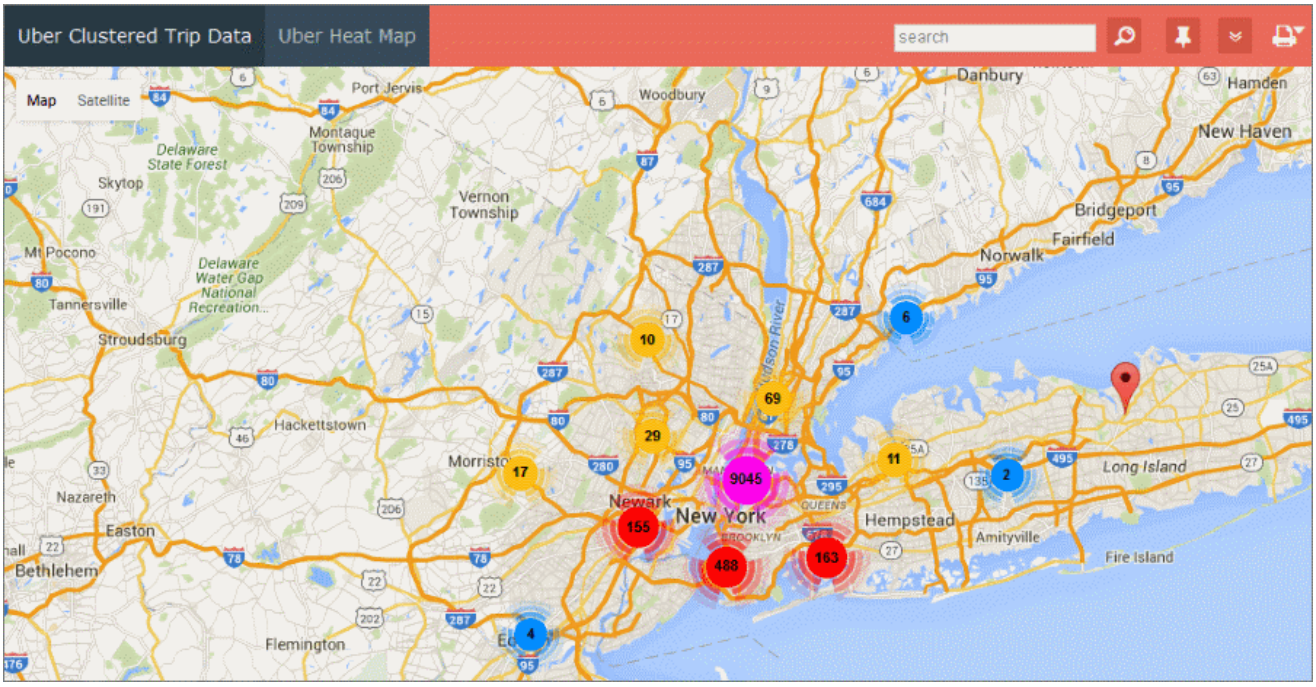
If you use Google Maps and store the latitude/longitude information, you can enable the clustering or heat map option.

To enable clustering, add this code:

```
$mapSettings['clustering'] = true;
```

You can use custom clustering icons as well. There is a "*source*" subfolder in your project folder. Proceed there and create an "*images*" folder. Place images named *m1.png*, *m2.png*, *m3.png*, *m4.png*, and *m5.png* into the folder. These icons are used when the clustering is turned on.

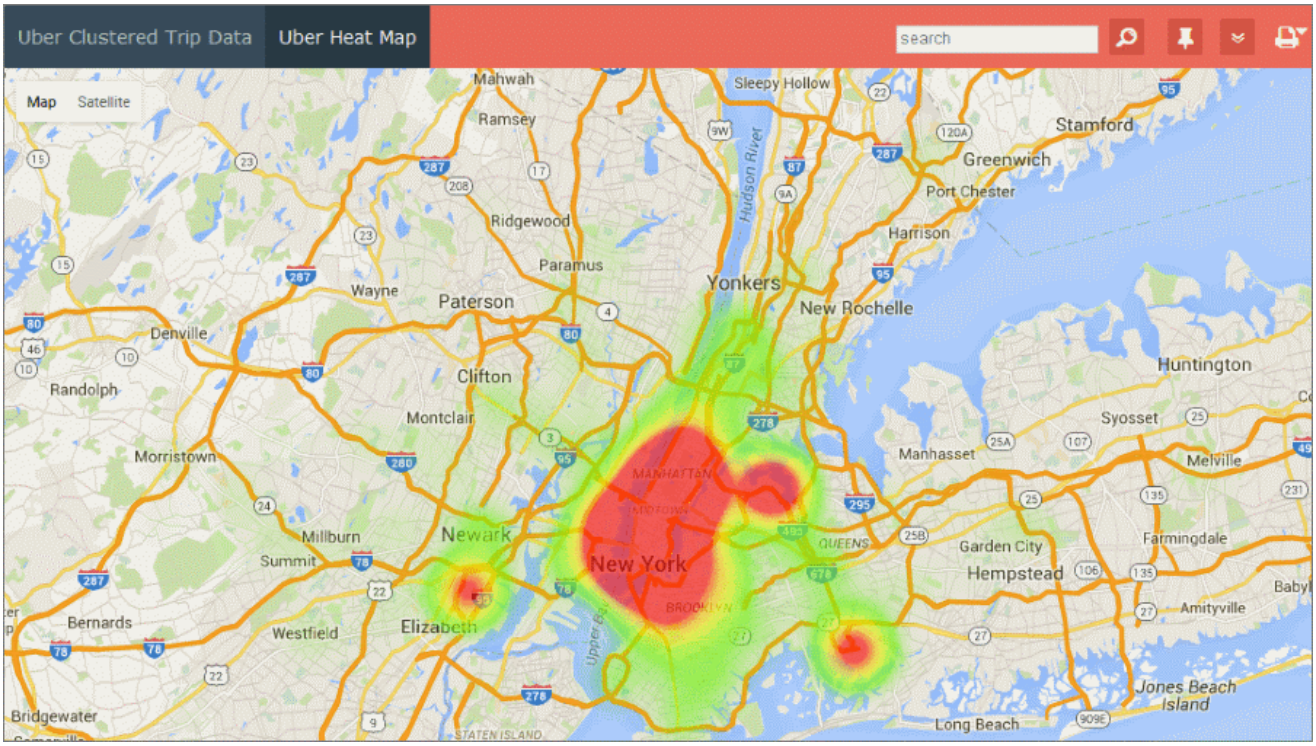
An **example** of a clustered map:



To enable the heat map, add this code:

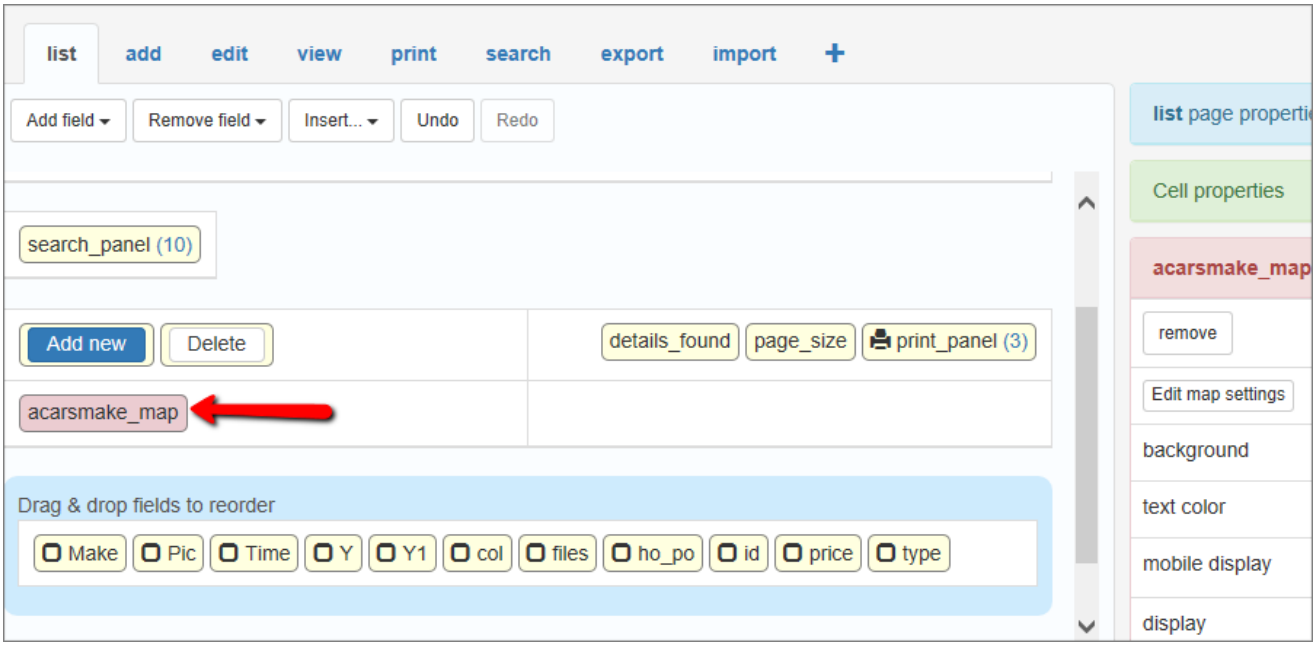
```
$mapSettings['heatMap'] = true;
```

An **example** of a heat map:

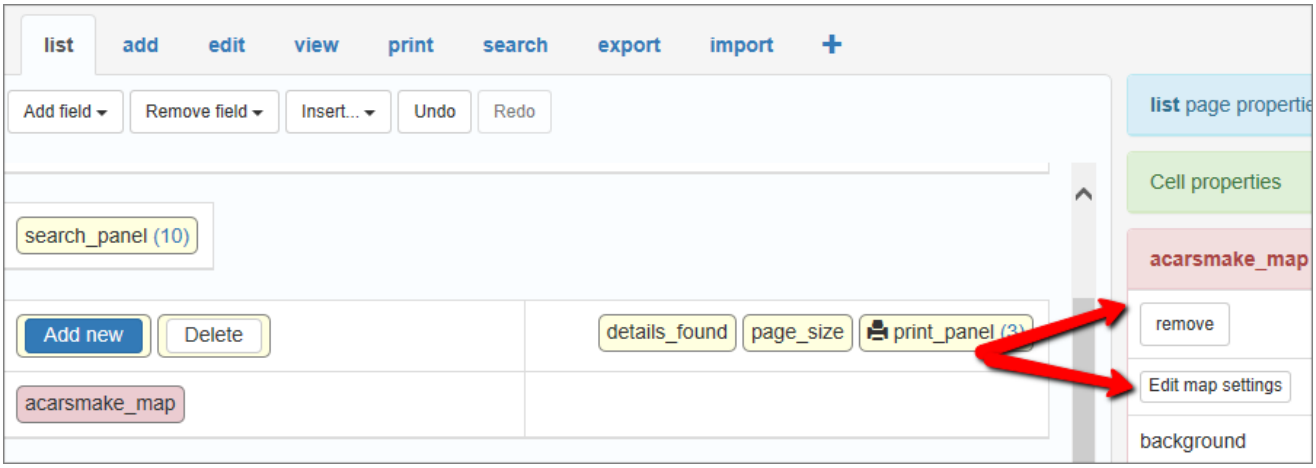


4. Click **OK**.

5. Now the map is added to your web page. You can drag the map element to change its location.

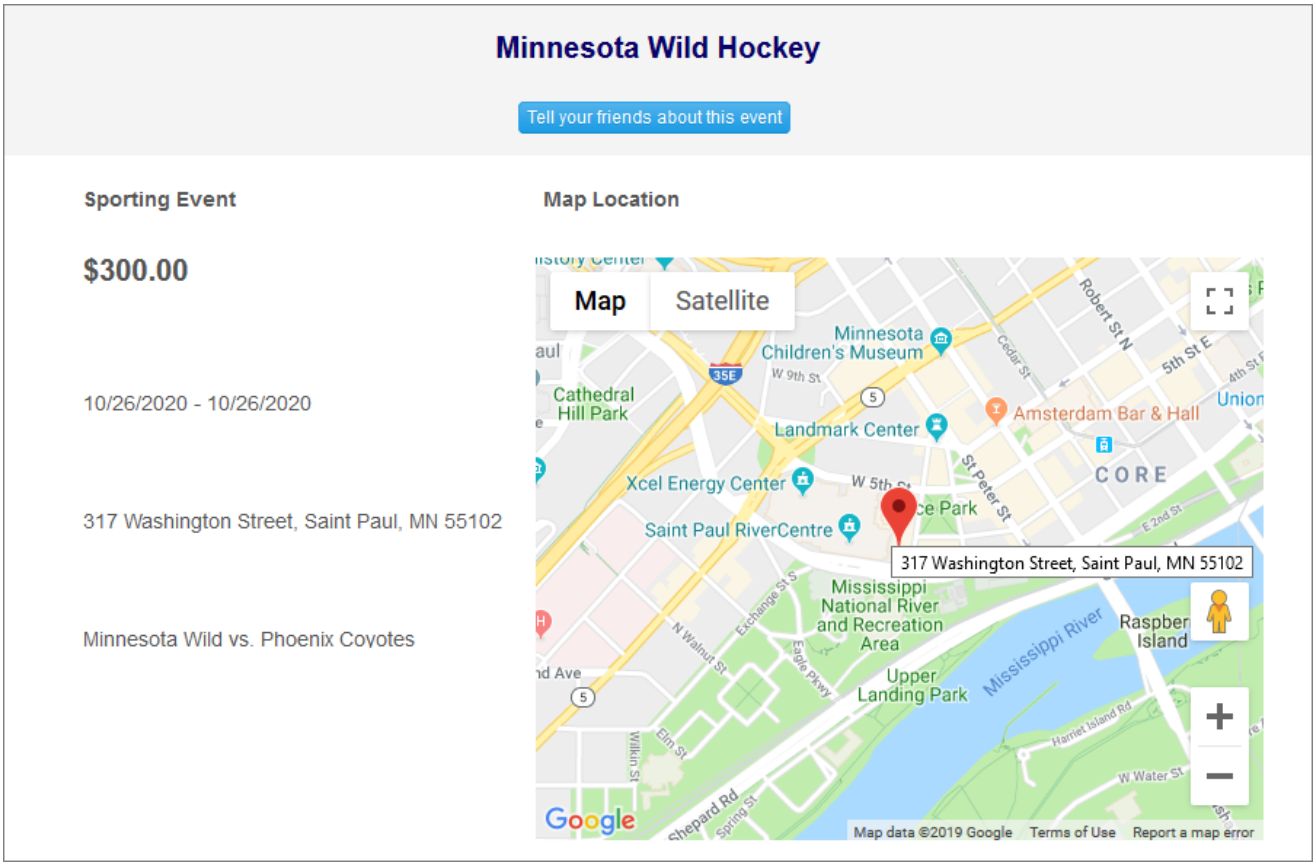


Click **Edit map settings** to edit the code. Click **Remove** to delete it.



You can also edit the code on the Events page.

The result might look like this:



Note: clickable markers on the large map point to the **View record** page.

How to place a description on each map marker

1. Modify an SQL Query to create a calculated field with a custom name and address:

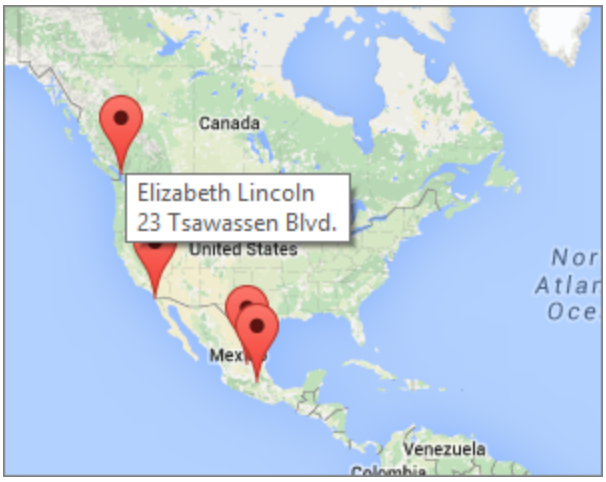
```
SELECT
        CustomerID,
        CompanyName,
        ContactName,
        ContactTitle,
        Address,
        Lat,
        Lng,
concat(ContactName, '\n', Address) as DisplayOnMap
FROM customers
```

Notice the *'\n'* argument: it allows you to create multiline descriptions. *concat()* function is MySQL specific, similar functions exist in all databases.

2. Insert a map into the **List** page. Here are the settings you may use for this specific table:

```
// Longitude and latitude or address field should be specified
// name of the field in the table that used as an address for map
$mapSettings["addressField"] = "DisplayOnMap";

// name of field in table that used as a latitude for map
$mapSettings["latField"] = "Lat";

// name of field in table that used as a longitude for map
$mapSettings["lngField"] = "Lng";
```

Latitude and *Longitude* fields are required, as we use the *address* field for marker description purposes.



How to display GPS locations (map markers) by date

For example, you have new geo-coordinates that were inserted into the database on a specific day. If you to display the GPS locations (map markers) by date, add a WHERE clause to the SQL query for the table you insert your map into.

You can use this query to select today's data:

```
SELECT * FROM mytable WHERE DATE(posted) = CURDATE()
```

Maps API and geocoding

Maps API providers have limits on the number of geocoding requests you can send per day, and also on a request rate. For example, if you use Google Maps API and have a page with 20 records with a map for each one, only about 10 maps will be displayed. To overcome those limits, you need to open a Premier account with Google that costs $10,000 per year.

The solution is to use latitude/longitude pairs instead of addresses for mapping purposes. Enable the Geocoding option on the **Choose pages** screen to update latitude/longitude information each time when a record is added or updated. The main question is how to convert the existing addresses to latitude/longitude pairs? We have added a small utility to PHPRunner that does the job for you. Here are the instructions:

1. Proceed to the table that stores addresses and add two new fields to that table. Fields need to be able to store floating-point numbers. Use Decimal(10,6) or Double in MySQL. In MS Access use Number with 'Field size' Double.

2. Proceed to the field that is set up as a Map. Make sure that the *Address* and *Latitude*/*Longitude* fields are selected.

3. Proceed to the **List** page, choose to display all records, select all records, click **Edit**, then **Save all**. It takes some time to update all records, but you only need to do this once. New/updated records will update longitude/latitude automatically.

To speed up this process, you may want temporarily remove **View as** Map field from the **List** page and add it back once the geocoding process is finished.

Page Designer articles:

- About Page Designer
- Working with common pages
- Working with table pages
- Working with cells
- Working with page elements
- Working with additional pages

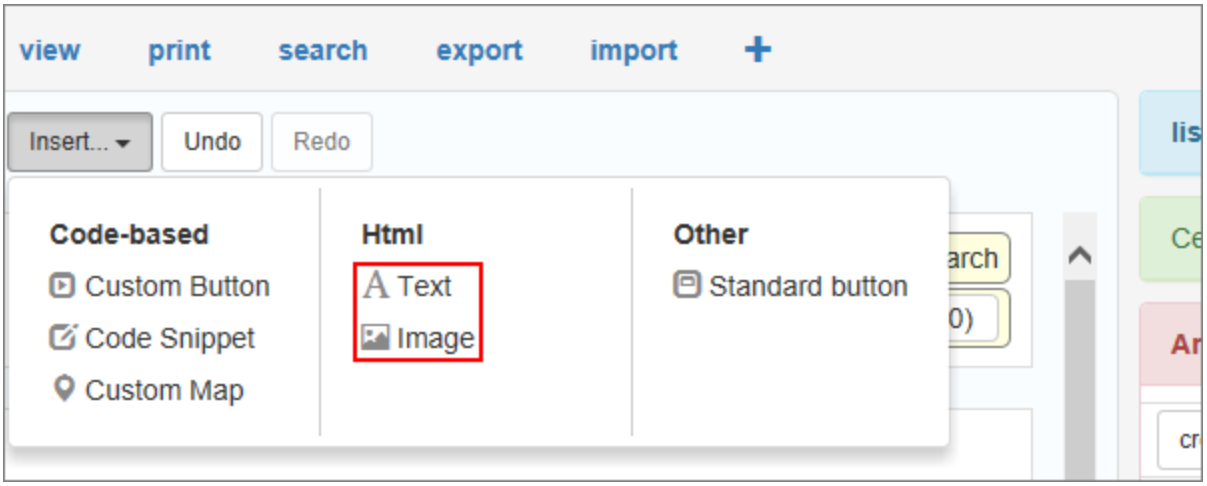
- Page layout and grid type
- Tabs and sections
- Insert custom button
- Insert code snippet
- Insert Text / Image
- Insert standard button
- "View as" settings
- "Edit as" settings
- "Filter as" settings

See also:

- Choose pages screen
- Choose fields screen
- Miscellaneous settings
- Editor screen
- Event editor

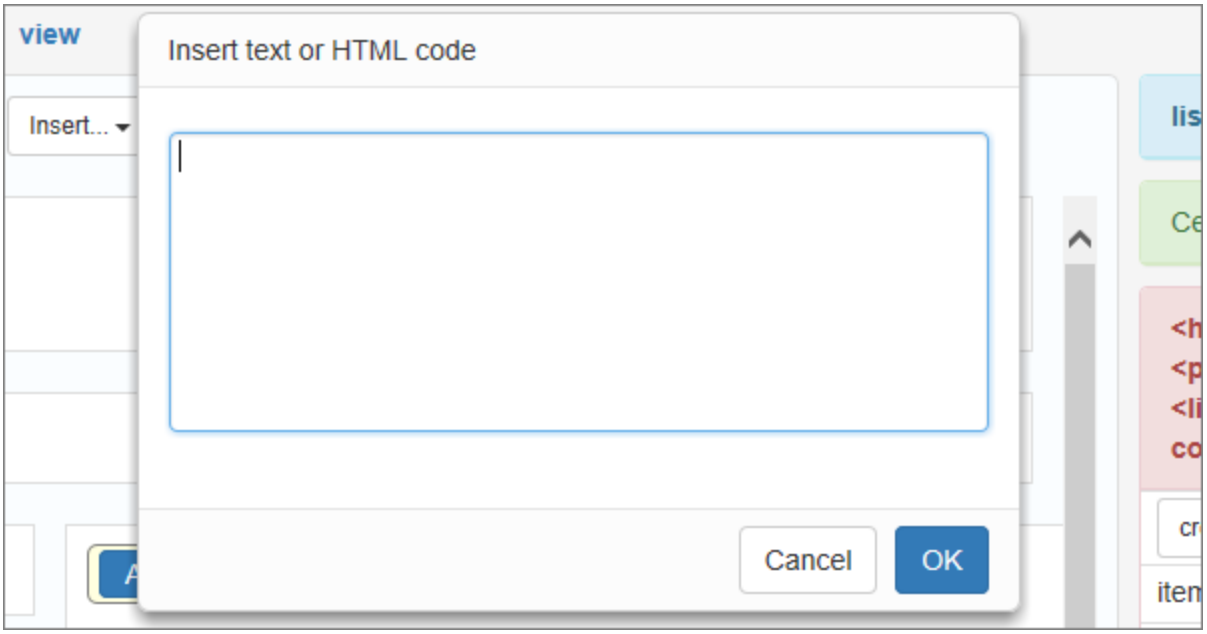
2.16.12 Insert Text / Image

The **Insert** button also allows adding custom texts/HTML code and images. They behave as standard page elements: have adjustable properties, and support drag-n-drop.

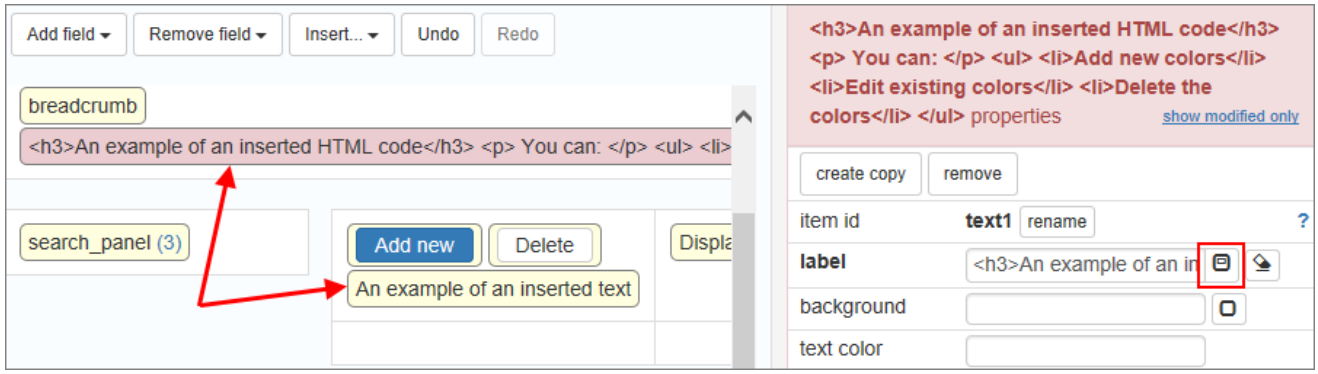


Insert Text/HTML code

To insert a custom text/HTML code, click the **Insert** button, and choose **Text**. Type or paste the text or code you want into the popup window.

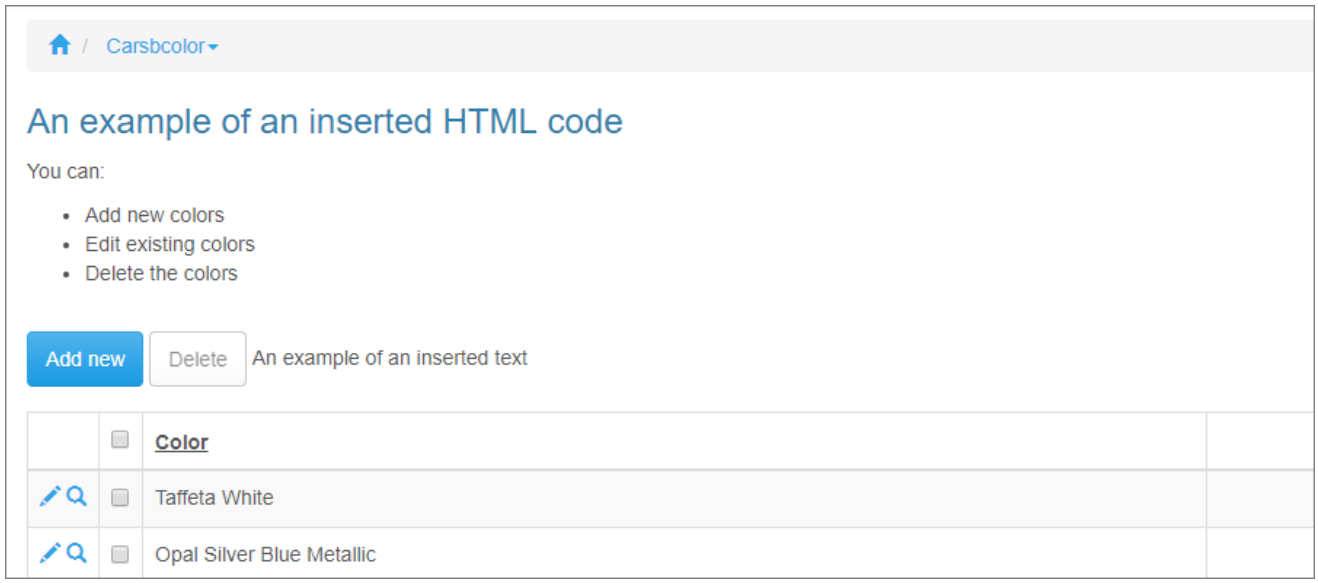


Let's add, for example, a text and an HTML code to the page:



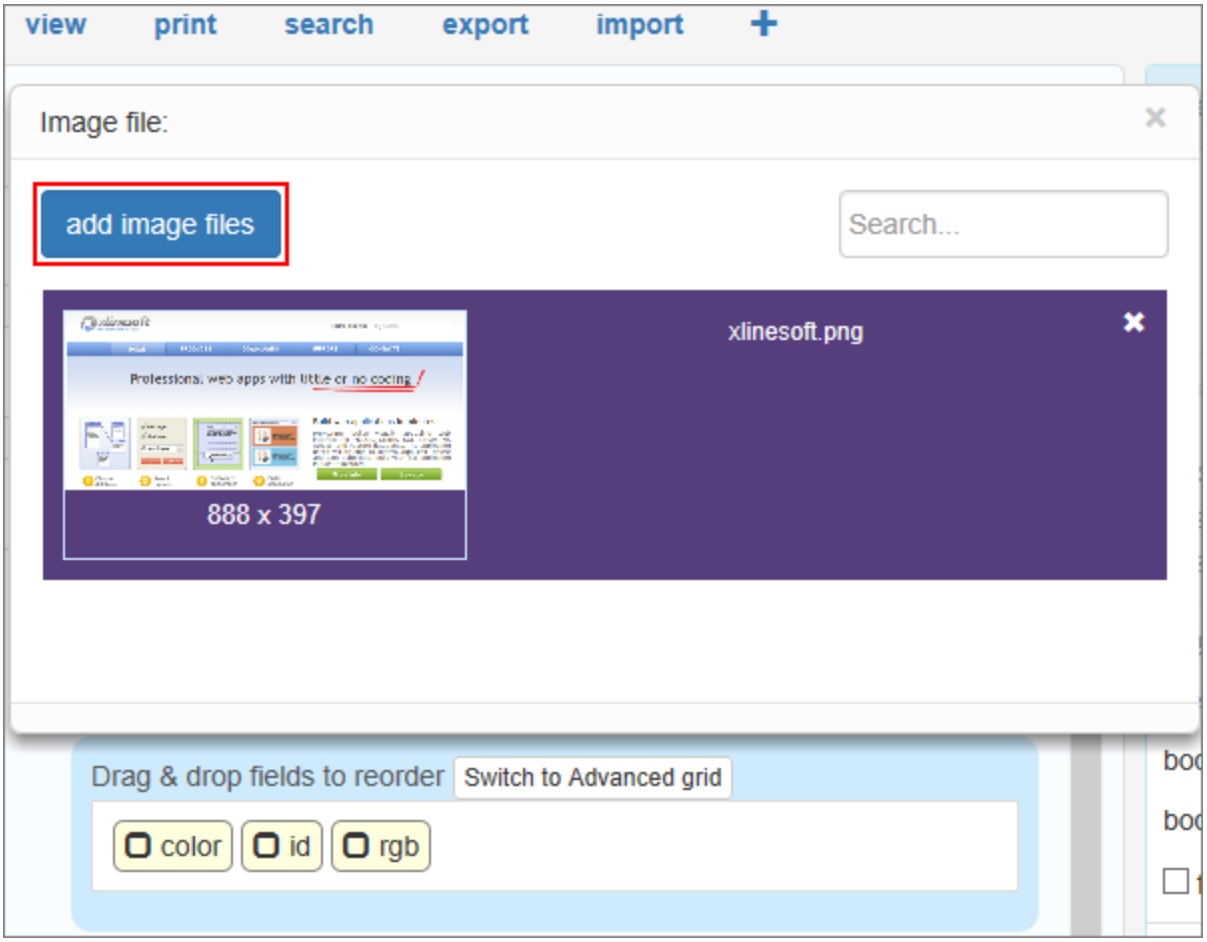
> **Note**: you can edit the inserted text or code in the *Label* field in the properties. If there is a lot of text/code, you can edit it in a popup window with the **edit in popup** button.

Here is how the inserted text and HTML code look like in the generated app:



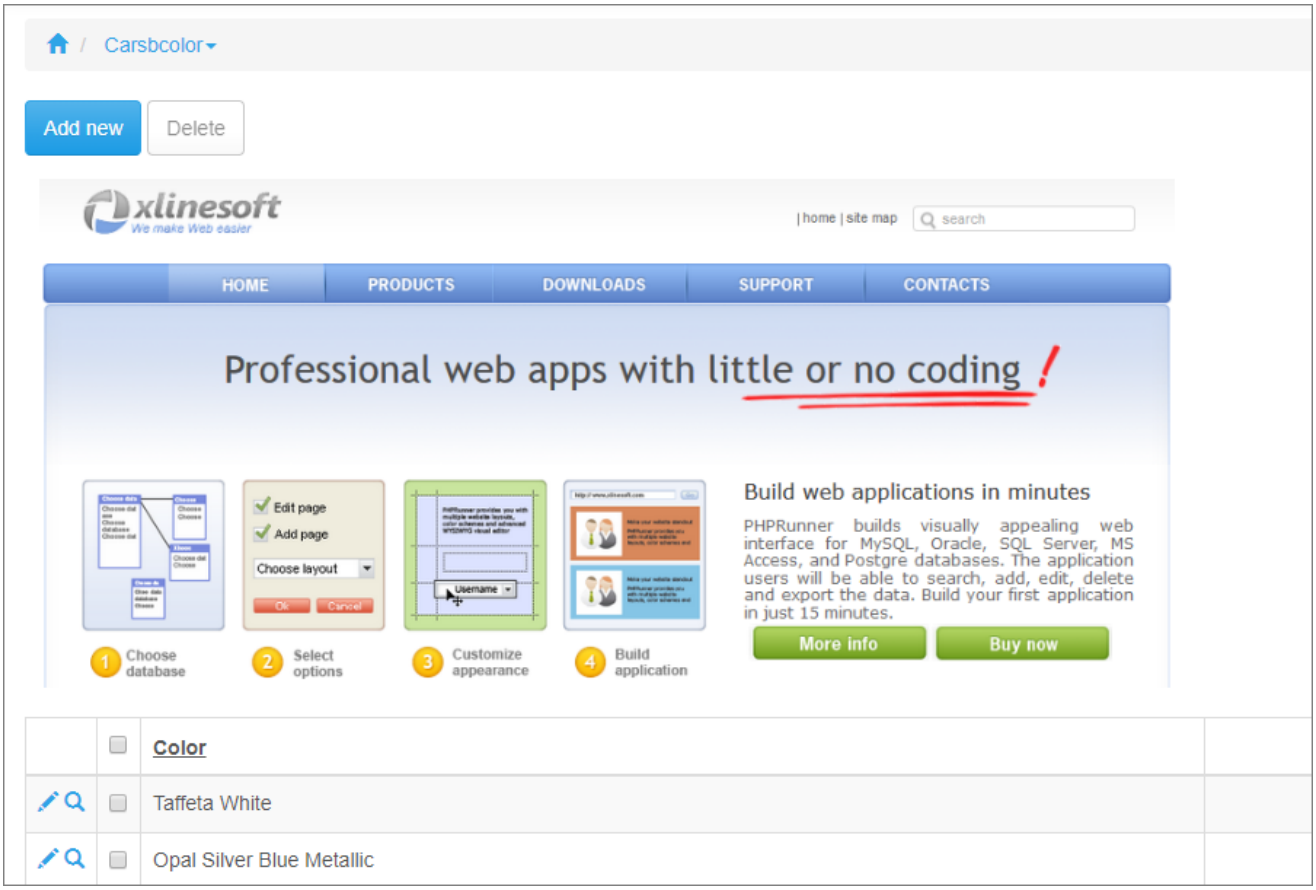
Insert Image

To insert a custom image, click the **Insert** button, and choose **Image**. If you haven't added any images yet, click the **add image files** button and choose an image to add. Click the image to add it to the page.



Note: if there are many images in the *Image file* popup, use the search panel to find the image you want.

Here is how the inserted image looks like in the generated app:



Page Designer articles:

- About Page Designer
- Working with common pages
- Working with table pages
- Working with cells
- Working with page elements
- Working with additional pages
- Page layout and grid type
- Tabs and sections
- Insert custom button
- Insert code snippet
- Insert map

- Insert standard button
- "View as" settings
- "Edit as" settings
- "Filter as" settings

See also:

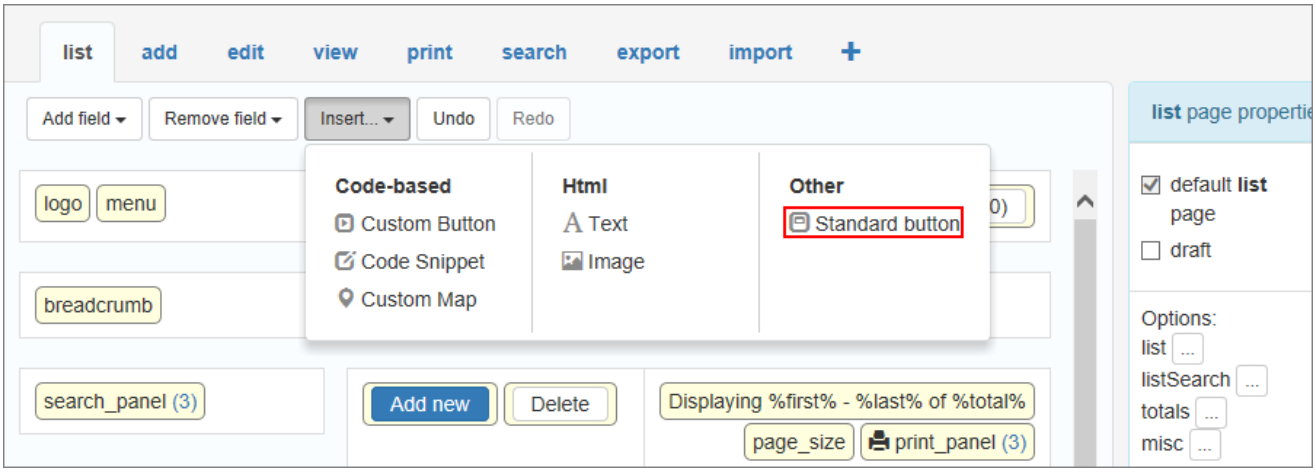
- Choose pages screen
- Choose fields screen
- Miscellaneous settings
- Editor screen
- Event editor

2.16.13 Insert standard button

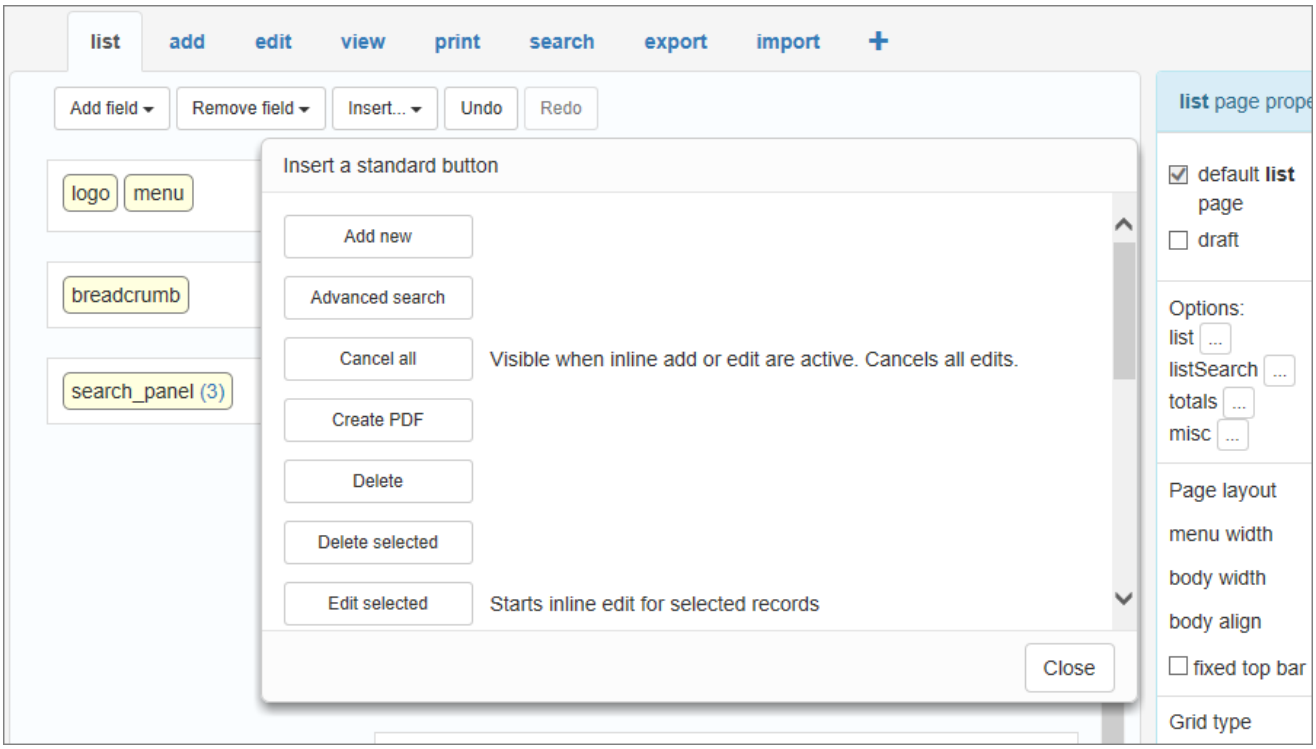
Quick jump

How to insert standard buttons

Standard buttons are buttons like **Add new**, **Create PDF**, **Delete**, and **Print**. To add a standard button to the page, click **Insert**, then - **Standard button**.



A popup with the list of all standard buttons appears. Click on the button you wish to add, and drag-n-drop it to any place on the page.

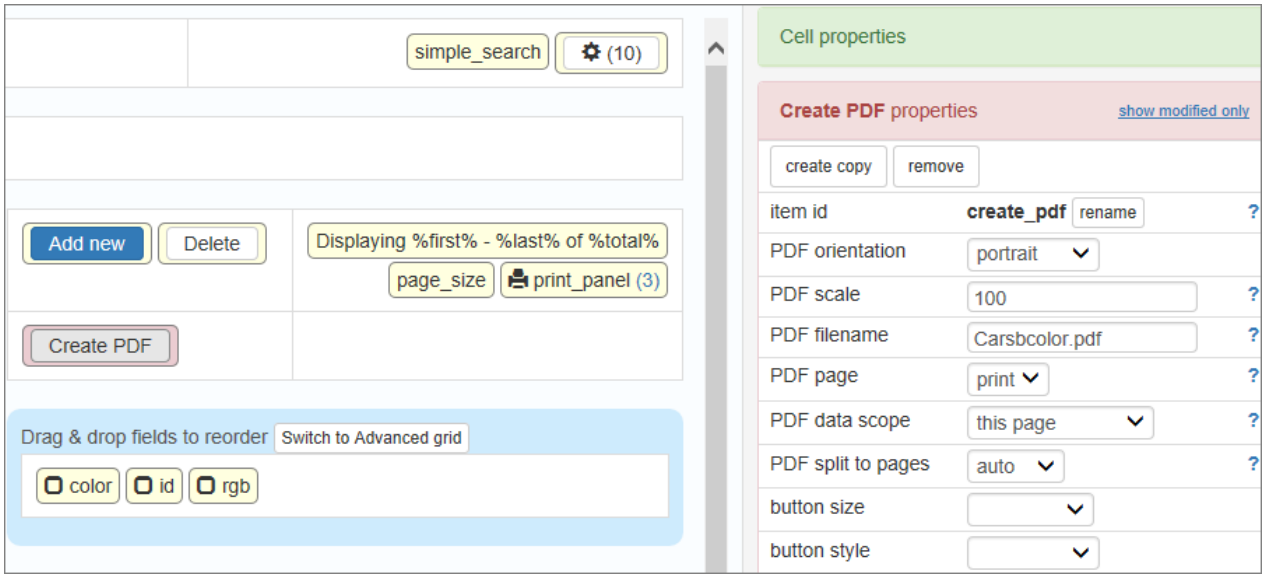


Standard buttons lists for different types of pages

The list of available buttons depends on the type of page:

List page

- **Add new**. Opens the **Add** page.
- **Advanced search**. Opens the advanced search page. For more information, see Search and Filter settings.
- **Cancel all**. Visible when an inline add or edit is active. Cancels all edits.
- **Create PDF**. Downloads the table as a PDF document. You can set the orientation, scale, filename, page type, data scope and split to pages for the resulting file in the properties.



- **Delete/Delete selected**. Both these buttons delete the selected records.
- **Edit selected**. Edits the selected records.
- **Export results/Export selected**. Exports all of the results/the selected records into an .xls, .doc, or .csv file to download. For more information, see Export/Import pages.
- **Hide/Show search panel**. Hides or shows the search panel. For more information, see Search and Filter settings.
- **Import**. Allows importing data from .xls, .csv files, or a simple text. For more information, see Export/Import pages.
- **Inline Add**. Visible when the **List** page allows inline adding. This button opens an inline adding interface.
- **Print**. Opens the **Print** page.
- **Save all**. Visible when an inline add or edit is active. Saves all records that are being edited.
- **Save search**. Saves the current search for later retrieval. For more information, see Search and Filter settings.

- **Update selected**. Allows quickly changing multiple records. For more information, see Update selected page.

Standard buttons: Add page

- **Back to list**. Opens the **List** page.
- **Cancel**. Cancels the adding and closes the window. Visible only in a popup window.
- **Next step**. Proceeds to the next step. For more information, see Working with table pages.
- **Reset**. Resets the values in the fields.
- **Save**. Saves the values into a new record.

Standard buttons: Edit page

- **next/prev**. These buttons switch between the records in the table.
- **Back to list**. Opens the **List** page.
- **Cancel**. Cancels the editing and closes the window. Visible only in a popup window.
- **Next step**. Proceeds to the next step. For more information, see Working with table pages.
- **Reset**. Resets the values in the fields.
- **Save**. Saves the edited values into the current record.
- **Update N records**. Visible on the **Update selected** page. Saves the edited values of the selected records. "N" is the number of selected records.
- **View**. Opens the **View** page.

Standard buttons: View page

- **next/prev**. These buttons switch between the records in the table.
- **Back to list**. Opens the **List** page.
- **Close window**. Closes the window. Visible only in a popup window.
- **Edit**. Opens the **Edit** page.
- **Next step**. Proceeds to the next step. For more information, see Working with table pages.
- **PDF View**. Creates a PDF document of the **Print** or **View** page. For more information, see PDF view settings.

Standard buttons: Print page

- **PDF View**. Creates a PDF document of the **Print** or **View** page. For more information, see PDF view settings.

Standard buttons: Search page

- **Back to list**. Opens the **List** page.
- **Cancel**. Cancels the search and closes the window. Visible only in a popup window.
- **Reset**. Resets the values in the fields.
- **Search**. Searches the table for the entered values.

Standard buttons: Export page

- **Cancel**. Cancels the export and closes the window. Visible only in a popup window.
- **Export**. Exports the results into an .xls, .doc, or .csv file to download. For more information, see Export/Import pages.

Standard buttons: Import page

The **Import** page doesn't have any standard buttons.

Page Designer articles:

- About Page Designer
- Working with common pages
- Working with table pages
- Working with cells
- Working with page elements
- Working with additional pages
- Page layout and grid type
- Tabs and sections
- Insert custom button
- Insert code snippet
- Insert map
- Insert Text / Image
- "View as" settings

- "Edit as" settings
- "Filter as" settings

See also:

- Choose pages screen
- Choose fields screen
- Miscellaneous settings
- Editor screen
- Event editor

2.16.14 "View as" settings

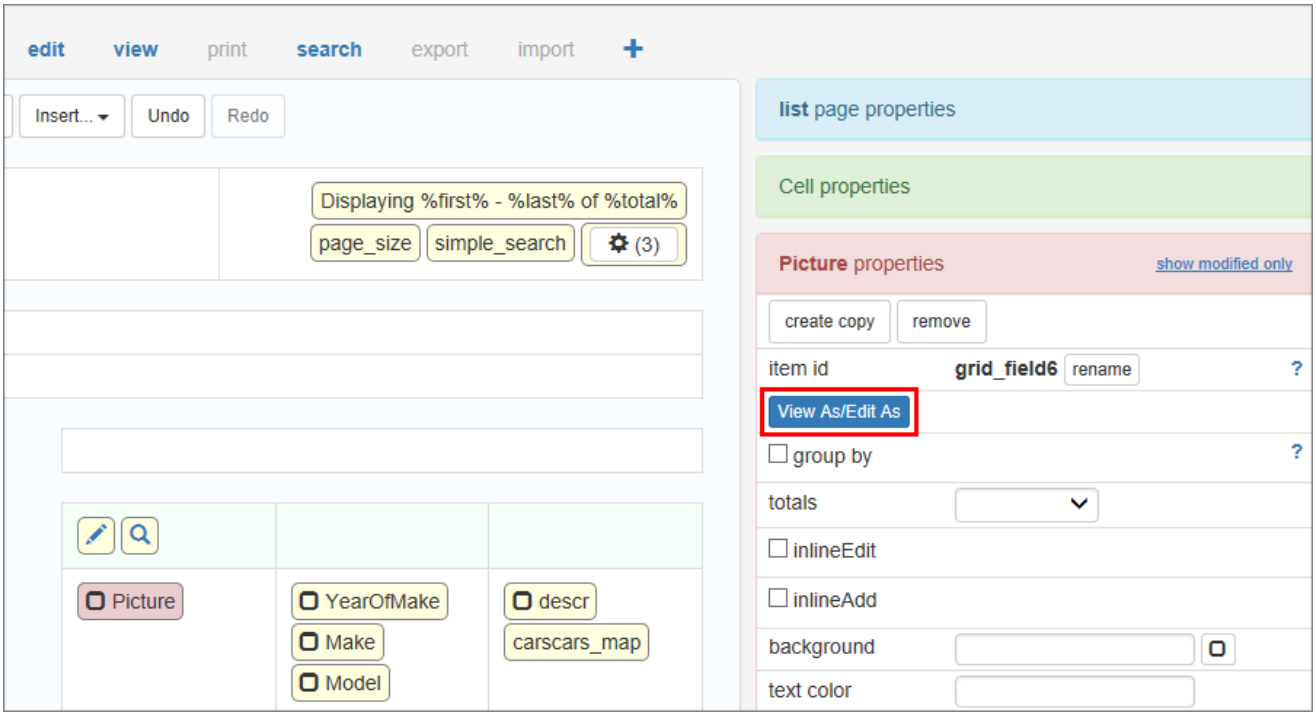
2.16.14.1 "View as" settings

Quick jump

Using "View As" settings

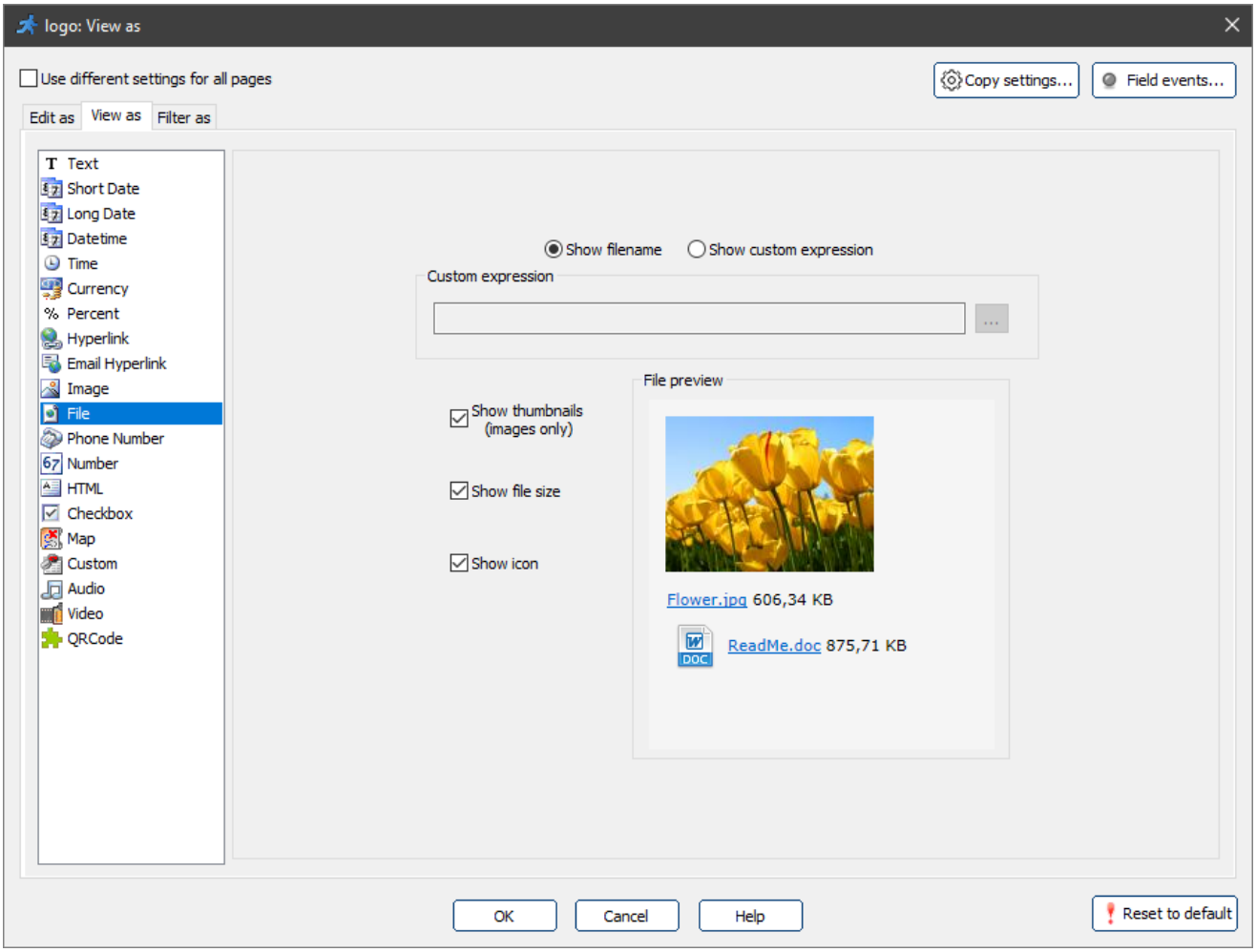
You can customize the data appearance on the **List**/**View**/**Print**/**Export** pages using formatting options on the **View as** settings dialog. You can define the field settings for all pages at once or separately for each page.

To control the field appearance on the page, click on the field and then on the **View As**/**Edit as** button in the *properties*.



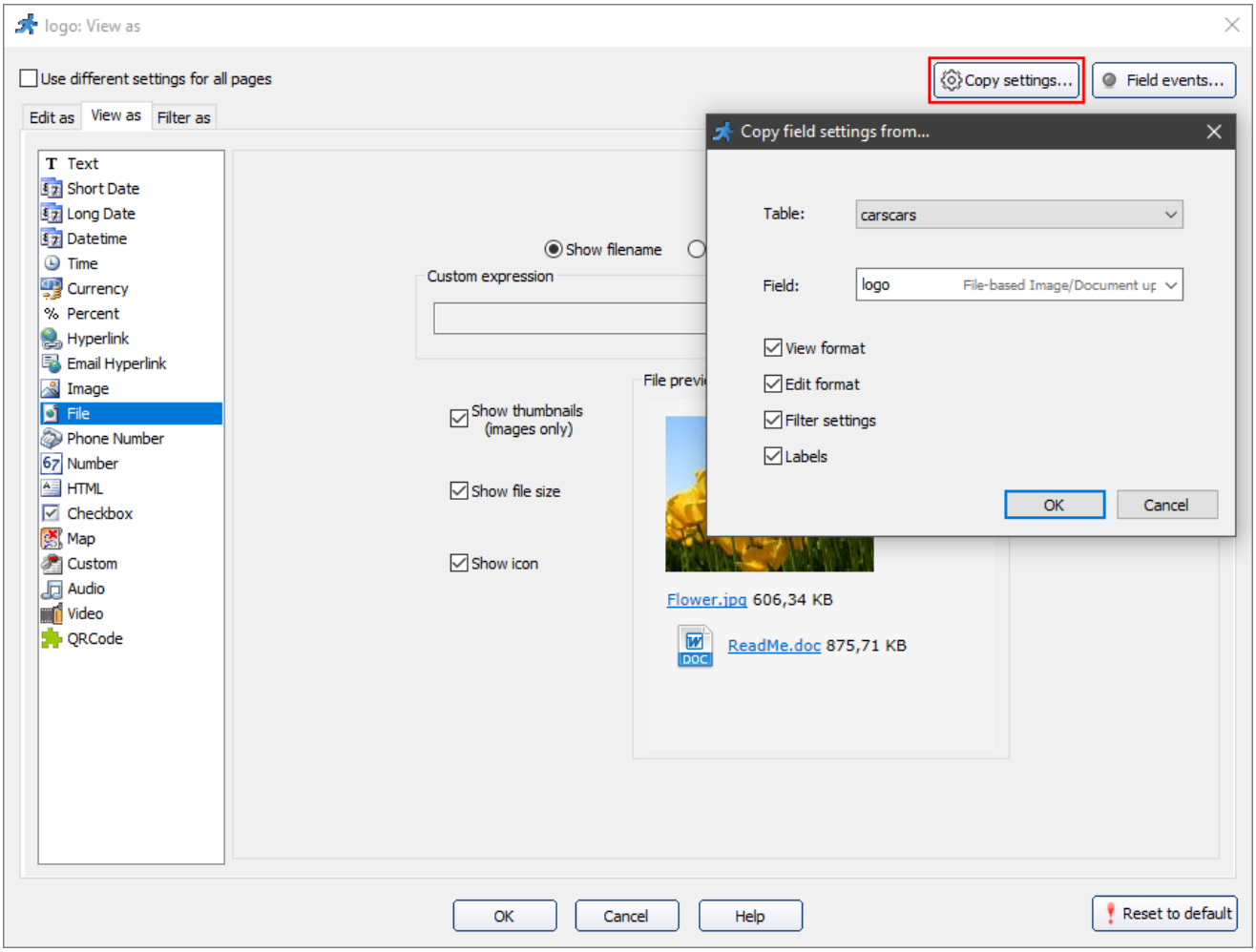
To set different field formats for different pages enable the **Use different settings for all pages** checkbox.

Depending on the selected format, you will see different dialogs.



Copy settings

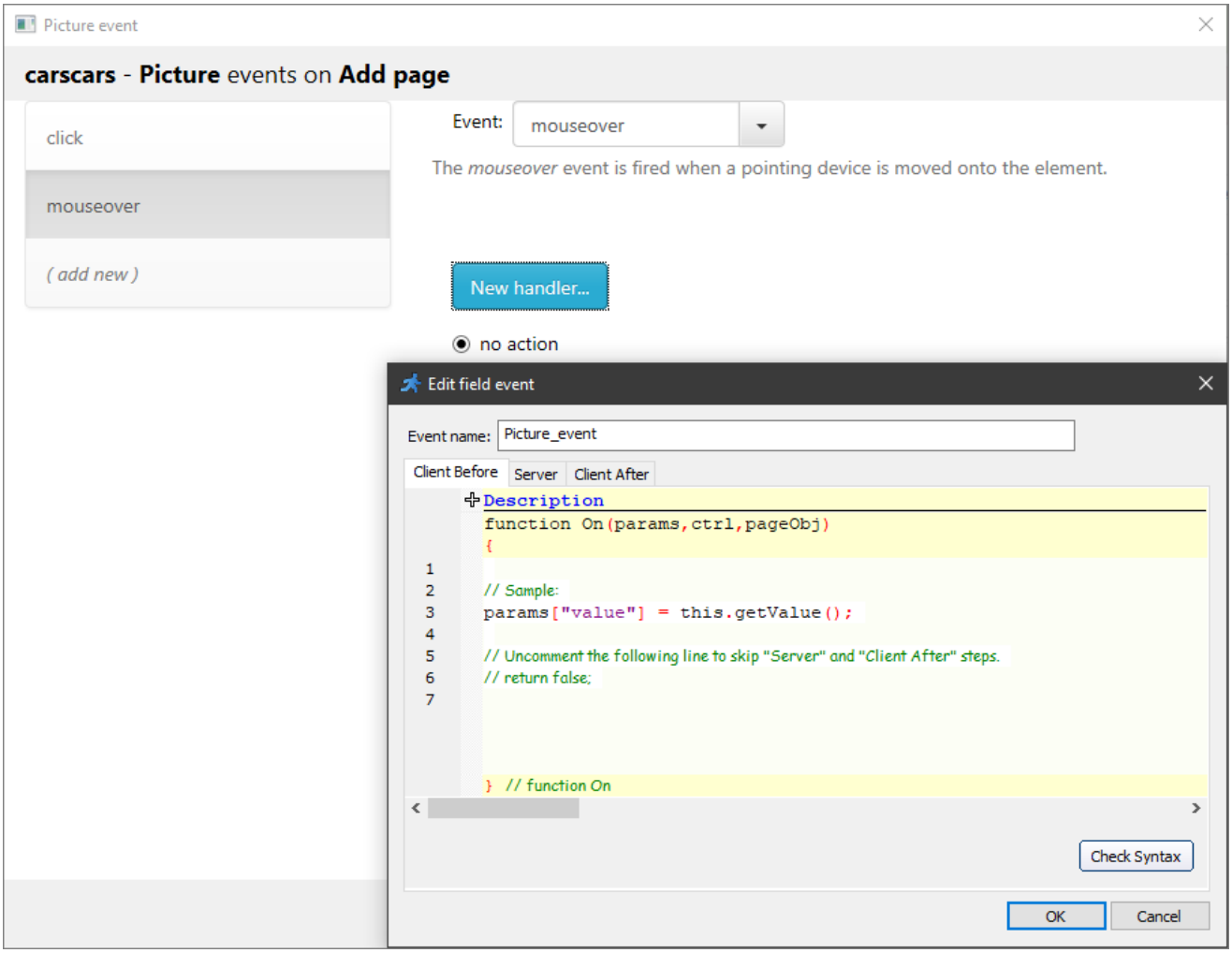
Copy settings option allows you to copy field settings from another field.



Field events

Field events option allows performing an action when the cursor enters, leaves or is over an edit field. Perform any sort of validation, make other fields hidden or required, etc. Field events are designed to work on **Add**, **Edit**, **View** and **Register** pages.

For example, the *mouseover* event occurs when the pointer is over the selected element.



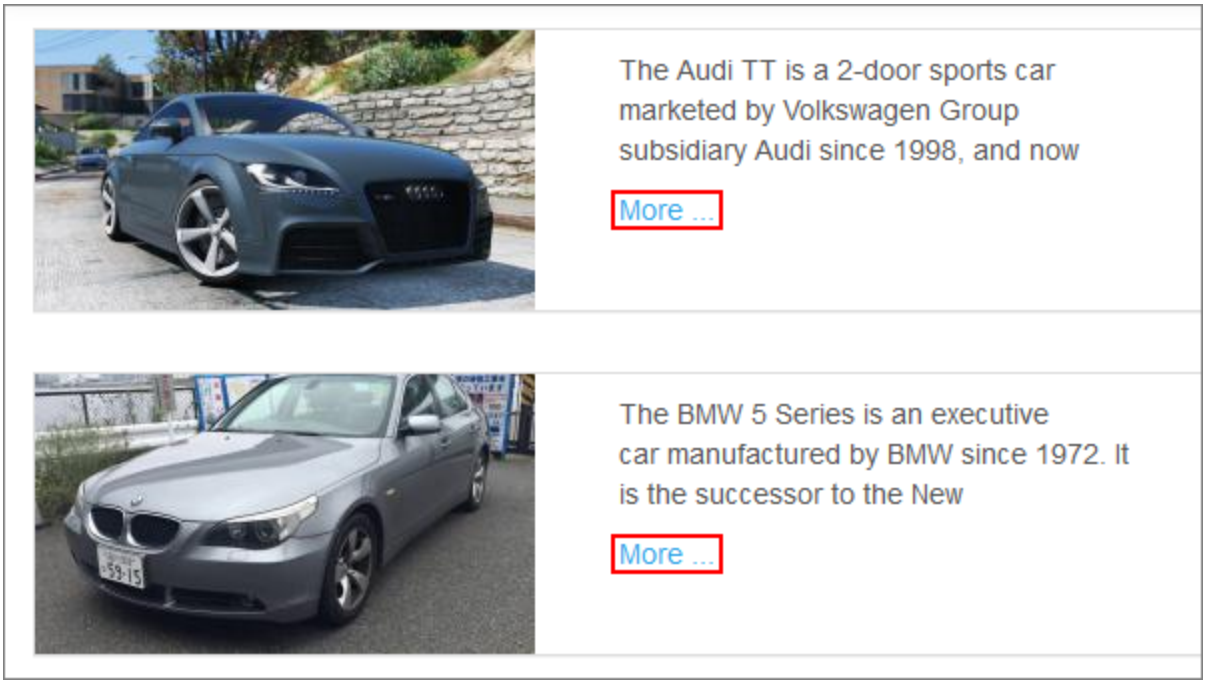
"View as" types

Text

The values are displayed as text.

Truncate large text fields

This option limits the number of characters that can be displayed in a single text field on a page. The "*More ...*" link appears if the number of characters is higher than the number you have entered into the *Characters to display* field.



Short Date

Dates will be displayed in a short format (02/17/2003).

Long Date

Dates will be displayed in a long format (17 February 2003).

Datetime

Datetime values will be displayed as date and time (02/17/2003 14:22:03).

Time

Datetime values will be displayed as time (14:22:03).

Currency

Numeric values like 14000 will be displayed as $14 000.00 (actual format depends on your system regional settings like currency symbol, decimal symbol, etc.).

Percent

Example: 0.38 will be displayed as 38%.

Hyperlink

Choose this format if you store hyperlinks in this database field. Those hyperlinks will be made clickable automatically.

Email Hyperlink

Choose this format if you store email addresses in this database field. It will be converted into *mailto HTML* code automatically.

File

Choose this format if you store files in this field. For more information, see "View as" settings: File.

Image

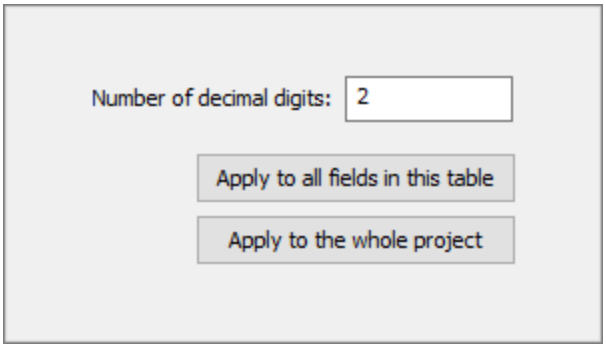
Choose this format if you store images in this field. For more information, see "View as" settings: Image.

Phone number

Formats number as a US phone number. Supports 7-digit or 10-digit numbers (123) 456-7890 or 123-4567.

Number

Choose this format if you like to format this field as a number. You can set the default number of digits after the comma for all numeric fields.



HTML

Use this view type when you store formatted HTML code in the database field and wish to display this HTML code on the **List** page.

Checkbox

Use this view format to present field value as a check box. Works best with the following data types:

- MS Access: *Yes/No* field
- SQL Server: TINYINT or BIT field
- MySQL: TINYINT
- Oracle: NUMBER(1)

Map

Allows adding maps to the web pages. For more information, see "View as" settings: Map.

Custom

Allows formatting field values by adding PHP code. For more information, see "View as" setting: Custom.

Audio

Choose this format if you store audio files in this field. For more information, see "View as" settings: Audio.

Video

Choose this format if you store video files in this field. For more information, see "View as" settings: Video.

QRCode

The QRCode control allows you to add QR Codes to your pages. For more information, see "View as" settings: QRCode.

See also:

- Field events

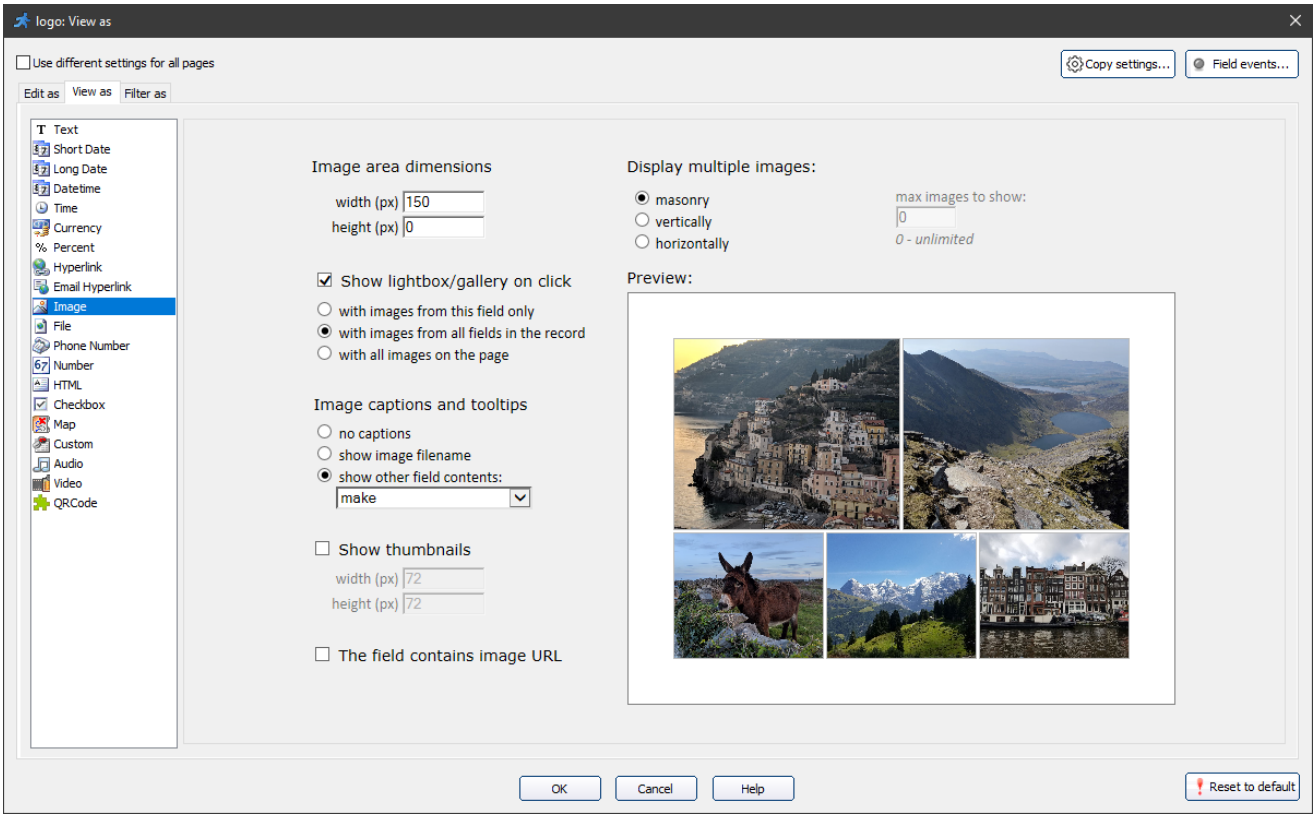
- How to access fields in the field events
- Connecting to the database
- About Date Control API
- "Edit as" settings
- About Page Designer
- About Editor

2.16.14.2 Image

PHPRunner creates the code that extracts images from the database on the fly. Choose this format if you store images in the selected field. Supported image formats are JPEG, GIF, and BMP.

A **View as**: **Image** type is available for Binary and Text fields.

View as Image for Text fields



Text field control options:

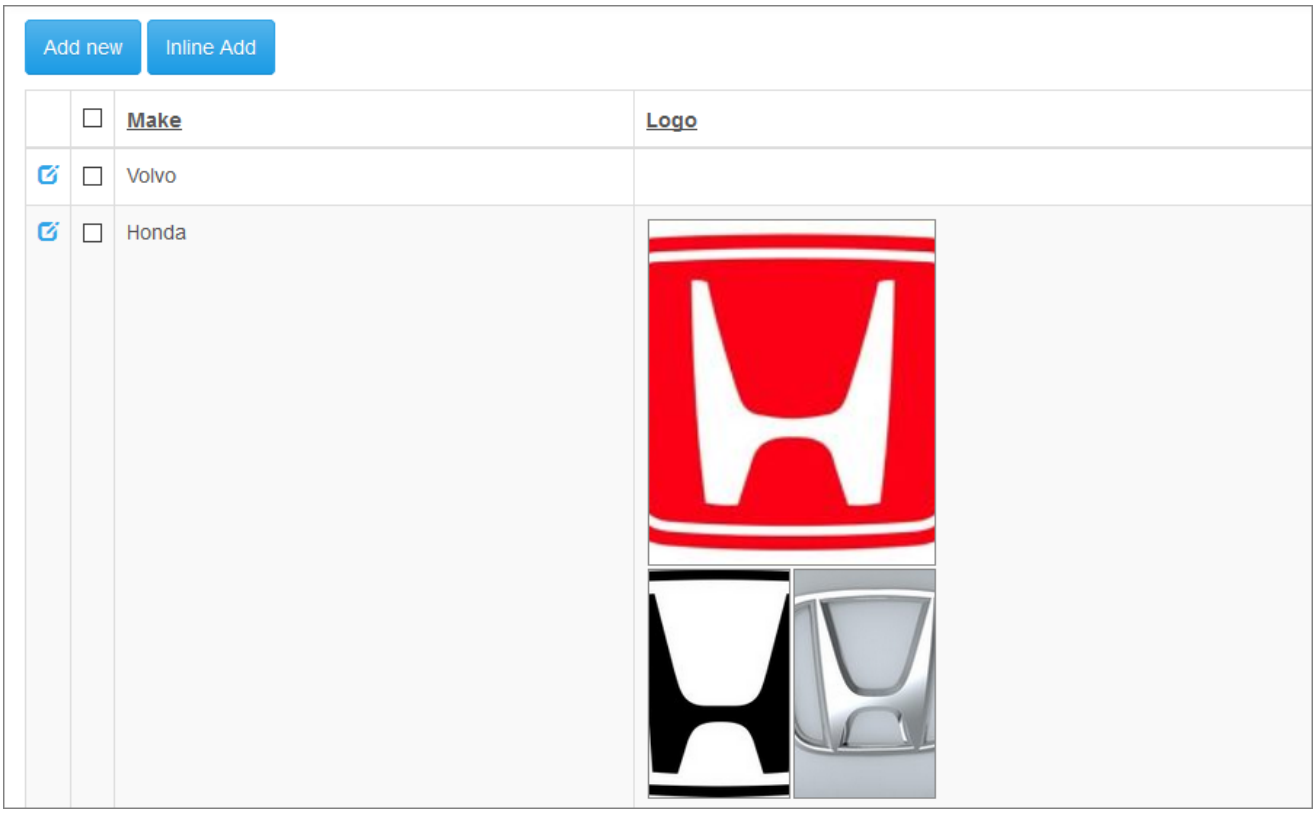
- **Display multiple images**. You can choose between masonry, vertical, and horizontal layout for the multiple images. The masonry layout works by placing elements in optimal position based on available vertical space, like a mason fitting stones in a wall.
- **Max images to show**. With a vertical and horizontal layout, you can select how many images to show in the grid. You can view additional images if you enable the gallery popup with the Show lightbox/gallery on click option.
- **Image size**/**Image area** dimensions. While vertical and horizontal layouts have an Image size option, that allows setting the width and height of the images, the masonry layout has the Image area dimensions option. This option sets the width and the height of the area the images are displayed in.

  > **Note**: to make sure, that all of the images fit correctly into the masonry layout, set the **Image area height** to 0. This way, the height is unlimited, and the maximum area width restricts the size of the images.

- **Show lightbox/gallery** on click. Enable this checkbox to show enlarged images in a lightbox/gallery popup when you click them. You can select, what images to show in the popup: from this field only, from all fields on the record, or with all images on the page.

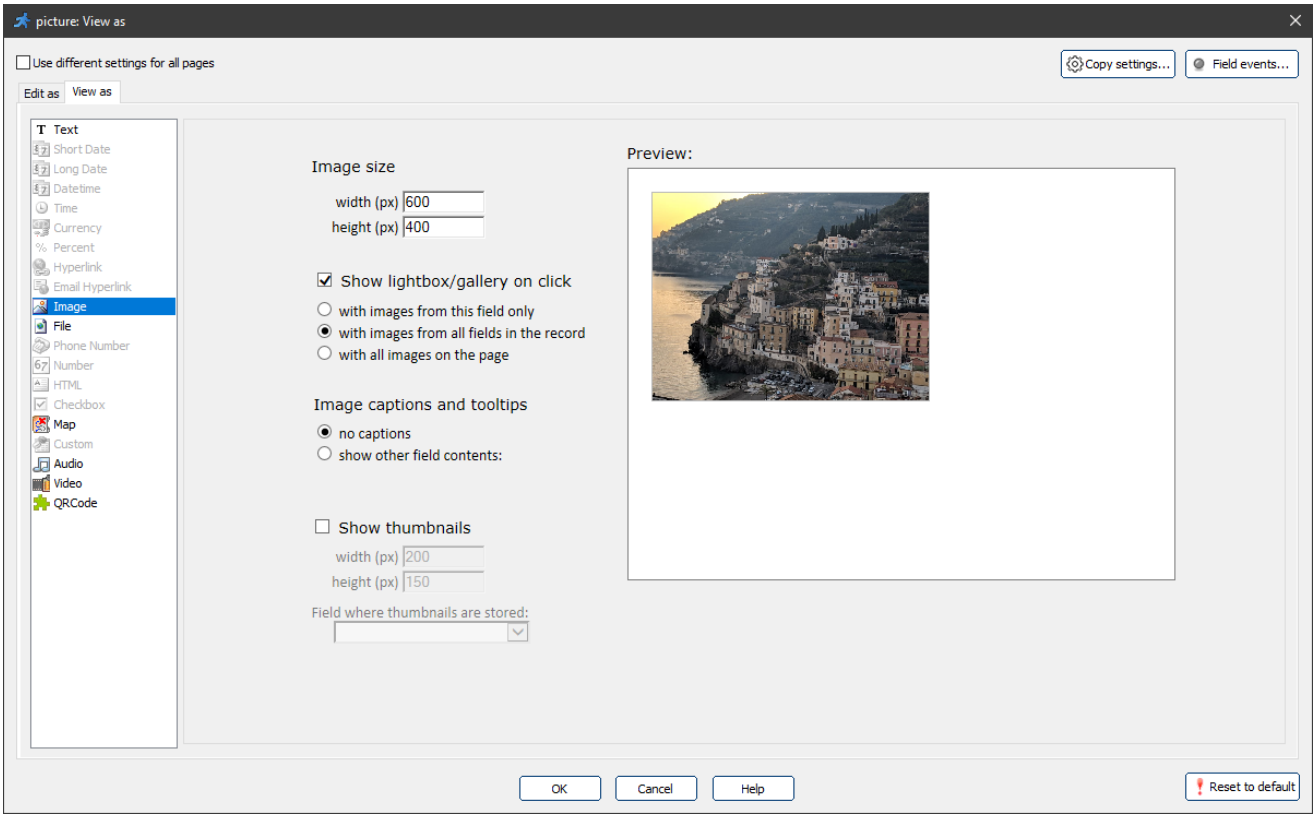
- **Image captions and tooltips**. This option allows showing the text from the selected field or the image filename in the gallery.
- **Show thumbnails**. Select this checkbox to display the thumbnails image. You can define the width and height of the thumbnails with the respective fields when vertical and horizontal layouts are selected.
- **This field contains image URL**. Use this checkbox if the **Edit as** type of this field is a Text field, and it contains the image URL.

Here is an example of a *masonry* layout with a width of 200 px:



Here is an example of a gallery displaying all the images on the page, with a *make* field as an image caption:

View as Image for Binary fields



The binary fields allow storing only one image, so they don't have the **Display multiple images** option.

To show the thumbnails for the images, you need to select an additional *binary* field (field of MEDIUMBLOB type) to store the thumbnails. This field is auxiliary, and we do not recommend to display it on the pages. You can display/hide fields on the Choose fields screen.

See also:

- "Edit as" settings: File/Image
- Choose fields screen
- "View as" settings
- About Page Designer
- About Editor

2.16.14.3 File

The file format is available for *Binary* and *Text* fields. Depending on the field type you will be able to choose either the folder where files are located (for the *text* field) or the field name that stores *filenames* (for the *binary* field).

Binary field

In this case, the file is stored in the database, and you need to choose a field that stores the name of the database file.

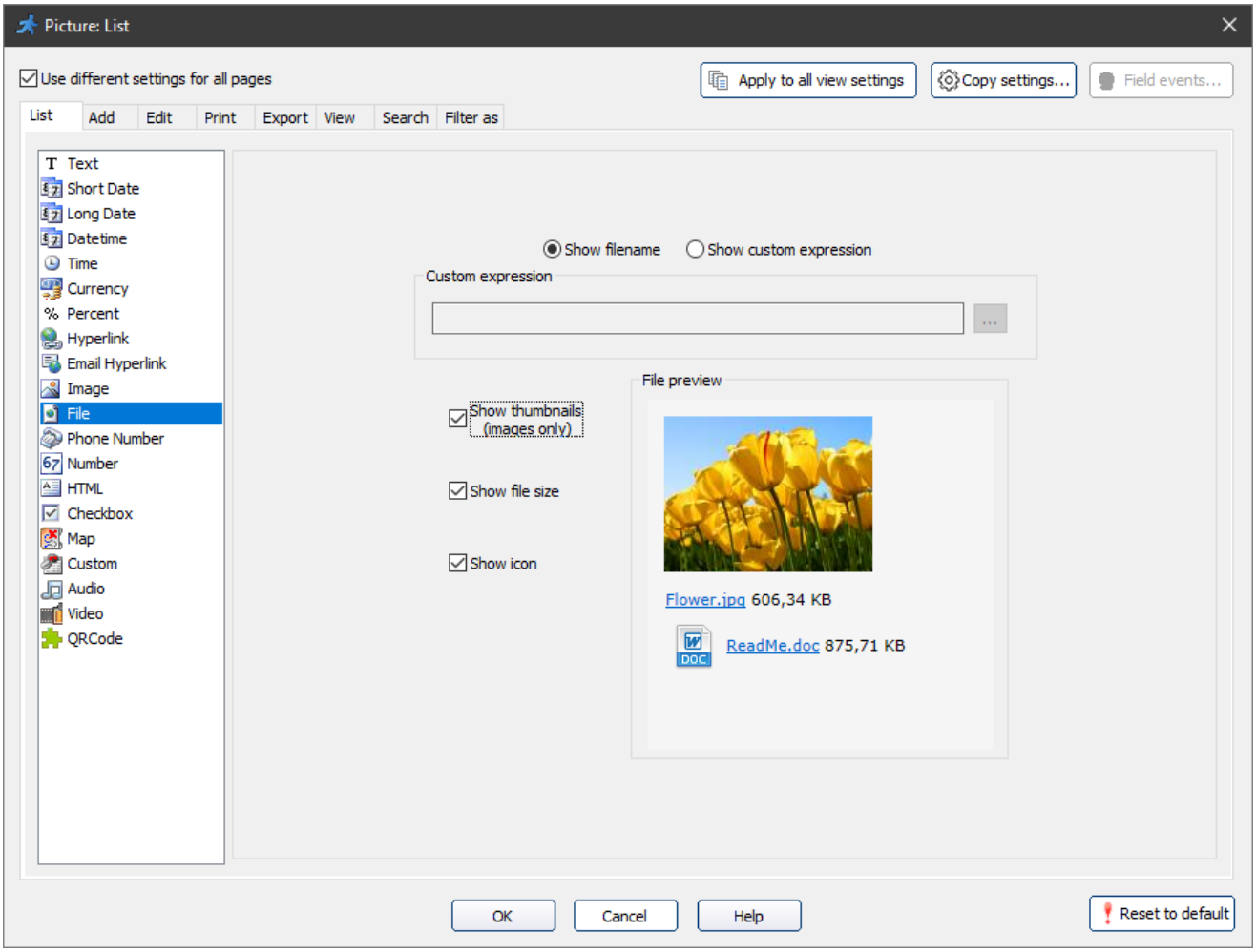
This *filename* is required to set correct file type when you retrieve uploaded file from the database.

If you don't choose the *filename* field or leave it empty, you will be presented with **Open with** dialog every time you download this file from the database.

Text field

You can select to display *filename* or a custom expression instead of the *filename*.

If you want to display the thumbnail image, file size or icon, select the corresponding checkboxes.



See also:

- "Edit as" settings: File/Image
- "View as" settings
- About Page Designer
- About Editor

2.16.14.4 Map

Quick jump

View as Map

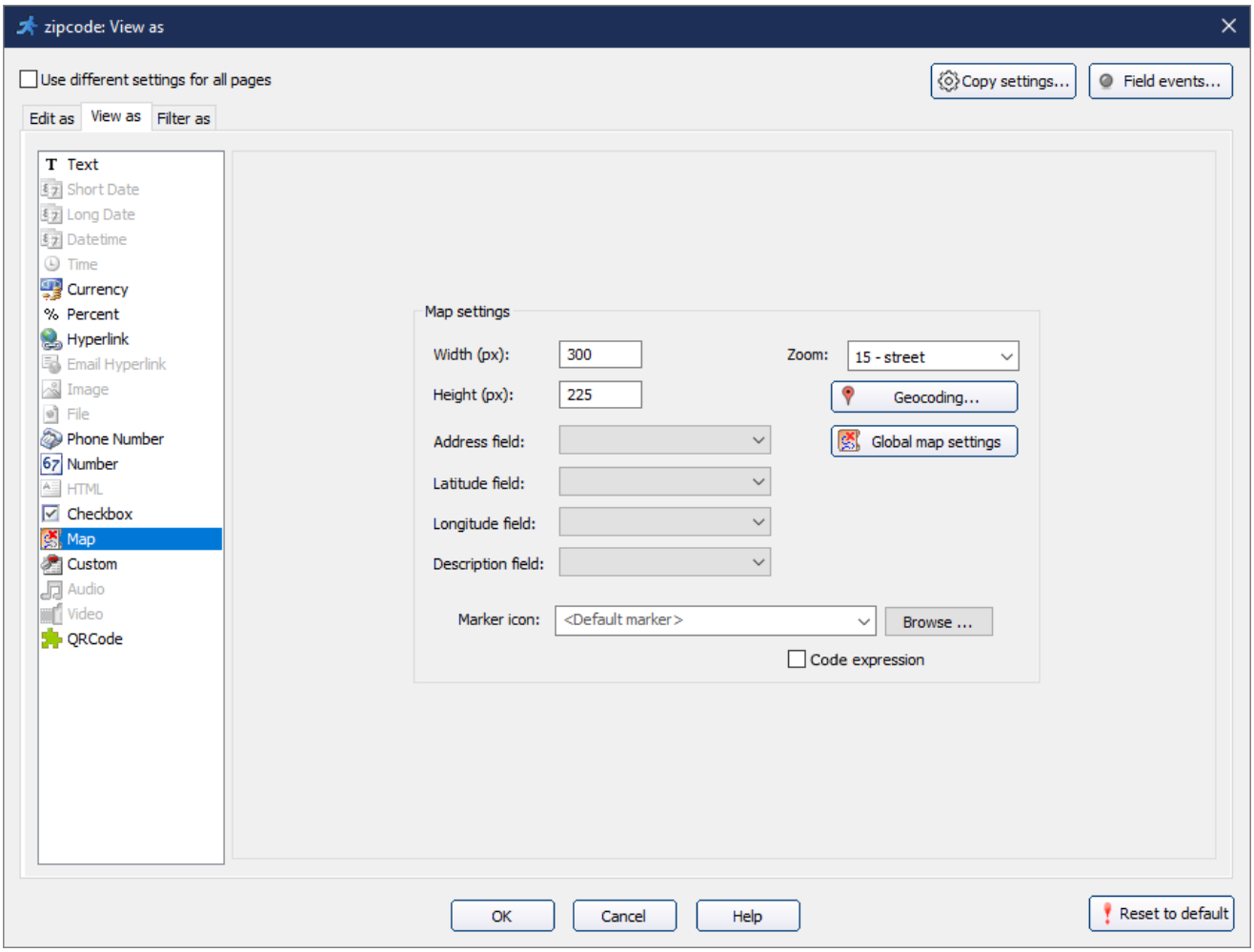
Global map settings

Geocoding

Maps API and geocoding

View as Map

This setting allows you to display the location data as a separate map for each table record. You can set the width and height of the map in pixels.



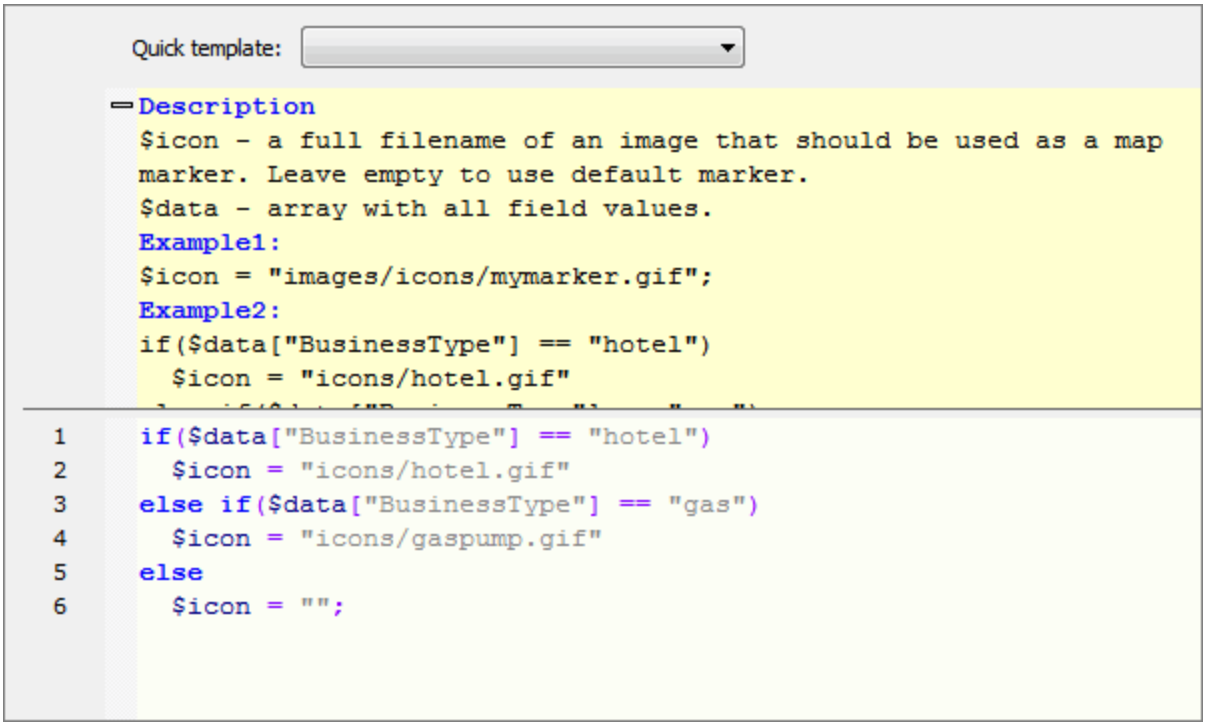
Use the address or latitude/longitude coordinates to set the location. It is highly recommended to use the *Latitude* and *Longitude* fields to make maps function properly.

The number given in the *Zoom* field will define the level of zoom on the map. Use the description field to set the descriptions for the markers.

> **Note**: The address field accepts any widely-used address format (e.g. "1600 Amphitheatre Parkway, Mountain View, CA").

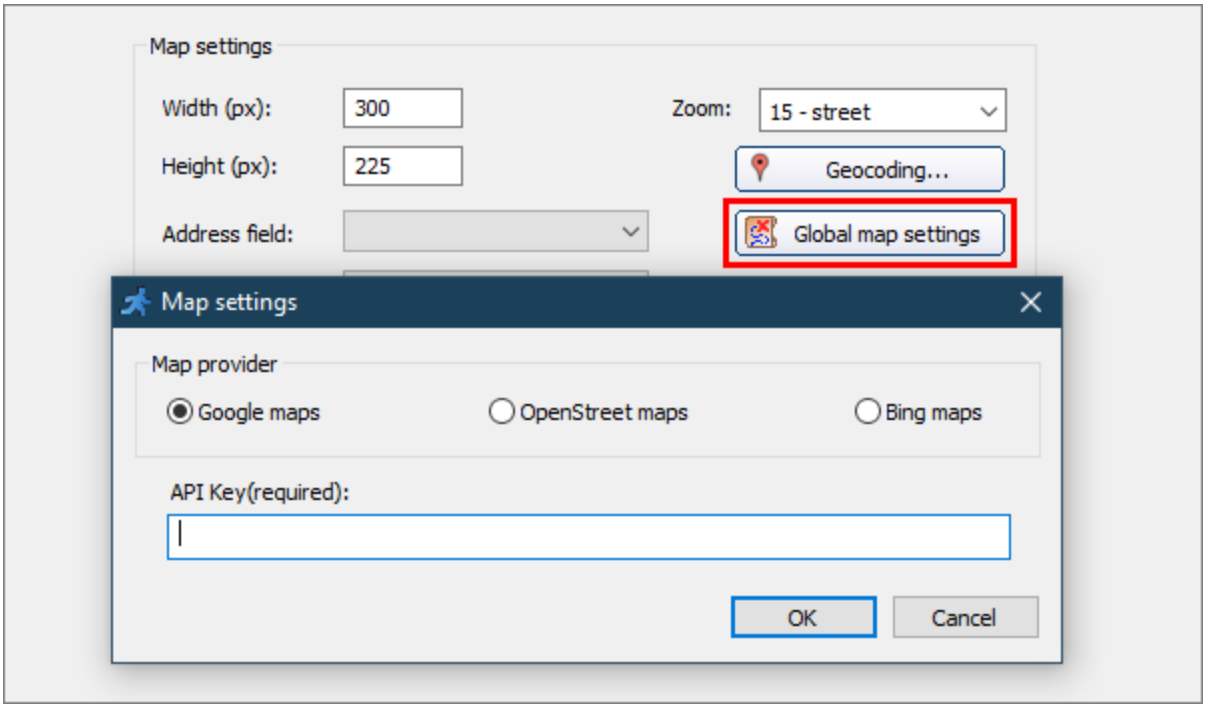
You may set a custom icon for map marker pins. Click **Browse** next to the *Marker* icon and select the icon image file.

You may also set different icons for different map objects by using the PHP expression option:



Global map settings

You may access **Global map settings** to select a map provider. Enter *API Key* (if required).

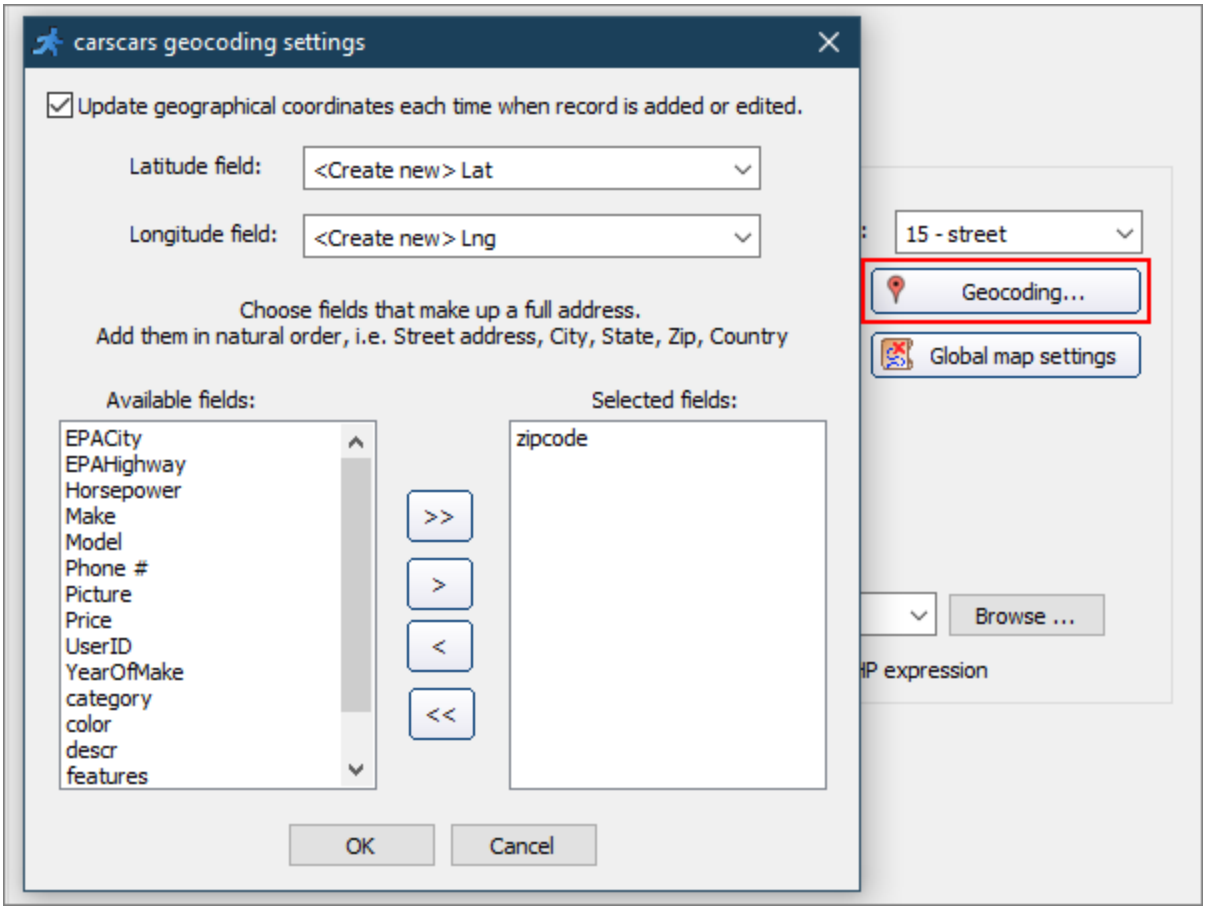


Note: you can get API keys for development purposes for free at https://developers.google.com (for Google maps) or at bingmapsportal.com (for Bing maps).

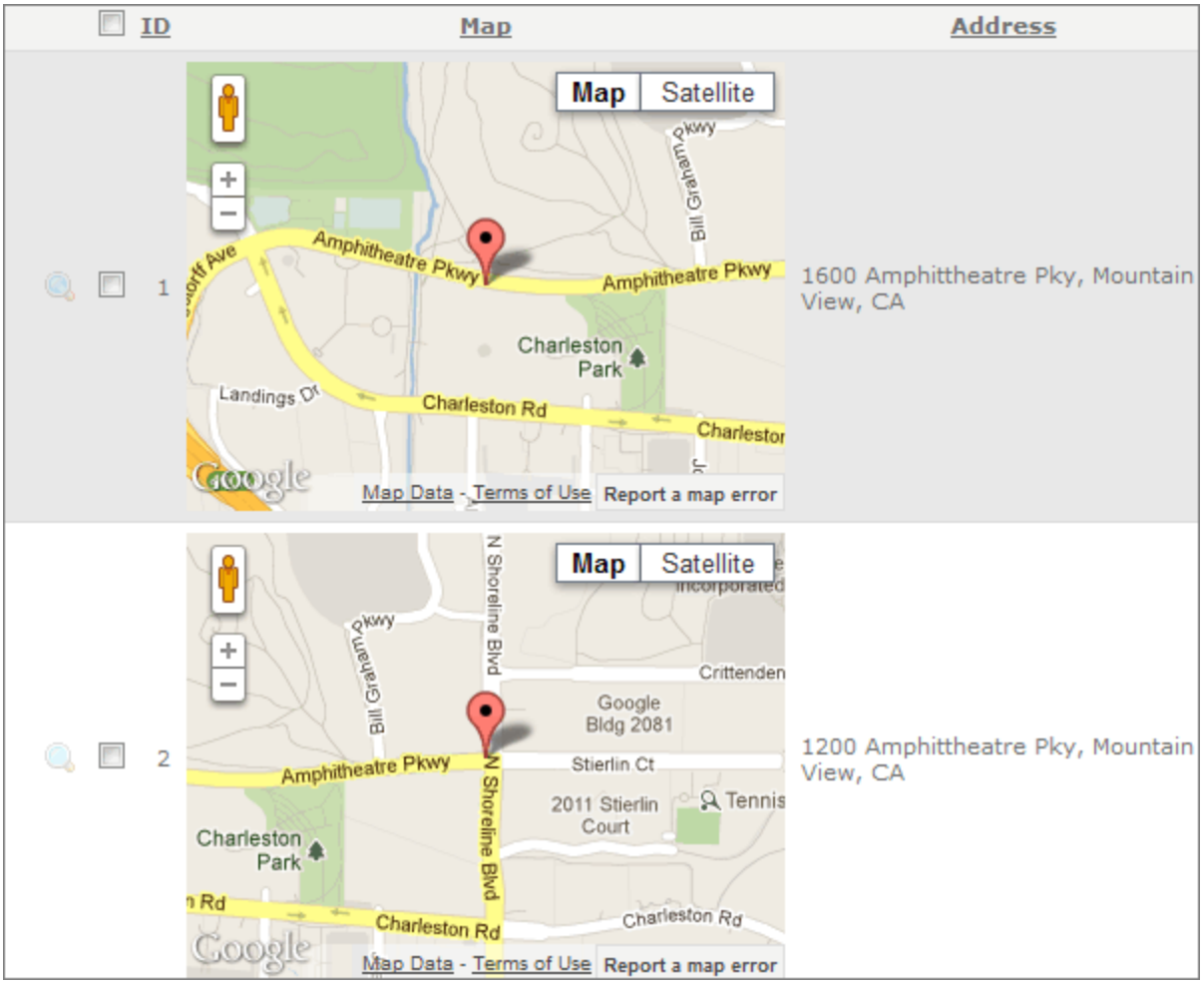
Geocoding

Use the **Geocoding** option to update the latitude/longitude information each time a record is added or updated.

Select existing fields for storing latitude and longitude data or create new ones. Then select address field(s) to be used during geocoding.



An example of how maps are displayed in a browser. Note that double left click will zoom in and double right click will zoom out.



Maps API and geocoding

Maps API providers have limits on the number of geocoding requests you can send per day, and also a limited requests rate.

For example, if you use Google Maps API and have a page with 20 records with a map for each one, only about 10 maps will be displayed properly. To overcome those limits you need to open a Premier account with Google that costs $10,000 per year.

The solution is to use latitude/longitude pairs instead of addresses for mapping purposes.

Enable the Geocoding option on the Choose pages screen to update latitude/longitude information each time when a record is added or updated.

The main question is how to convert the existing addresses to latitude/longitude pairs? The instruction is available at Insert Map - Geocoding.

See also:

- Insert map
- Miscellanneous settings: Map settings
- "View as" settings
- About Page Designer
- About Editor

2.16.14.5 Custom

You can format field values by adding PHP code.

```
$value - a value to be displayed on the page.
Example:
$value = strtoupper($value);

$data - array with all field values.
Example:
$value = $data["FirstName"].$data["LastName"];
where FirstName and LastName are actual field names.
1  $value = strtoupper($value);
```

The input value is stored in the variable value. The output value is assigned to the same variable value. You can access other fields of the same data record as *data["FieldName"]*.

If you chose Lookup wizard in the "Edit as" settings, use value to access the *Display* field value and *data["LinkFieldName"]* to access the *Link* field value.

Examples

1. Convert a string into the upper case:

```
$value = strtoupper($value);
```

2. Format a 10-digit phone number into the following format (xxx) xxx-xxx:

```
if (strlen($value)==10)
{
  $value="(" . substr($value,0,3) . ") " . substr($value,3,3) . "-" .
substr($value,6);
}
```

3. Display the value of the field *FirstName* as *<FirstName> <LastName>* (if the *LastName* field defined):

```
if ($data["LastName"])
  $value = $value." ".$data["LastName"];
```

4. Display a number in black if the number is positive, and in red if the number is negative:

```
if ($value>0)
  $color="black";
else
  $color="red";
$value="<font color='$color'>$value</font>";
```

5. Display a field containing email address as an email hyperlink (*mailto* function used). The value of subject in *mailto* function is set to a value of another field:

```
$value = "<a href='mailto:".$value."?subject=".$data["SubjectField"]."'>Send
email</a>";
```

6. Display a field value in one line (without line breaks):

```
$value = str_replace(" ", " ",$value);
```

7. Display all images uploaded via *multiupload*:

```
$filesArray = my_json_decode($value);
foreach ($filesArray as $imageFile) {
   $imageValue .= "<img alt=\"".htmlspecialchars($imageFile["usrName"])."\"
src=\"".htmlspecialchars($imageFile["name"])."\">";
}
$value = $imageValue;
```

8. Display a phone number as a click-to-call link for mobile browsers:

```
$value = '<a href="tel:'.$value.'">Call us!</a>';
```

9. Display a hyperlink to another page passing the field value as a parameter:

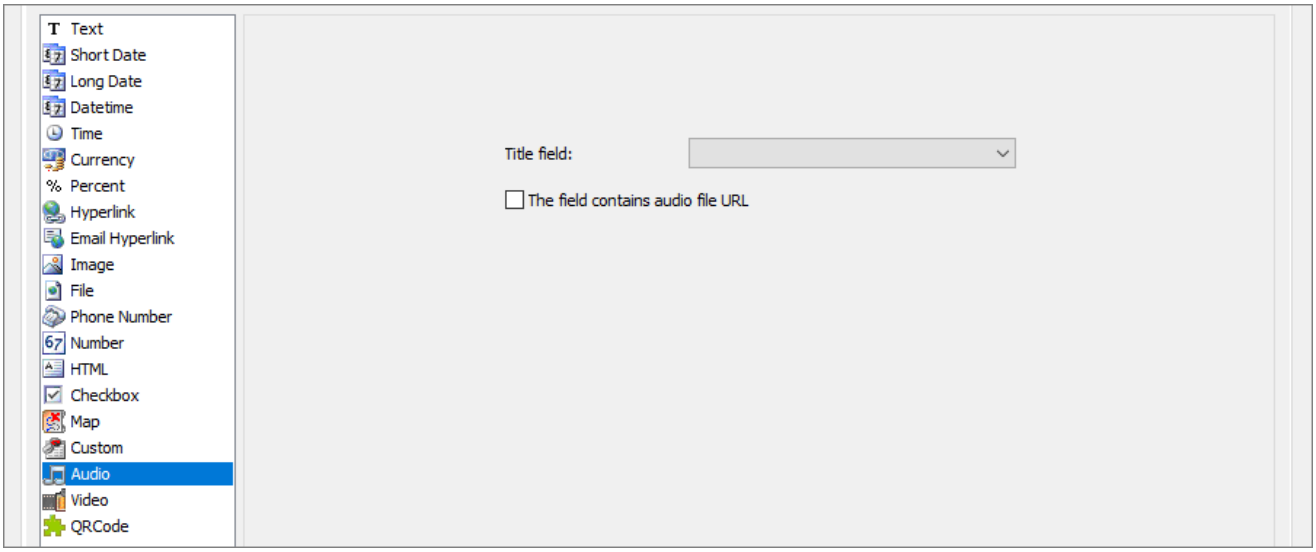
```
$value = "<a href='http://www.mysite.com/reports/sales_rpt.php
order_id=invoice_id".$data["invoice_id"]."'>Link</a>";
```

See also:

- Conditional formatting
- How to create a custom Edit control plugin
- "View as" settings
- About Page Designer
- About Editor

2.16.14.6 Audio

Select the field that contains the audio titles. Audio titles are displayed instead of the audio file path/URL.



If the selected field contains an audio file URL, select the corresponding checkbox.

The "Edit as" type can be either Text field (enter audio file name there) or File/Image (upload files).

An example of how the audio files appear in the browser:

See also:

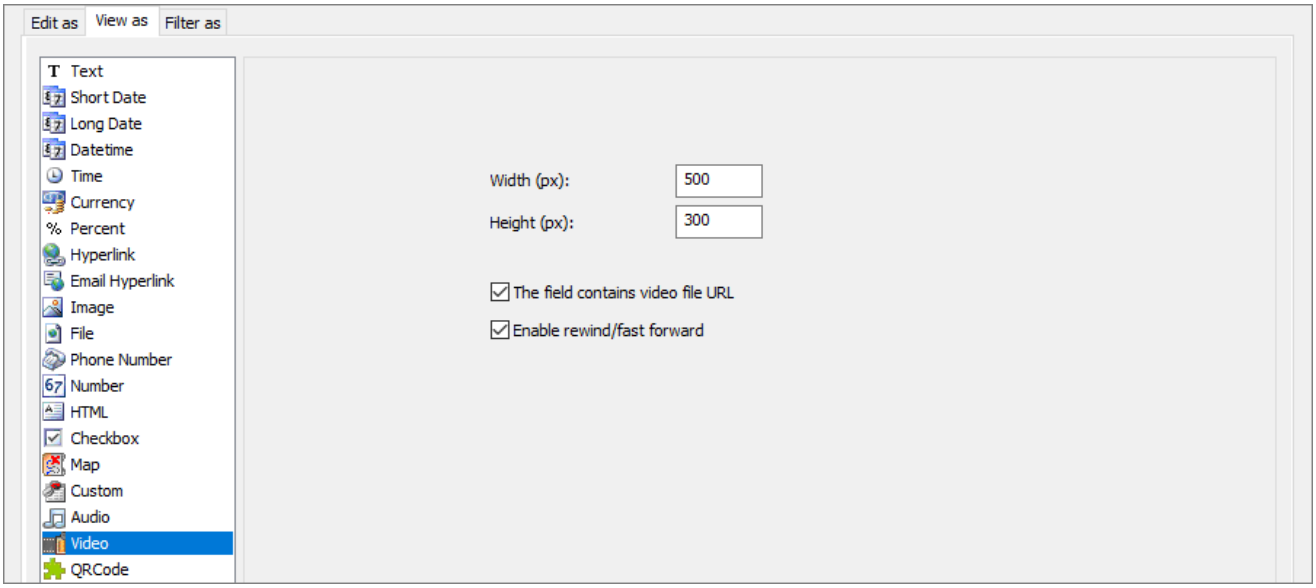
- "Edit as" settings: Text field
- "Edit as" settings: File/Image

- "View as" settings
- About Page Designer
- About Editor

2.16.14.7 Video

You can define the *width* and *height* of the video in the **View as Video** settings.

The "Edit as" type for the **View as Video** can be either a Text field (enter video file name or URL there) or a File/Image (upload files).

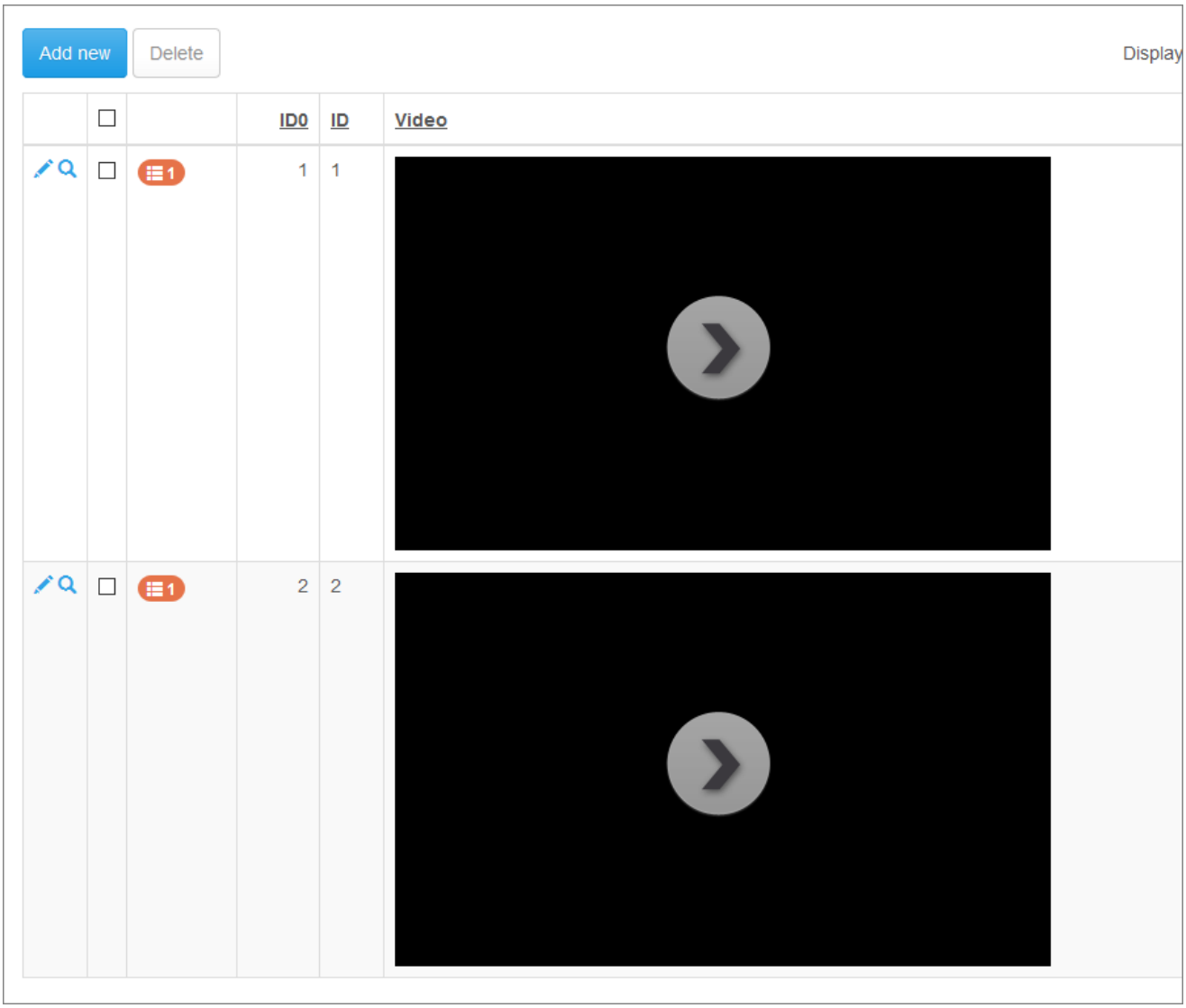


Select **The field contains Video file URL** checkbox if the selected field stores the URL of the video file, for example, *http://yourwebsite.com/folder/video.mp4*.

> **Note**: do not use this option if you use **File/Image** as the "Edit as" type for this field.

Select the **Enable rewind/fast forward** checkbox to enable additional controls in the resulting page.

An example of how video files appear in the browser:



See also:

- "Edit as" settings: Text field
- "Edit as" settings: File/Image
- "View as" settings
- About Page Designer
- About Editor

2.16.14.8 PDF

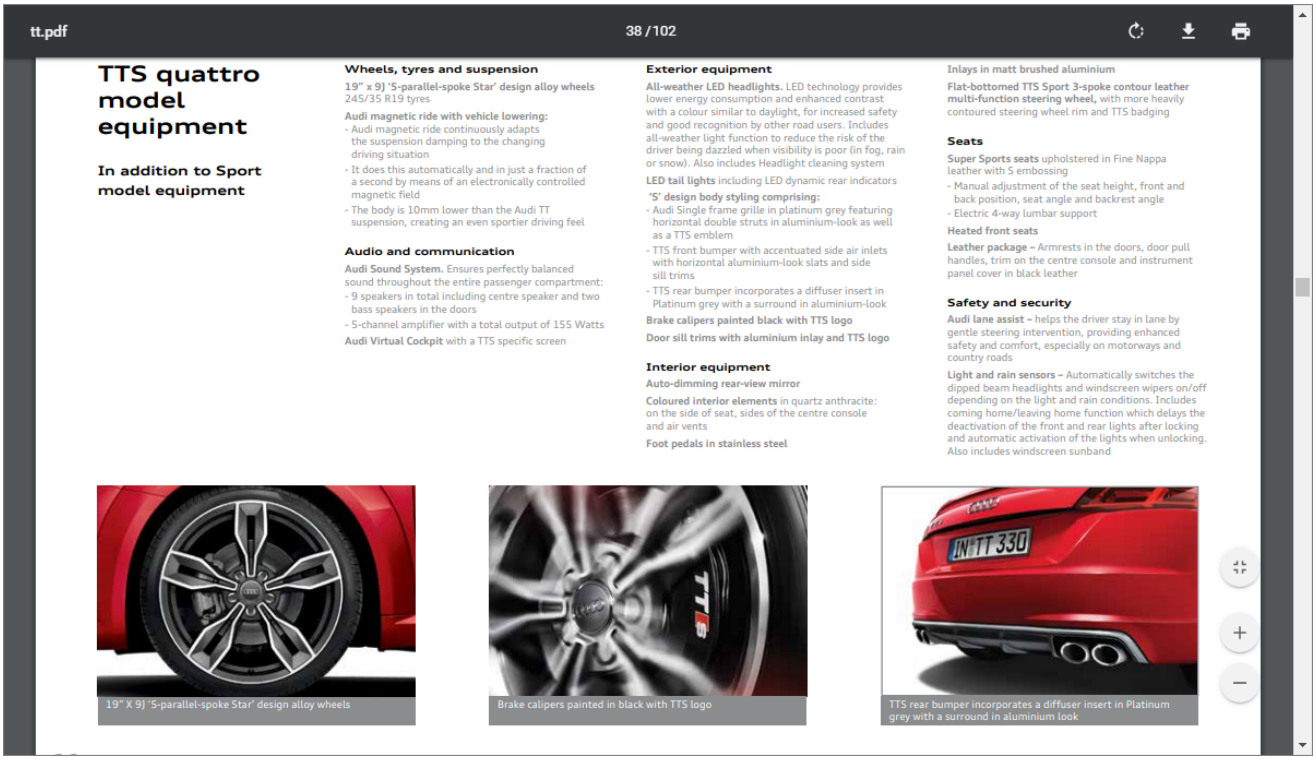
The **View as PDF option** works similarly to View as File, but it allows you to view uploaded PDF files directly in the browser.

Click the *uploaded PDF document* on the **List**/**View** page to open it.

> **Note**: this option only works with PDF files. Other types of files would still display the *Download file* dialog when clicked.

Click the **Add initialization script** button to customize the PDF control.

Here is how the PDF document looks like in the browser:



See also:

- About PDF API

- Printer-friendly/PDF view settings
- PDF View settings
- "View as" settings
- About Page Designer
- About Editor

2.16.14.9 QRCode

The QRCode control allows you to generate QR Codes and display them on your web pages.

QR Code (it stands for "*Quick Response*") is a cell phone readable bar code that can store website URLs, plain text, phone numbers, email addresses, and any other alphanumeric data.

An example of how it may look in the browser:

Click the **Add initialization script** button to customize the QRCode control.

See also:

- "View as" settings
- About Page Designer
- About Editor

2.16.14.1 Conditional formatting

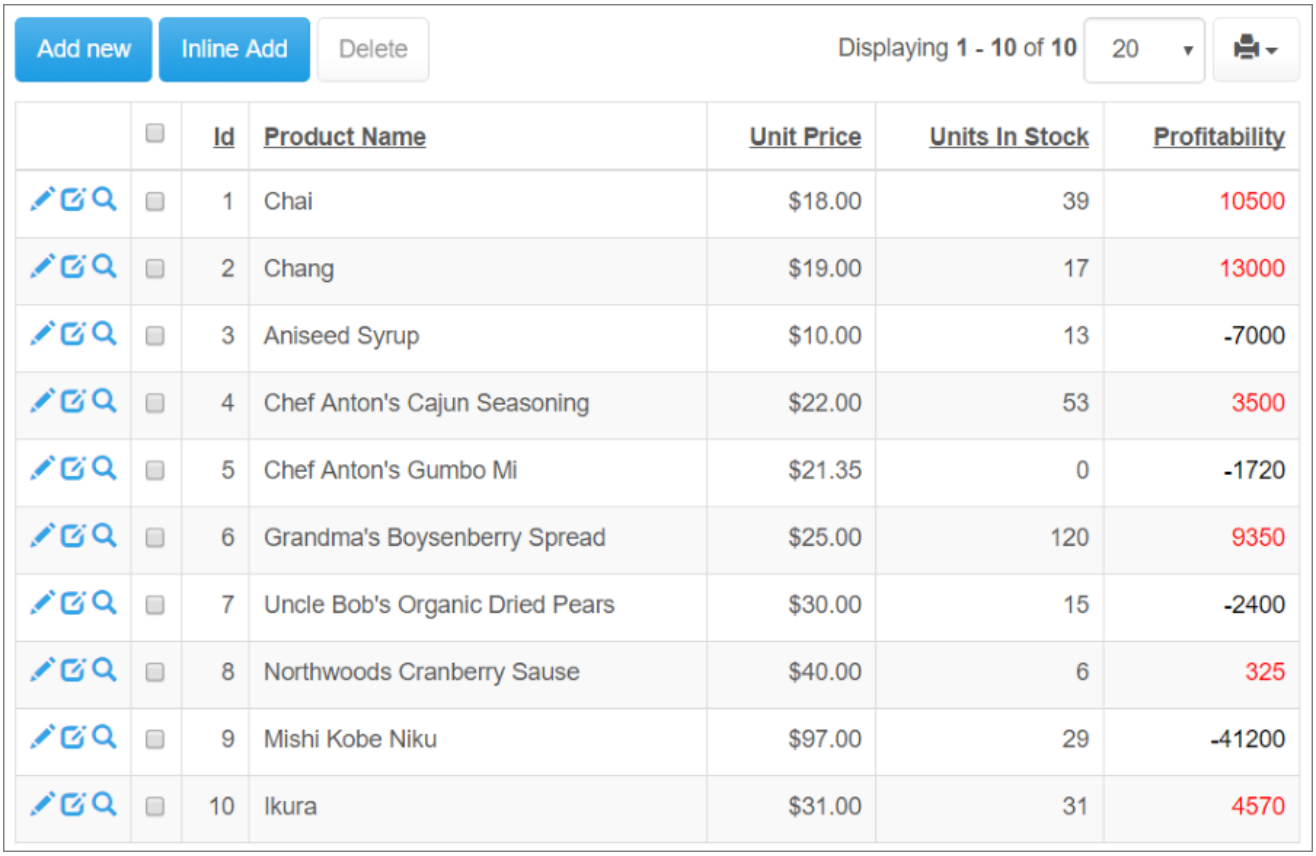
Quick jump

Using conditional formatting in PHPRunner projects

Displaying font in different colors

This example illustrates the profitability of the product line, where we have a list of products with their respective monthly profit figures.

The positive numbers are displayed in red and the negative - in black. Changing the font color of the field with negative values helps spot the losses immediately.



Id	Product Name	Unit Price	Units In Stock	Profitability
1	Chai	$18.00	39	10500
2	Chang	$19.00	17	13000
3	Aniseed Syrup	$10.00	13	-7000
4	Chef Anton's Cajun Seasoning	$22.00	53	3500
5	Chef Anton's Gumbo Mi	$21.35	0	-1720
6	Grandma's Boysenberry Spread	$25.00	120	9350
7	Uncle Bob's Organic Dried Pears	$30.00	15	-2400
8	Northwoods Cranberry Sause	$40.00	6	325
9	Mishi Kobe Niku	$97.00	29	-41200
10	Ikura	$31.00	31	4570

To set the formatting, proceed to the **Page Designer**, click on the field you wish to format (the *profitability* field in our case), then click **View as/Edit as**. Select **View as Custom** option. Add the code in the custom code editor:

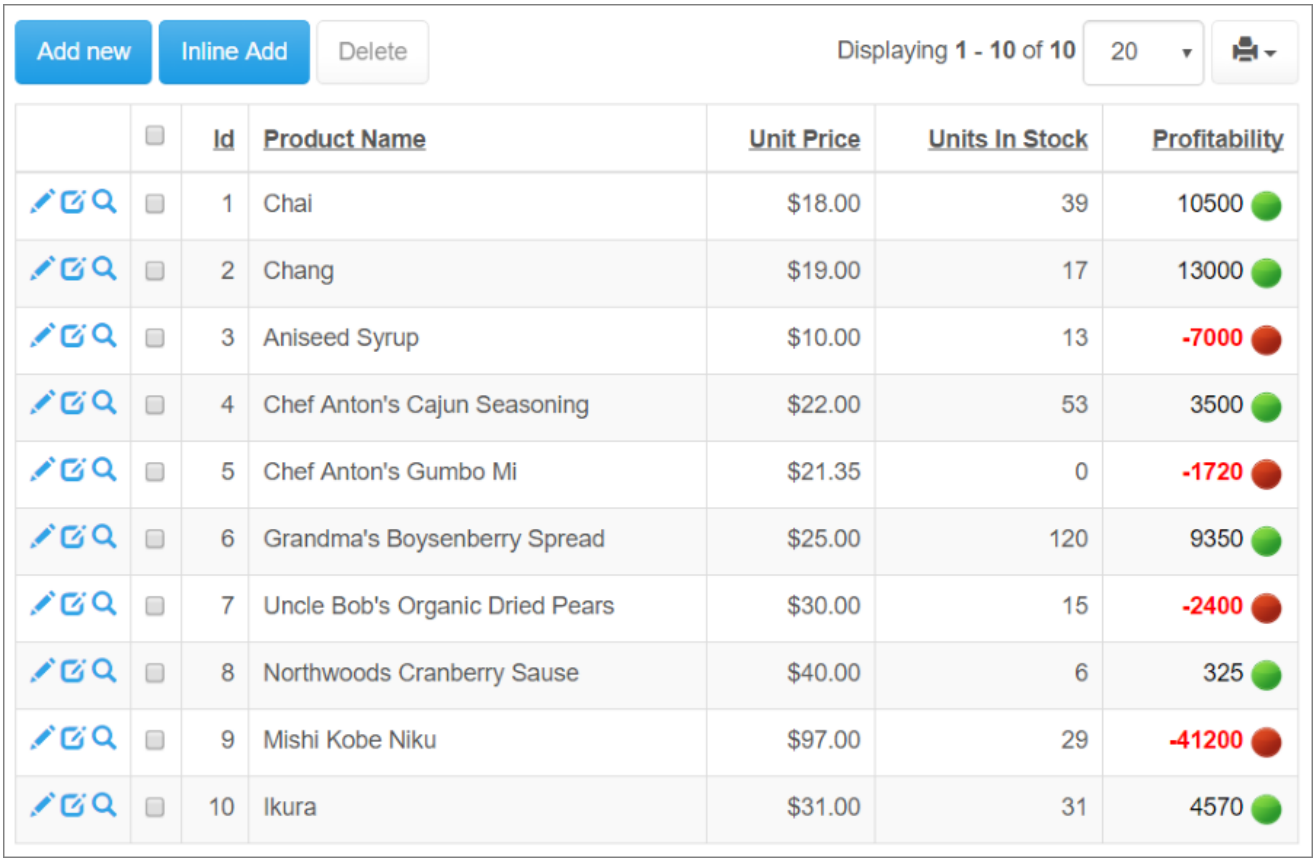
```
if ($value > 0) {
     $color="red";
} else {
   $color="black";
}
$value="<span style='color: " . $color . "'>" .$value . "</span>";
```

Conditional formatting with images

You can represent positive and negative results with different images. If you want to highlight the performance of the products with images of red and green circles, place the actual images into the project folder first.

Then add two more lines to your custom code to show these images:

```
if ($value > 0) {
  $value =$value. ' <img src="green.png" alt="" />';
   $color="black";
} else {
   $value ='<strong>'.$value. '</strong> <img src="red.png" alt='' />';
   $color="red";
}
$value="<span style='color: " . $color . "'>" . $value . "</span>";
```



	Id	Product Name	Unit Price	Units In Stock	Profitability
	1	Chai	$18.00	39	10500
	2	Chang	$19.00	17	13000
	3	Aniseed Syrup	$10.00	13	-7000
	4	Chef Anton's Cajun Seasoning	$22.00	53	3500
	5	Chef Anton's Gumbo Mi	$21.35	0	-1720
	6	Grandma's Boysenberry Spread	$25.00	120	9350
	7	Uncle Bob's Organic Dried Pears	$30.00	15	-2400
	8	Northwoods Cranberry Sause	$40.00	6	325
	9	Mishi Kobe Niku	$97.00	29	-41200
	10	Ikura	$31.00	31	4570

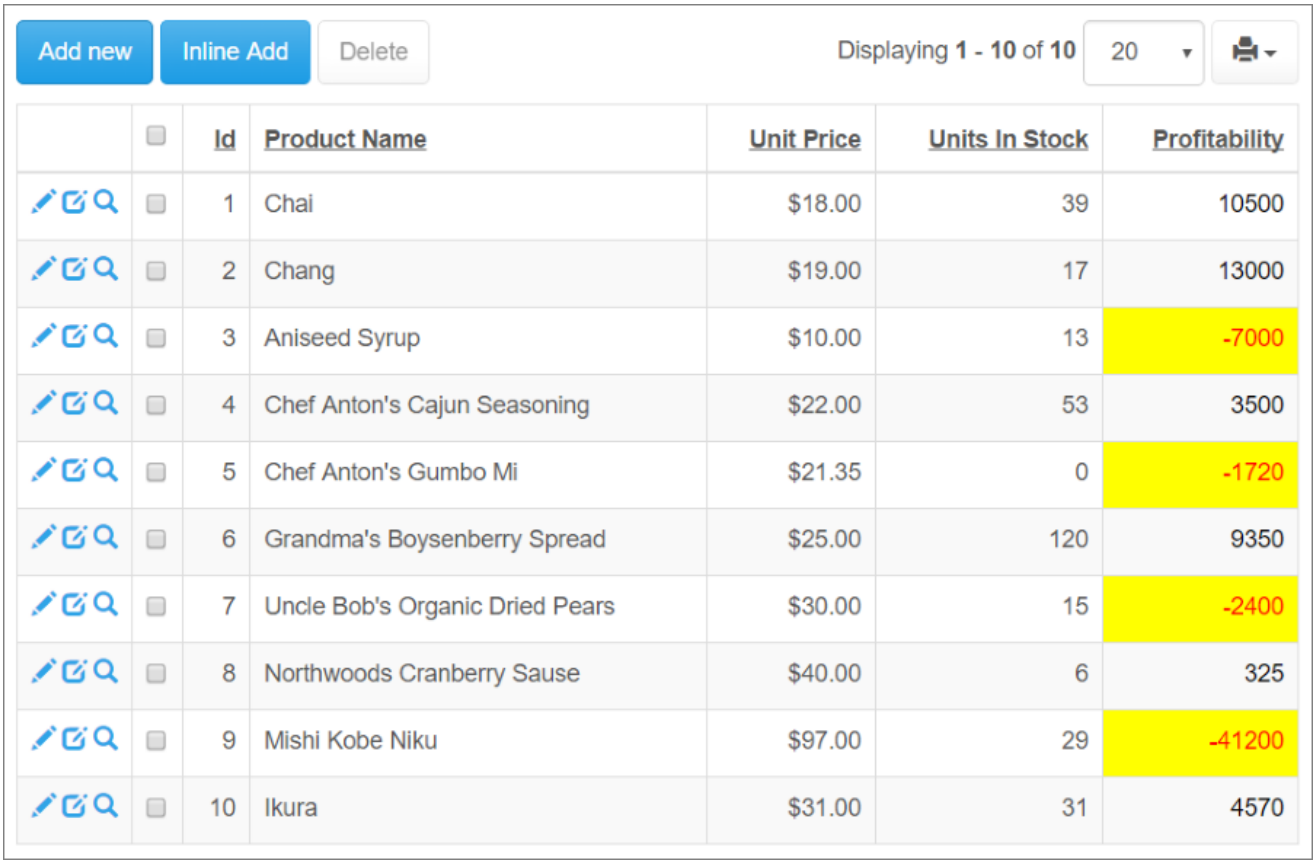
Changing the background of a cell

You may change the background of the cell to yellow, based on its value. As you are working with the table structure here, you will have to use the events.

Proceed to the Events screen in the PHPRunner and select the After record processed event for the **List** page of the *Products* table.

The code is similar to the previous examples:

```
if ($data["Profitability"]<0)
   $record["Profitability_css"].='background:yellow';
```



Id	Product Name	Unit Price	Units In Stock	Profitability
1	Chai	$18.00	39	10500
2	Chang	$19.00	17	13000
3	Aniseed Syrup	$10.00	13	-7000
4	Chef Anton's Cajun Seasoning	$22.00	53	3500
5	Chef Anton's Gumbo Mi	$21.35	0	-1720
6	Grandma's Boysenberry Spread	$25.00	120	9350
7	Uncle Bob's Organic Dried Pears	$30.00	15	-2400
8	Northwoods Cranberry Sause	$40.00	6	325
9	Mishi Kobe Niku	$97.00	29	-41200
10	Ikura	$31.00	31	4570

Highlighting an entire row of a table

You can use the same condition check in the After record processed event - change the code to set the entire row yellow. Your code may look like this:

```
if ($data["Profitability"] < 0)
$record["css"]="background:yellow;";
```

And if you want to change the background color of all rows regardless of the condition, you can simply delete the condition in this event:

```
$record["css"]="background:yellow;";
```

See also:

- A video tutorial on conditional formatting
- How to change the row background color
- How to change the cell background color
- Customizing CSS
- "View as" settings
- About Page Designer
- About Editor

2.16.15 "Edit as" settings

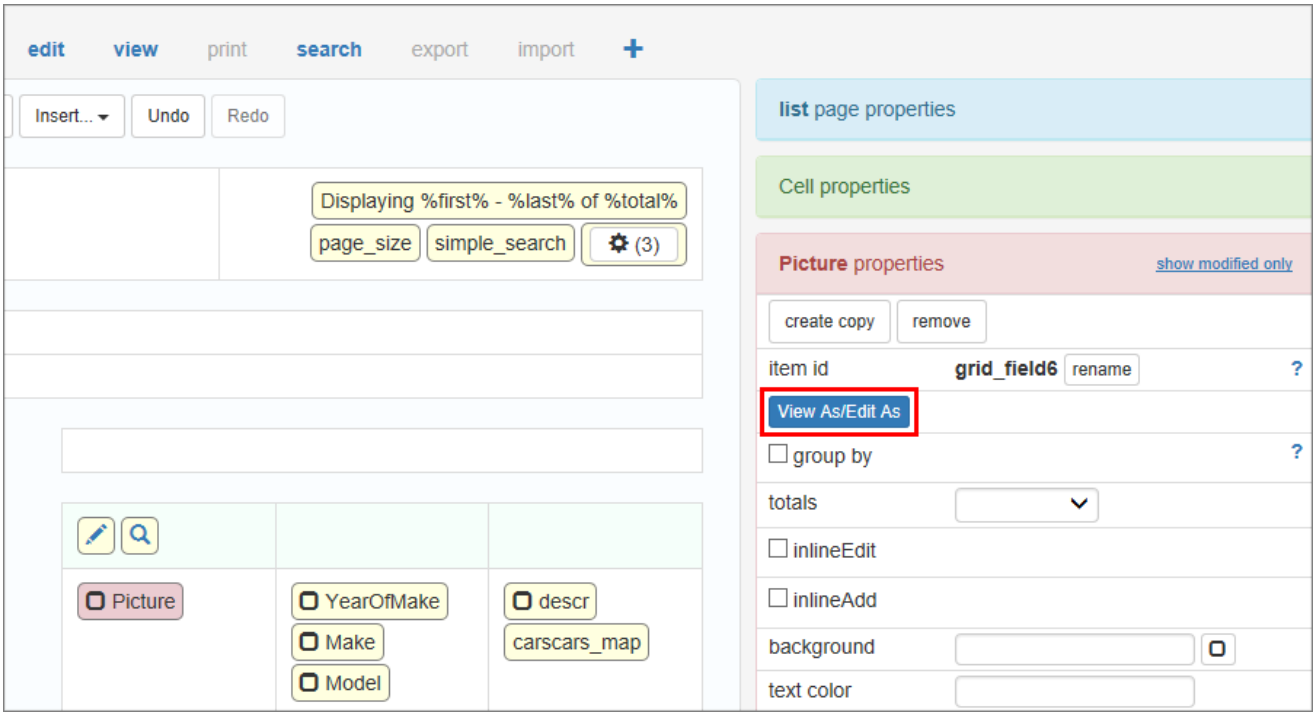
2.16.15.1 "Edit as" settings

Quick jump

Using "Edit as" settings

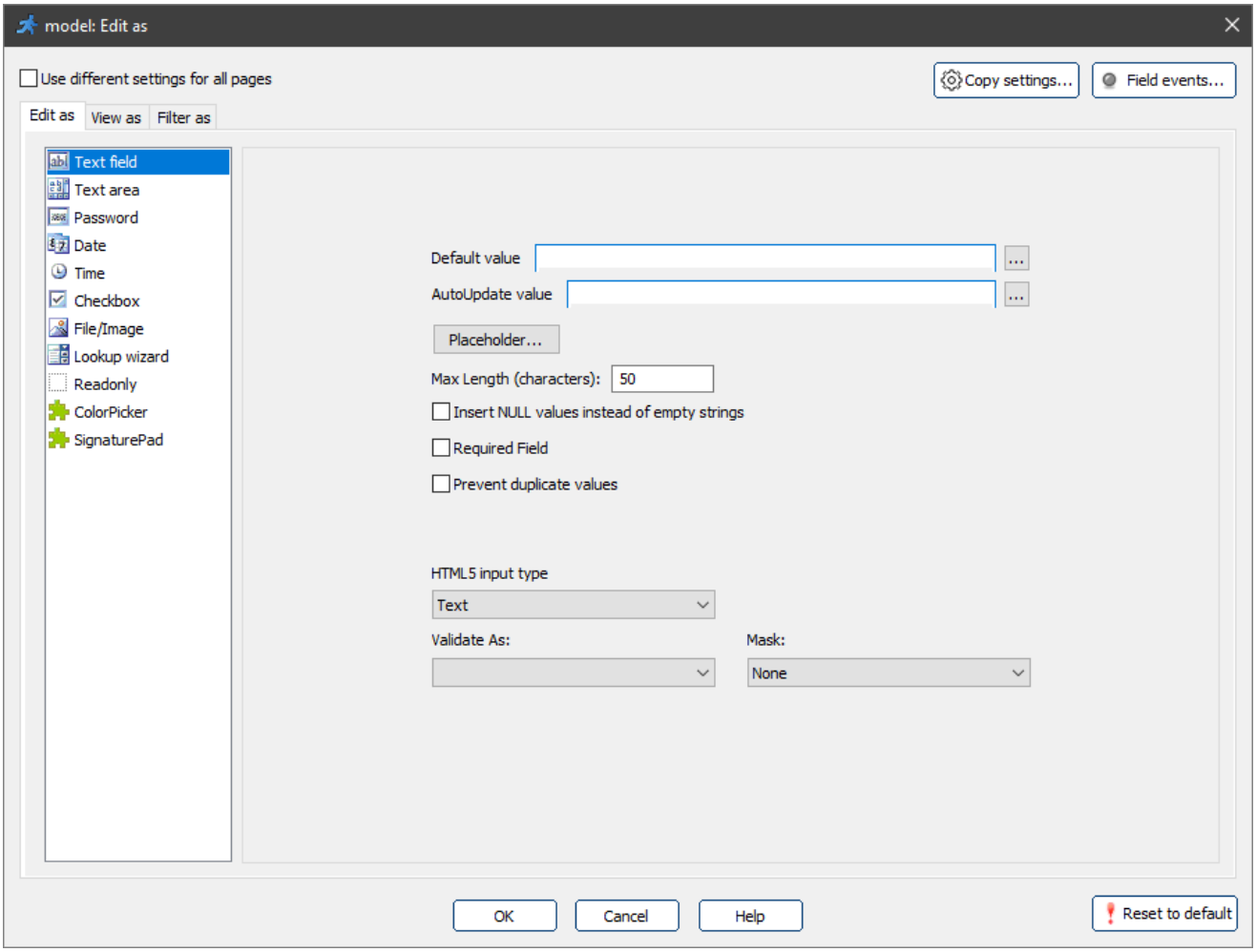
You can customize your data appearance on the **Add**/**Edit**/**Search** pages using formatting options in the **"Edit as" settings** dialog. You can define the field settings for all pages at once or separately for each page.

To control the field appearance on the page, click on the field and then on the **View As**/**Edit as** button in the *properties*.



To set different field formats for different pages enable the **Use different settings for all pages** checkbox.

Depending on the selected format, you will see different dialogs.



Common options for "Edit as" formats

Default value

Default value will be assigned to a field directly on the **Add**/**Search** pages. **Default value** should be a valid PHP expression.

Text expressions must be quoted.

Example of the default value	Description

35	Number.
"ABC"	Text.
now()	Current date/time.
$_SESSION["UserID"]	Username of the person who created or updated the record.
$_SERVER["REMOTE_ADDR"]	IP address of the user.

> **Note**: the **Default value** option appears only when you have not selected the **Use different settings for all pages** checkbox.

Validate as

For more information about validation, see Validation types.

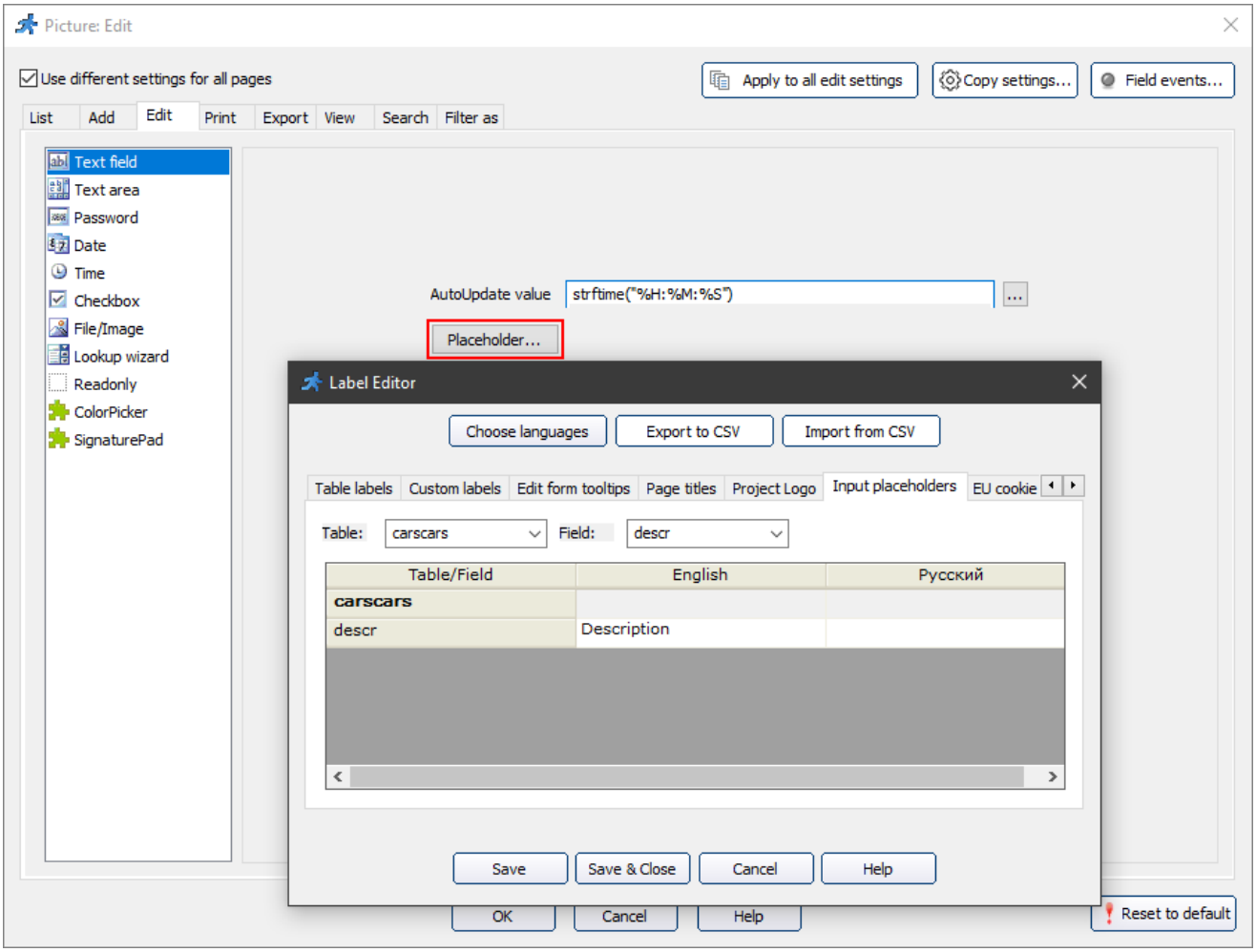
AutoUpdate value

An **AutoUpdate value** will be assigned to a field every time a record is updated on the **Edit** page. You can use this feature to keep track of who and when updated the record.

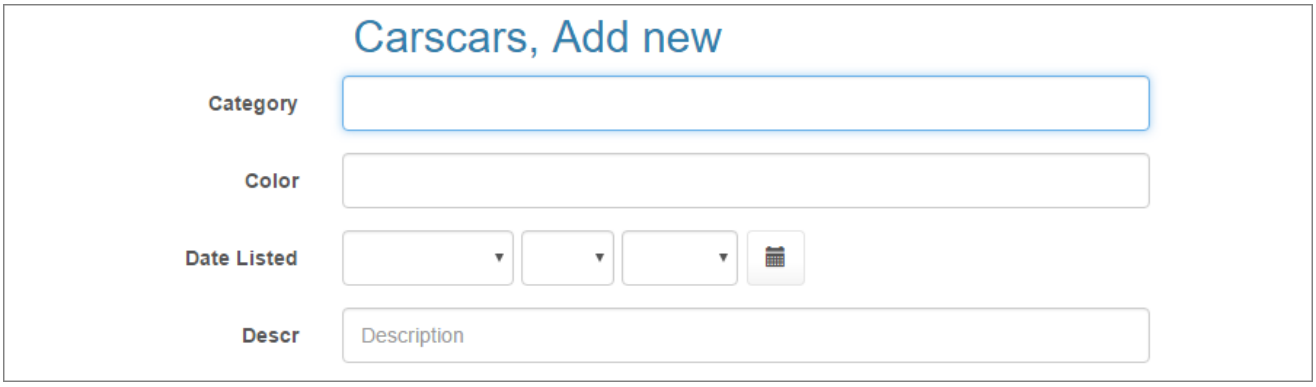
An **AutoUpdate value** should be a valid PHP expression. See sample expressions for the **Default value**.

Placeholder

Placeholders are the tooltips within the field that disappear as soon as the user starts typing something in that field.



The placeholder looks like this on the generated page:



You can also change the placeholders in the Label Editor or using the Labels/Titles API.

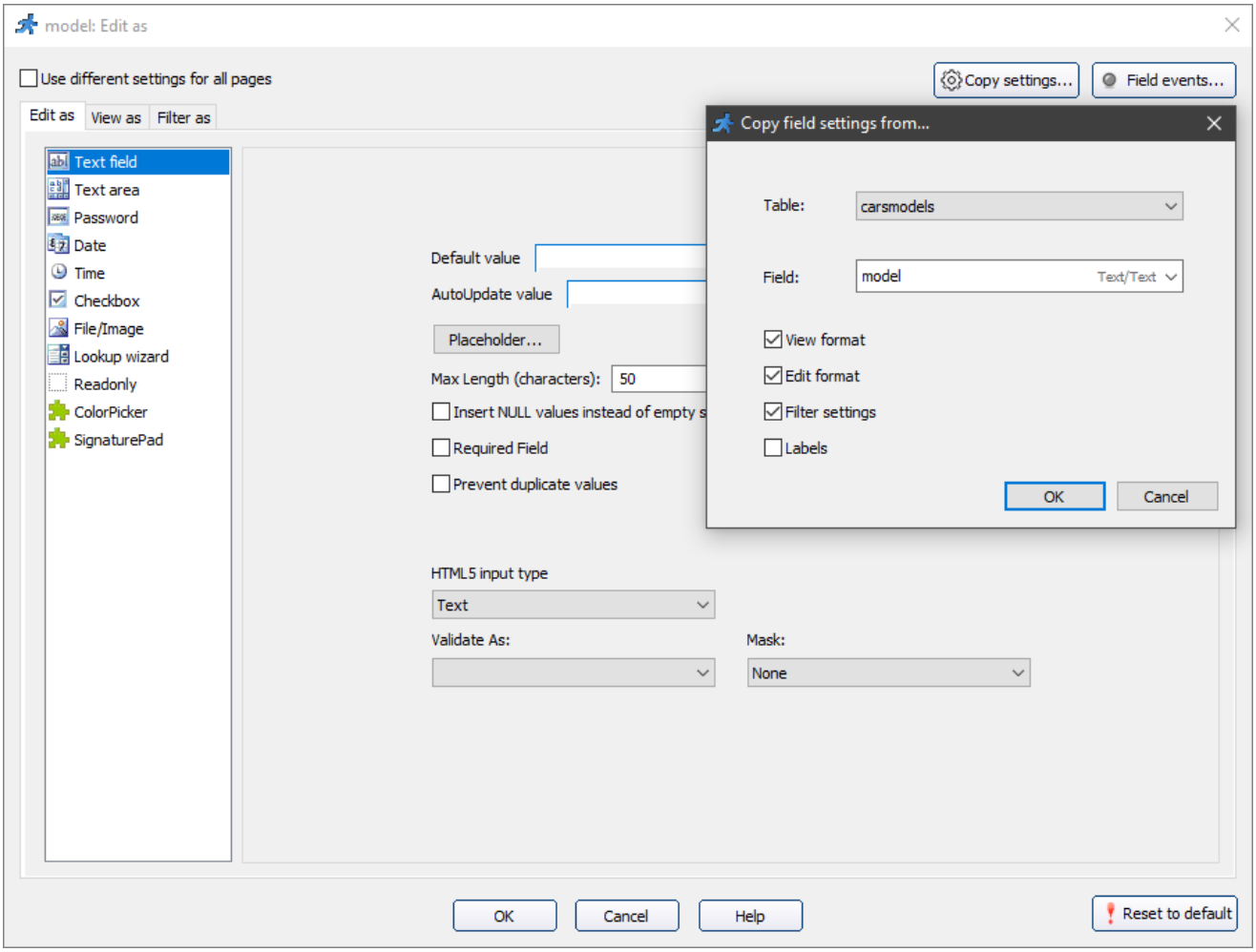
Feel free to check the following live demo that showcases placeholders.

Prevent duplicate values

Use this option to prevent users from entering duplicate values. Enter a message that will be displayed when users enter a value that already exists in the database.

Copy settings

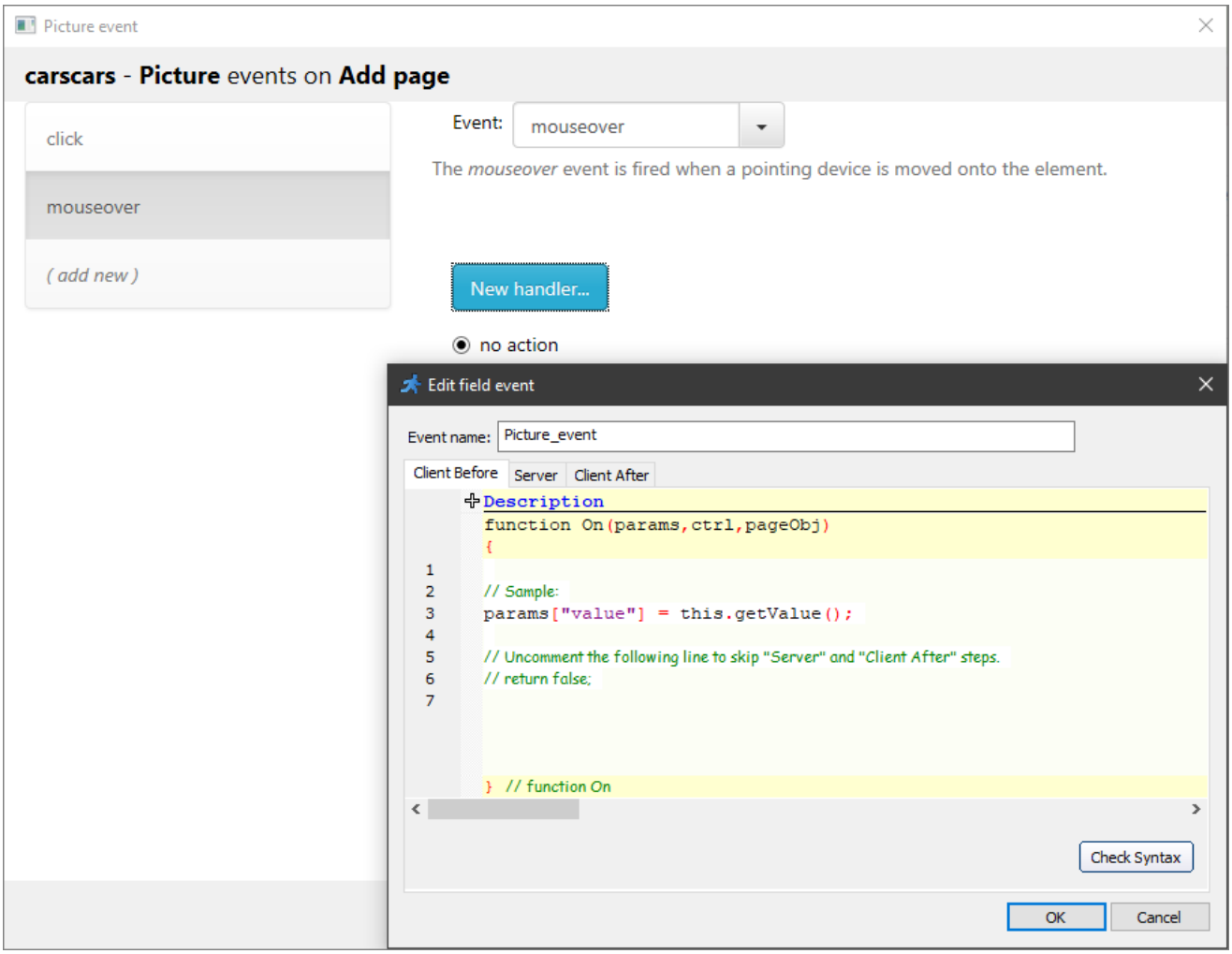
Copy settings option allows you to copy the settings from another field.



Field events

The Field events option allows performing an action when the cursor enters, leaves or is over an edit field. Perform any sort of validation, make other fields hidden or required, etc. Field events are designed to work on **Add**, **Edit**, **View** and **Register** pages.

For example, the *mouseover* event occurs when the pointer is over the selected element.



"Edit as" types

Text field

A simple text box. For more information, see "Edit as" settings: Text field.

Text area

A multiline text area. For more information, see "Edit as" settings: Text area.

Password

A password field. All entered characters appear as "*".

> **Note**: if you want the empty password field to insert NULL into the table, select the **Insert NULL values instead of empty strings** checkbox.
>
> This option may come as useful for indicating that no data is available for the field.

Date

A date edit control. For more information, see "Edit as" settings: Date.

Time

A time edit control. For more information, see "Edit as" settings: Time.

Checkbox

A checkbox control. Works best with the following data types:

- MS Access: Yes/No field;
- SQL Server: TINYINT or BIT field;
- MySQL: TINYINT field;
- Oracle: NUMBER(1) field.

Select the **Required field** option to make the checkbox required.

That way it should be selected before submitting a form.

Use this option for the fields such as the *Accept License Agreement* and *Terms*.

File/Image

Choose this format if you store files/images in this field. For more information, see "Edit as" settings: File/Image.

Lookup wizard

A drop-down box with a list of values. For more information, see "Edit as" settings: Lookup wizard.

Readonly

Use this format for a field that should not be edited.

ColorPicker

The **ColorPicker** control allows users to select the color the same way they do in Adobe Photoshop. For more information, see "Edit as" settings: ColorPicker.

SignaturePad

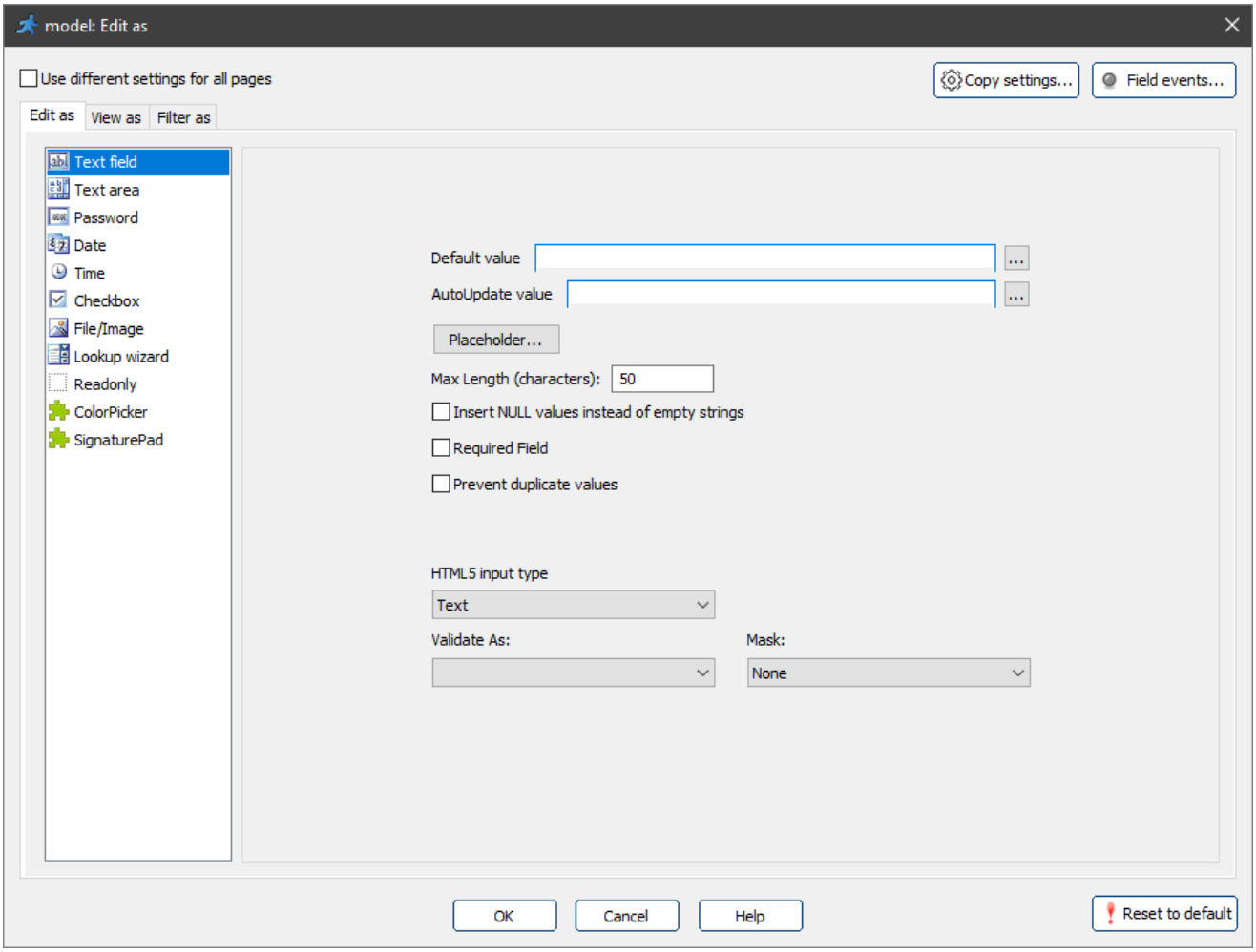
The **SignaturePad** control allows you to add a signature pad to your forms. For more information, see "Edit as" settings: SignaturePad.

See also:

- Field events
- How to access fields in the field events
- About Labels/Titles API
- How to create a custom Edit control plugin
- About Date Control API
- "View as" settings
- About Page Designer
- About Editor

2.16.15.2 Text field

The **Text field** is a simple text box control. **Max length (characters)** option defines the maximum amount of characters that can be typed into the field.



> **Note**: if you want the empty text field to insert NULL into the table, select the **Insert NULL values instead of empty strings** checkbox.
>
> This option may come as useful for indicating that no data is available for the field.

Validate As

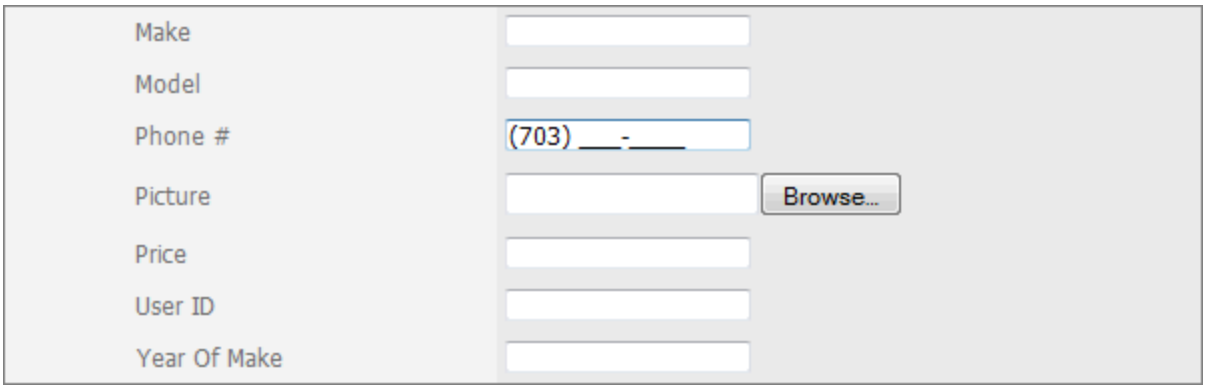
For more information about validation, see Validation types.

Mask

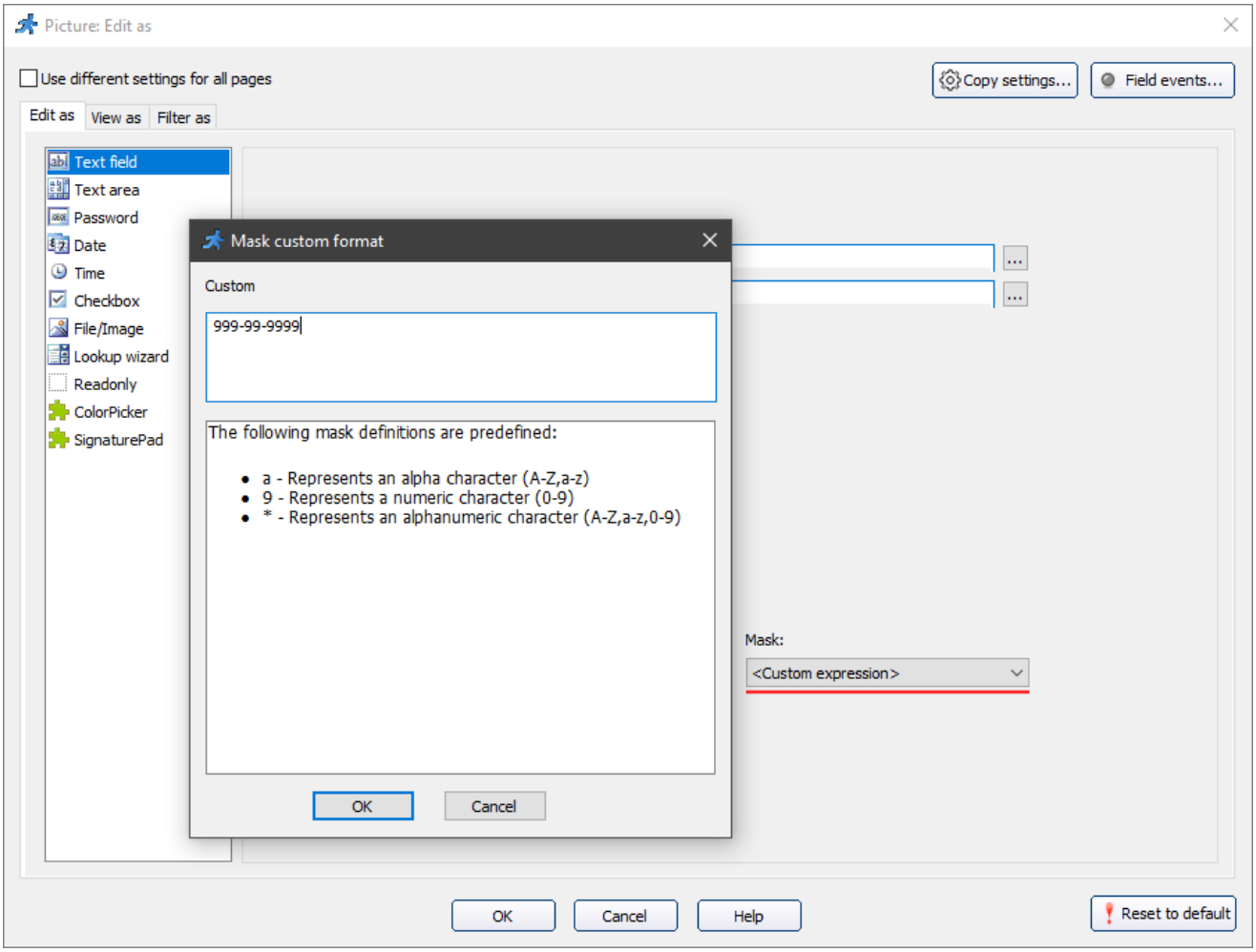
Masked input is a template that applies a specific format of input. It helps users enter the data (phone numbers, dates, IP addresses, etc.) into the field in a predefined way.

Note: when you use a mask make sure to set HTML5 input type to Text.

An example of a phone number masked input:



You can use one of the predefined mask formats or create your own custom expression.



The following mask definitions are predefined:

- a - Represents an alpha character (A-Z,a-z)
- 9 - Represents a numeric character (0-9)
- * - Represents an alphanumeric character (A-Z,a-z,0-9)

A few examples of custom expressions:

- US social security number: 999-99-9999
- Date (mm/dd/yyyy): 99/99/9999

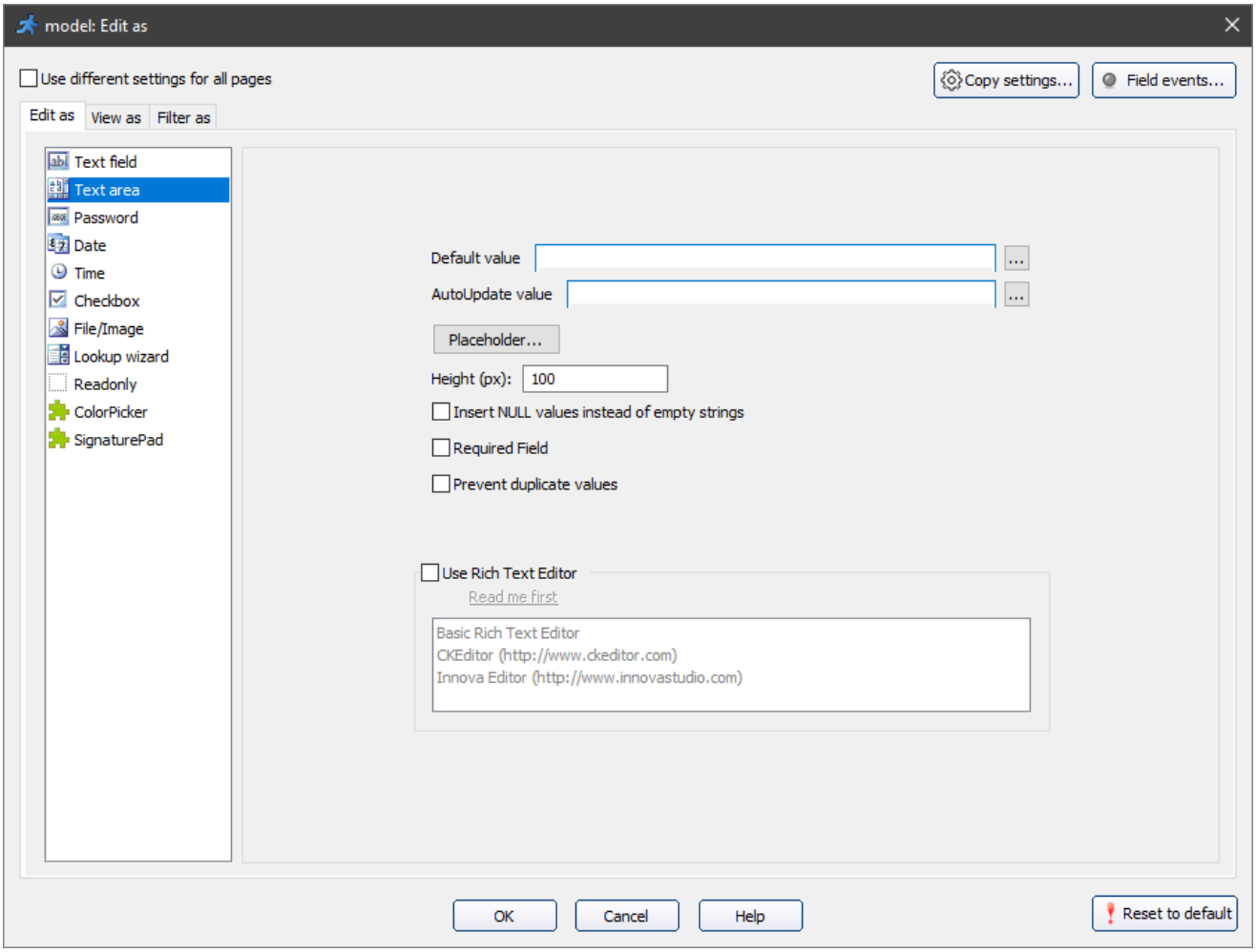
- Time (hh:mm:ss): 99:99:99
- Phone number: (999) 999-9999
- Phone number with optional extension: (999) 999-9999? x99999

See also:

- "Edit as" settings: Text Area
- "Edit as" settings: Validation types
- "Edit as" settings
- About Page Designer
- About Editor

2.16.15.3 Text area

The **Text area** is a multiline text area control. You can set the height of the text area in pixels with the *Height (px)* option.



Select the **Use Rich Text Editor** checkbox to enable one of the advanced WYSIWYG editors for the field.

For more information about using **Rich Text Editor**, see Rich Text Editor plugins.

> **Note**: if you want the empty text area to insert NULL into the table, select the **Insert NULL values instead of empty strings** checkbox.
>
> This option may come as useful for indicating that no data is available for that field.

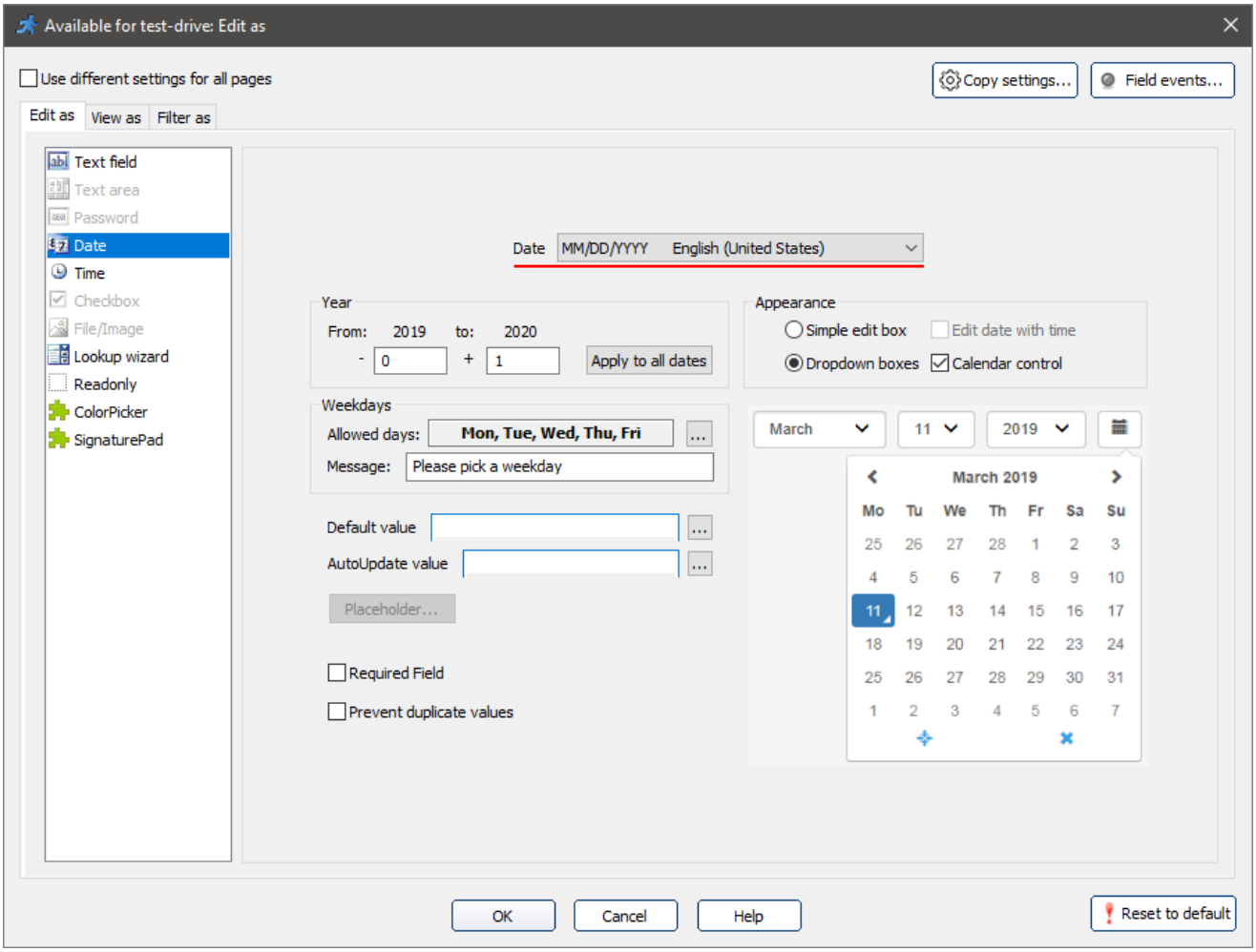
See also:

- "Edit as" settings: Text field

- "Edit as" settings
- About Page Designer
- About Editor

2.16.15.4 Date

The **Date** is a date edit control. The regional date appearance can be selected in the **Date** dropdown.



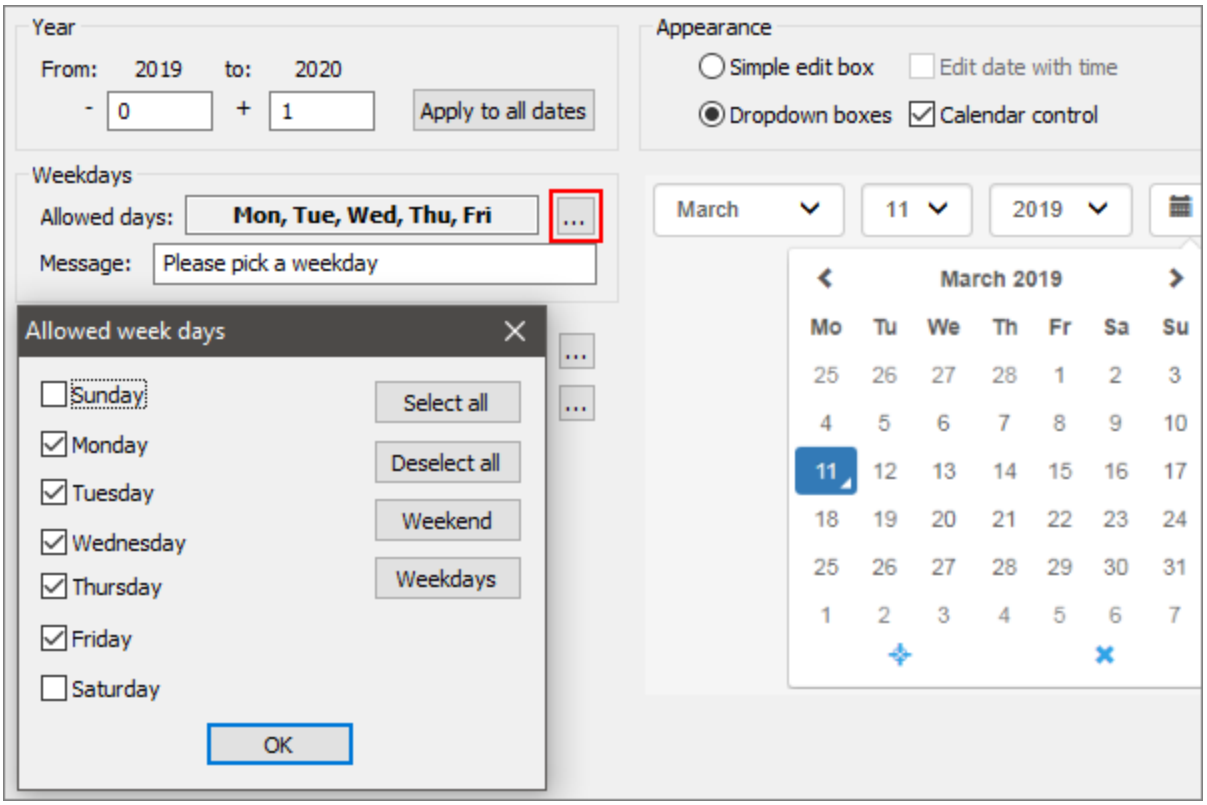
Date edit control options:

- **Appearance**. You can choose between *Simple* edit box and *Dropdown* boxes styles of the *Date* field. Select the **Calendar control** checkbox to enable a calendar popup for picking a date.

  > **Note**: a *Simple* edit box allows entering time as well as the date. Select the **Edit date with time** *checkbox* to do so.

- **Year**. Limit the range of years to choose from with the *From* and *To* fields. The **Apply to all dates** option applies the selected limit to every *Date* field in the app.

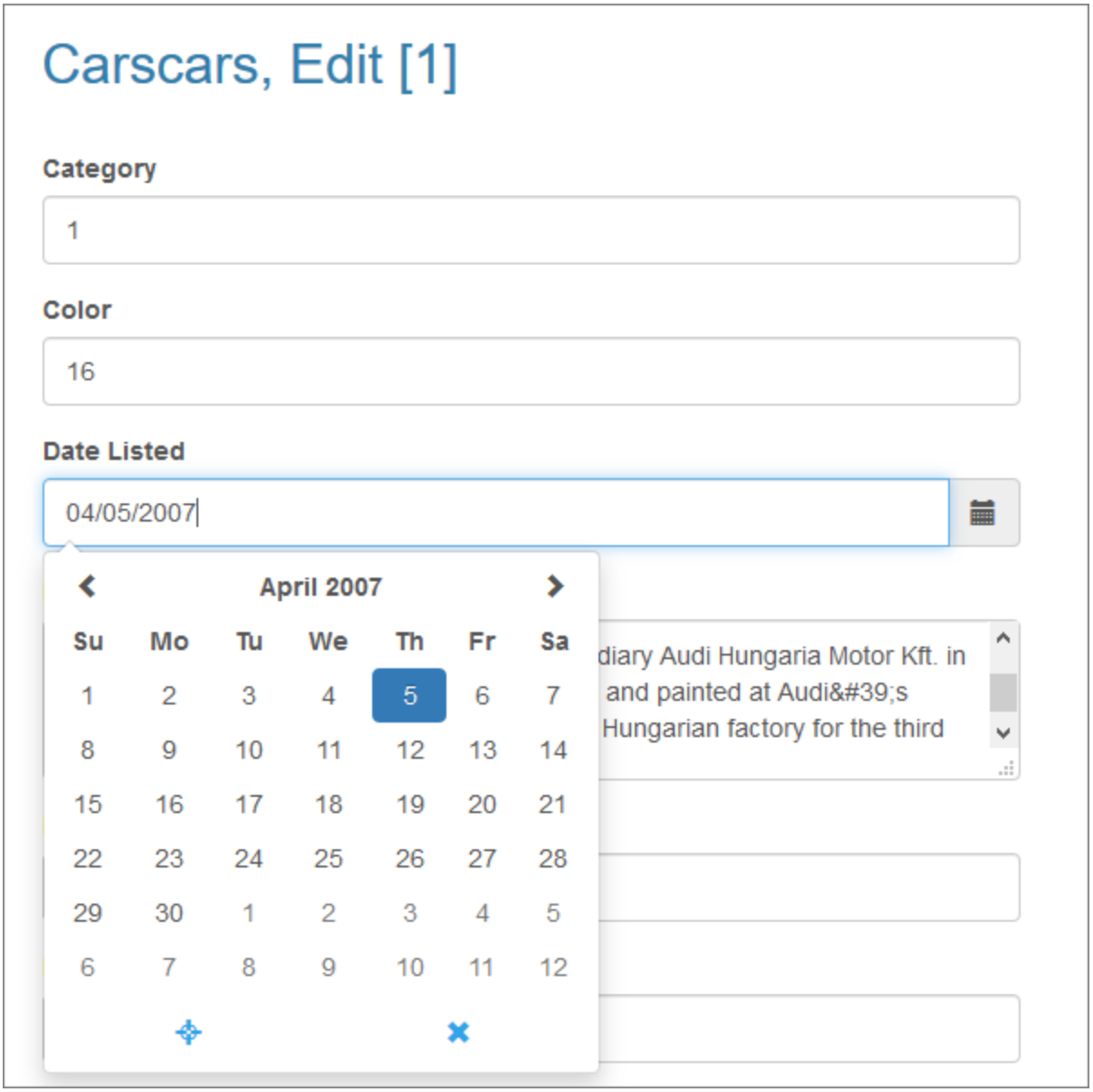
- **Weekdays**. This option allows limiting the weekdays, that the user can choose in the *Date* control. To select the allowed days in the popup, press the **...** button. The *Message* field allows showing a custom message when the user picks a disallowed day.



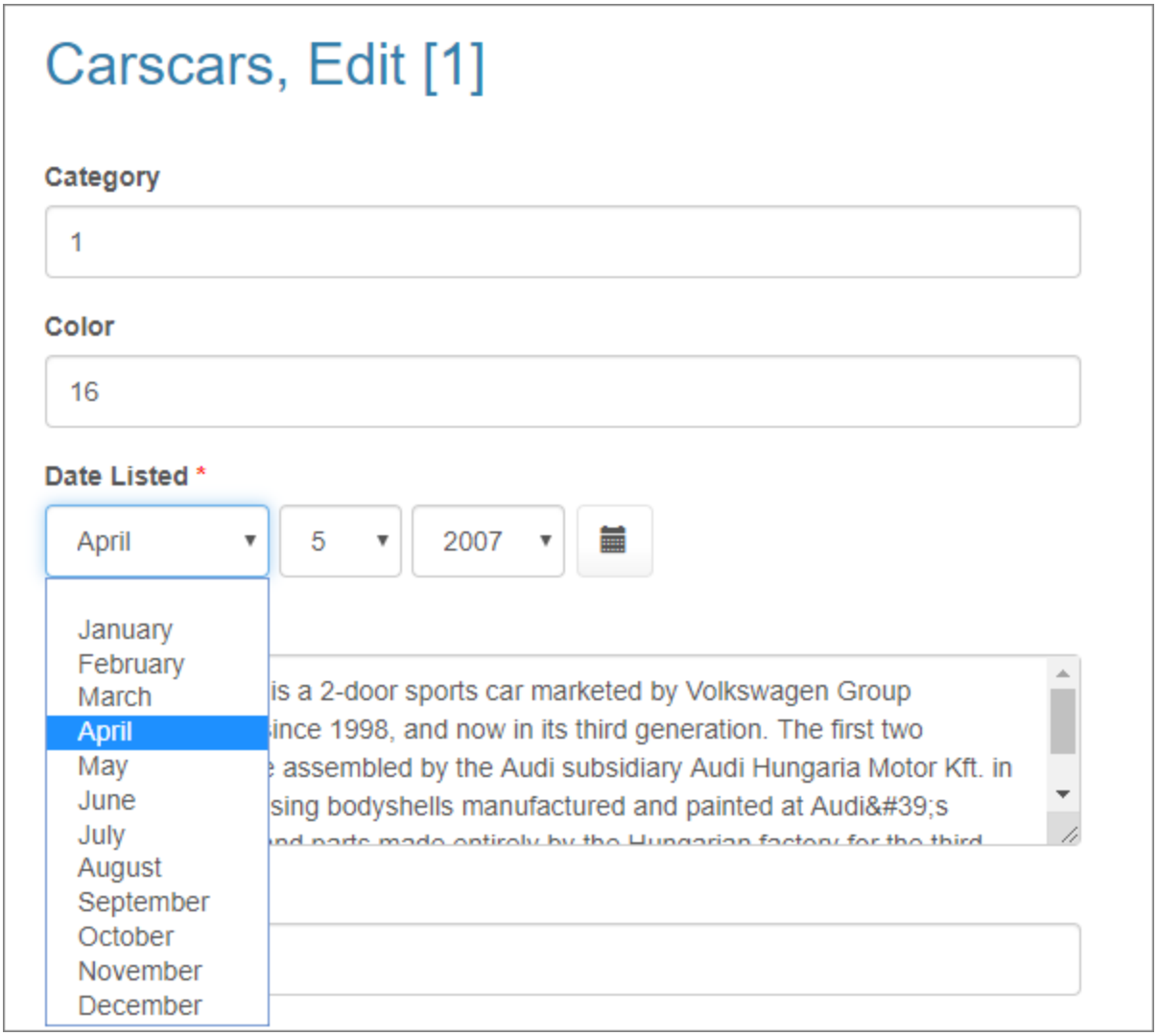
> **Note**: you can also allow certain days, or limit the allowed date intervals with the Date API.

Examples

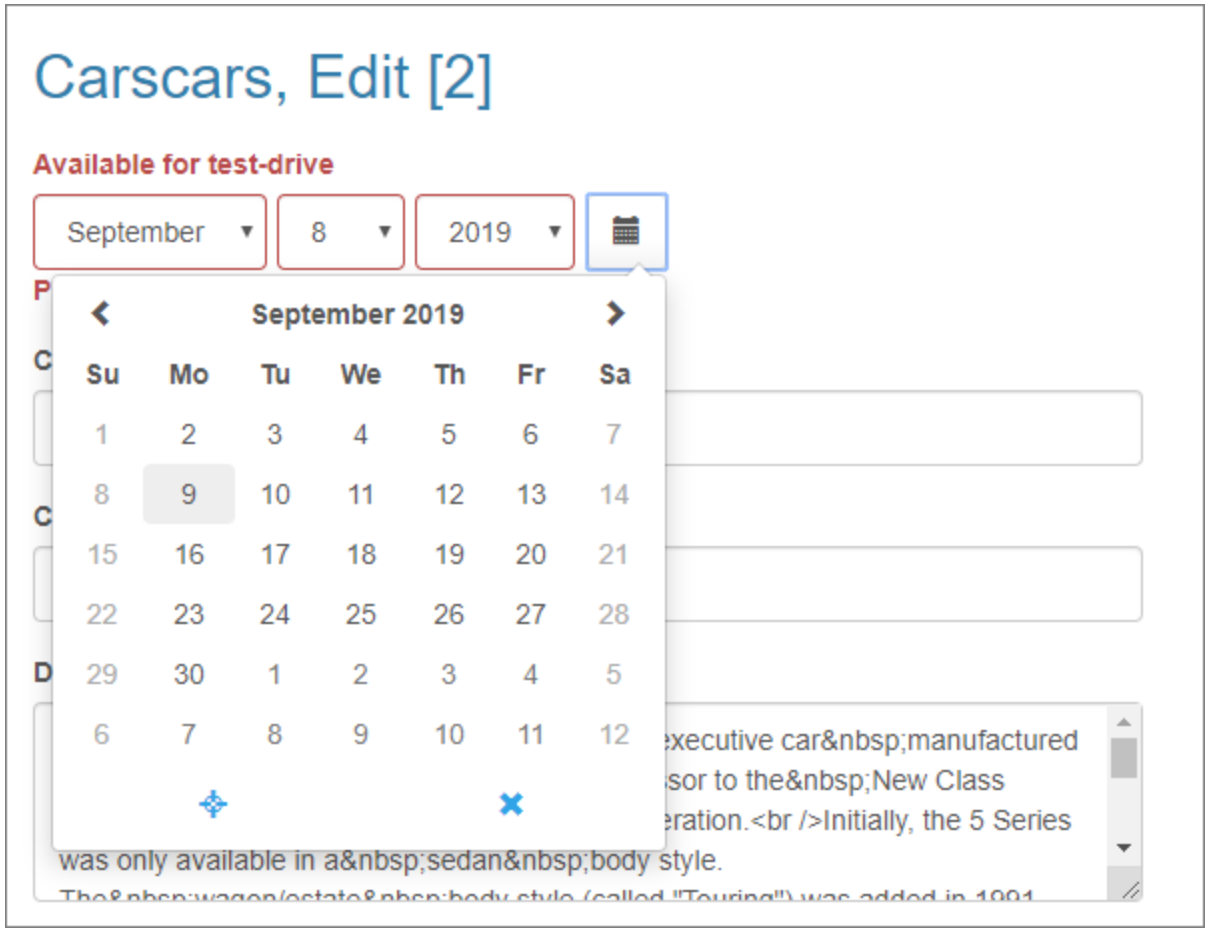
1. A simple edit box style *Date* field with a **Calendar control** (inline mode):



2. A dropdown boxes style required Date field with a **Calendar control**:



3. A dropdown boxes style *Date* field with a **Calendar control**, limited to weekdays only.

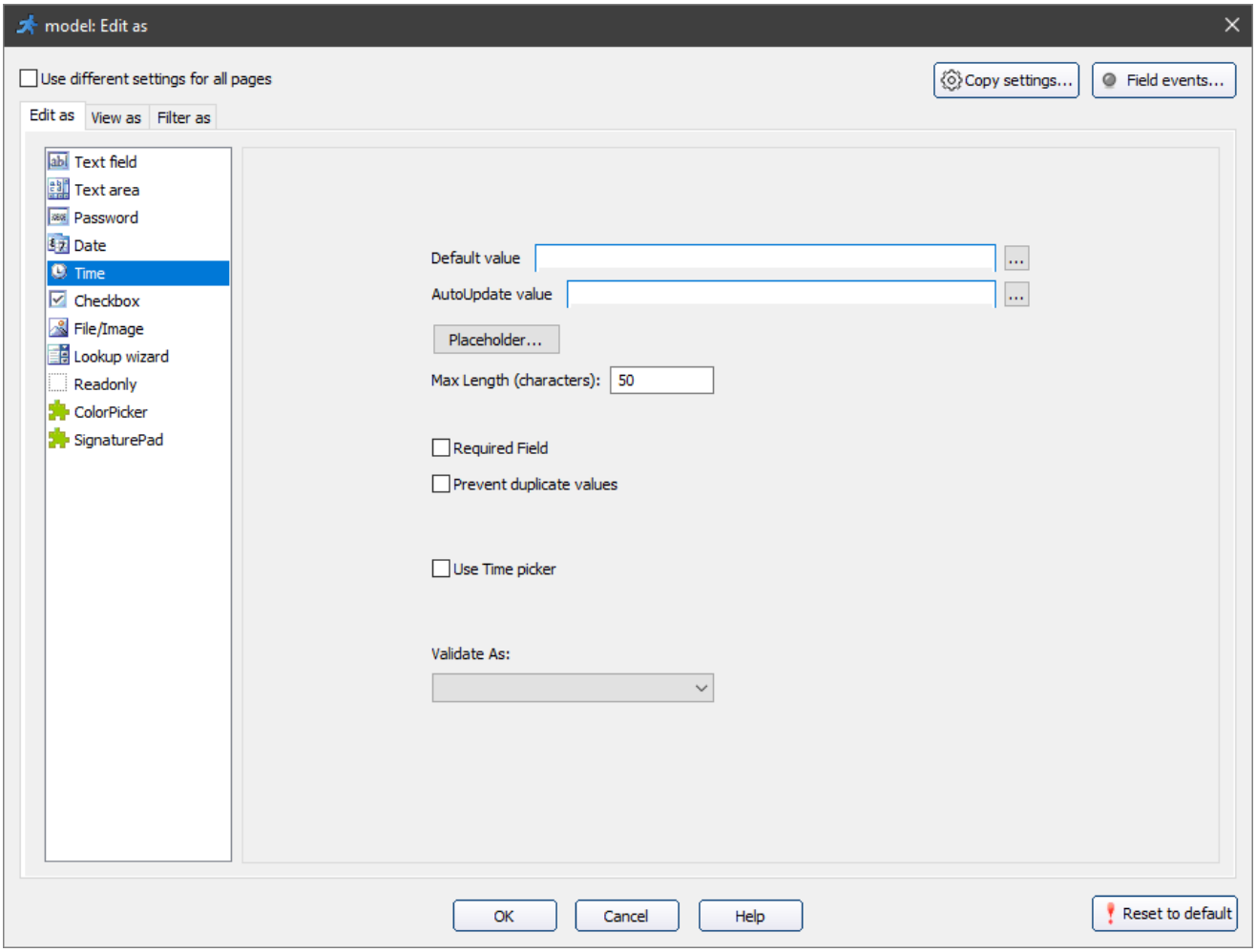


See also:

- About Date Control API
- "Edit as" settings
- About Page Designer
- About Editor

2.16.15.5 Time

The **Time** is a time edit control.



The **Use Time picker** checkbox gives access to the time picker screen, which allows you to pick the time in a popup instead of typing it in.

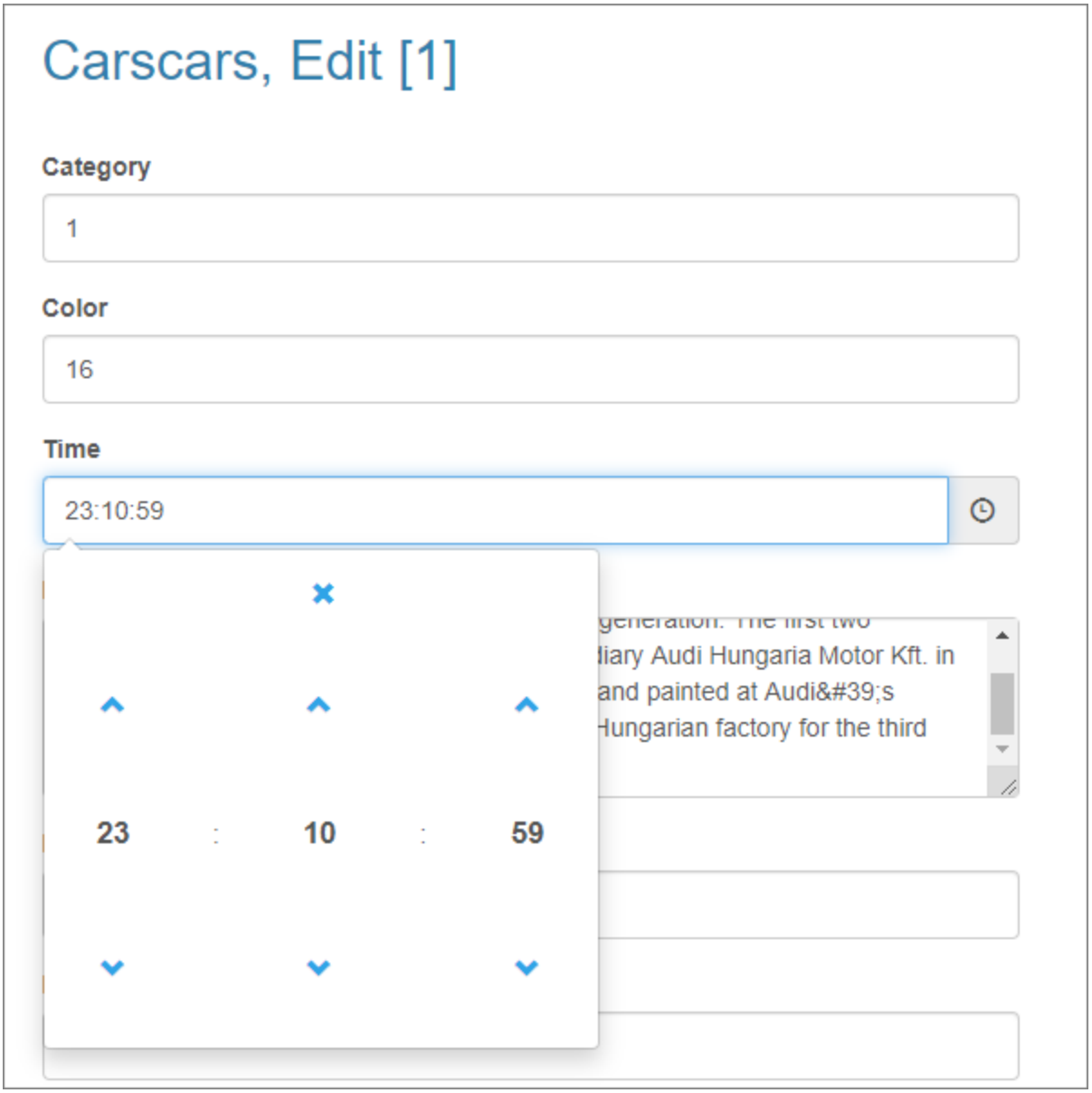
Select the 12 or 24-hour format and one of the available minute intervals.

Enable the option **Display seconds** to show seconds within the field.

Validate As

For more information about validation, see Validation types.

An example of how the *Time edit* with the **Time picker** looks like on the generated page:



See also:

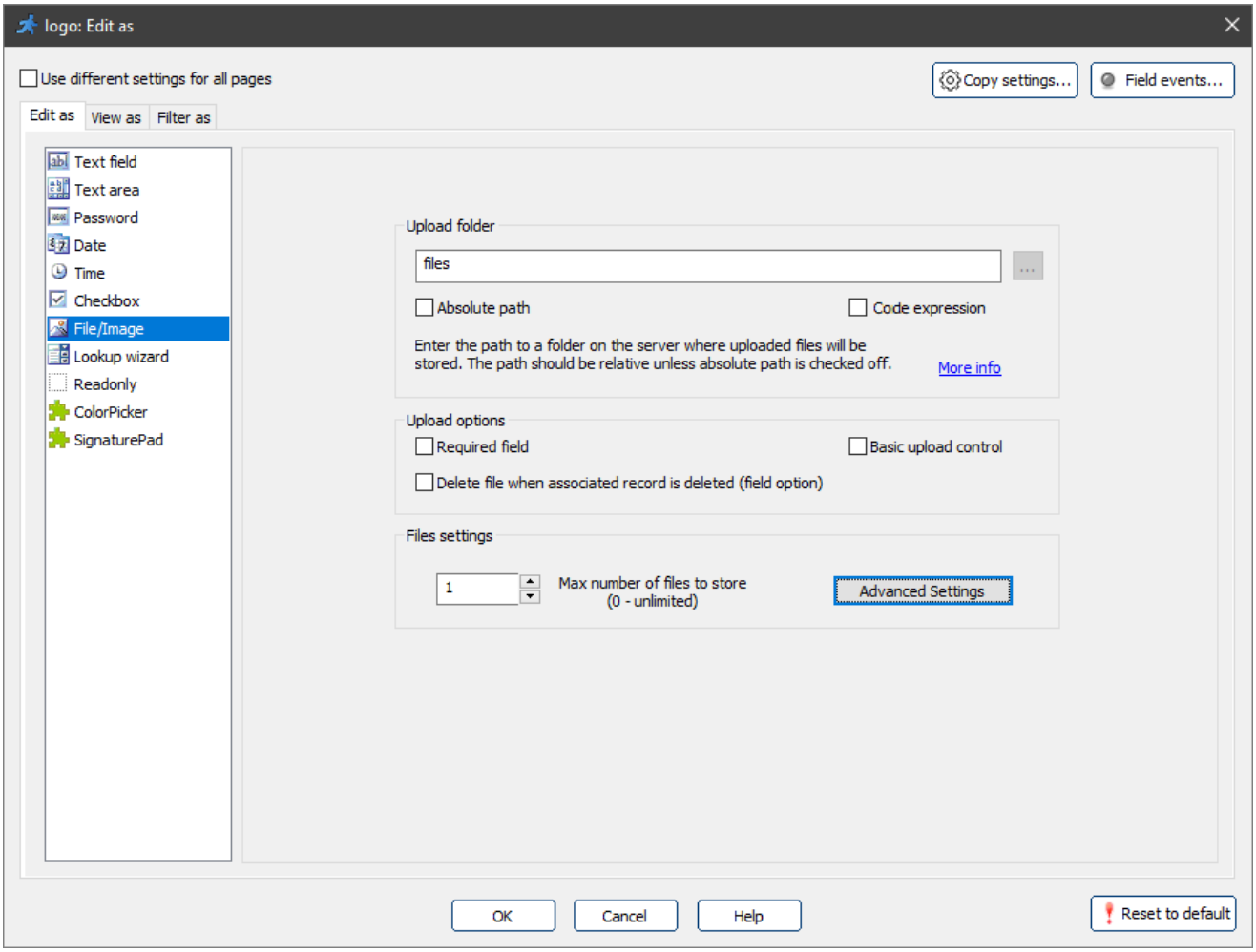
- "Edit as" settings
- About Page Designer
- About Editor

2.16.15.6 File/Image

Quick jump

Edit as File/Image

Depending on the field type, this control allows uploading images and files to the database (*binary* field) or some directory on the Web server (*text* field).



Text field

File/Image upload control allows selecting and uploading multiple files at once. You only need one database field to store all the file names. A *long* text field is recommended for multiple file upload, for example, *Memo* in MS Access, *Mediumtext* in MySQL, TEXT or *Varchar(max)* in SQL Server.

Since images and files are uploaded to some directory on the server, you need to enter the path to that directory. The path should be relative to the folder with generated pages. However, since some hosting providers do not allow referring to the directories above the current one, you can use the **Absolute path** option and specify the full path to the directory on the server to which to upload the files. E.g., *C:\\project1\files*.

The path to the upload folder may contain PHP expression (select *Code (PHP)* expression option). You can use this option, for example, to save each user's files to a separate folder. Sample upload path in this case is:

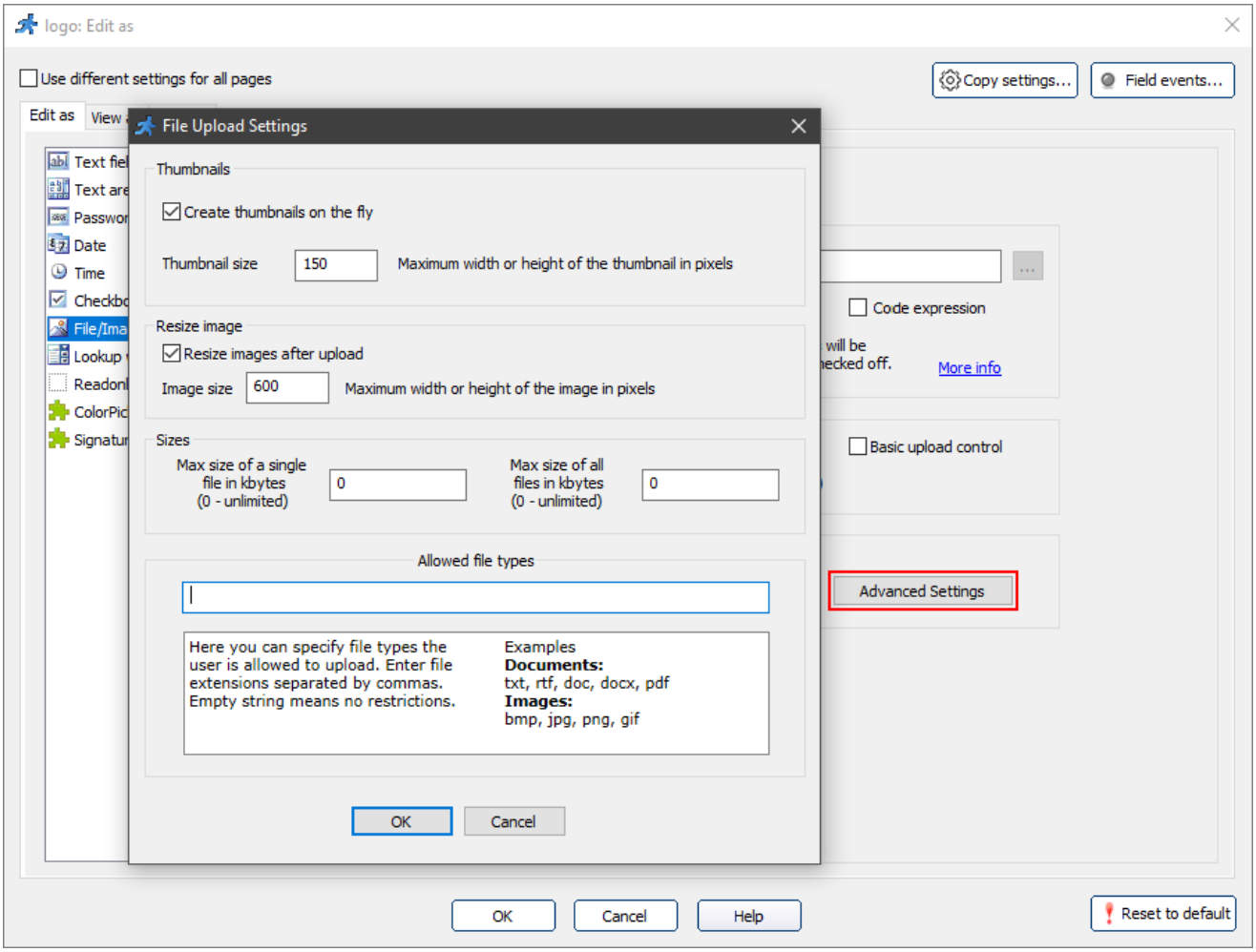
```
$folder = $_SESSION["UserID"];
```

Use the **Delete file when associated record is deleted** option to make sure that the record from the database is deleted along with the associated file. This option is global and applies to all tables and fields.

You can restrict the maximum number of files to store. Enter "0" for an unlimited number of files. Note that the default value is "1" and you need to increase it to enable multiple upload.

Enable **Basic upload control** option to select the old style file upload control that lets users upload one file at a time.

Advanced settings



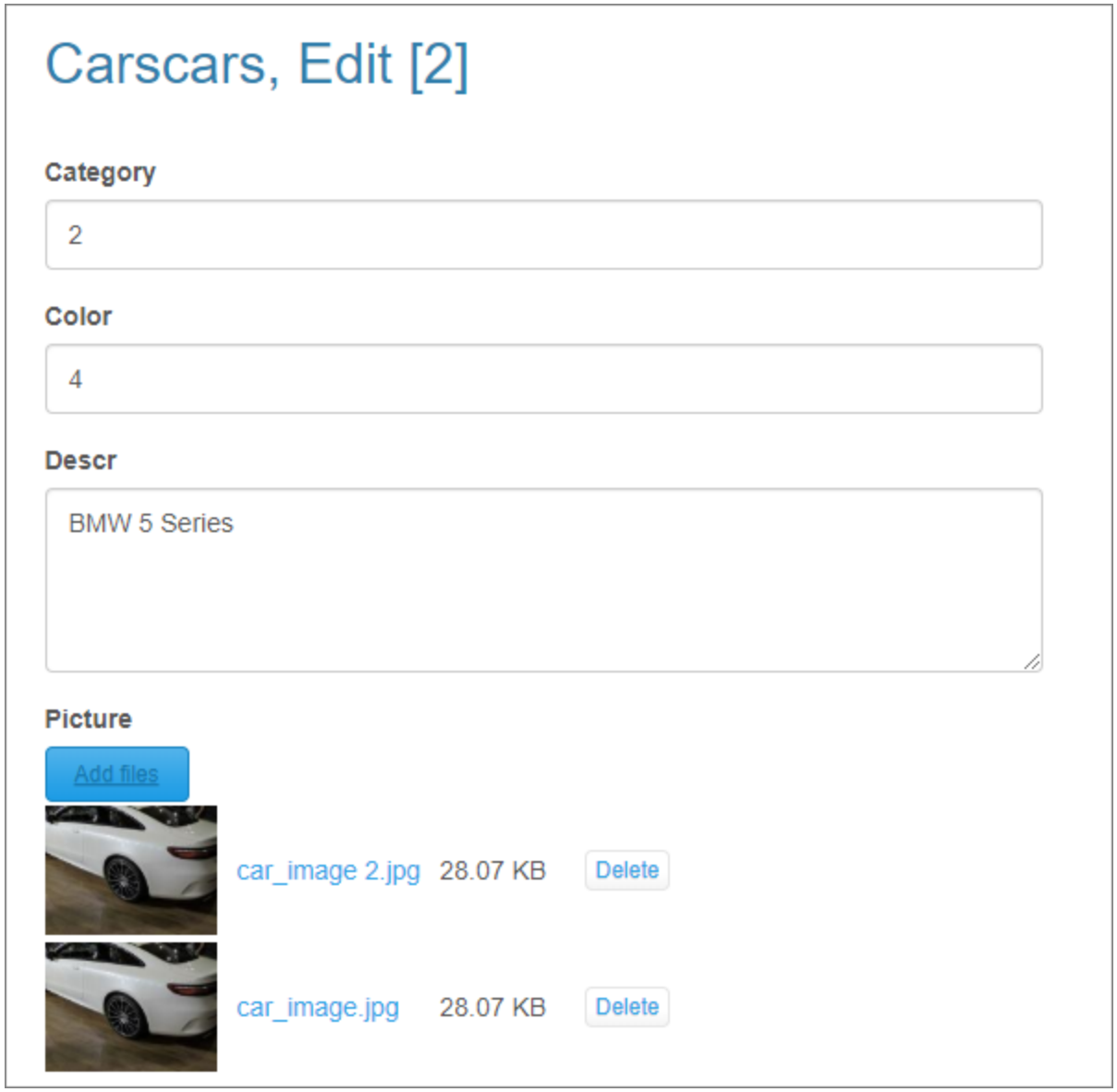
- **Create thumbnails on the fly**. PHPRunner allows creating thumbnails on the fly. To do this, select this checkbox and choose the thumbnail size. Thumbnail images are stored in the same folder as the images.

> **Note**: to use thumbnails functionality, you need to have the GD library installed. On Windows you need to uncomment PHP_gd2.dll extension in your PHP.ini file.

- **Resize images after upload**. To resize images on upload, select this option and type in the *max width* or *height* of the resulting image.
- **Sizes**. You can restrict the max size of a single file or all of the files stored in the field by changing the *Max size of a single file* and *Max size of all files* fields.
- **Allowed file types**. Add a list of the file extensions allowed to upload in the *Allowed file types* section. An empty string means there are no restrictions.

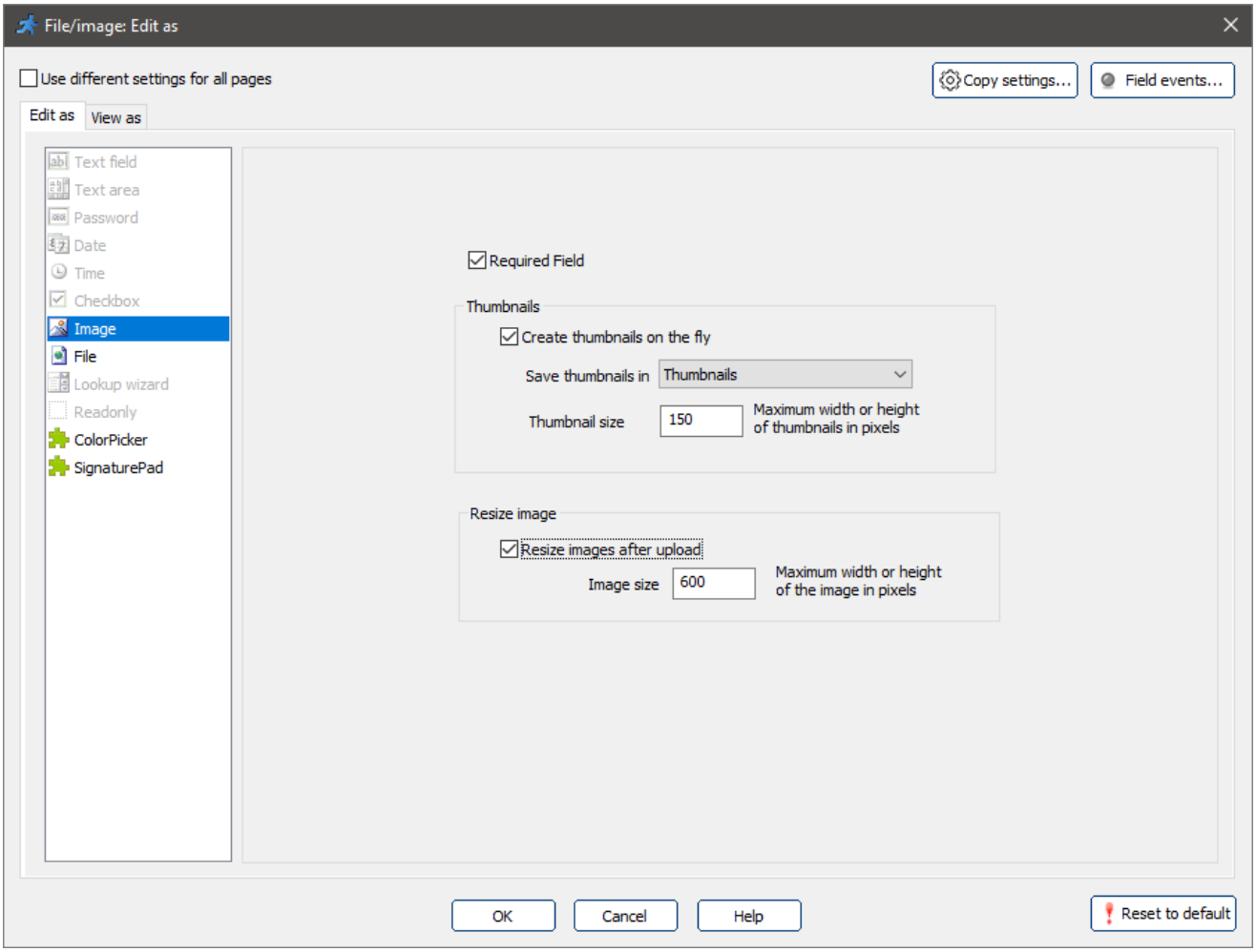
Upload control in the generated app

You can drag one or more files into the upload control on the web page or add them via a browser popup window.



Note: Internet Explorer does now support the drag-n-drop feature.

Binary field: Image

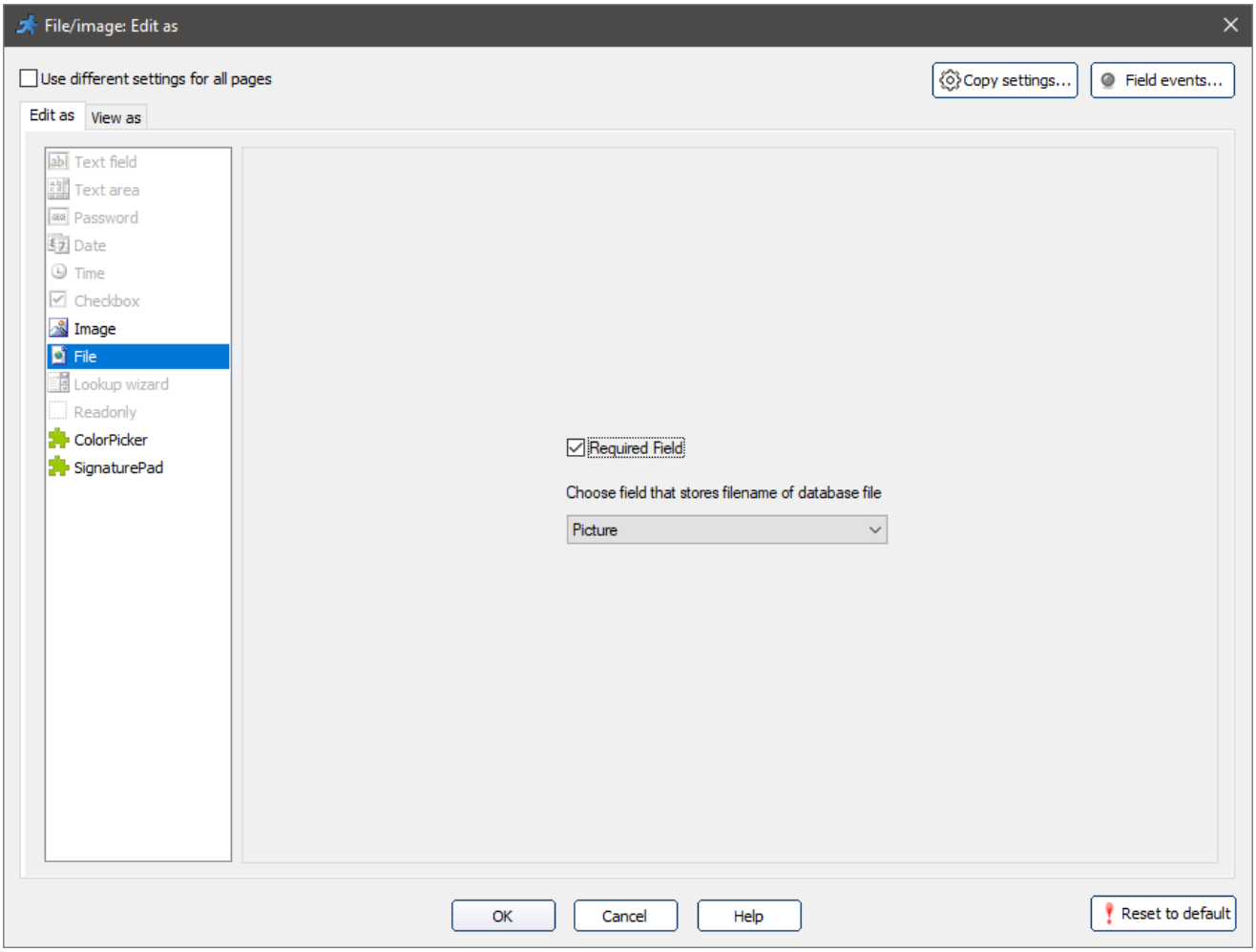


To create thumbnails on the fly, select the **Create thumbnails of the fly** checkbox and choose the field name in which to save the thumbnails.

> **Note**: you need an additional binary field (*MEDIUMBLOB* type) to store thumbnails. This field is an auxiliary one, and we do not recommend displaying it on the pages. You can display/hide fields on the Choose fields page.

To resize images on upload, select the **Resize images after upload** option and select the *max width* or *height* of the resulting image.

Binary field: File



Choose the field that stores the name of the database file. This filename is required to set the correct file type when you retrieve the uploaded file from the database.

If you don't choose the filename field or leave it empty, you will see an **Open with** dialog every time you download this file from the database.

See also:

- "View as" settings: File
- "View as" settings: Image
- "Edit as" settings
- About Page Designer

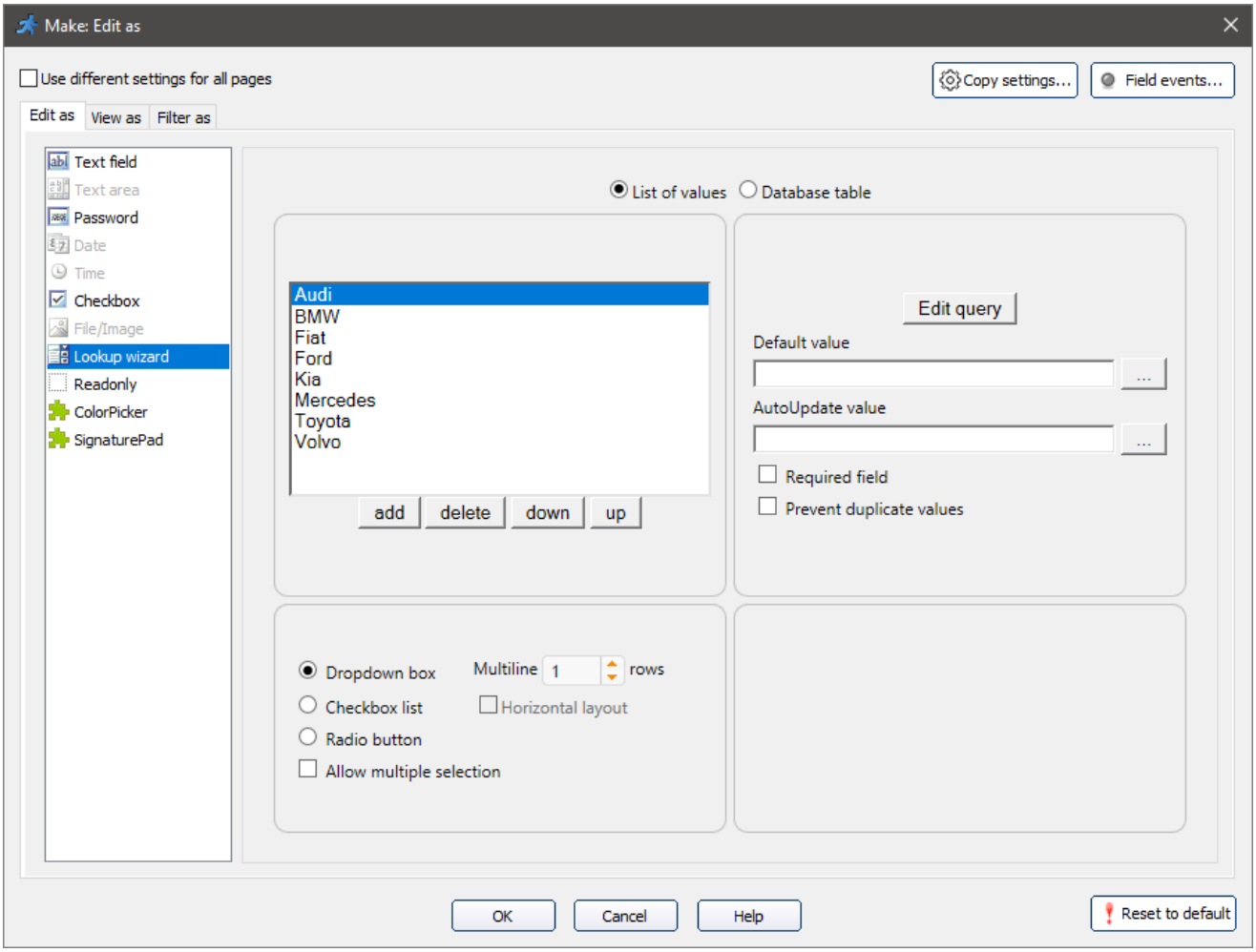
- About Editor

2.16.15.7 Lookup wizard

The **Edit as** option **Lookup wizard** makes the element appear as a list of values. The values can be entered manually (List of values option) or retrieved from the database table (Database table option).

List of values

The **List of values** option allows entering or removing the values manually via **add**/**delete** buttons. Use **up**/**down** buttons to re-order the values on the page.

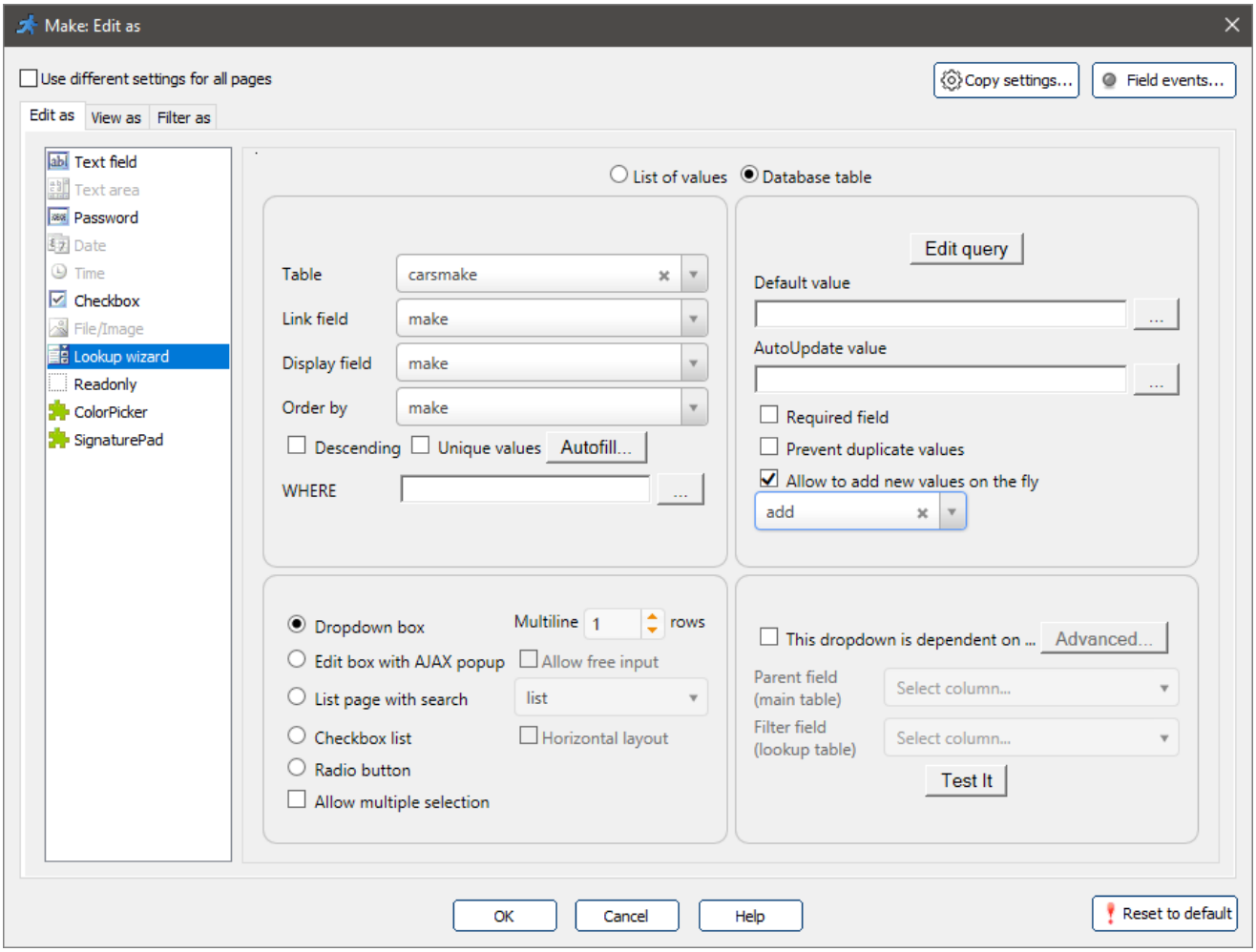


Edit query

The **Edit query** button opens the **Query** screen in a new window. For more information, see About SQL query.

Database table

With a **Database table** option, you can select the existing database table to retrieve the values from.

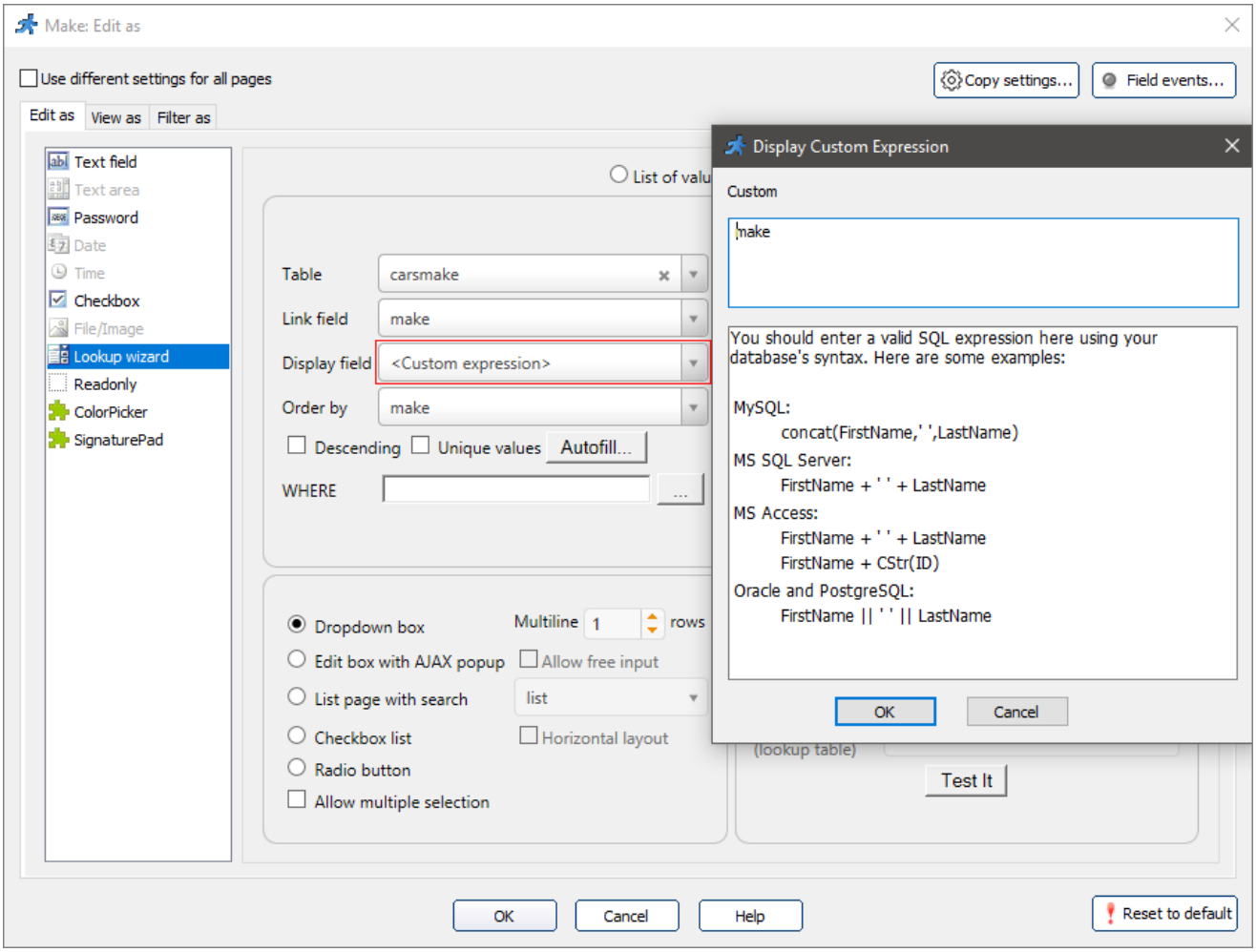


Note: the rules applied to the project tables (such as modified SQL query, Advanced Security settings, changes made to the SQLQuery object in the After table initialized event), also apply to the **Lookup wizard**.

For example, you can limit the list of resulting items with Advanced Security settings.

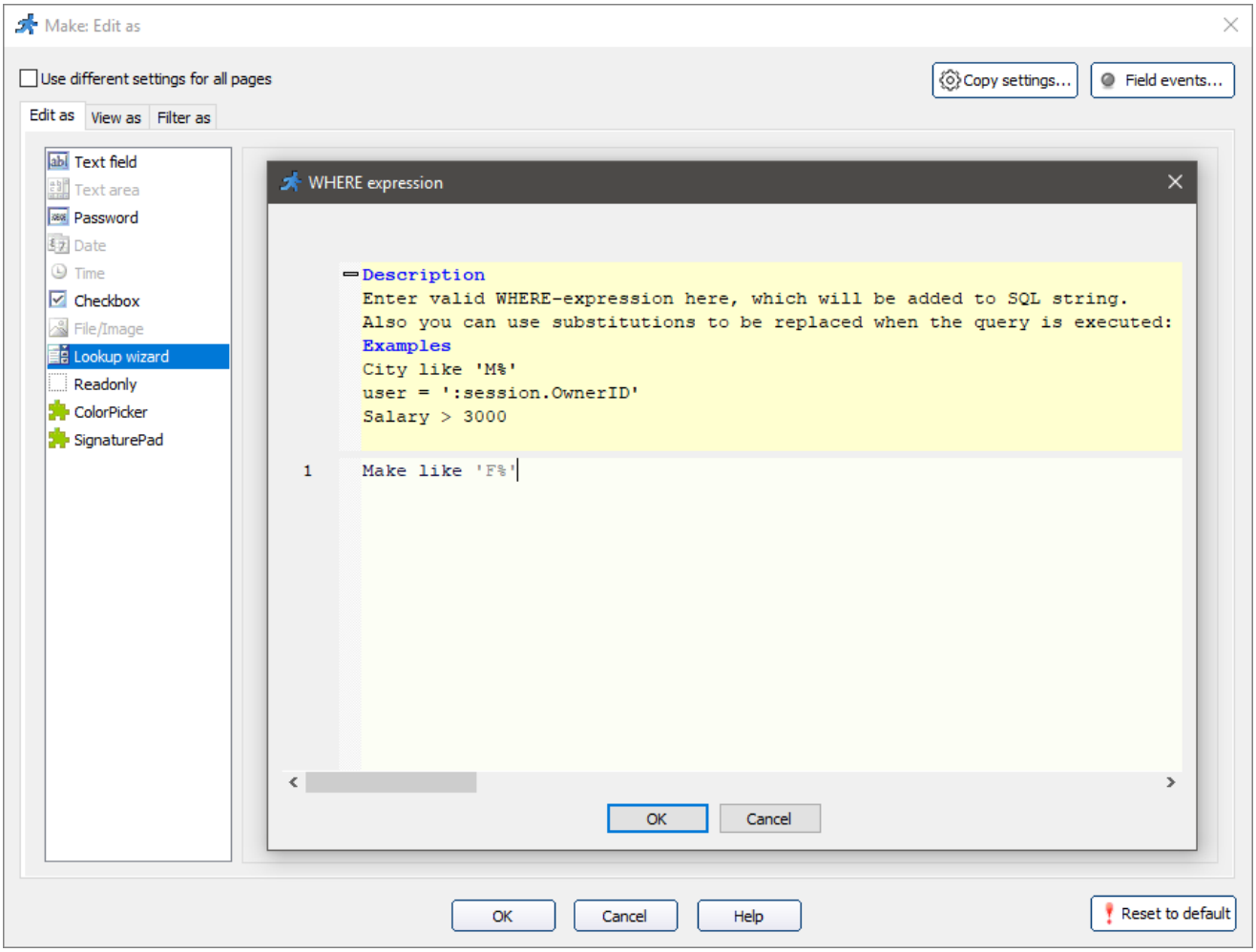
Custom expression in the Display field

You can use custom expressions in the *Display* field to display several values from different fields with a custom design. Click the **Display field dropdown** and select **<Custom expression>** to enter the expression in a popup window.



WHERE expression

The WHERE expression allows filtering the resulting values. In our example, if you put "*Make like 'F%*'" into the WHERE box, only the makers starting with "*F*" will appear in this field.



If you want to use the session table variables in a WHERE clause, write the table names like this:

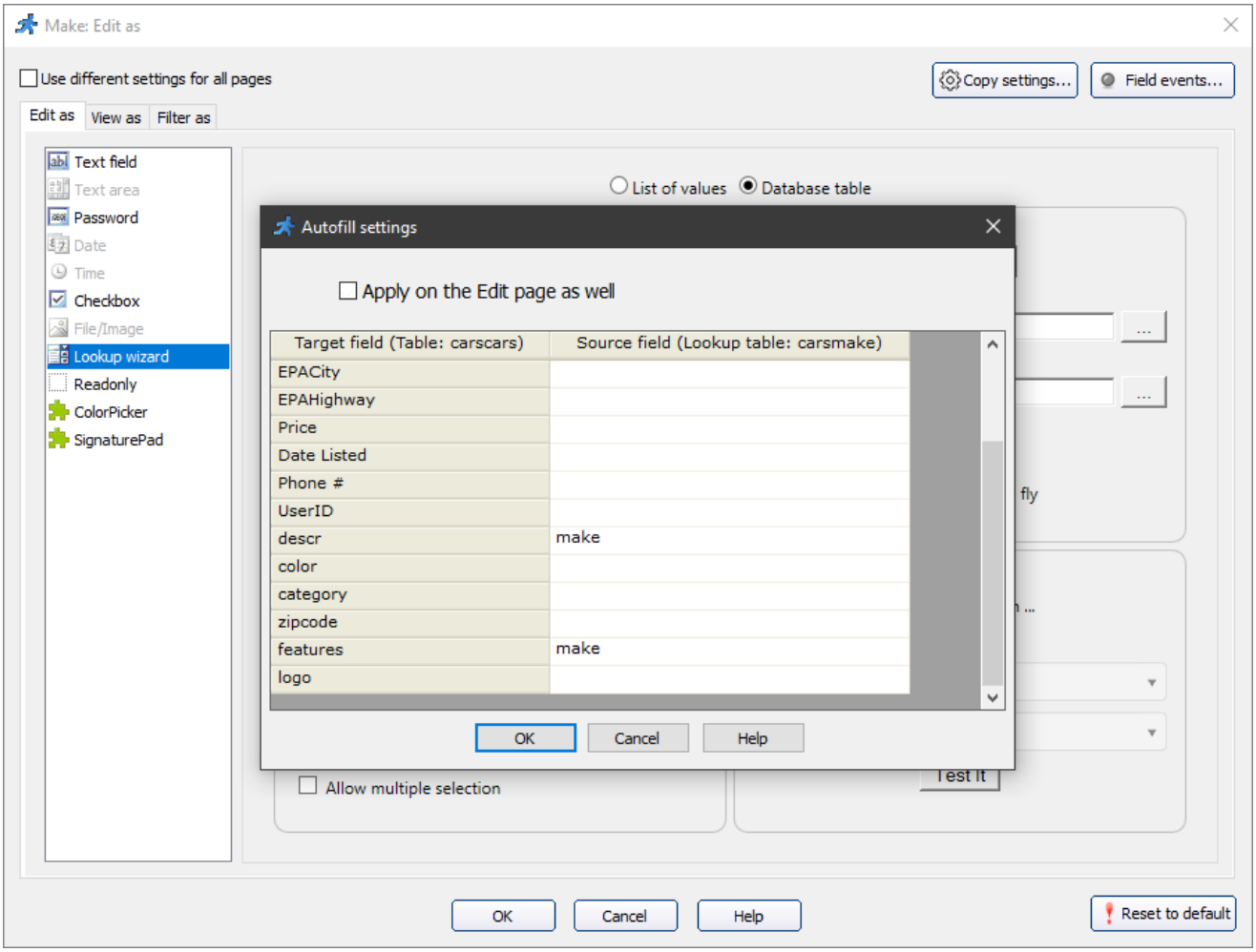
```
$_SESSION["Cars_masterkey1"]
$_SESSION["_Cars_OwnerID"]
```

You can use SQL variables in a WHERE expression:

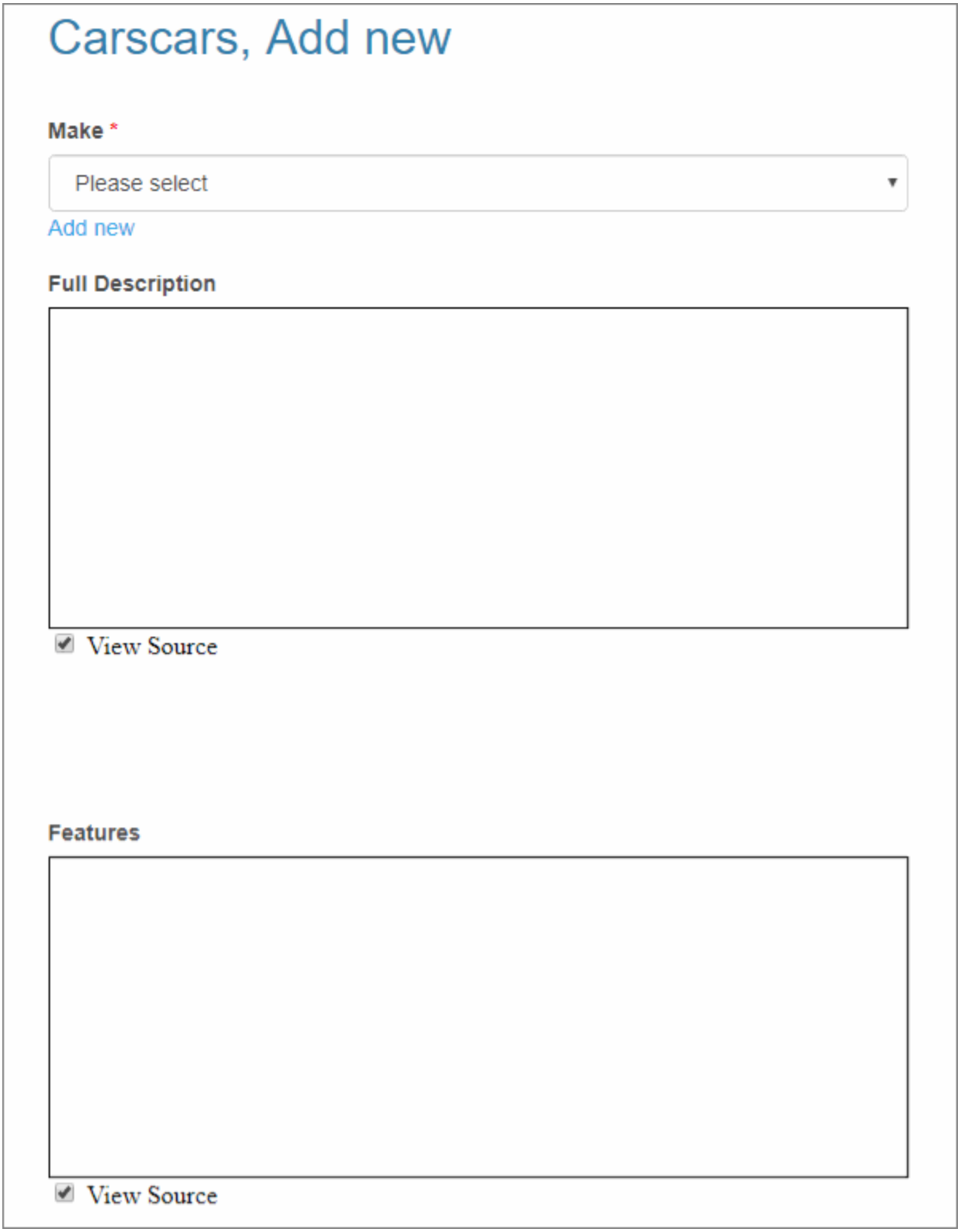
```
CustomerID= ':user.CustomerID'
CustomerID= ':session.UserID'
```

Autofill

You can autofill several fields on the **Add**/**Edit** pages with values from the lookup table. Let's say, that you want to autofill the *Description* and *Features* fields with the *Make* value from the *carsmake* table on the **Add** page. Click **Autofill** and select the corresponding source fields of the lookup table.

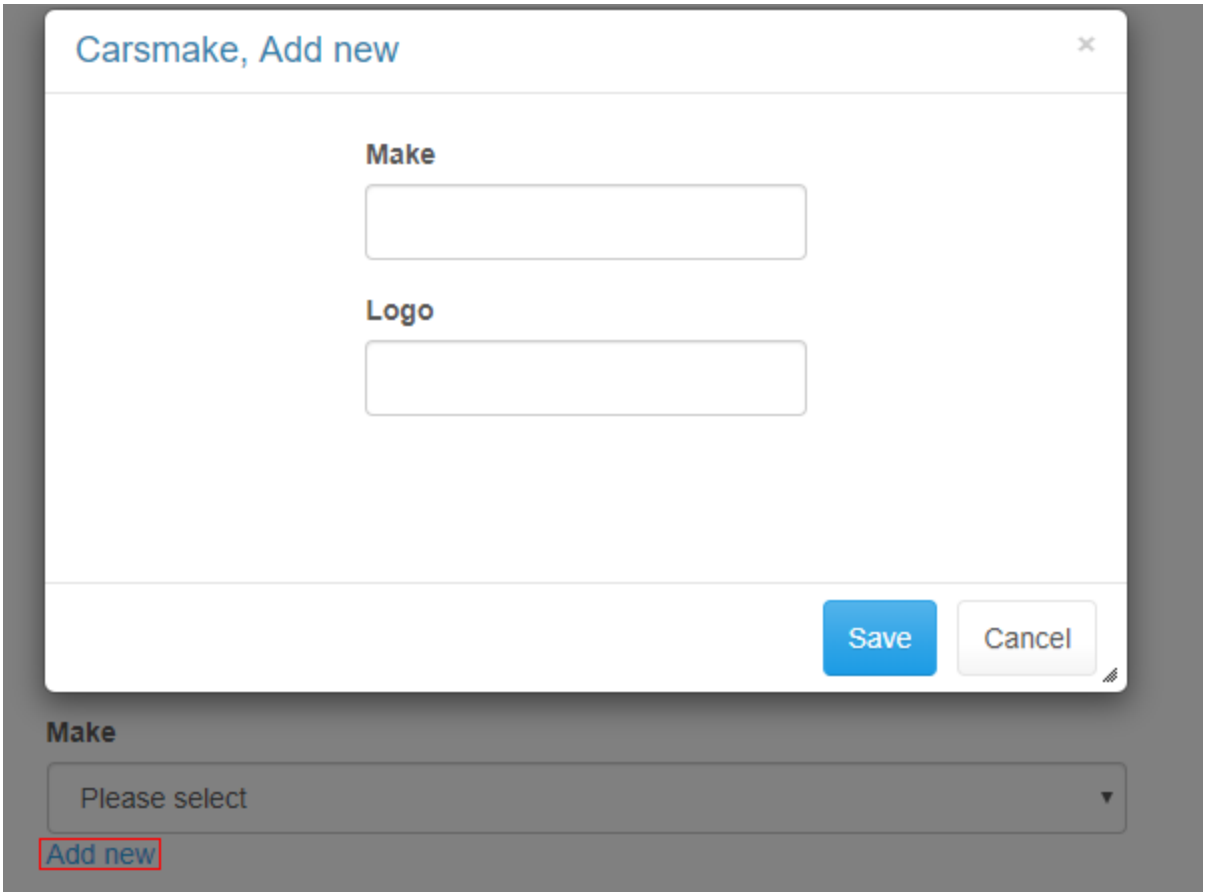


When you select the make of a car, it is also automatically added to the *Description* and *Features* fields:



Allow to add new items on the fly

This option puts "**Add new**" link next to the list of the values, allowing to add new items right on **Edit**/**Add** page. *Add new* item popup is a fully-featured **Add** page. You can specify which **Add** page to open in the popup with a dropdown under the checkbox.



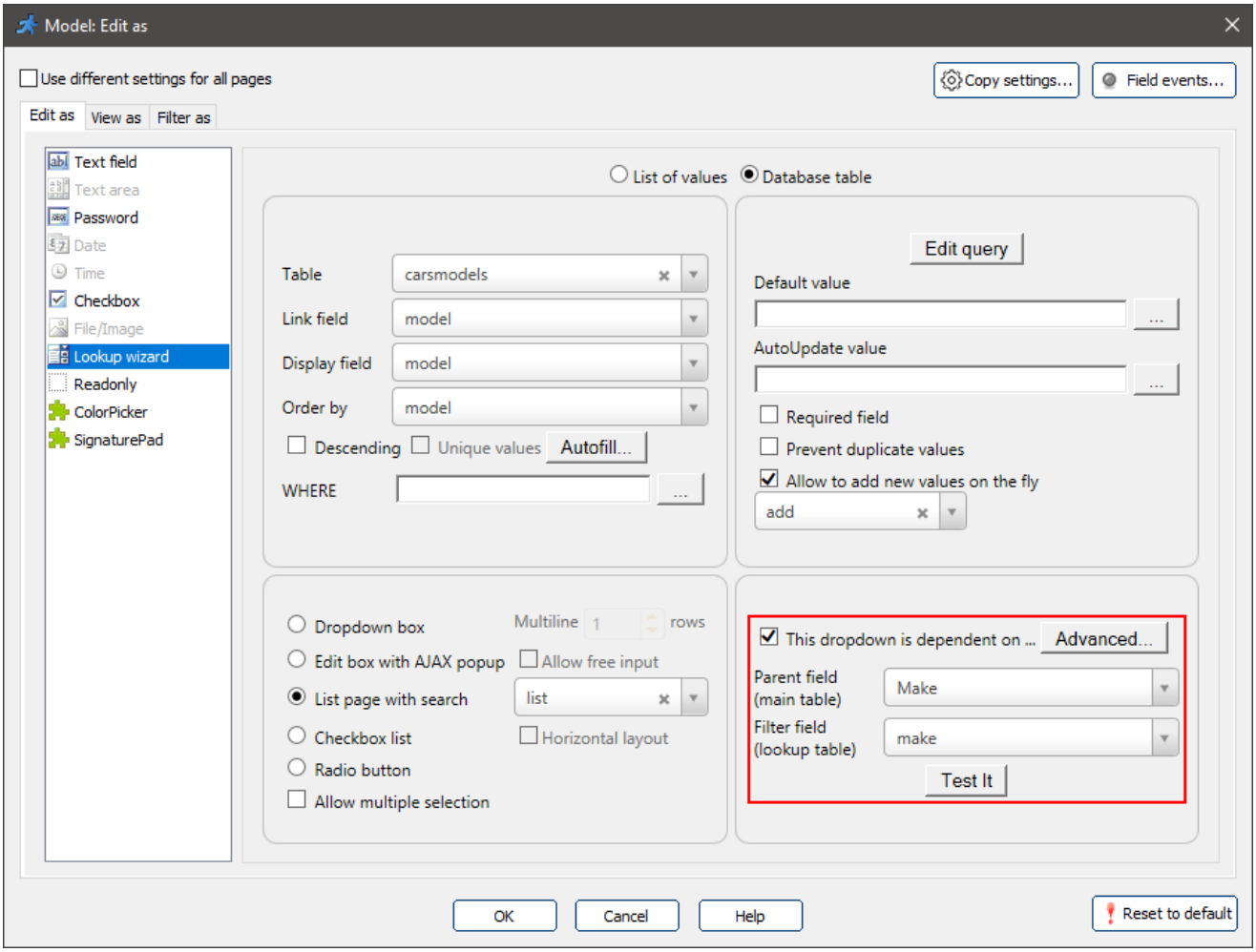
Dependent value lists

You can use dependent value lists, where the values shown in the second list depend on the value you've chosen in the first one.

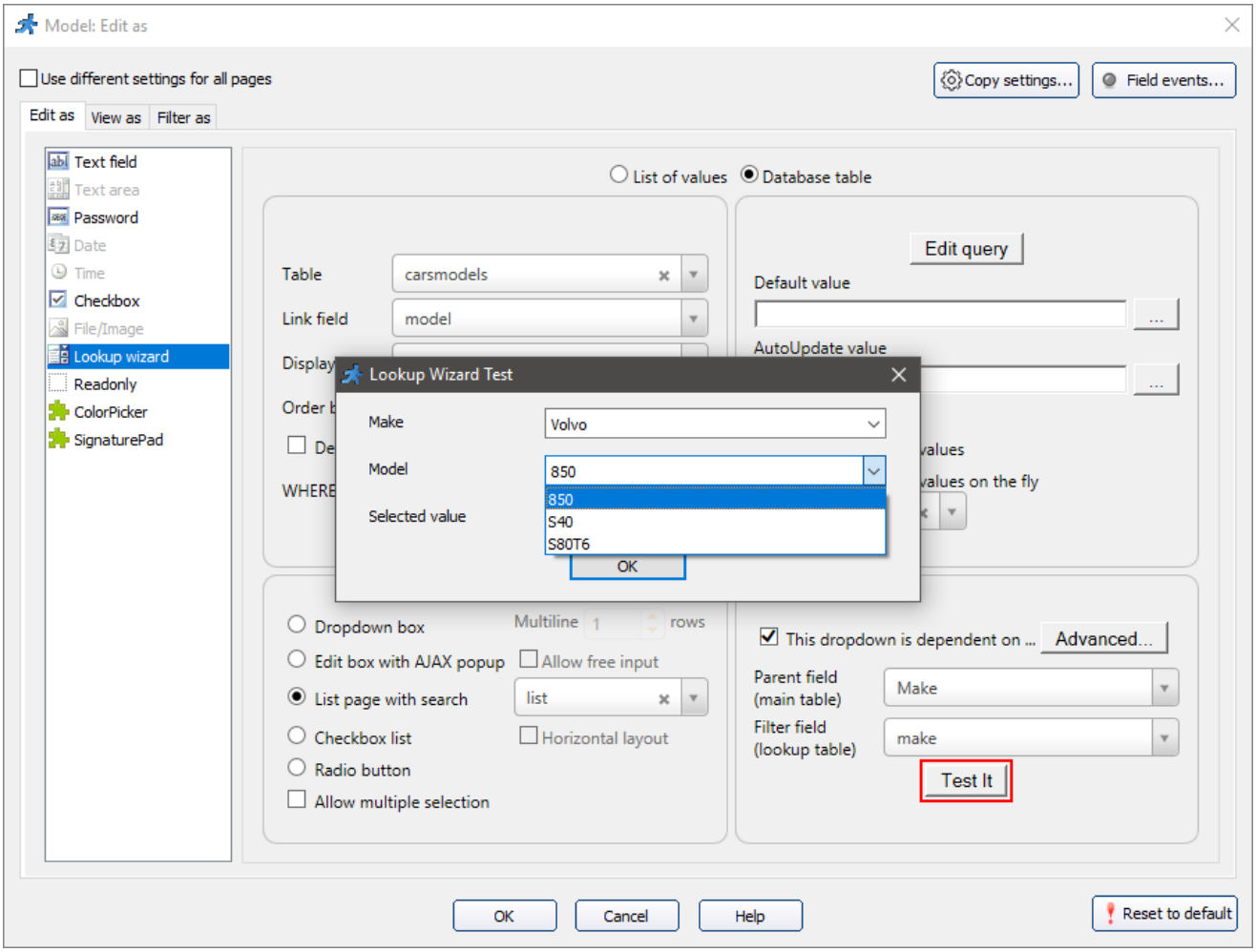
Let's make the *Model* field content depend on the *Make* field value:

1. Set **Lookup wizard** as the "Edit as" type for the *Make* and *Model* fields.

2. For the *Model* field, select **This dropdown is dependent on** checkbox. Select *Make* as the parent field from the main table and the filter field from the lookup table.



3. Click **Test it** to check how it works.



An example of an **Edit** page with a dependent dropdown list:

Cascading value lists

You can also create a chain of dependent value lists, where one list depends on two or more master controls.

Let's say, for example, that you store the year the models were produced in the *Year* field of the *carsmodels* table.

You can make the *Model* field content of the *carscars* table depend on the values of *Make* and *YearOfMake* fields.

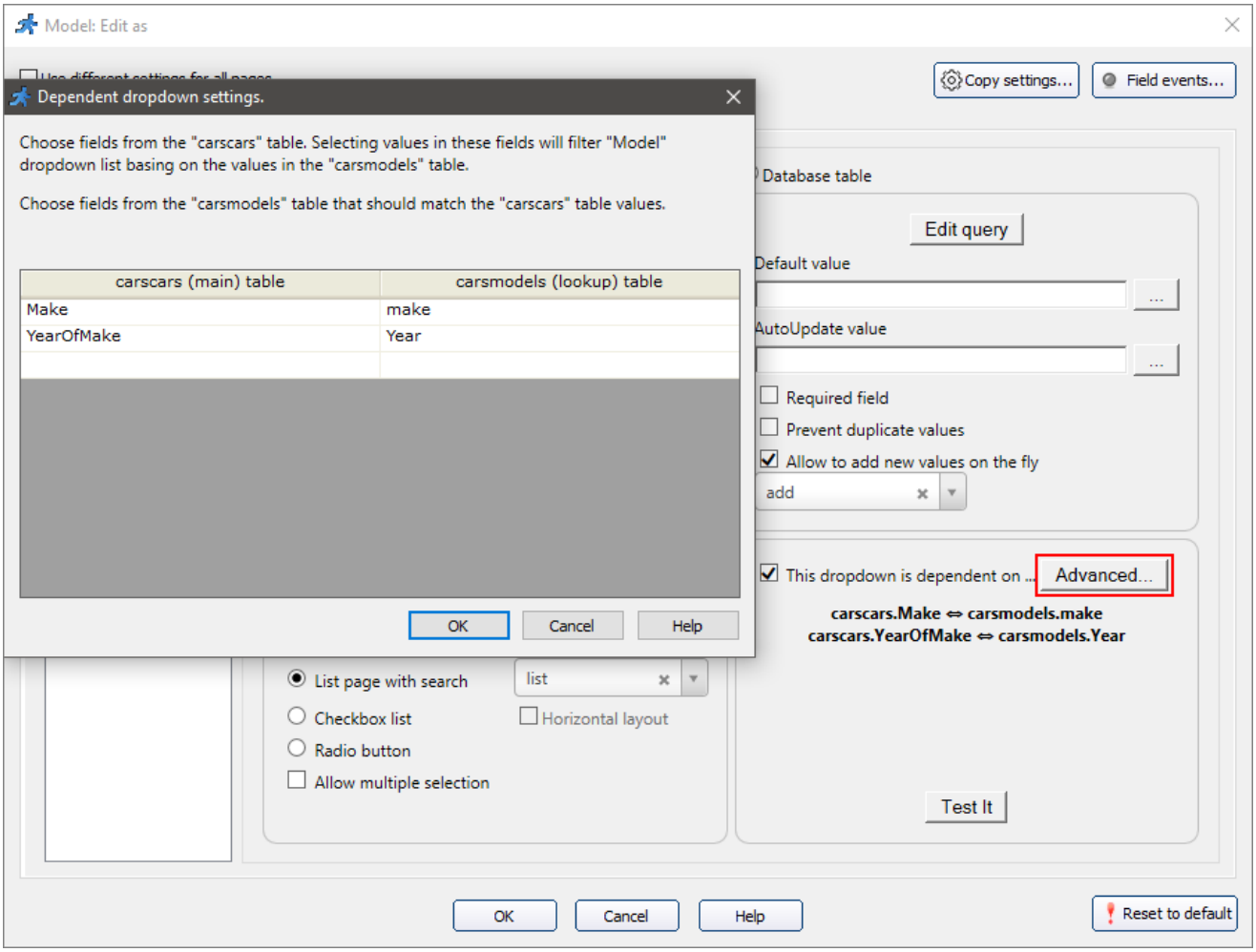
That way, you can select only those models that were produced by the selected company in the selected year. To perform this:

1. Set **Lookup wizard** as the "Edit as" type for the *Make*, *YearOfMake*, and *Model* fields.

2. Make the *YearOfMake* depend on *Make* field.

3. For the *Model* field, select **This dropdown is dependent on** checkbox and click **Advanced**.

Then set up both *Make* and *YearOfMake* fields and click **OK**.



4. Click **Test it** to check how it works.

5. The resulting list may look like this:

Note: more than one dependent list can be tied to the same master control.

The appearance of the Lookup wizard

Dropdown box

This option makes the list of values display as a dropdown box. If you set the **Multiline rows** option to any value greater then one, this field will appear as a listbox on **Add**/**Edit** pages.

An example of a multiple selection dropdown box:

Edit box with AJAX popup

This option makes the field show only the values matching (or containing) the string, that is typed into the selected field.

An example of a multiple selection AJAX popup:

By default, the application looks for the occurrence of the typed in string anywhere within the values of the list.

For example, when you type '*co*', it shows both *'Corolla'* and *'Accord'*.

If you want to change this behavior and make it look only for the values starting with the entered value, i.e. *'Corolla'* only, add the following code to the AfterAppInit event:

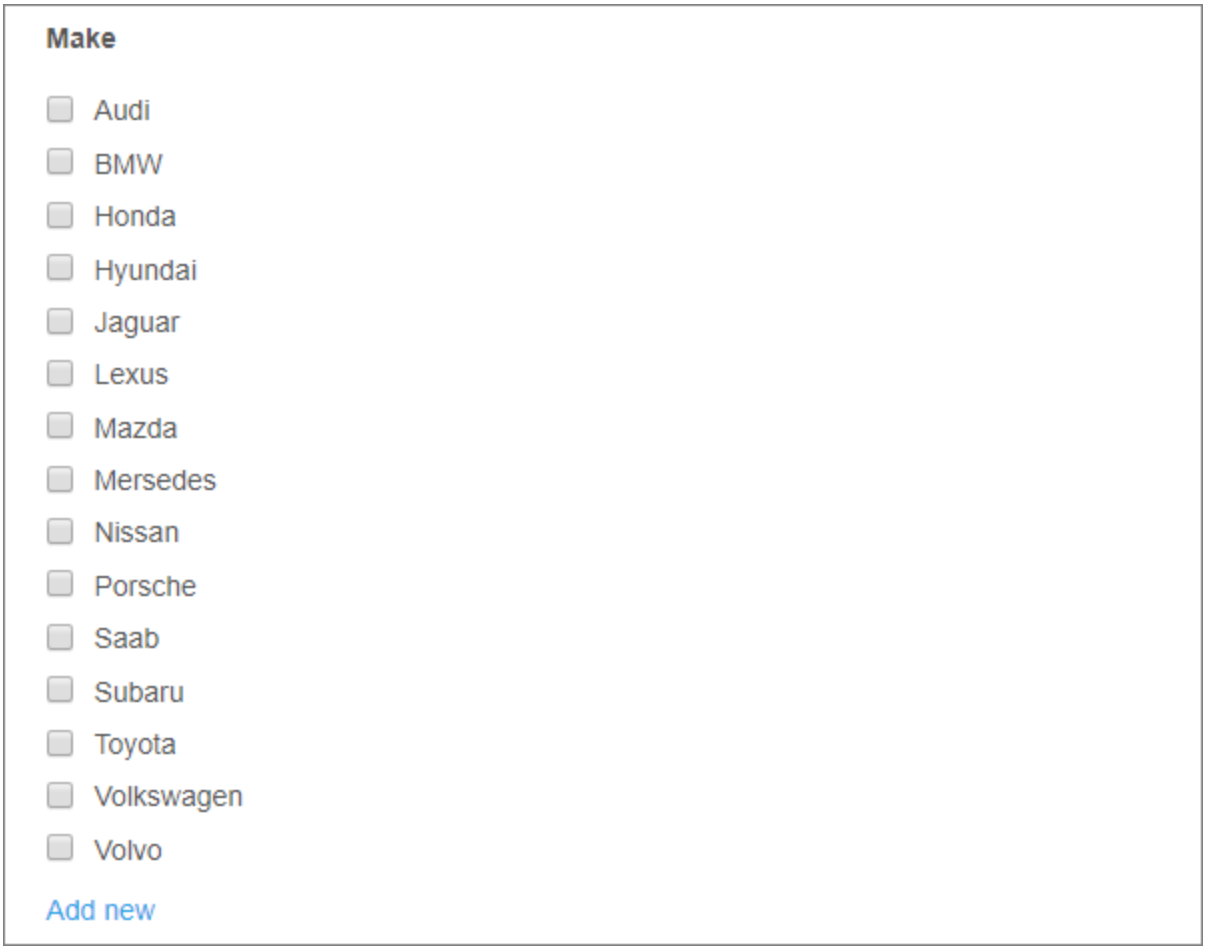
```
$ajaxSearchStartsWith = true;
```

Checkbox/Radio button list

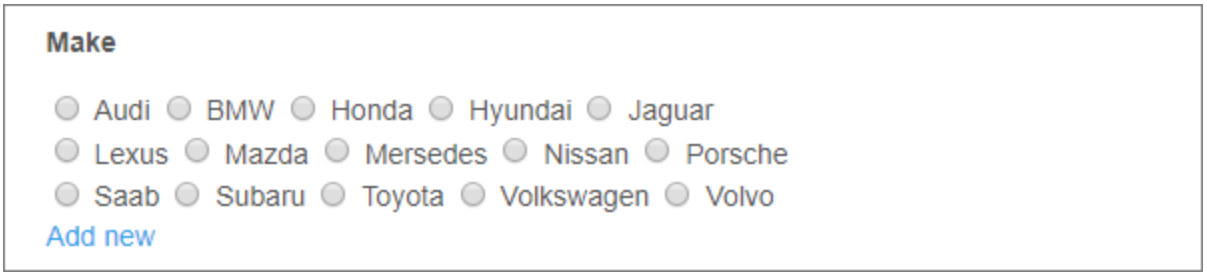
This option makes the list of values display as a set of checkboxes or radio buttons.

Users can select one or several values with the **Checkbox list**, and only one value - with the **Radio buttons list**. You can also select a **Horizontal layout** checkbox to display the elements horizontally.

A vertical checkbox list example:



A horizontal radio button list example:

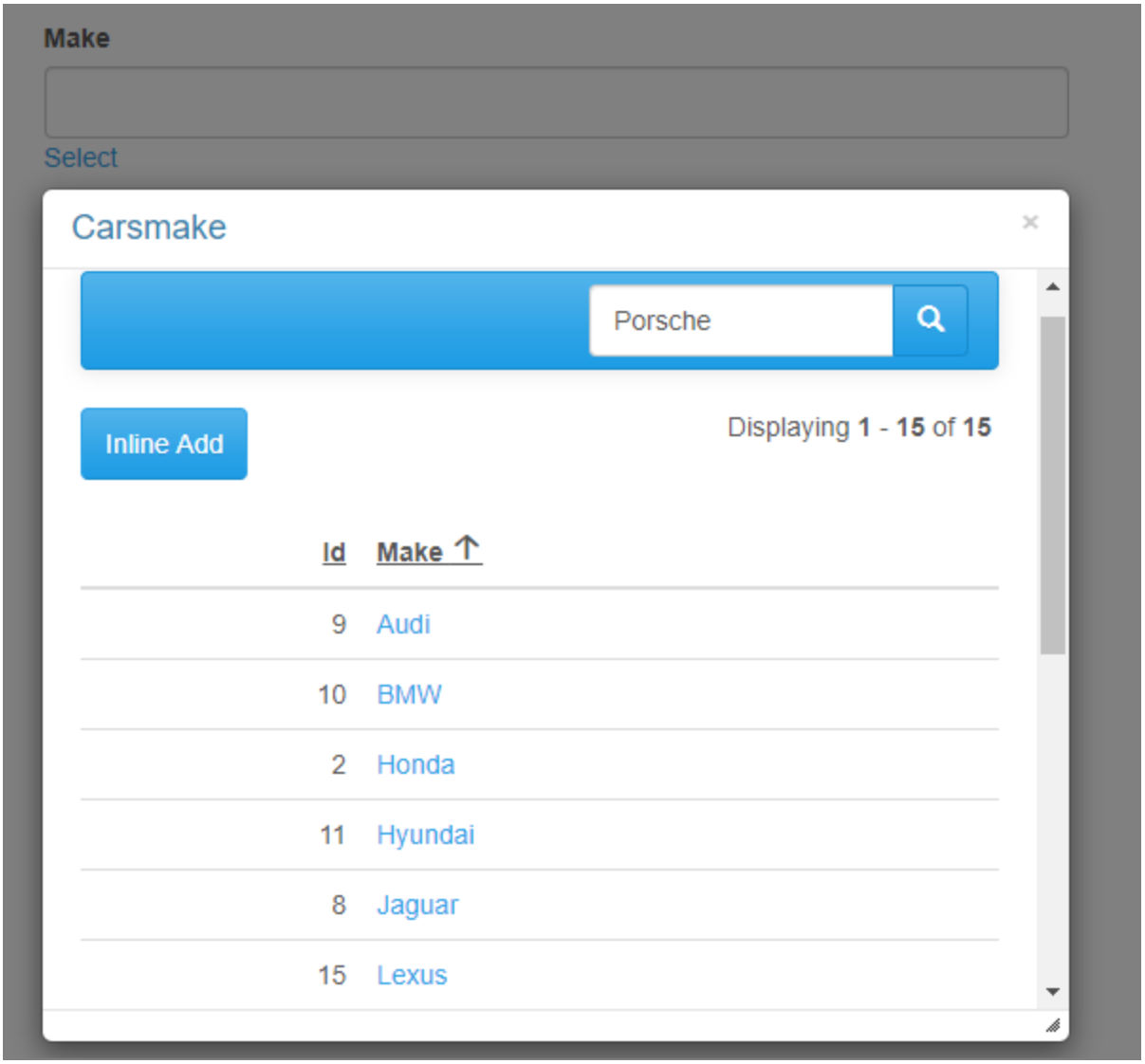


List page with search

This option displays the **Select** button under the field. When you click on the field or the **Select** button, a popup window appears with a searchable **List** page of the lookup table.

You can specify which **List** page to open in the popup with a dropdown next to the **List page with search** option.

A list page with search example:



Multiple selection

Select the **Allow multiple selection** checkbox to allow users to select multiple values.

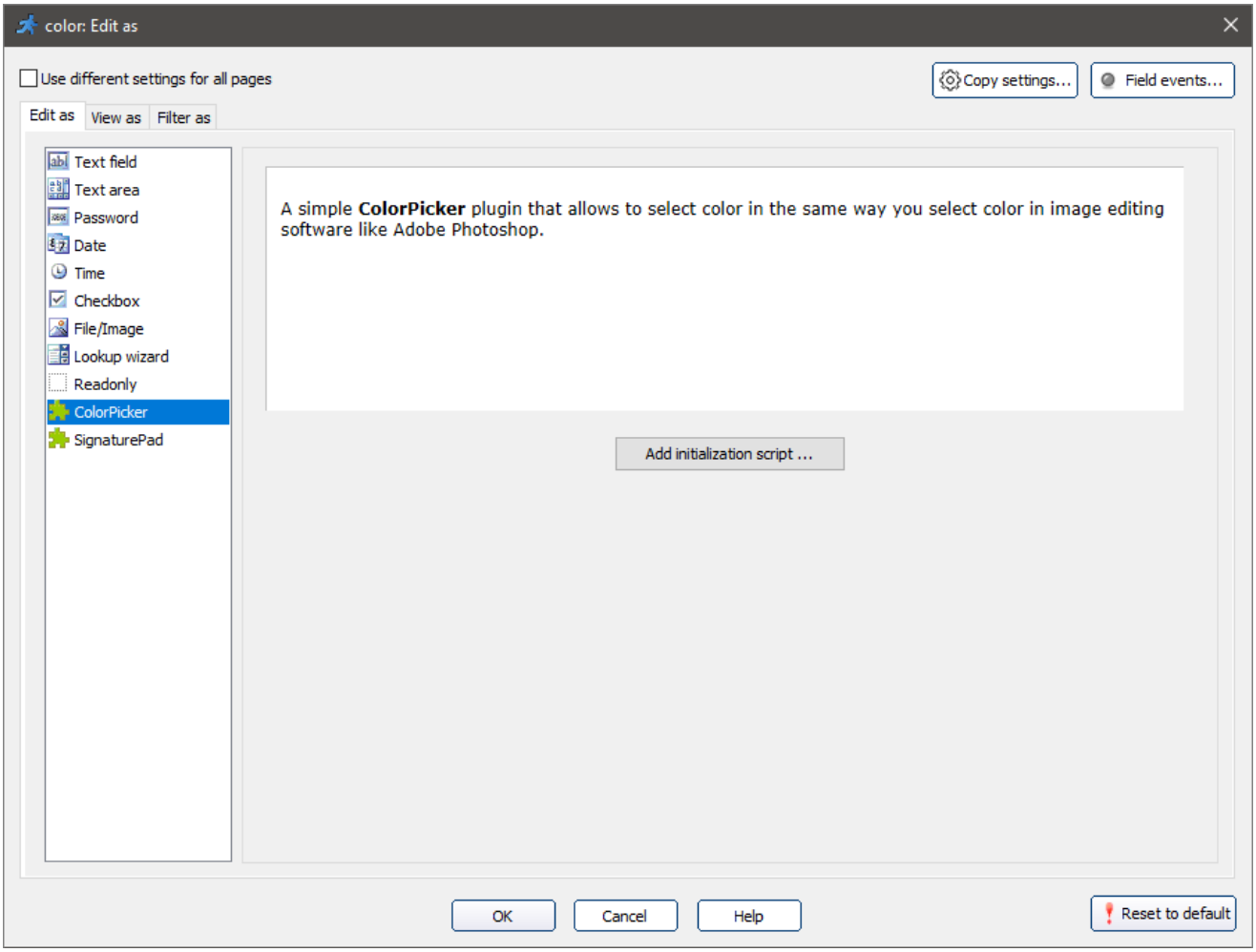
This option applies to every type of the **Lookup wizard** appearance.

See also:

- Examples of SQL variables
- AJAX-based features
- "Edit as" settings
- About Page Designer
- About Editor

2.16.15.8 ColorPicker

The **ColorPicker** control allows users to select a color the same way they do in Adobe Photoshop. You may select the **ColorPicker** as an **Edit as** type for any text field on the **Add**/**Edit** pages.



A **ColorPicker** example on the **Add** page:

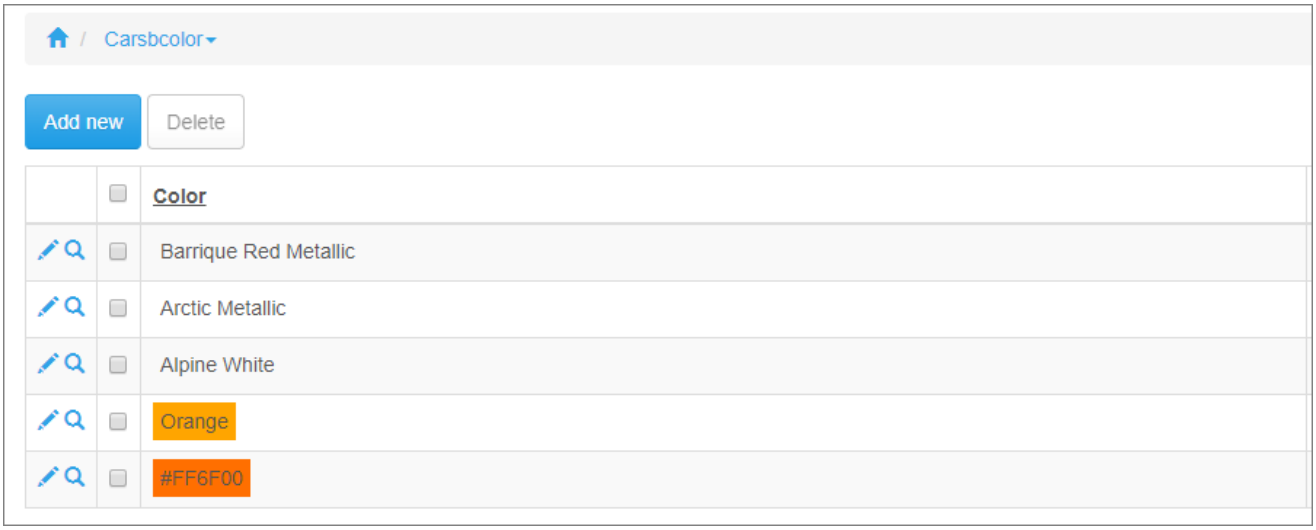
Click the **Add initialization script** button to customize the **ColorPicker** control.

You may consider some additional enhancements:

1. You can show the visual representation of the selected color on the **List**/**View** pages. Choose Custom as the **View as** type and insert the following code there to do so:

```
$value="<span style='padding: 5px; background: ".$value."'>".$value."</span>";
```

As a result, the **List** page looks like this:



2. By default, PHPRunner sets focus on the first edit control when an **Add** or **Edit** page is loaded.

This may not be a desired behavior for the **ColorPicker** control, as the popup window will open every time the page is loaded.

To prevent this from happening, implement the setFocus function, and return "*false*" every time.

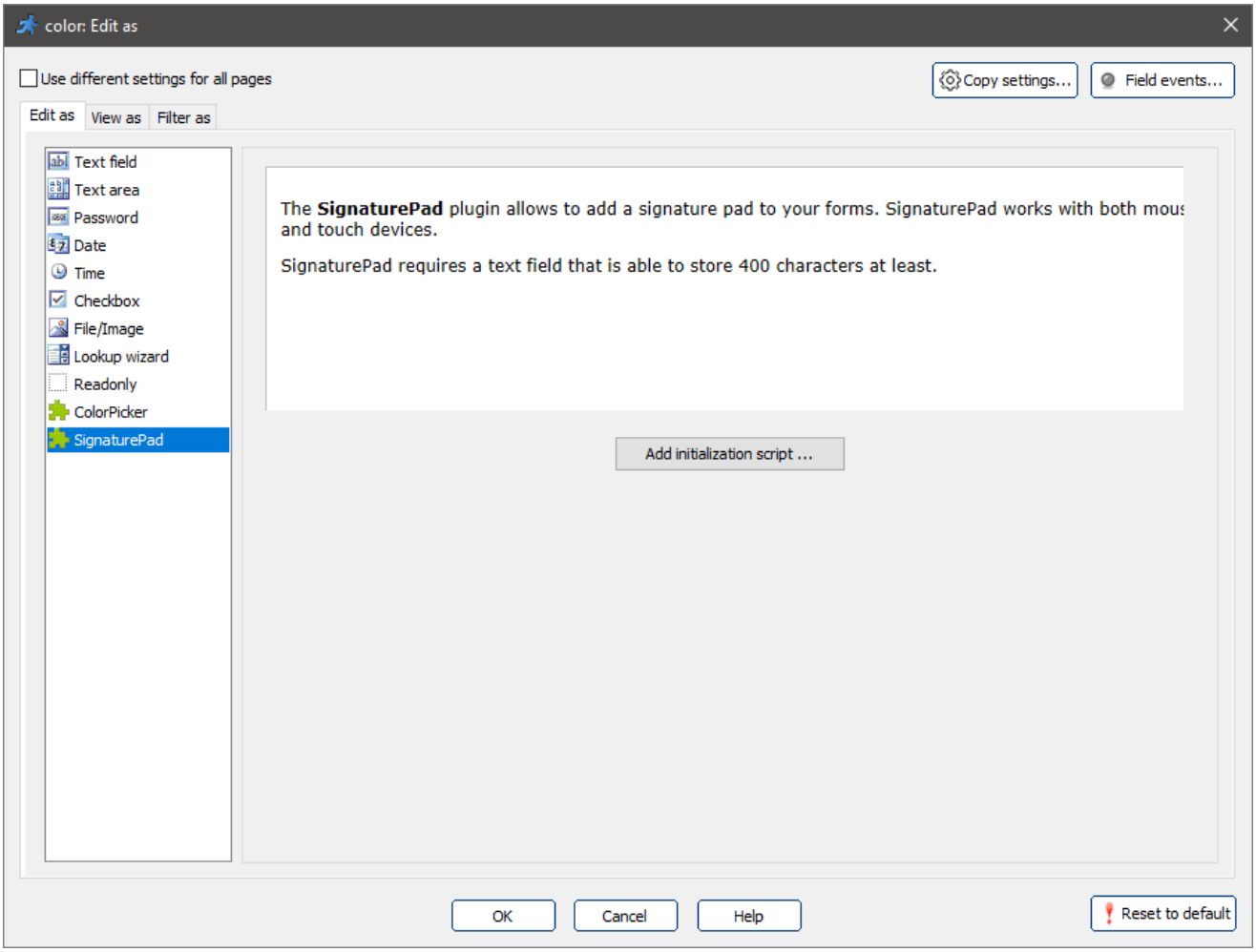
> **Note**: **ColorPicker** is a custom plugin. You can create custom Edit control plugins to use in PHPRunner.
>
> For more information, see How to create a custom Edit control.

See also:

- "Edit as" settings
- About Page Designer
- About Editor

2.16.15.9 SignaturePad

The **SignaturePad** control allows you to add a signature pad to the page. **SignaturePad** works with mouse and touch controls.



Select **SignaturePad** as the **Edit as** type for any text field with a length of 200 characters or more on the **Add**/**Edit** pages.

Select Image as the **View as** type for the selected text field to see the signatures on the **View**/**List** pages.

A **SignaturePad** example on the **Add** page:

Click the **Add initialization script** button to customize the **SignaturePad** control.

Here is the list of its settings:

```
// signature field height
$this->settings["height"] = 100;
// signature field width
$this->settings["width"] = 300;
// signature background color
$this->settings["bgcolor"] = "#ffffff";
// set it to true to make signature field required
$this->settings["required"]=false;
// folder to store signature files
$this->settings["folder"]="files";
// signature background image
// $this->settings["bgimage"] = "http://website.com/images/background.png";
$this->settings["bgimage"] = "";
// signature pen color
$this->settings["color"] = "#145394";
// signature line width
$this->settings["linewidth"] = 2;
```

> **Note**: **SignaturePad** is a custom plugin. You can create custom Edit control plugins to use in PHPRunner.
>
> For more information, see How to create a custom Edit control.

See also:

- "Edit as" settings
- About Page Designer
- About Editor

2.16.15.1 Validation types

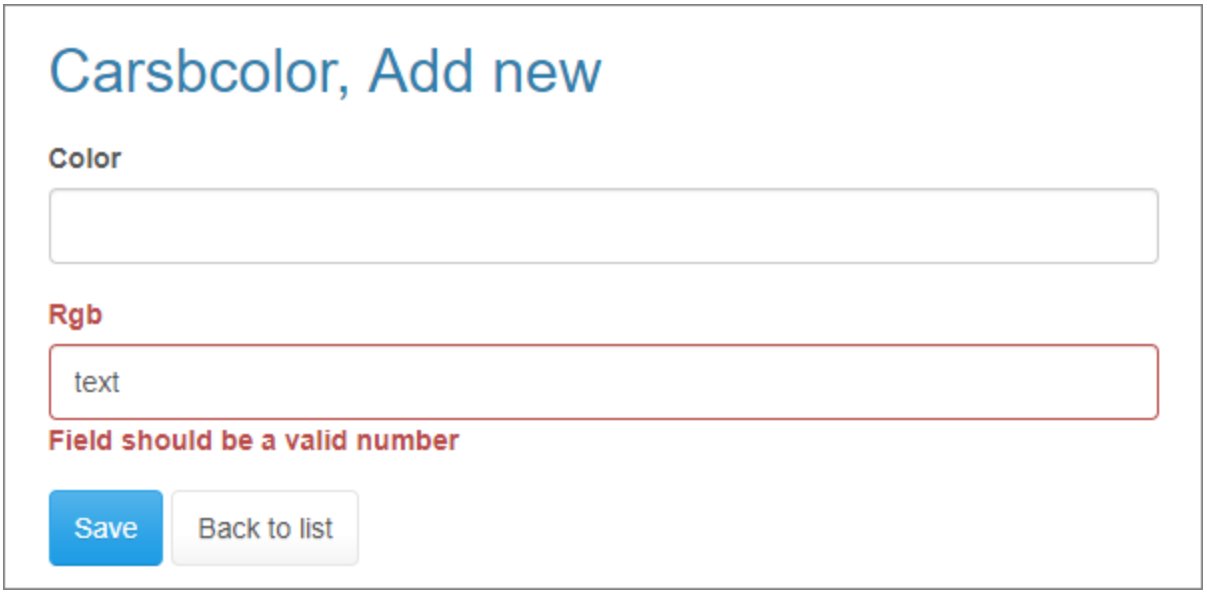
Quick jump

Validate As

The **Validate As** option allows you to validate the data that users enter on the **Add**/**Edit** pages. You can use standard validation types, create a regular expression, or even a custom validation plugin.

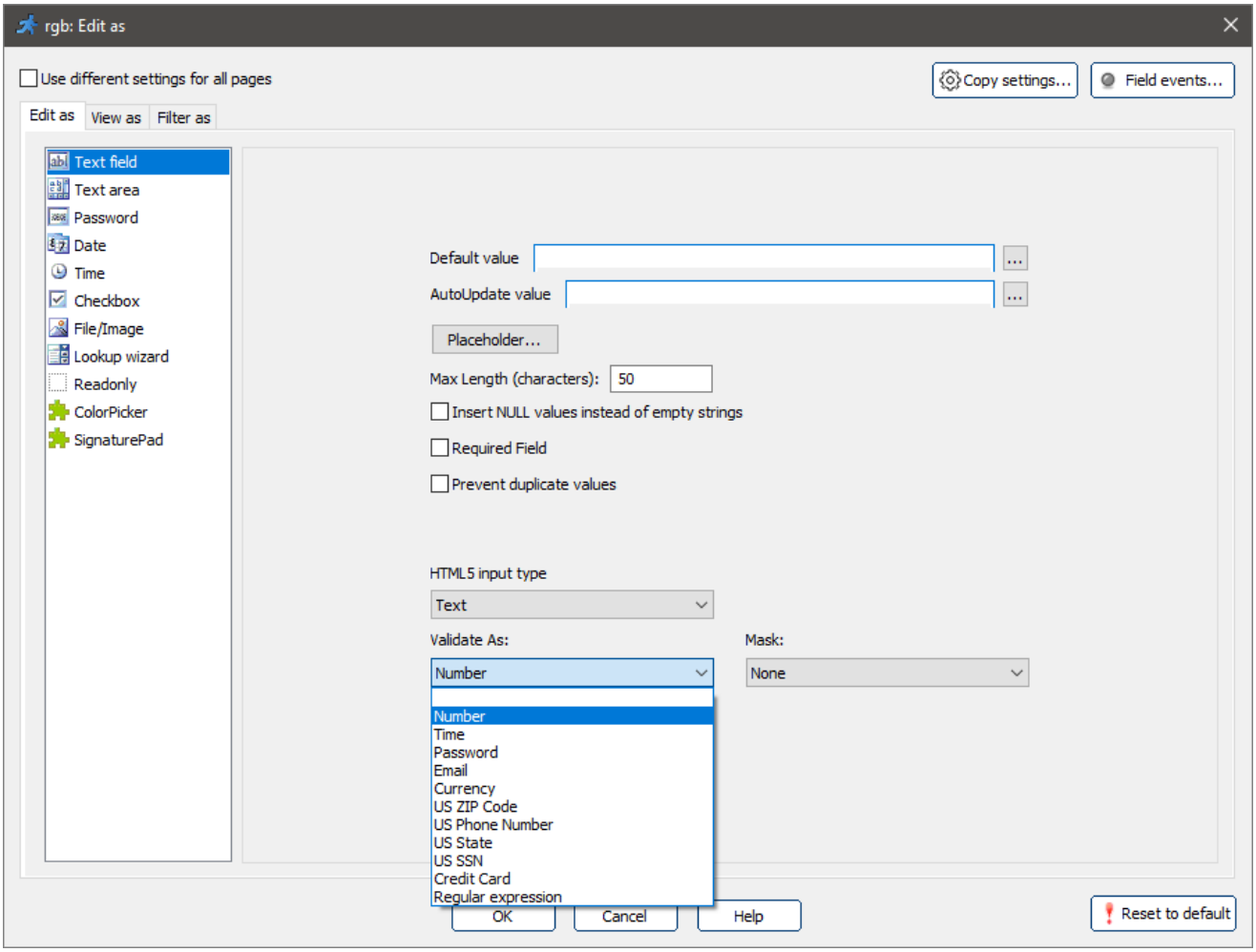
When the field value doesn't match the defined validation format, a message appears under the field with a predefined or a custom text.

Here is an example of a predefined message for a "*Number*" validation type:



To edit the validation rules, proceed to **Page Designer**, select a field and click the **View As/Edit As** button.

Choose one of the validation types from the list:



The **Validate As** dropdown is available for the **Text field**, **Password** and **Time** formats on the **Edit as** tab.

A list of standard validation types

- *Number* - the field value should be a number.
- *Time* - any valid time format that matches regional settings.
- *Password* - the password field cannot be blank, cannot be "Password" and should be at least 4 characters long.
- *Email* - the field value should be a valid email address.
- *Currency* - a numeric value. Decimal point is allowed. Examples: 24.95, 13.
- *US Zip Code* - a five or ten-digit number. Valid formats: 12345, 12345-6789, or 123456789.

- *US Phone Number* - numbers, spaces, hyphens, and parentheses are allowed. Examples: (123) 456-7890, 123 456 7890, 123 4567.
- *US State* - the field accepts a two letter US state abbreviation. Examples: AK, AL, CA, MN.
- *US SSN* - a nine-digit US social security number. Valid formats: 123-45-6789 or 123 45 6789.
- *Credit Card* - a valid credit card number.

Regular expression

A **regular expression** (*regexp* for short) is a specific text string for describing a search pattern.

You can get more information about the basic syntax of regular expressions at http://www.regular-expressions.info/reference.html.

Examples of regular expressions:

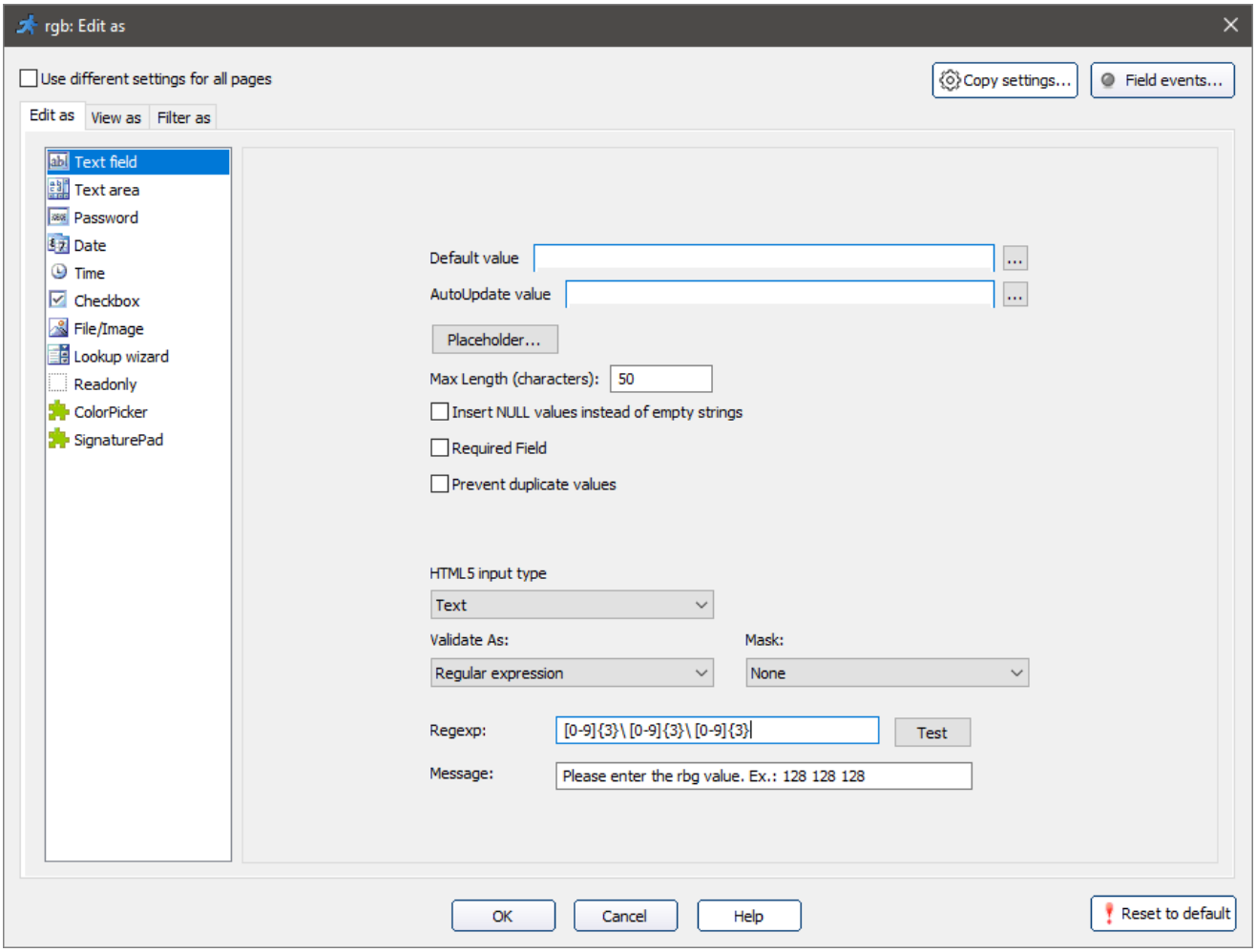
- *[abc]* matches a, b or c;
- *[a-zA-Z0-9]* matches any letter or digit;
- *[^a-d]* matches any character except a, b, c or d;
- *a{3}* matches *aaa*;
- regular expression for an ip address:

  \b(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\b matches 1.2.3.4;
- regular expression for an email address:

  ^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}$ matches contact@abc.com.

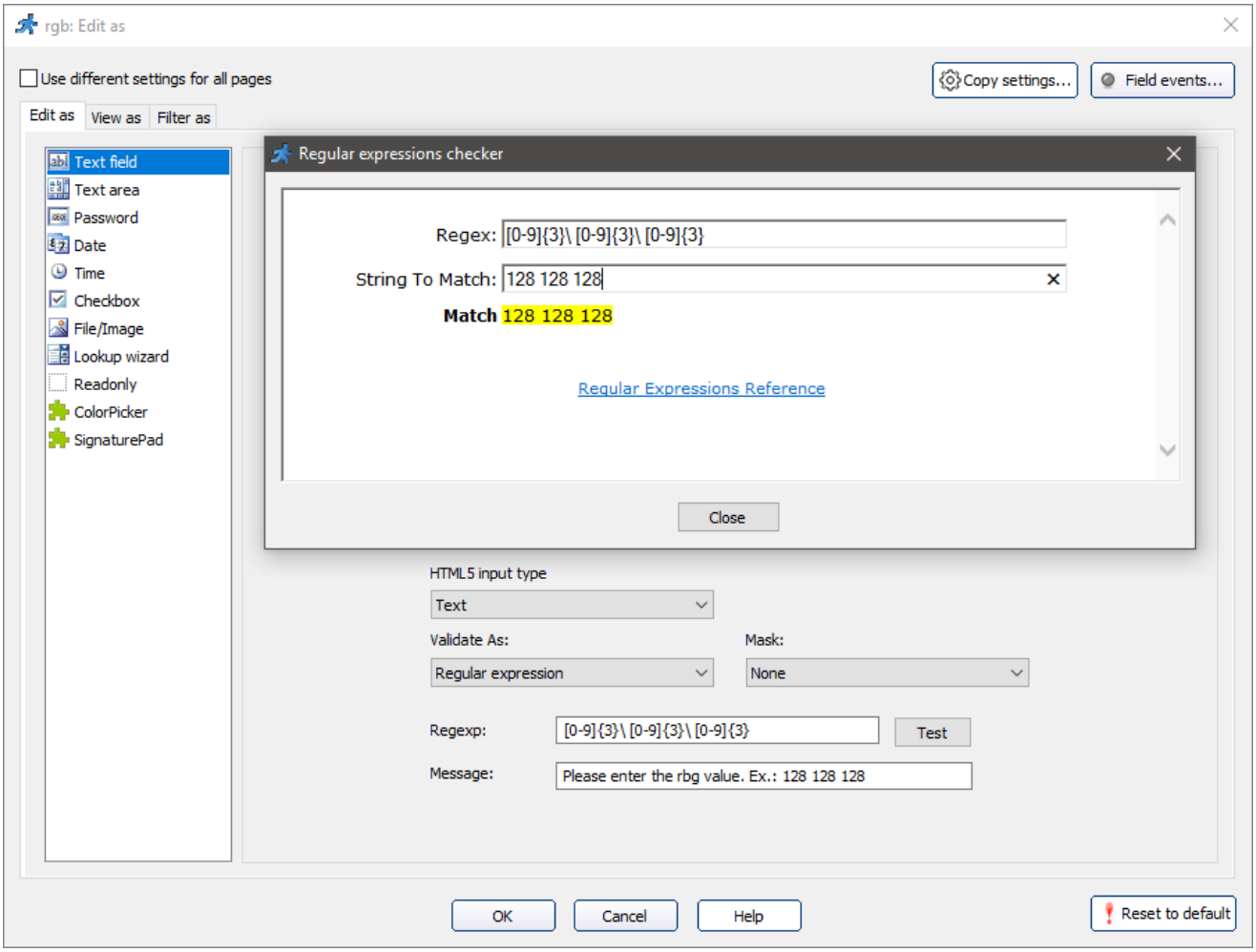
To define a regular expression, select **Regular expression** in the **Validate As** dropdown and type the regular expression you want into the *Regexp* field.

The *Message* field contains the message that appears when the entered value doesn't match the regular expression.



> **Note**: with the multilanguage support turned on, a **Multilanguage** button appears next to the *Message* field, which allows showing the translated message in different languages.
>
> For more information, see Miscellaneous_settings.

You can test the regular expression by clicking the **Test** button.



How to add a custom validation plugin

To add a new validation type, create a file with a JavaScript function that implements the data validation, and store it at the "*<PHPRunner install folder>\source\include\validate*" folder, where <PHPRunner install folder> is the folder PHPRunner is installed to.

You can add any JavaScript code to the validation plugin. The JavaScript function should return nothing if the value was validated successfully, and return an error message if the value was not validated.

> **Note**: The JS file name should match the function name.

As an example, let's create a validation plugin for an IP address.

1. Create a file named "*ip_address.js*".

2. Add the following JavaScript function to this file:

```
function ip_address(sVal)
  {
  var regexp = /\b(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?
[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|[01]?[0-9]
[0-9]?)/;
  if(!sVal.match(regexp))
    return 'Please enter a valid IP address';
  else
    return true;
  }
```

3. Move the file *ip_address.js* to the "*<PHPRunner install folder>\source\include\validate*" folder.

In our case the folder is *C:\Program Files\PHPRunner10.3\source\include\validate*.

4. Start PHPRunner. Now an *ip_address* validation type is available in the *Validate As* dropdown, and you can use it in your projects.

To add a multilingual validation message (i.e., an error message) to the validation plugin, do the following:

1. Create a custom label in the Label editor;

2. Write the translated message into each language field;

3. Use a *Runner.getCustomLabel("LABEL1")* function in the JavaScript file, where "*LABEL1*" is the custom label title.

> **Note**: the *GetCustomLabel function* is applicable only for editing fields with the **Inline add**/**edit**.

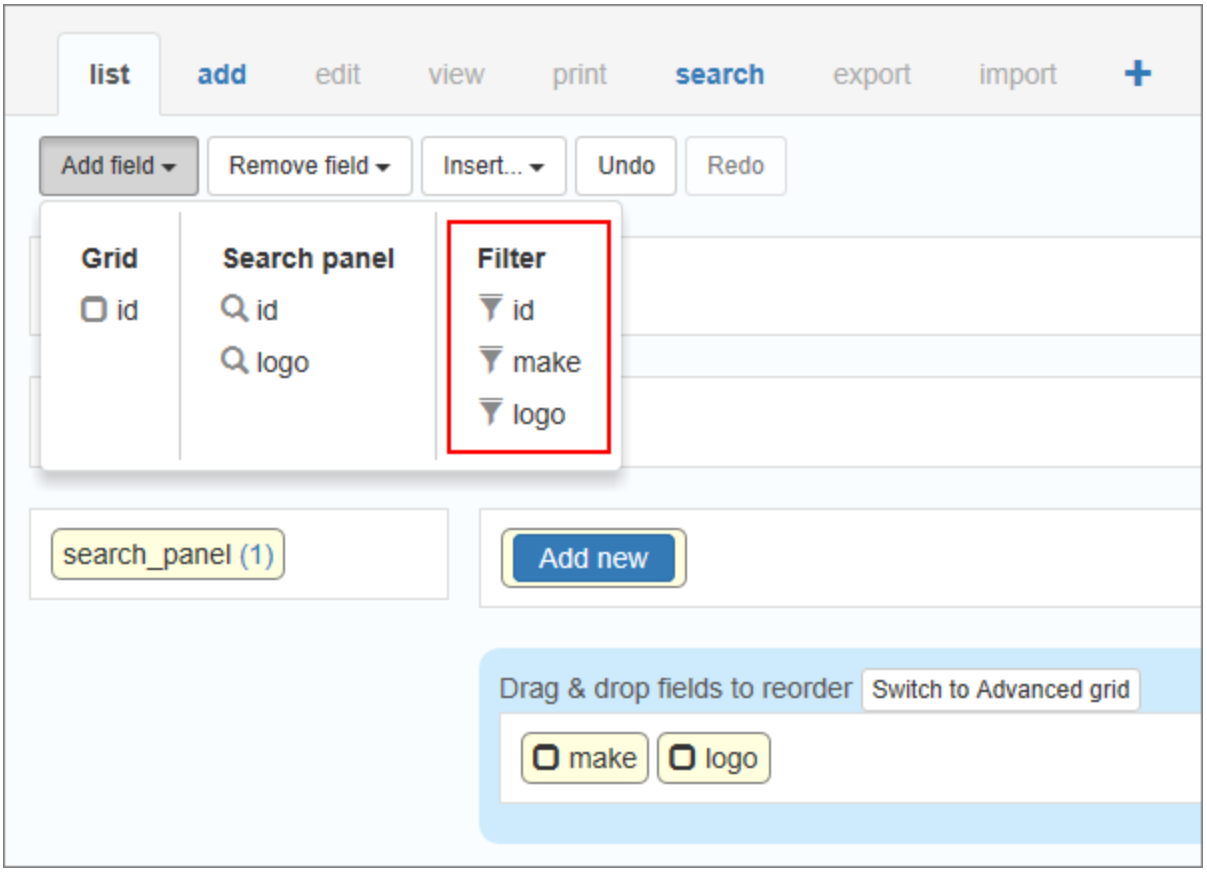
See also:

- JavaScript API: Control object > addValidation()
- Miscellaneous settings: Custom labels
- "Edit as" settings
- About Page Designer
- About Editor

2.16.16 "Filter as" settings

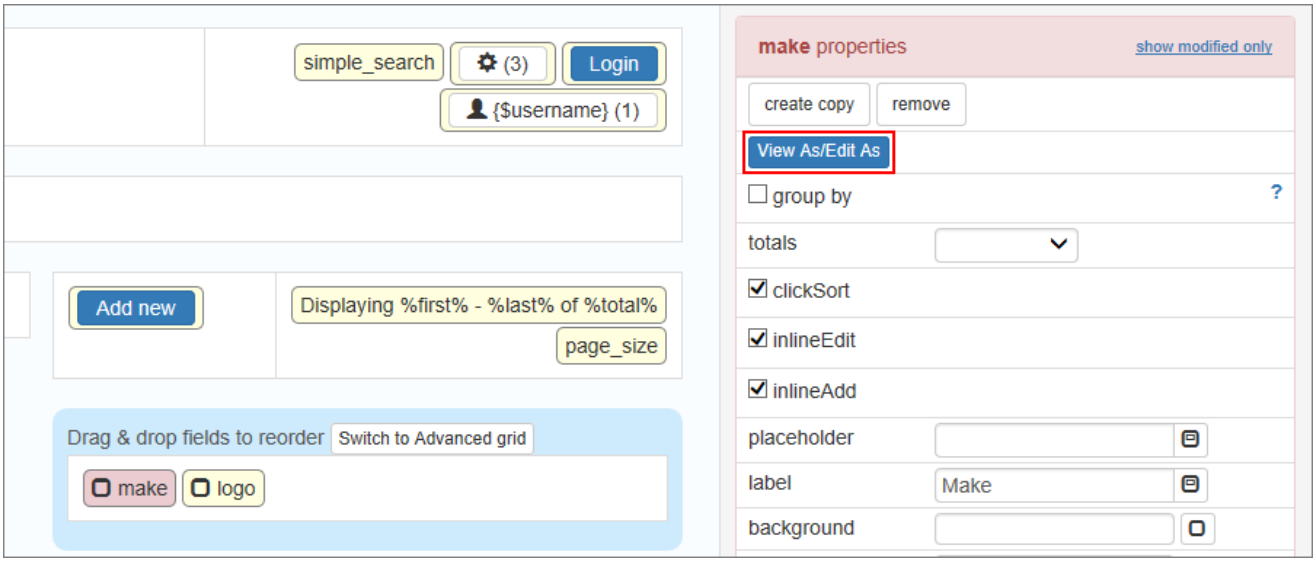
2.16.16.1 "Filter as" settings

You can choose what fields to show on the **Filter panel** by using the **Add**/**Remove** field buttons on the **Page Designer**.

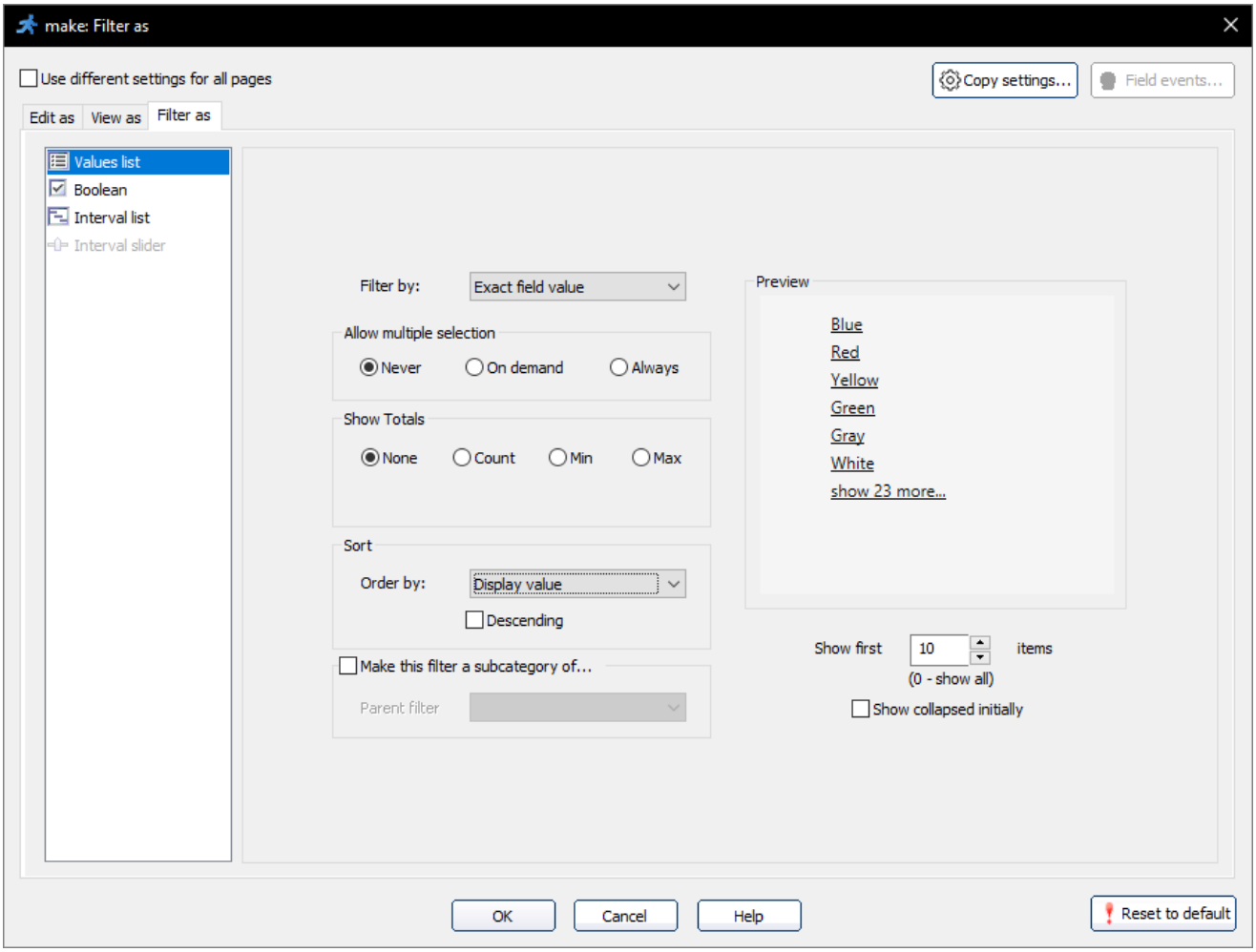


You can customize the data appearance on the **Filter Panel** using formatting options on the **Filter as settings** dialog.

To open this dialog, select the field and click on the **View As**/**Edit As** button in the *properties* panel on the right-side.

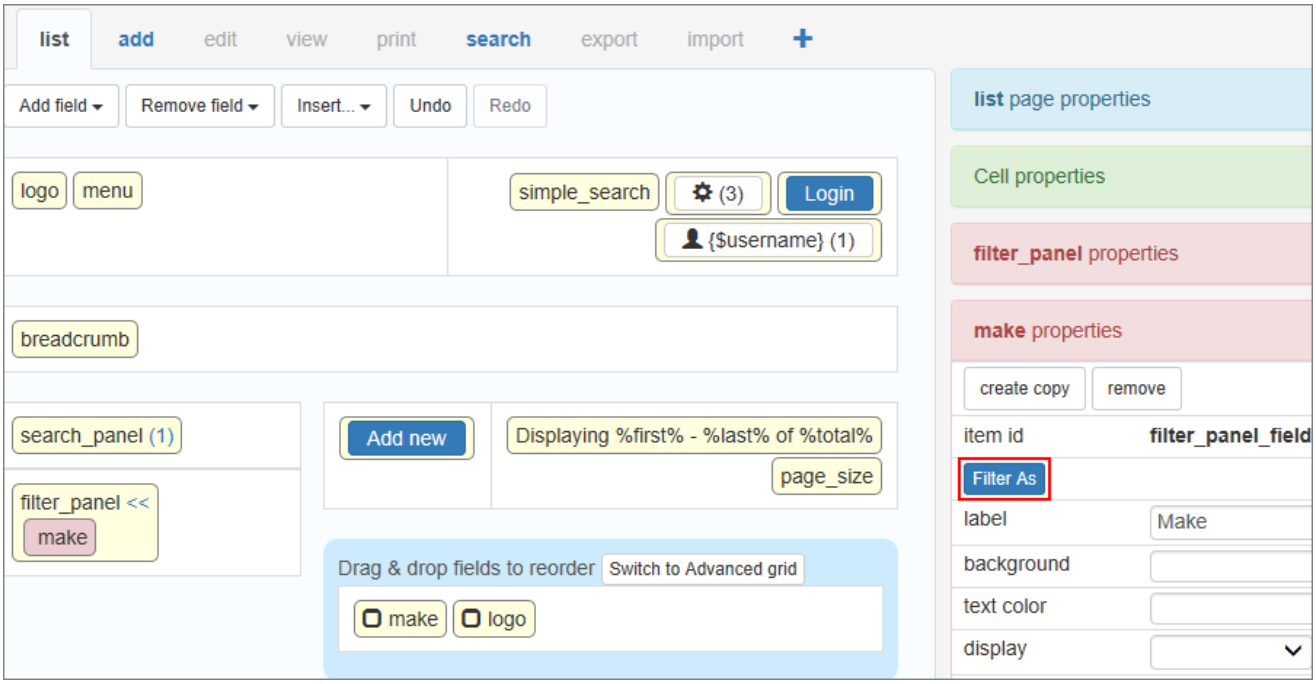


Select the **Filter as** tab, choose the **Filter** format on the left, and set the options to your liking.



Now you can see that the selected field appears inside the **filter_panel** element on the **Page Designer**.

The fields within the **filter_panel** have a **Filter As** button in their properties. You can click the **Filter As** button to open the **Filter as** settings of the selected field.



Another way to enable or disable the **Filter** on your pages is to use Search and Filter settings.

"Filter as" formats

- Values list
- Boolean
- Interval list
- Interval slider

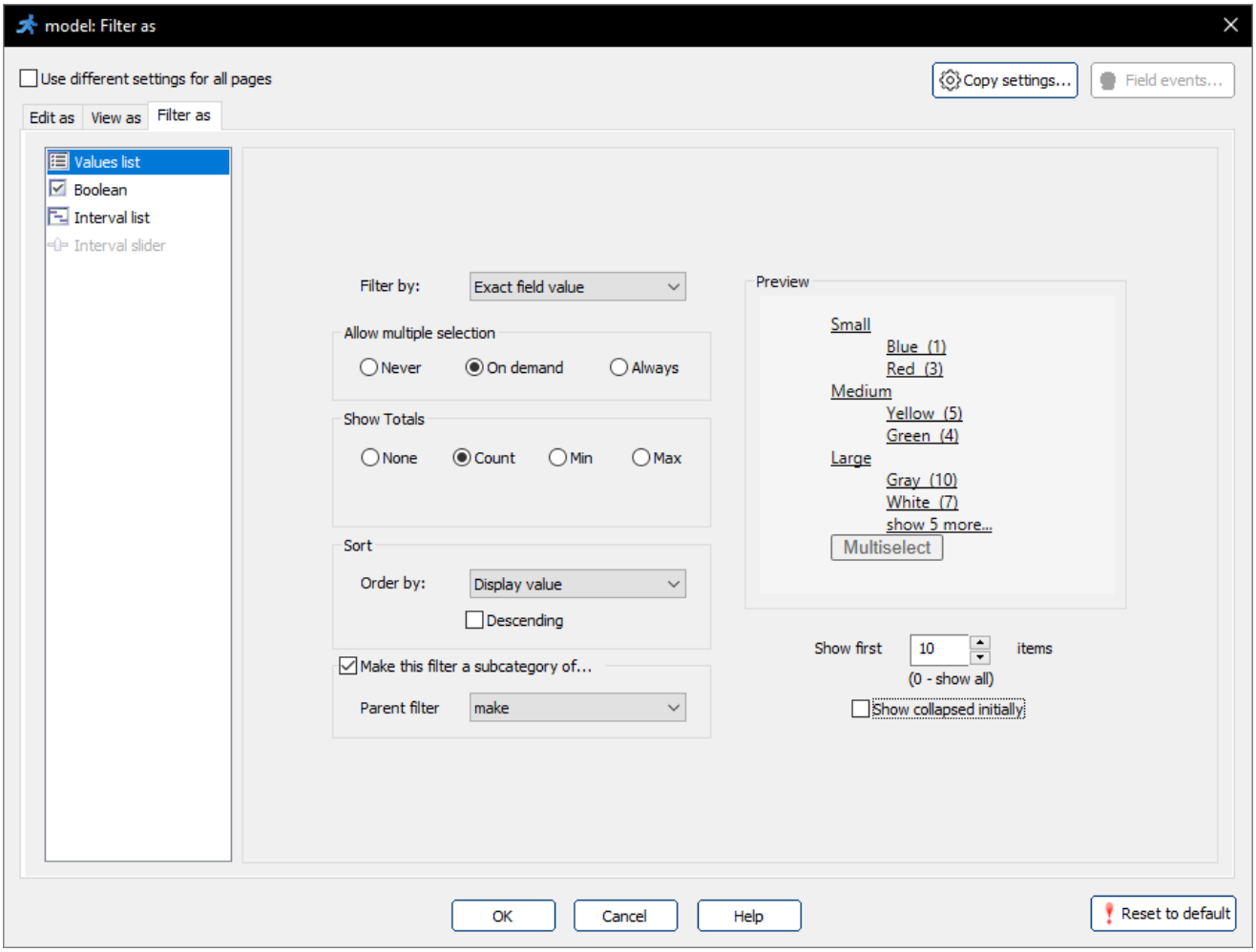
See also:

- Choose fields screen: Search and Filter settings
- About Page Designer
- About Editor

2.16.16.2 Values list

"Values list" filter type

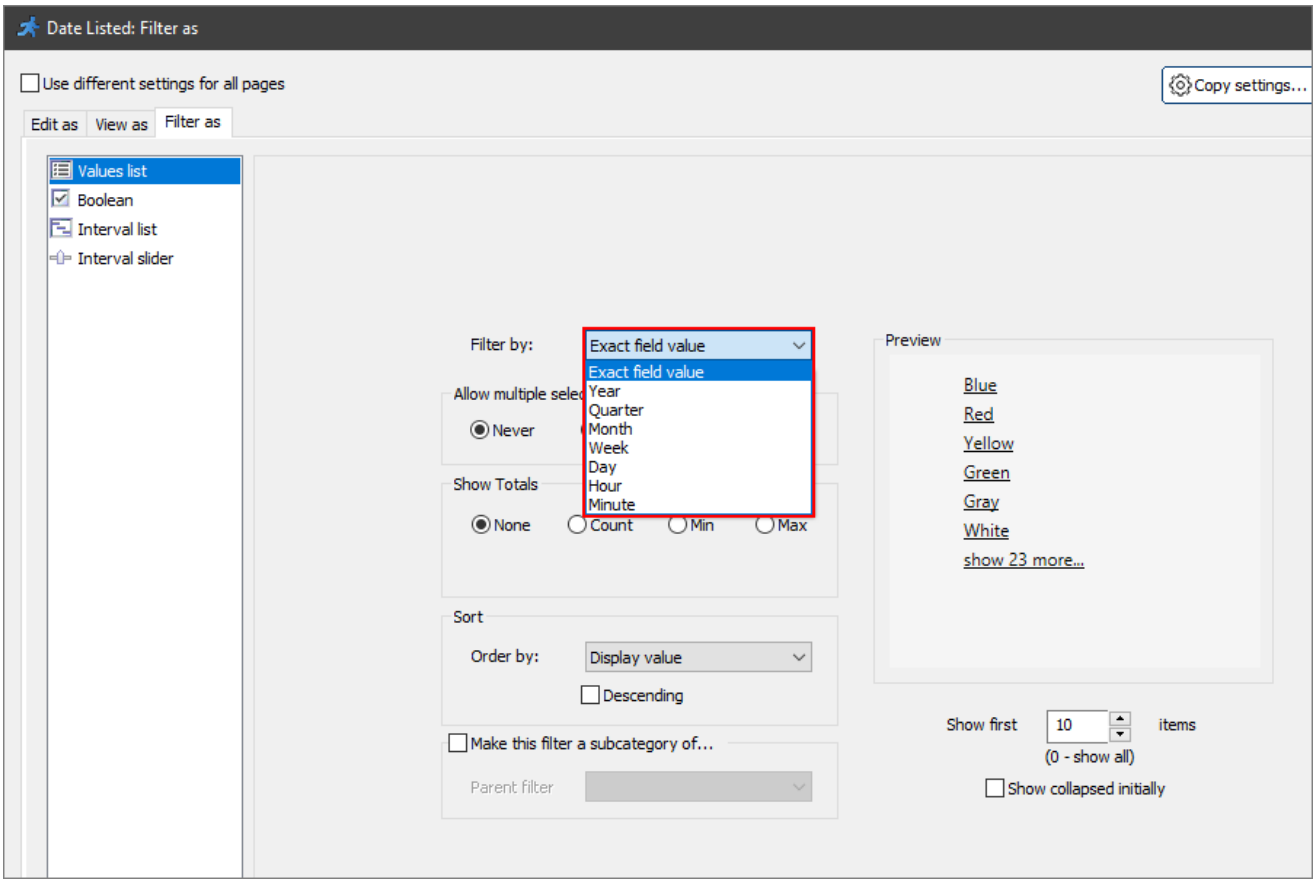
This filter type displays the filter as a list of values. There are several options available, along with a preview window, that shows how the result might look like.



Values list options

- **Filter by**. This option makes the filter show the exact field value or the first 1-5 initial letters. Useful for grouping the values, when you have a lot of them.

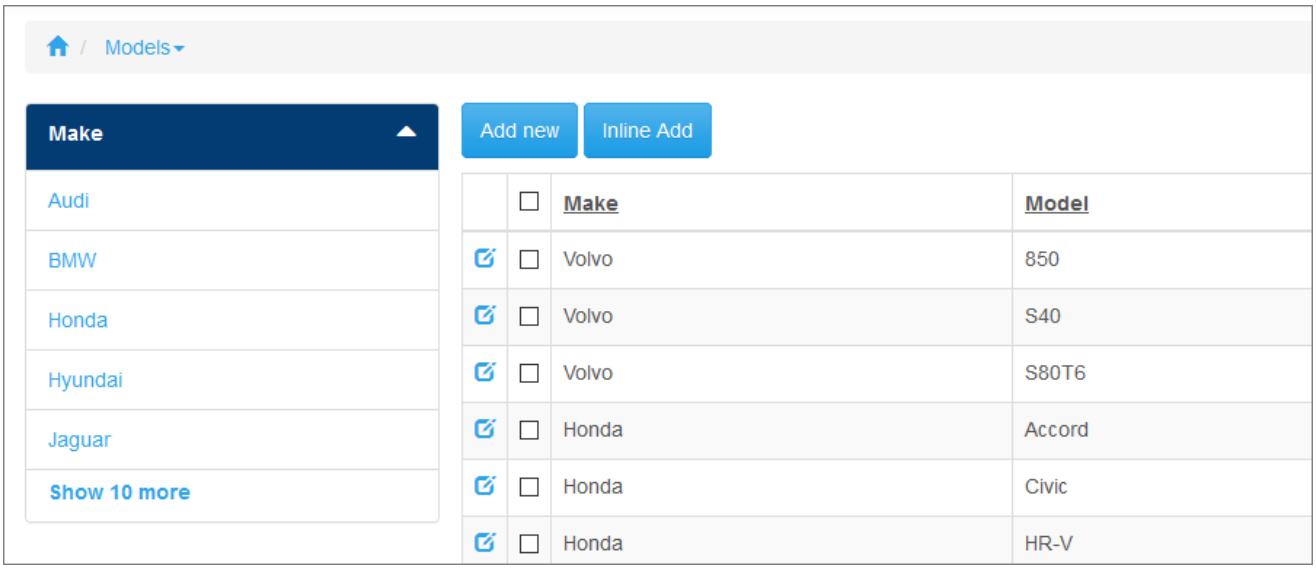
Note: if you have a *datetime* value, starting with PHPRunner version 10.3, you will find new convenient **Filter by** options. You can choose between filtering by *Year*, *Quarter*, *Month*, *Week*, *Day*, *Hour*, or *Minute*.



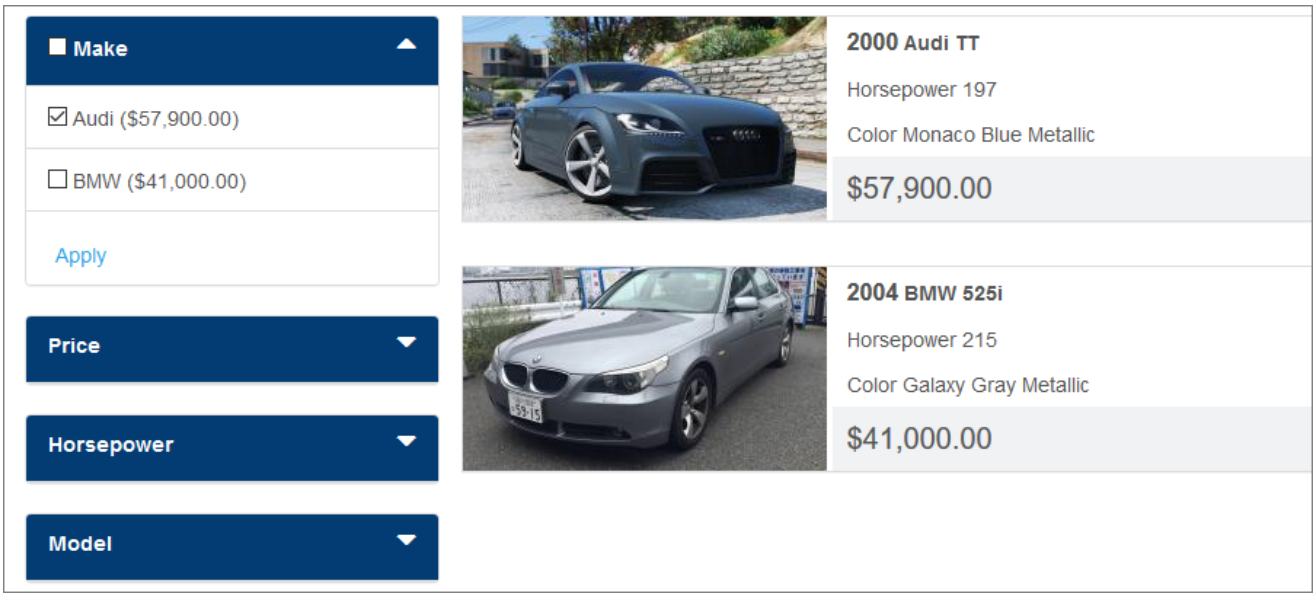
- **Allow multiple selection**. This option switches between different multiple selection modes. The multiple selection can be always on, off, or turned on by clicking the **Multiselect** button.
- **Show totals**. This option can show a total count or a min/max amount of the values of the field you select.
- **Sort**. This option allows sorting the filter by display value, database value, or totals value.
- **Show first N items**. This option allows limiting the number of items to show when the filter opens.
- **Make this filter a subcategory of**. This option allows grouping the values by a parent filter.
- **Show collapsed initially**. Select this checkbox to make the filter appear collapsed when the user loads the page for the first time.

Examples

1. Allow multiple selection - "*Never*"; show totals - "*None*"; show first "*5*" items.

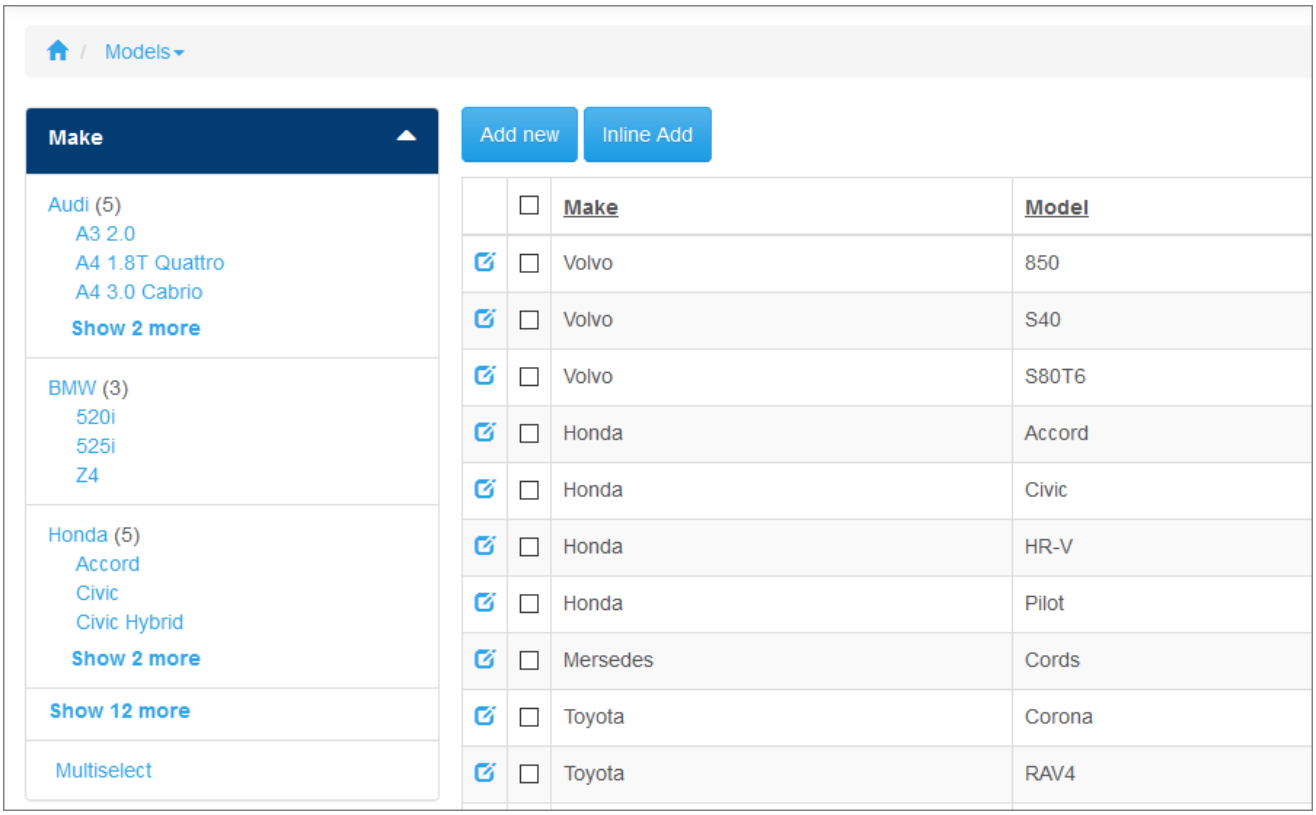


2. Filter "*make*": allow multiple selection - "*Always*"; show totals - "*Max*" (price); show all items. Other filters: *show collapsed initially*.



3. Filter "*model*": allow multiple selection - "*On demand*"; show totals - "None"; make this filter a subcategory of "*make*"; show first "*3*" items.

Filter "*make*": allow multiple selection - "*On demand*"; show totals - "*Count*"; show first "*3*" items.



See also:

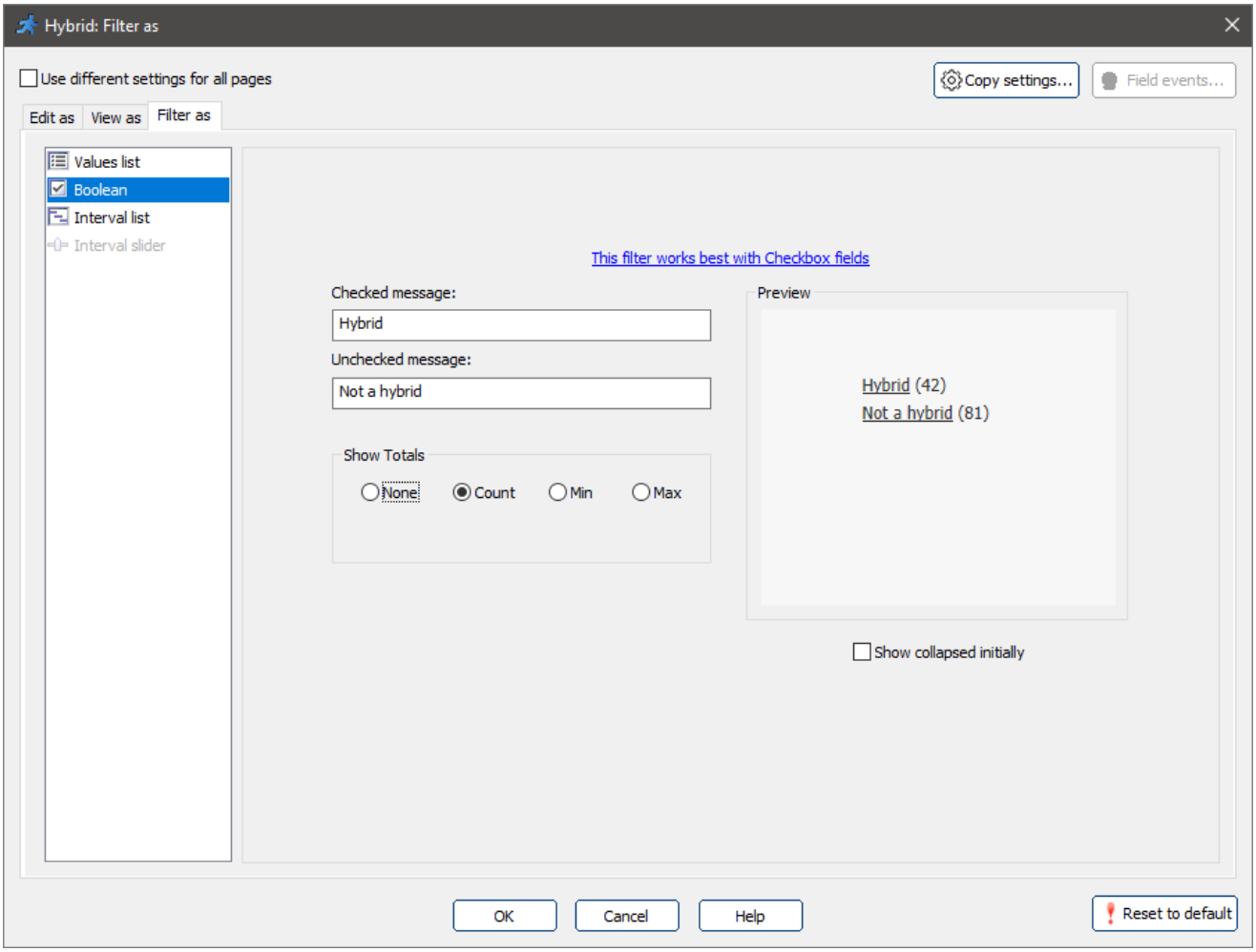
- "Filter as" settings
- About Page Designer
- About Editor

2.16.16.3 Boolean

"Boolean" filter type

This filter type displays the filter as two values: *Checked*/*Unchecked* by default or as your custom values (i.e., *Yes*/*No*, *True*/*False*, etc.).

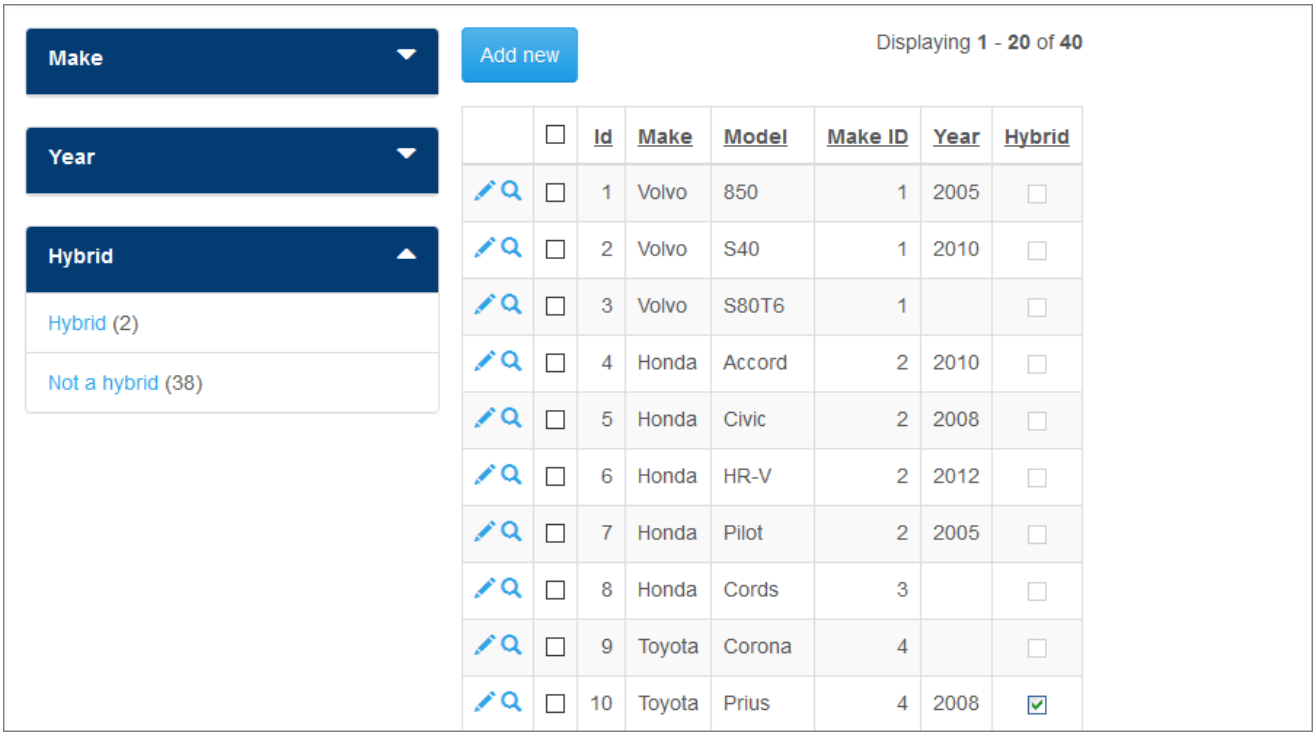
This filter works best with Checkbox fields.



Boolean filter options

- **Checked/unchecked message**. These messages appear in the filter to describe the state of the checkbox field. By default, the messages are "*Checked*" and "*Unchecked*". However, you can change the messages to your liking.
- **Show totals**. This option can show a total count or a min/max amount of the values of the field you select.
- **Show collapsed initially**. Select this checkbox to make the filter appear collapsed when the user loads the page for the first time.

Here is how the boolean filter looks like in the browser:



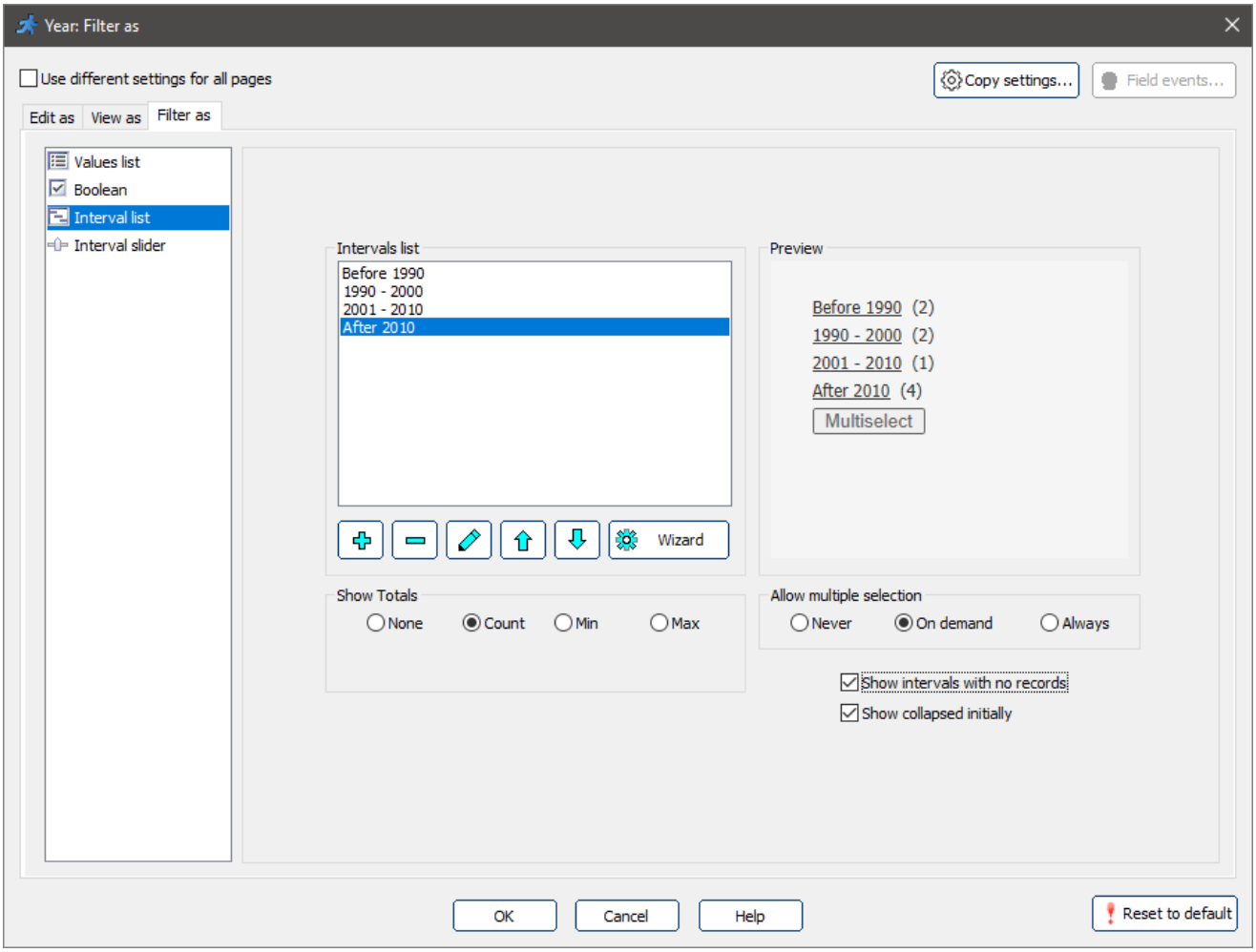
See also:

- "Filter as" settings
- About Page Designer
- About Editor

2.16.16.4 Interval list

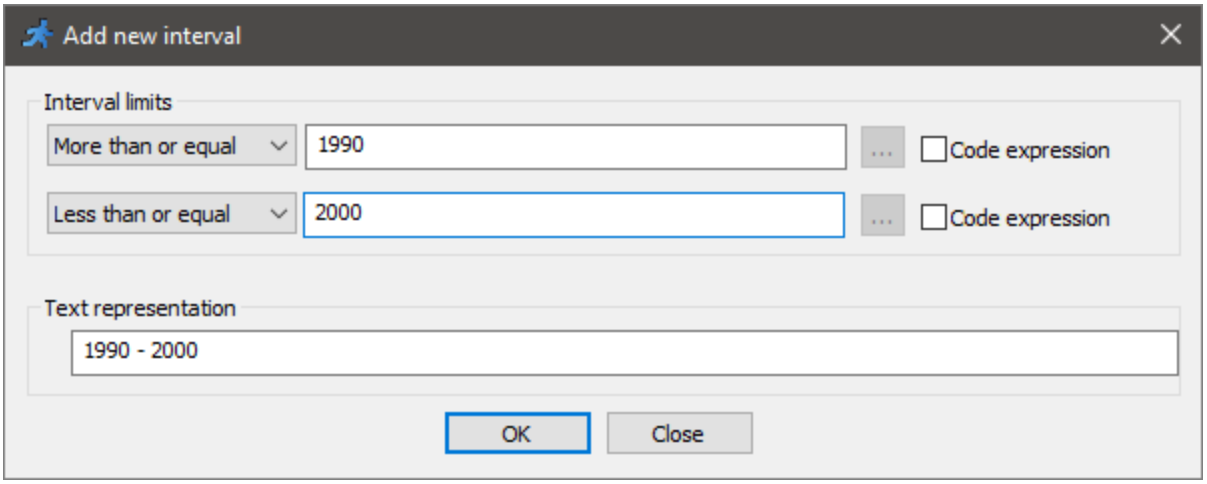
"Interval list" filter type

This filter type displays the filter as an interval list. You can create the intervals manually or using the **Wizard**.



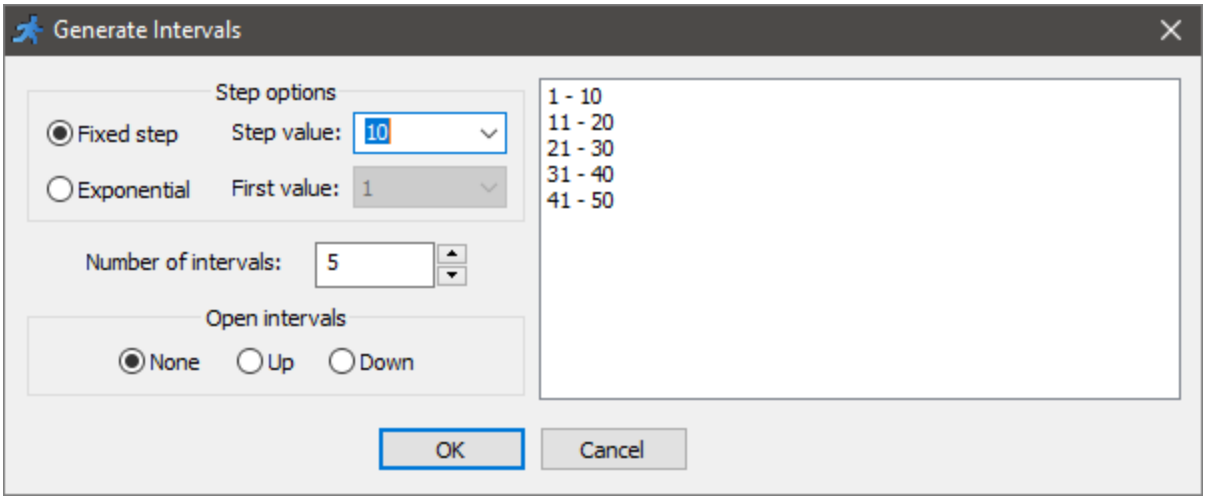
Interval list options

- **Add**. Click this button to open a popup window where you can define the interval limits. Select the **PHP expression** checkbox to define the limits using the code.



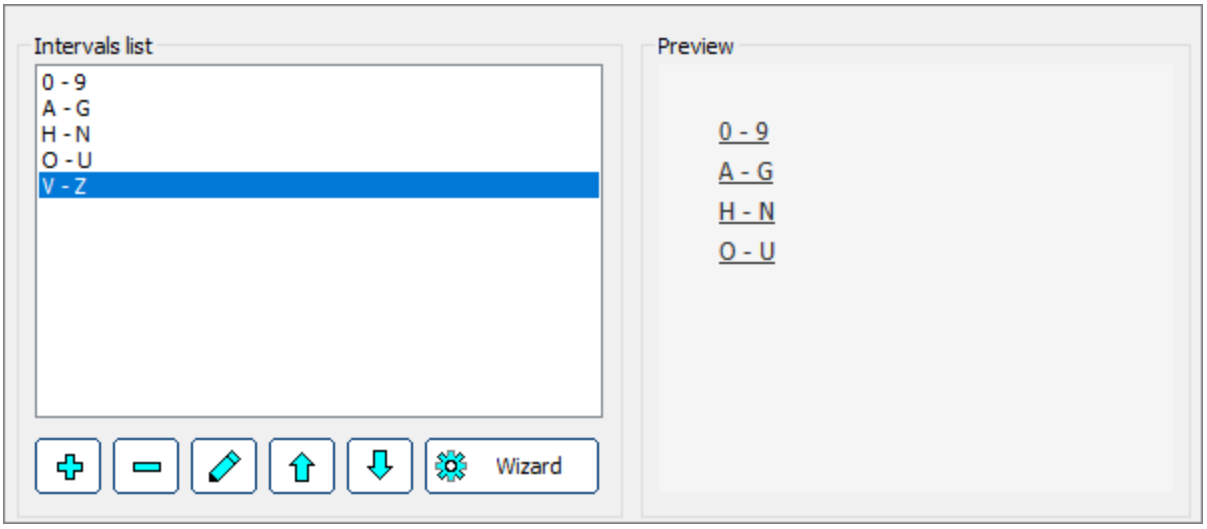
Note: the text representation field shows how the name of the interval appears in the generated app. It is generated automatically, but you can type in a custom name if you want.

- **Delete**. Click this button to delete the selected interval.
- **Edit**. Click this button to edit the selected interval in a popup window.
- **Move up/down**. Click these buttons to reorder the intervals.
- **Wizard**. Click this button to open a popup window where you can automatically generate the intervals for the numeric types of fields. You can select the number of intervals and choose whether to generate the intervals with a fixed step, or exponential steps. If you want to have an open interval, select one of the **Open intervals** options.

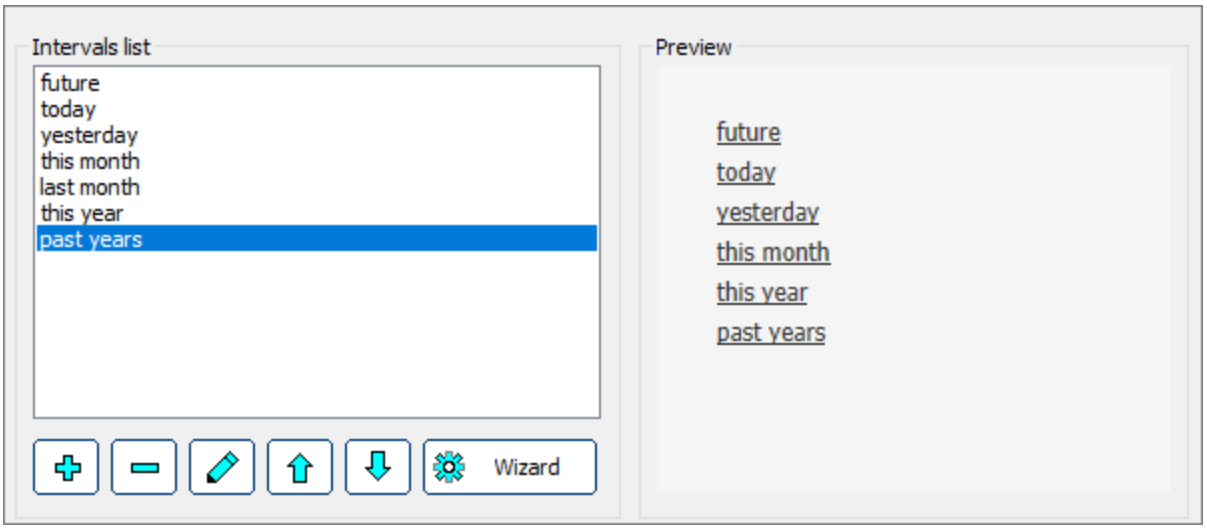


Note: if you click the **Wizard** button for the *varchar* or *datetime* types of fields, it sets up a several predefined intervals.

Varchar field:



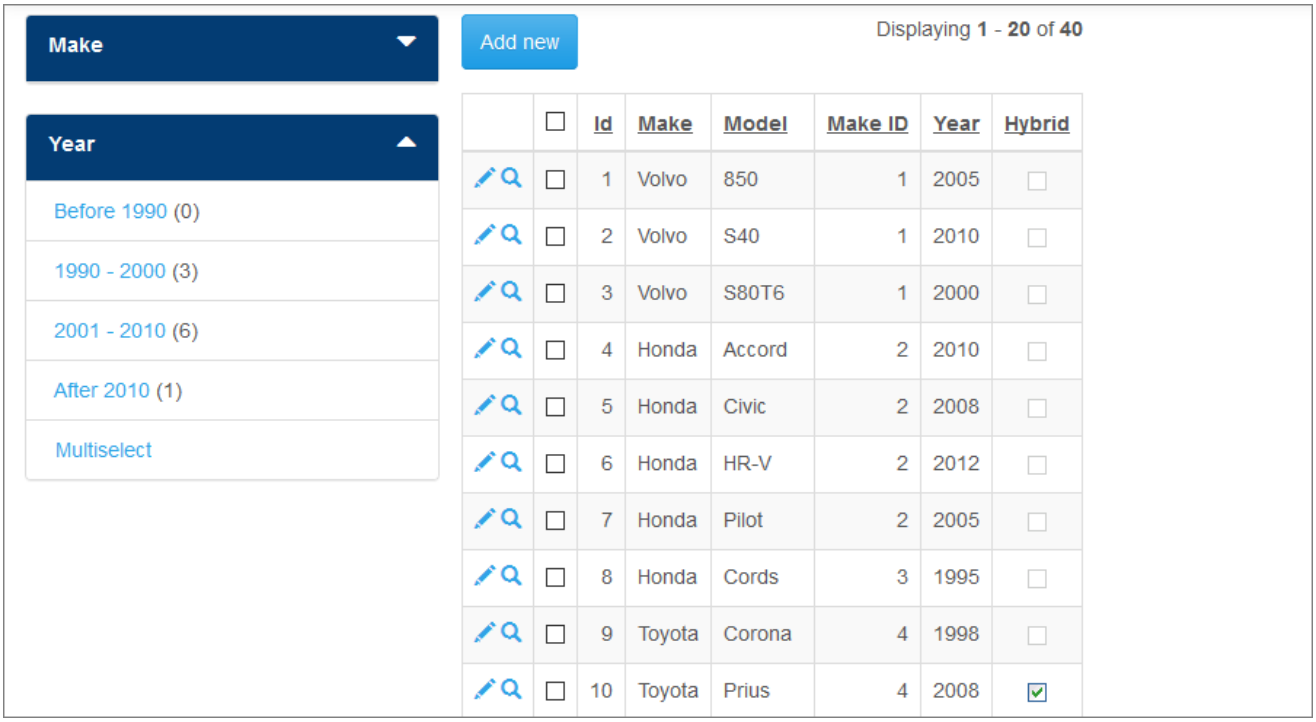
Datetime field:



- **Show totals**. This option can show a total count or a min/max amount of the values of the field you select.

- **Allow multiple selection**. This option switches between different multiple selection modes. The multiple selection can be always on, off, or turned on by clicking the **Multiselect** button.
- **Show intervals with no records**. Select this checkbox to make the filter show all of the intervals regardless of whether the interval contains any records or not.
- **Show collapsed initially**. Select this checkbox to make the filter appear collapsed when the user loads the page for the first time.

Here is an example of the interval list filter:



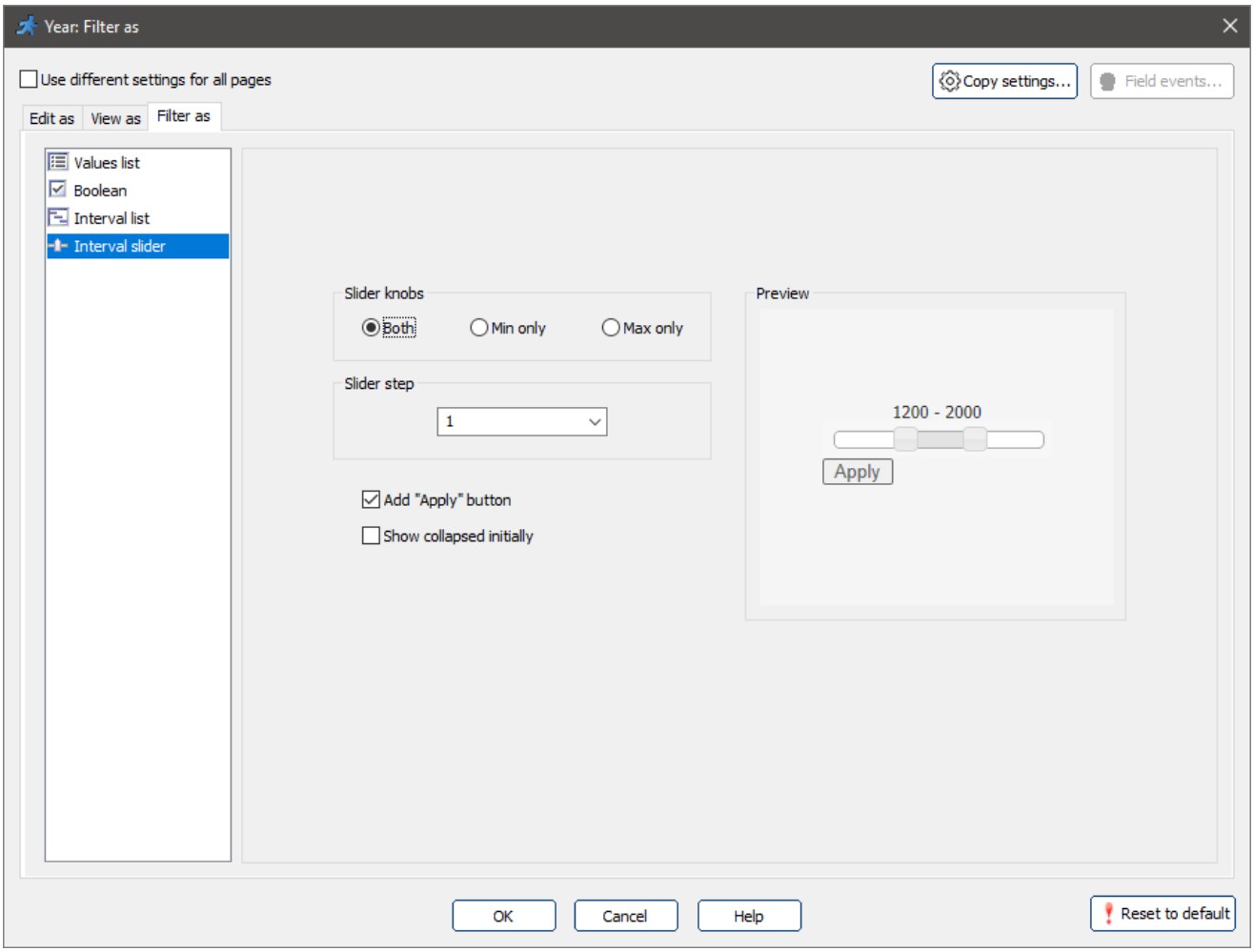
See also:

- "Filter as" settings
- About Page Designer
- About Editor

2.16.16.5 Interval slider

"Interval slider" filter type

This filter type displays the filter as an interval slider.

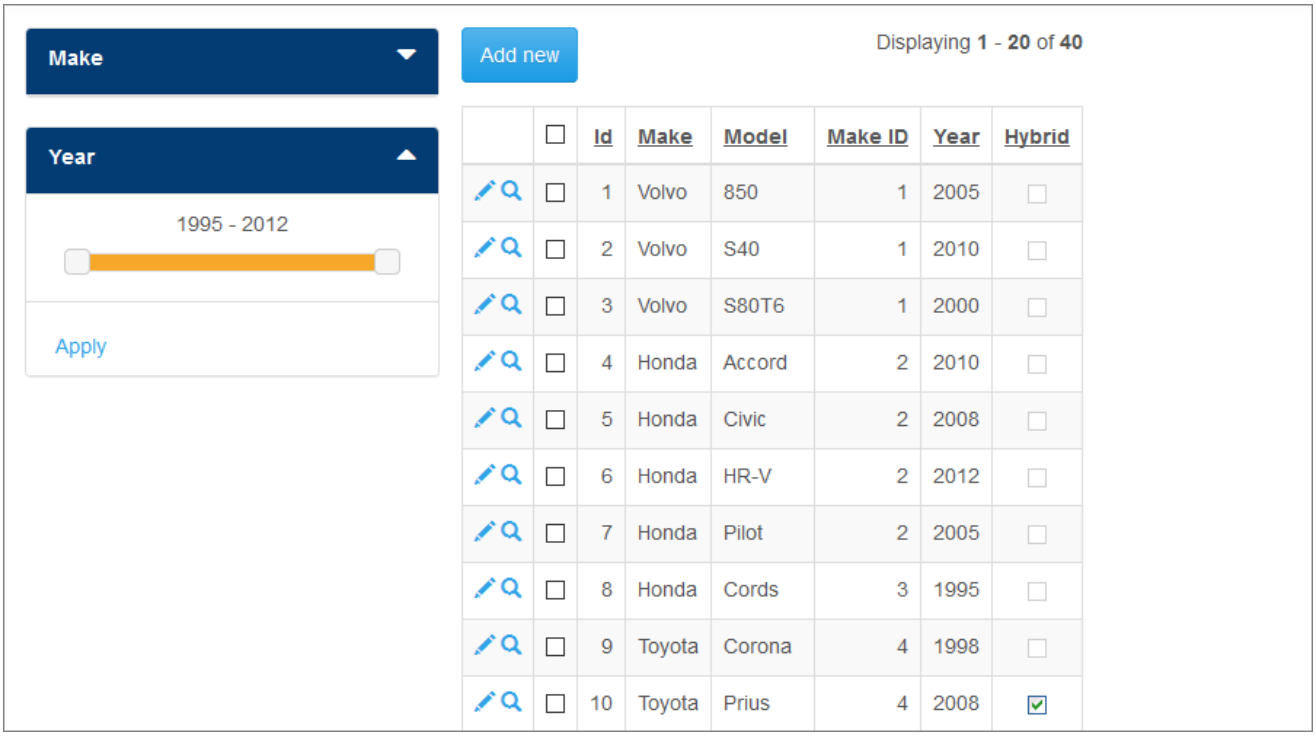


Interval slider options

- **Slider knobs**. Select whether to show both, only the "*Min*", or only the "*Max*" slider knob.
- **Slider step**. Use this dropdown to select the step for the slider. You can choose predefined values between 0.1 and 1000.
- **Add "Apply" button**. If you select this checkbox, clicking the "**Apply**" button filters the records to match the current values of the interval slider. Otherwise, the page reloads every time the user adjusts the filter.

- **Show collapsed initially**. Select this checkbox to make the filter appear collapsed when the user loads the page for the first time.

Here is an example of the interval slider in the browser:



See also:

- "Filter as" settings
- About Page Designer
- About Editor

2.17 Editor

2.17.1 About Editor

Quick jump

Choosing themes

Adding custom pages to your application

Setting the landing page

Editing HTML

On the **Editor** screen you can select a theme and set its size. You can also add your Custom CSS here that will be applied to the whole project.

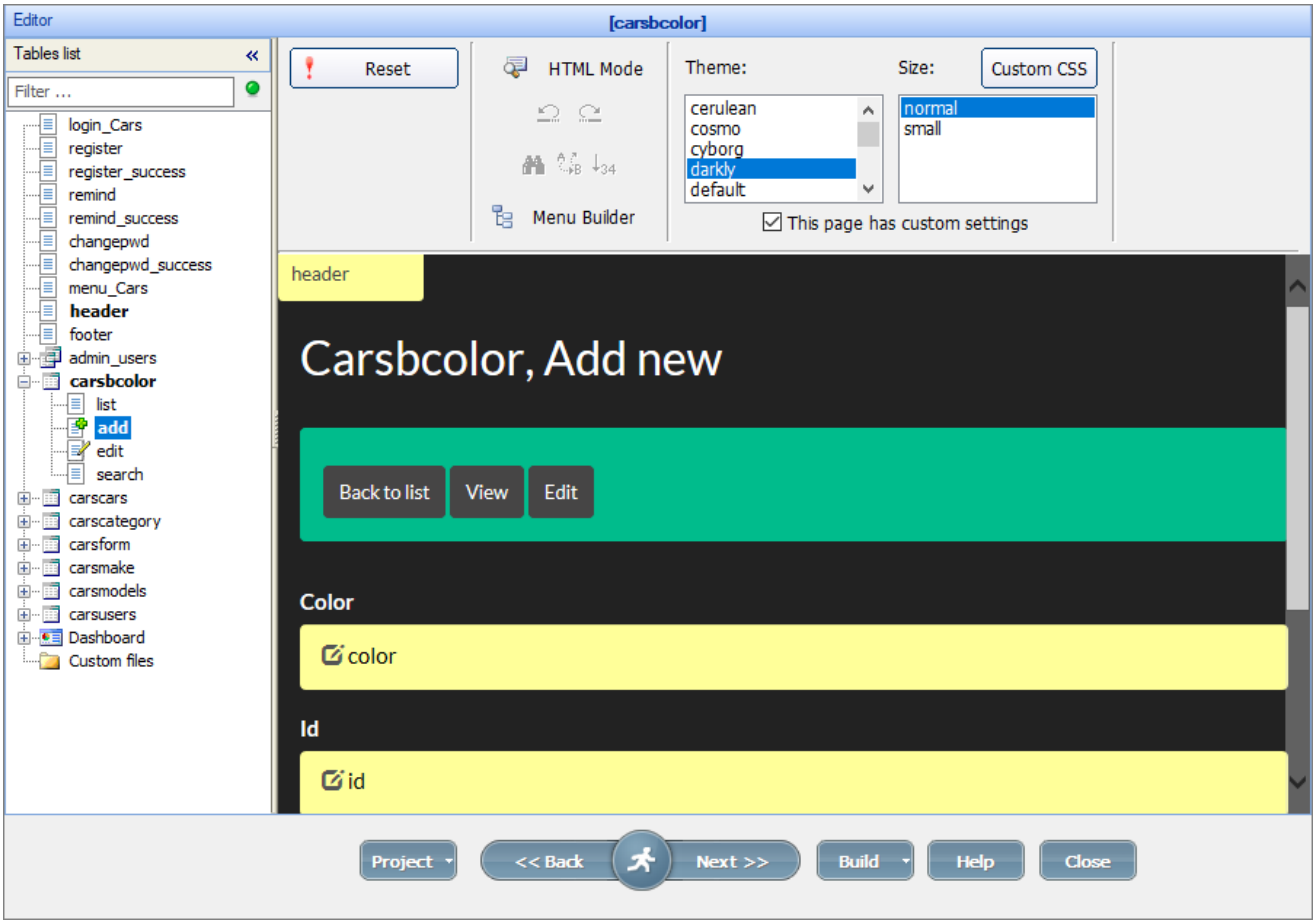
Choosing themes

With the **Editor** screen, you can modify the visual appearance of your pages and instantly preview the results.

There are three sections on the **Editor** screen:

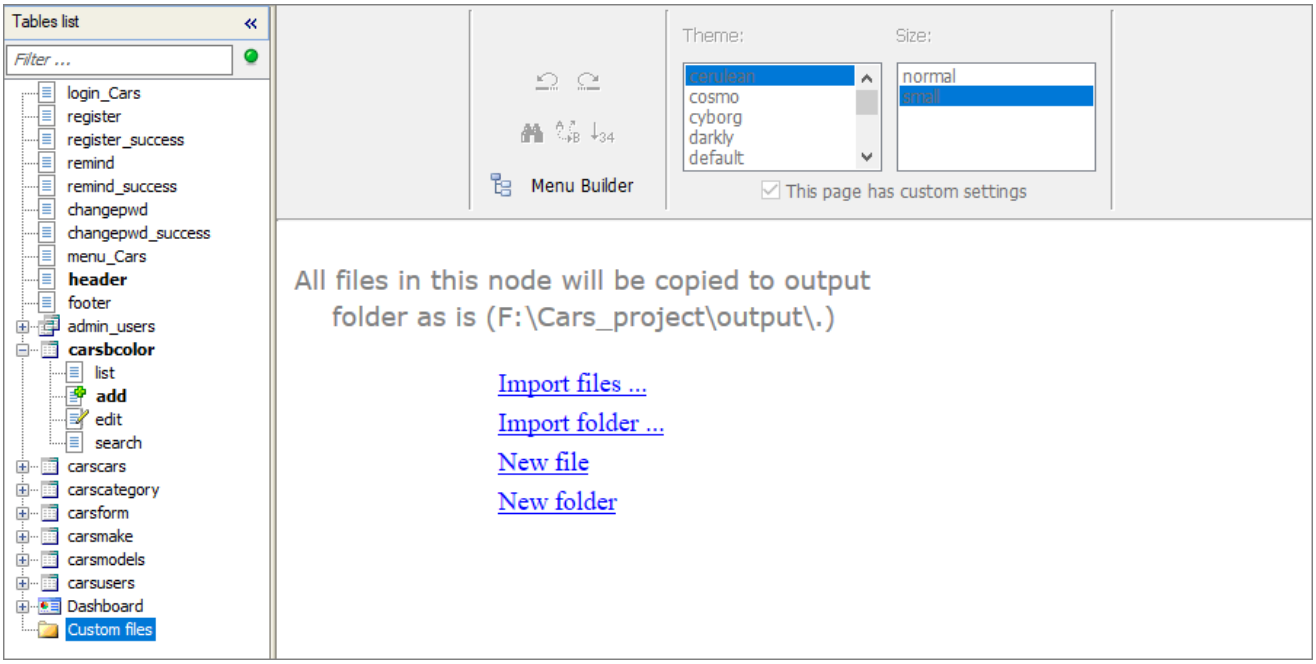
- **Tables list** on the left with the tables and their related pages. You can select the page to view/modify. To filter pages by name, use the text field above.
- **Control panel** at the top, where you can switch between **Editor** modes, open the Menu builder, alter the project theme, font size, and customize CSS.
- **Preview panel** on the right to see how the selected page will look like in the browser.

This page has custom settings is an option that enables setting the page to have a different appearance from the rest of the project:



Adding custom pages to your application

You can add a new page to your application or import an existing one. You can import the whole folder with HTML, images, CSS files and edit/preview them in **Editor**.



Setting the Landing page

You can set any page as a landing page - the page to open when the user enters the site (if login is disabled or guest user login is enabled) or just after login. To do that, right-click the page name and select **Set as Landing page**.

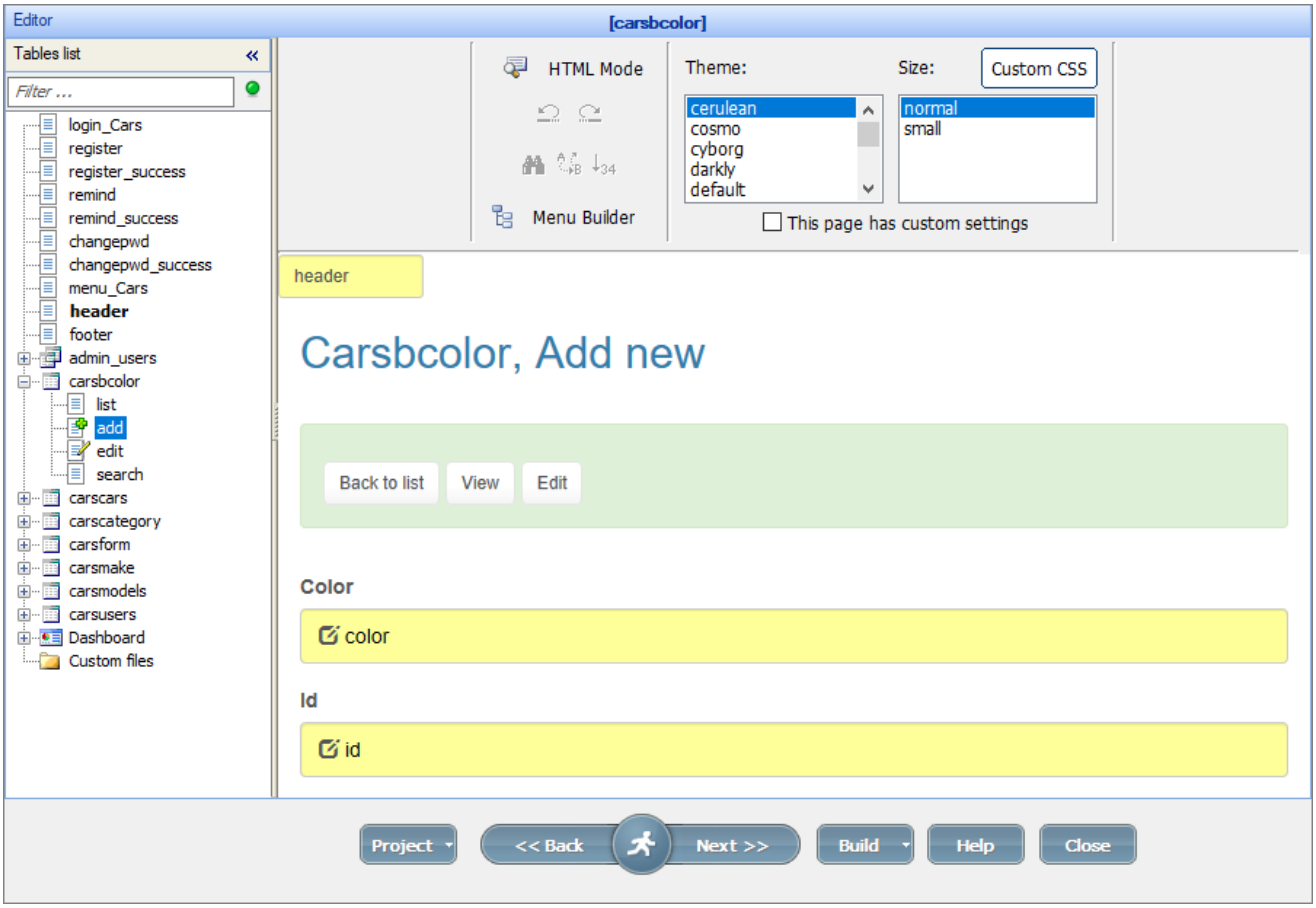
Editing HTML

You can also edit HTML of any page here.

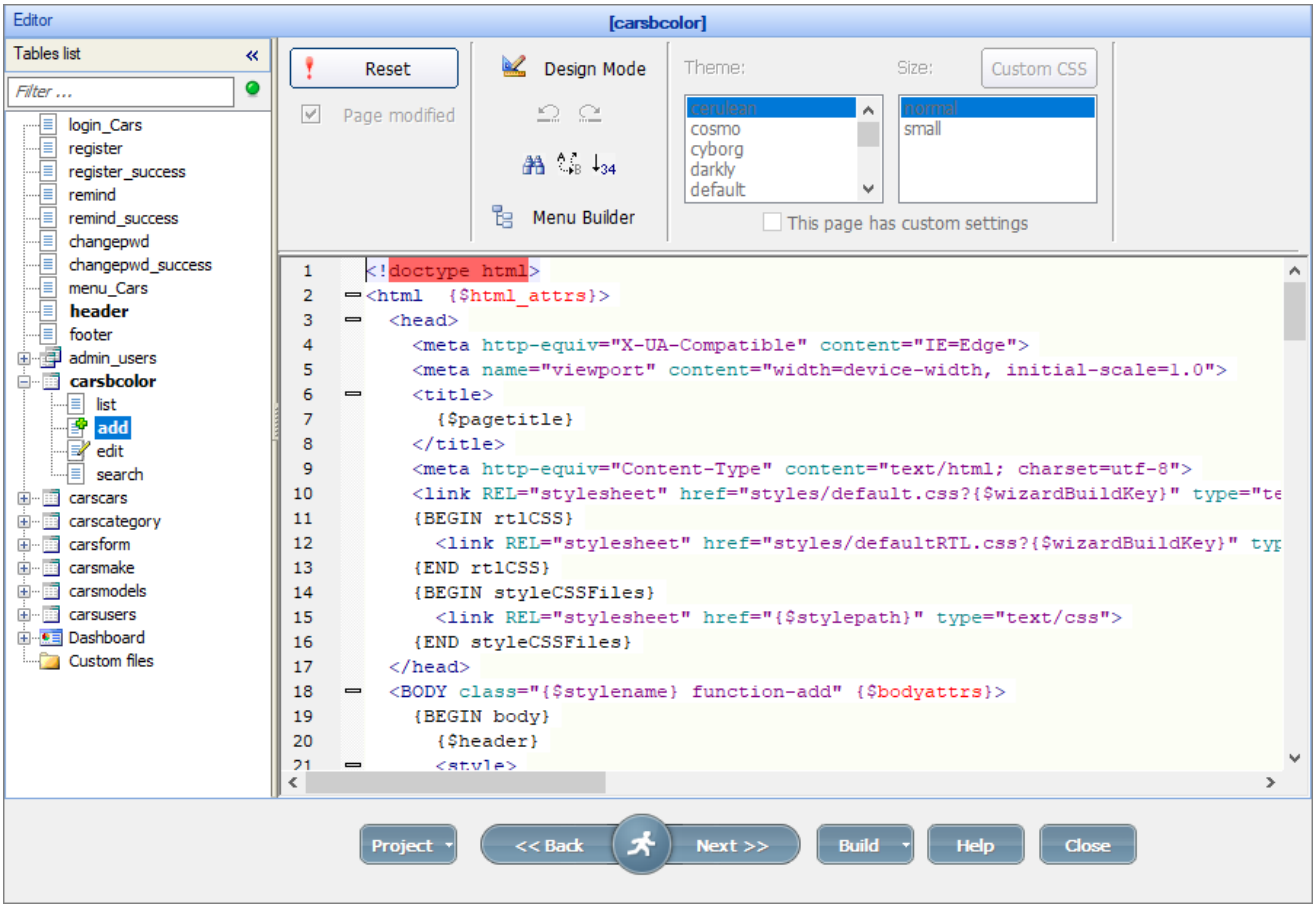
> **Warning!**
>
> Editing HTML is left for compatibility with old projects only and highly not recommended.

You can switch between the **Design** mode (page preview) or **HTML** (code view) mode.

Here is how the **Design** mode looks like:



Here is how the **HTML** mode looks like:



See also:

- Customizing CSS examples
- About Page Designer
- Menu builder
- Editing the <head> section

2.17.2 Customizing CSS examples

Quick jump	
How to change the font in a grid on the List page	How to change the cursor to a hand icon while hovering over a dashboard element
How to hide the "required" icon	How to turn on the word wrap for all table cells

Quick jump	
How to change the color of menu items and the project name	How to turn off the word wrap for all table cells
How to change the color of a regular button	Making the 'View details' icon bigger
How to change the color of the main button	How to change the 'View details' icon mouseover tooltip
How to change the color of the menu bar	How to change the icon into the word "Details"
How to change the text color of a breadcrumb item	How to change the width, font size and color of the search suggest window
How to change the text color of a breadcrumb container	How to change the color of the info on the Welcome page panels
How to change the color of dashboard panel headers	How to make the login form semi-transparent
How to set an image as the page background	

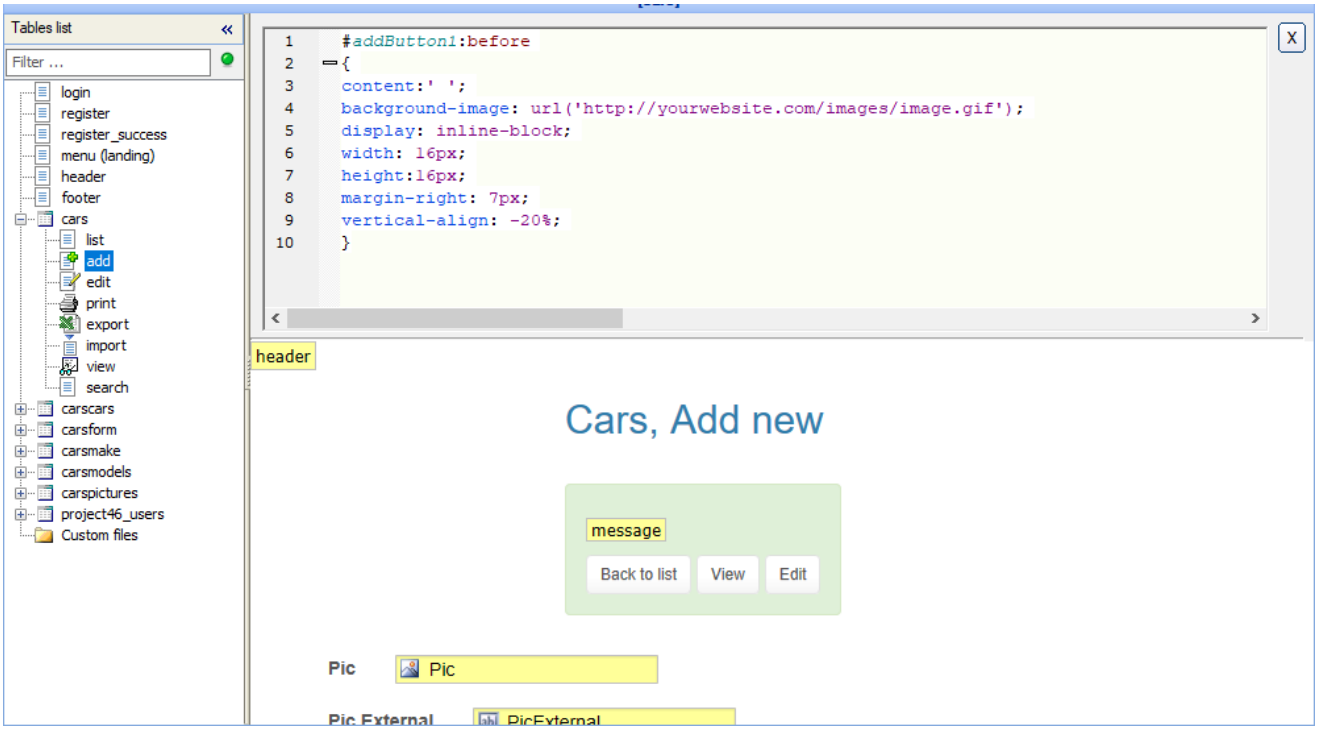
The **Custom CSS** button allows you to add custom CSS code to the pages of your project.

Let's say, for example, you want to add a background image to a single **List** page in the project. Select this page in the **Editor,** enable **This page has custom settings** checkbox. Click **Custom CSS** and add the following code:

```
body.function-list {
height:100%;
background:transparent url("http://mywebsite.com/images/some_pic.jpg") no-repeat
center center fixed;
background-size:cover;
}
```

When you have finished CSS customization, click the **Close** button to return to the preview mode.

You can also customize CSS in the Page Designer Cell/Element **Properties**.

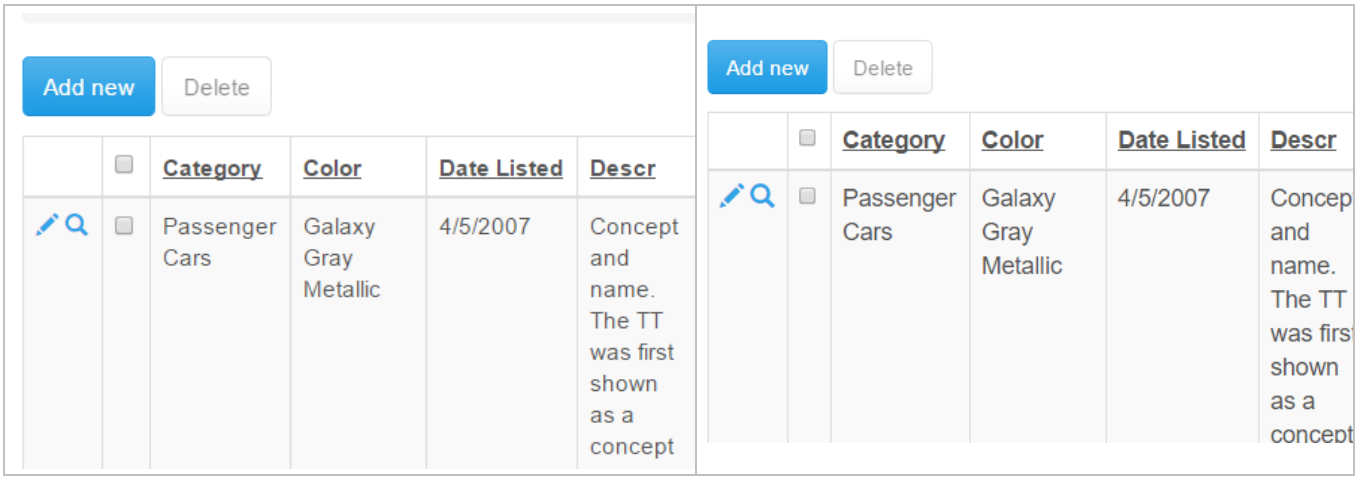


How to Customize CSS

Example 1. How to change the font in a grid on the List page.

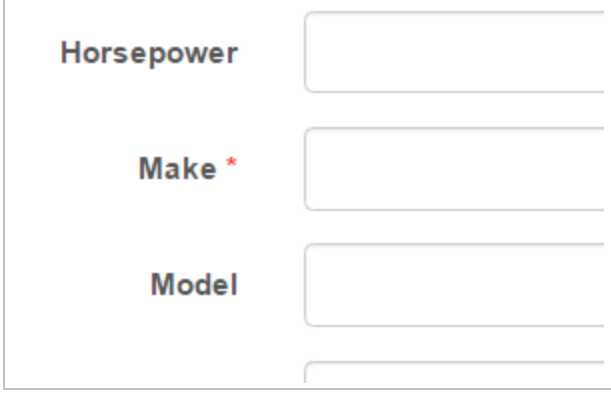
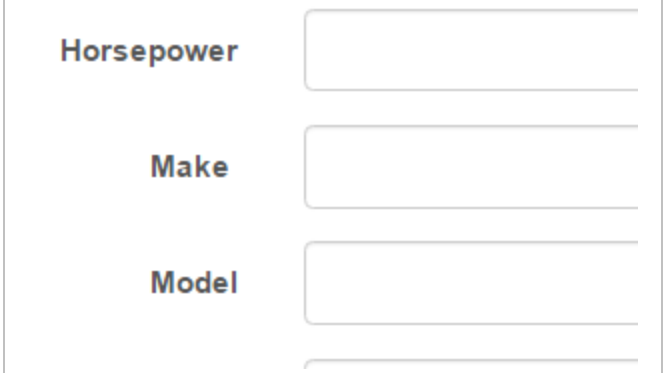
```
.r-grid {
font-size: 20px;
}
```

Before custom CSS was applied	After custom CSS was applied



Example 2. How to hide the "required" icon.

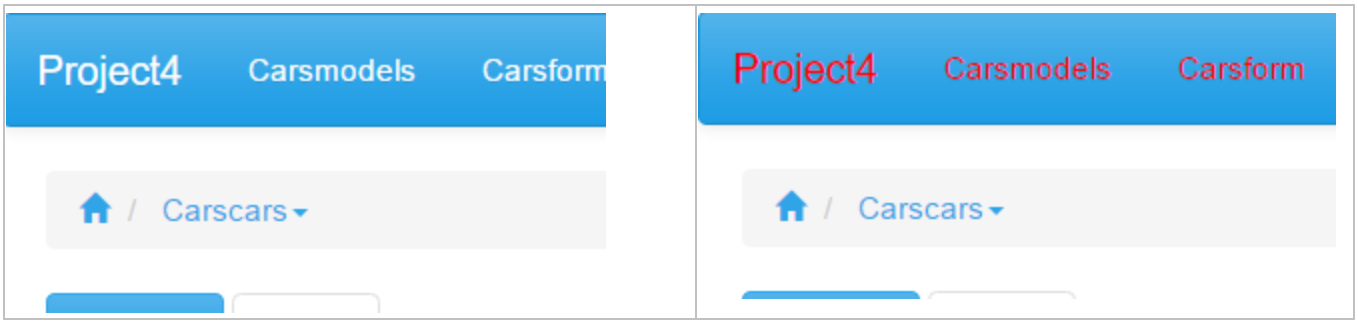
```
.icon-required {
display: none;
}
```

Before the custom CSS was applied	After the custom CSS was applied
	

Example 3. How to change the color of menu items and the project name.

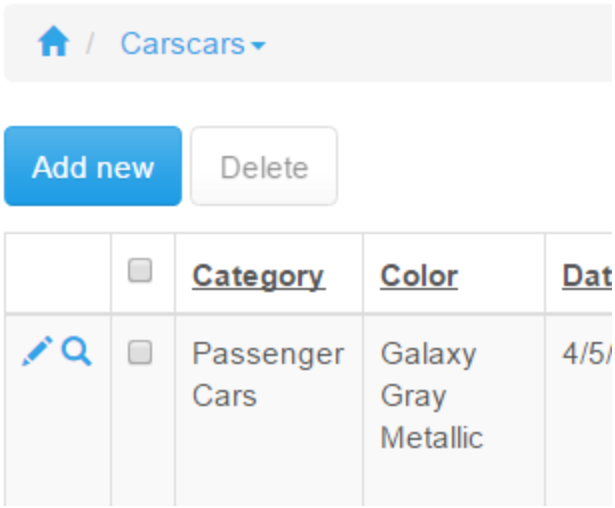
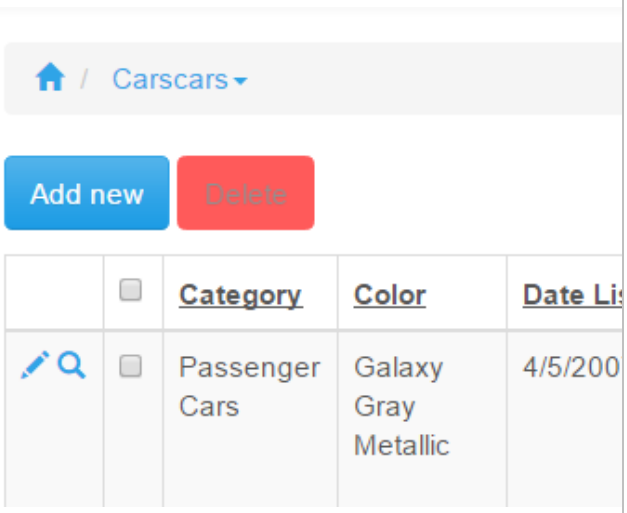
```
.navbar-nav>li>a, a.navbar-brand {
color: red !important;
}
```

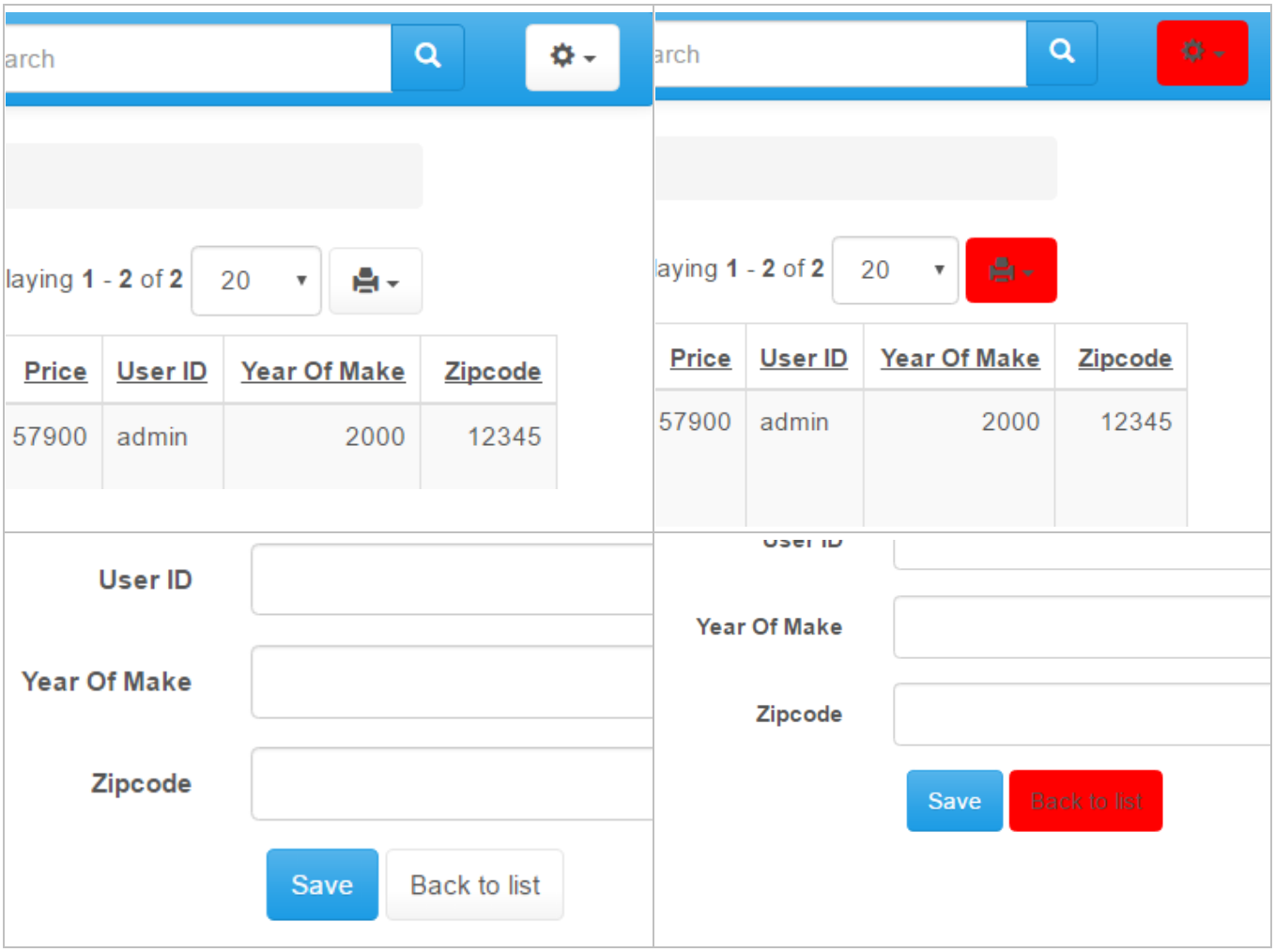
Before the custom CSS was applied	After the custom CSS was applied



Example 4. How to change the color of a regular button.

```
.btn.btn-default {
background: red;
border-color: red;
}
```

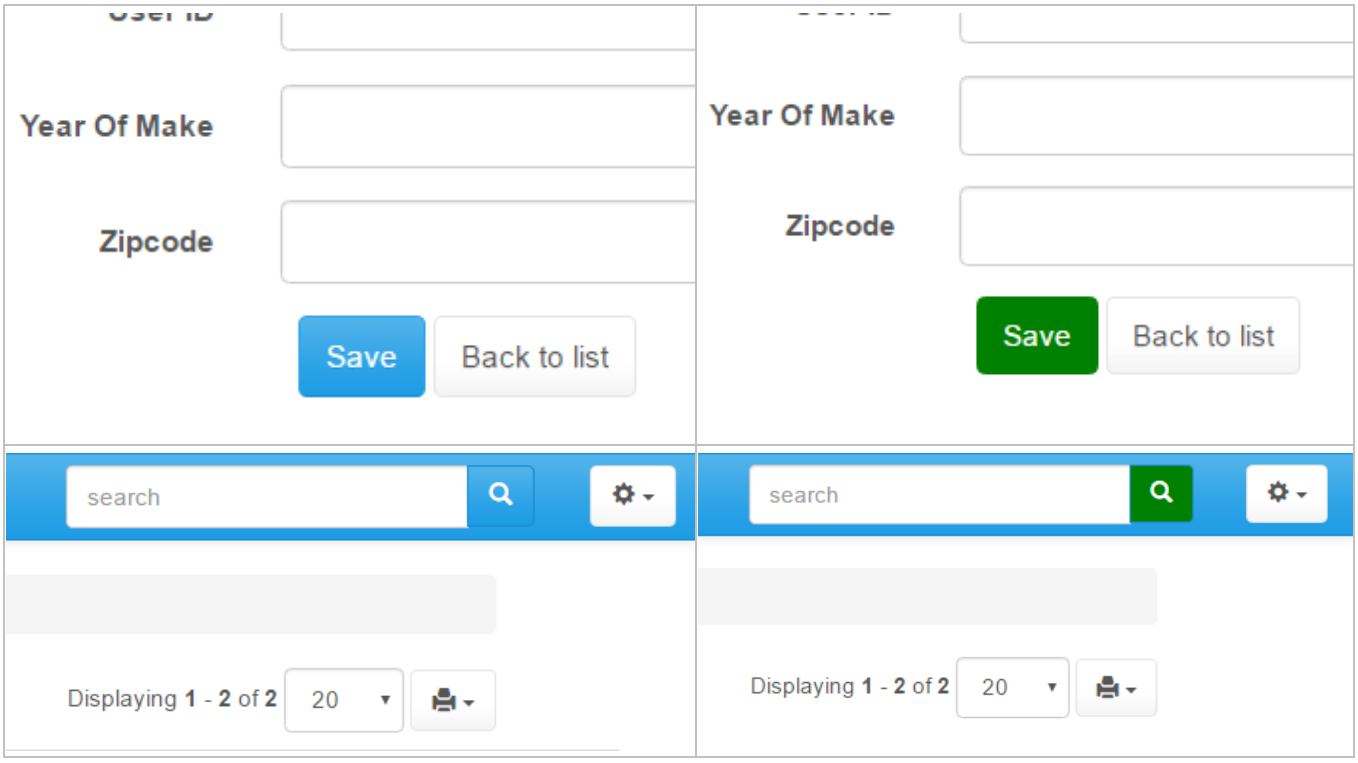
Before custom CSS was applied	After custom CSS was applied
	



Example 5. How to change the color of the main button.

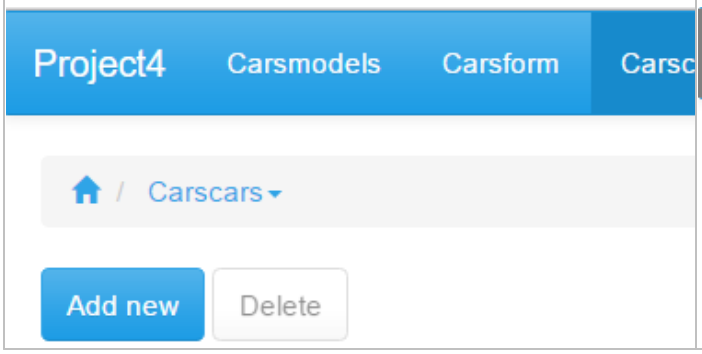
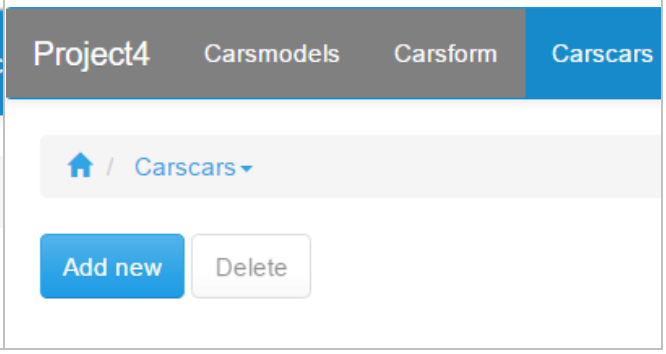
```
.btn.btn-primary {
background: green;
border-color: green;
}
```

Before custom CSS was applied	After custom CSS was applied



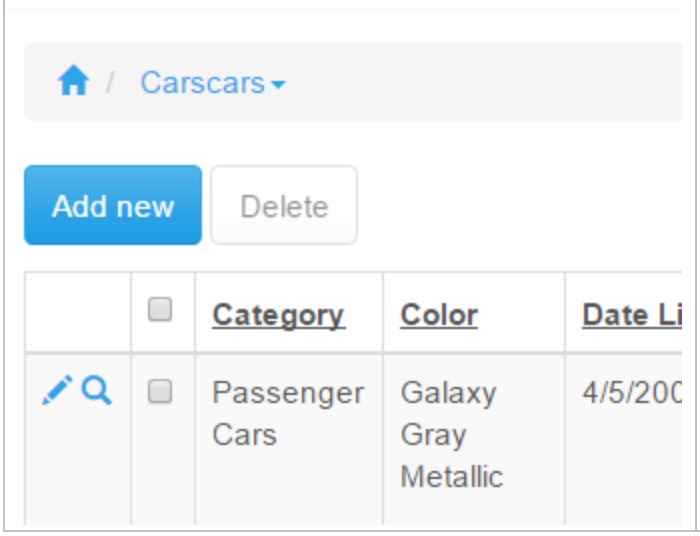
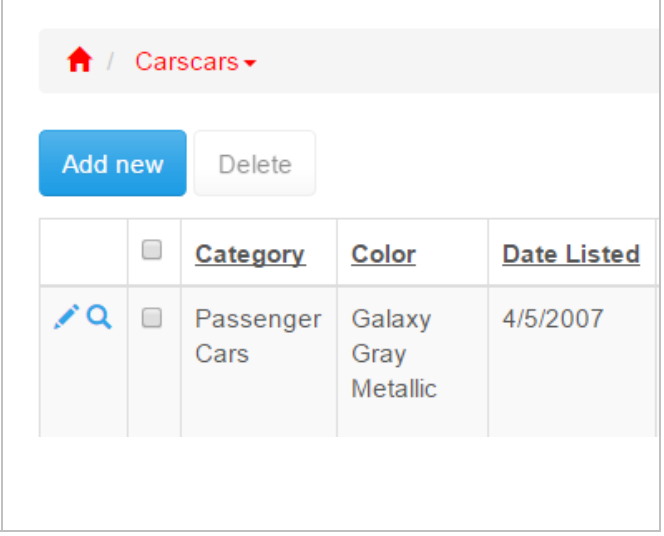
Example 6. How to change the color of the menu bar.

```
.navbar-nav {
background: grey;
}
```

Before custom CSS was applied	After custom CSS was applied
	

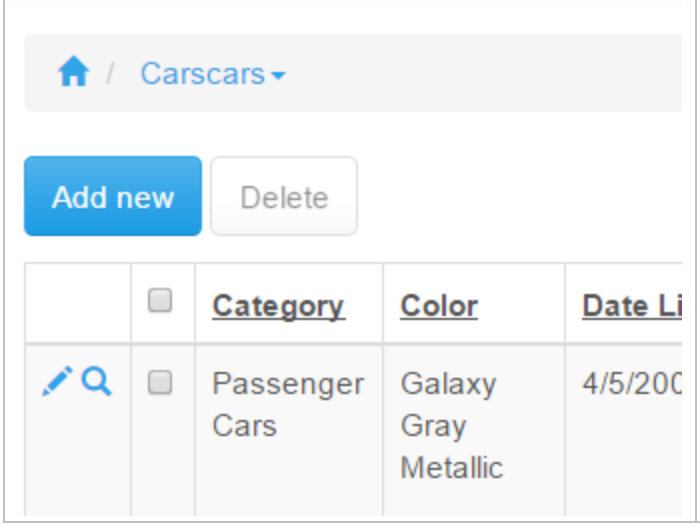
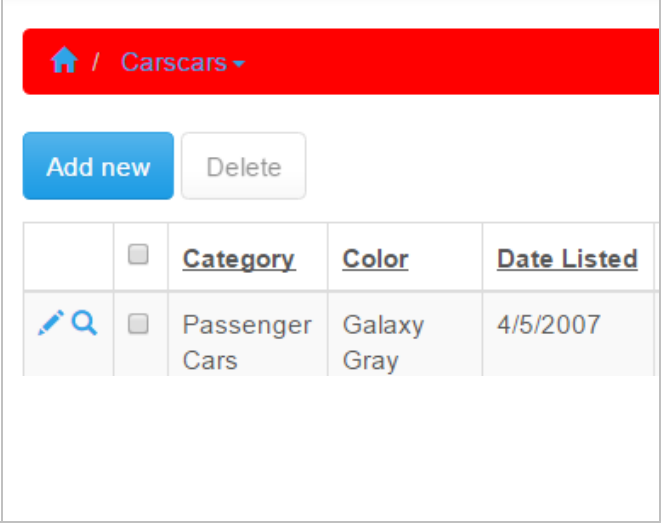
Example 7. How to change the text color of a breadcrumb item.

```
ol.breadcrumb a {
color: red;
}
```

Before custom CSS was applied	After custom CSS was applied
	

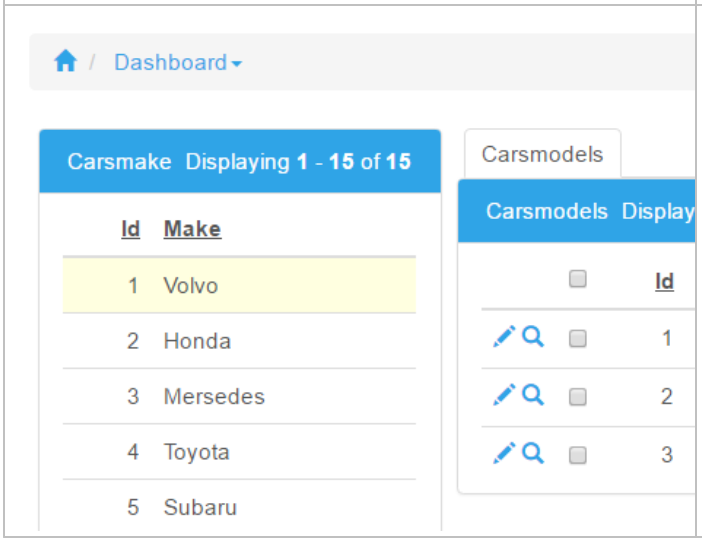
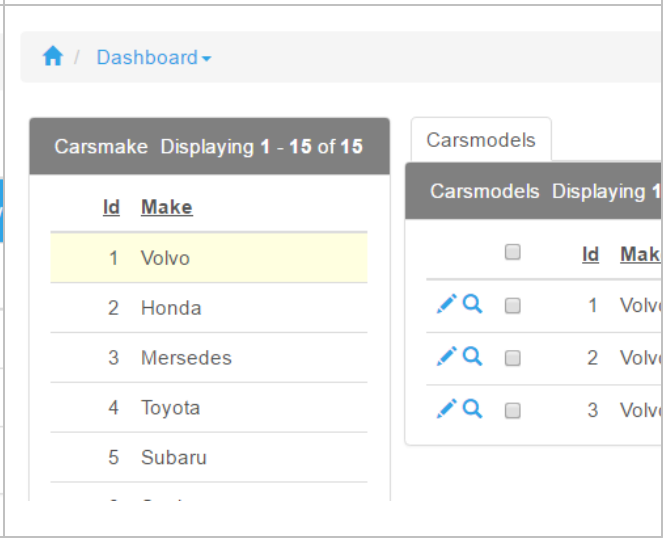
Example 8. How to change the text color of a breadcrumb container.

```
ol.breadcrumb {
background: red;
}
```

Sample screenshot before	After custom CSS was applied
	

Example 9. How to change the color of dashboard panel headers.

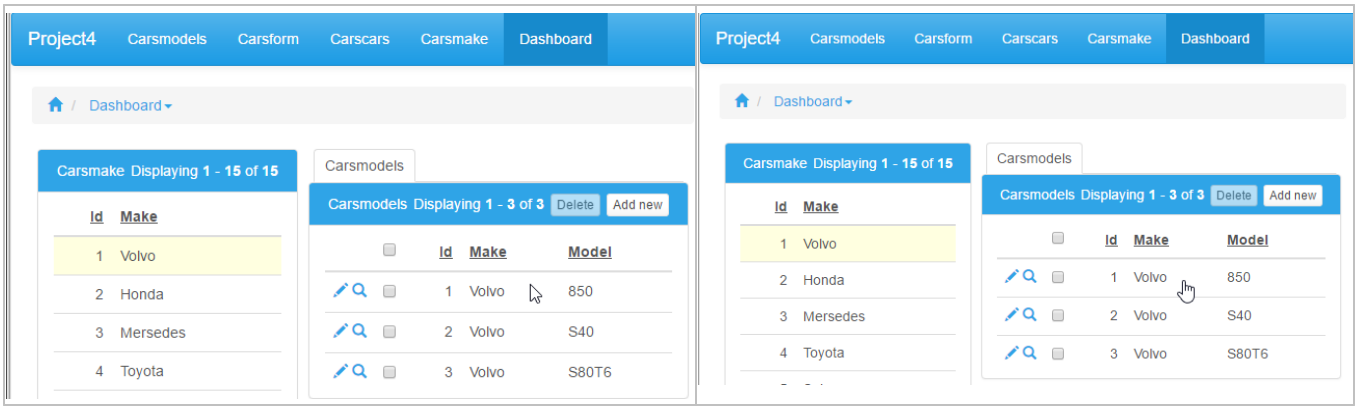
```
.panel-primary > div.panel-heading {
background: #808080;
}
```

Before custom CSS was applied	After custom CSS was applied
	

Example 10. How to change the cursor to a hand icon while hovering over a dashboard element.

```
.bs-dbelement {
    cursor: pointer;
}
```

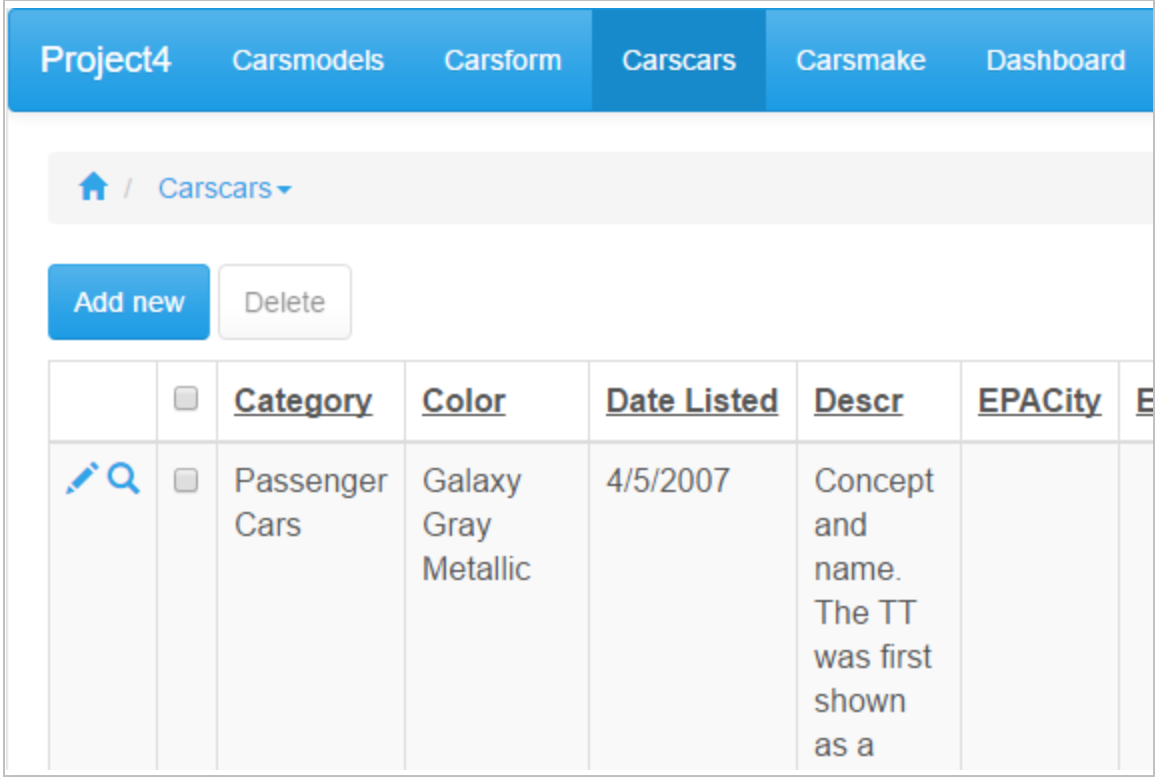
Before custom CSS was applied	After custom CSS was applied



Example 11. How to turn on the word wrap for all table cells.

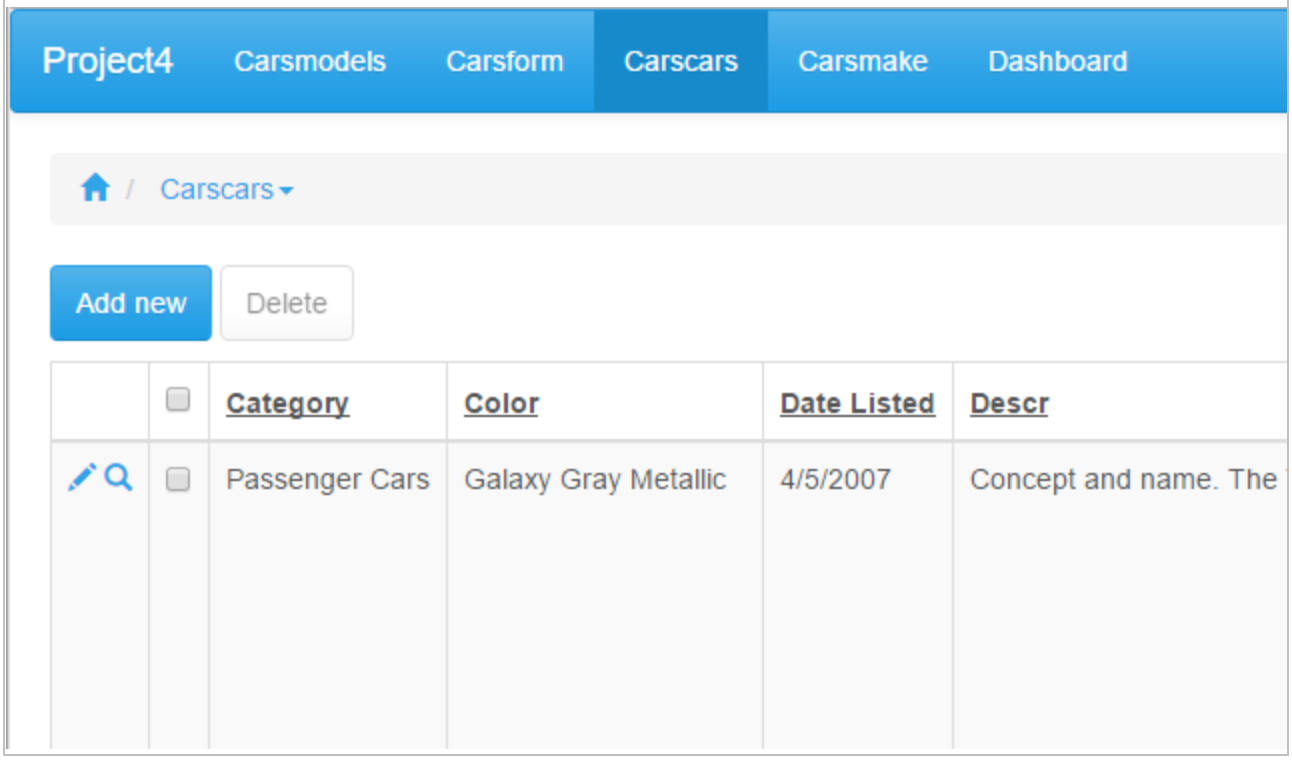
```
td.r-ori-vert {
white-space: normal;
}
```

After custom CSS was applied



Example 12. How to turn off the word wrap for all table cells.

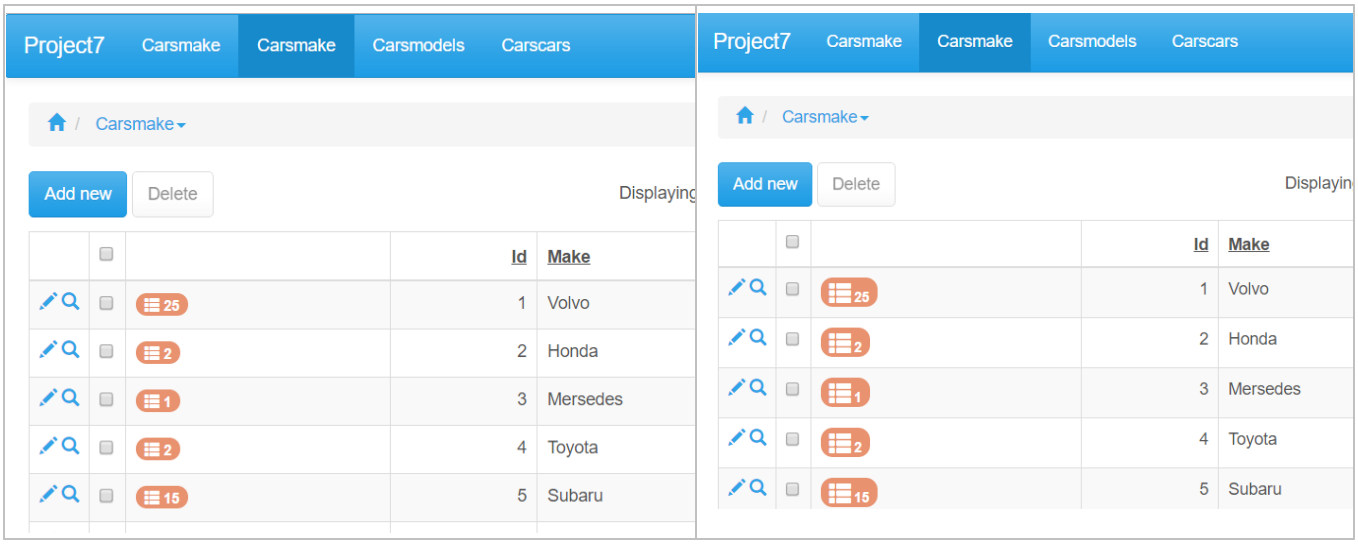
```
td.r-ori-vert {
white-space: nowrap;
}
```

After custom CSS was applied


Example 13. How to make the the 'View details' icon bigger.

```
span.glyphicon.glyphicon-th-list {
font-size: 150%;
}
```

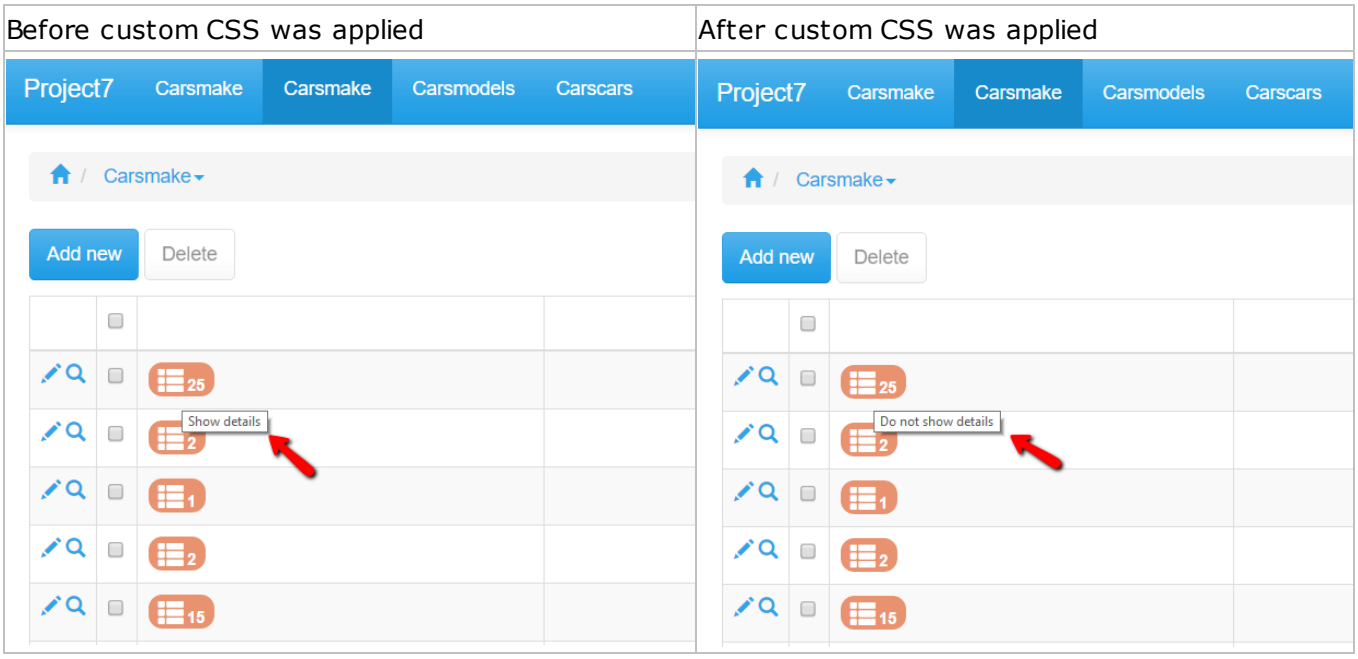
Before custom CSS was applied	After custom CSS was applied



Example 14. How to change the 'View details' icon mouseover tooltip.

Add the following to the JavaScript OnLoad event of the **List** page.

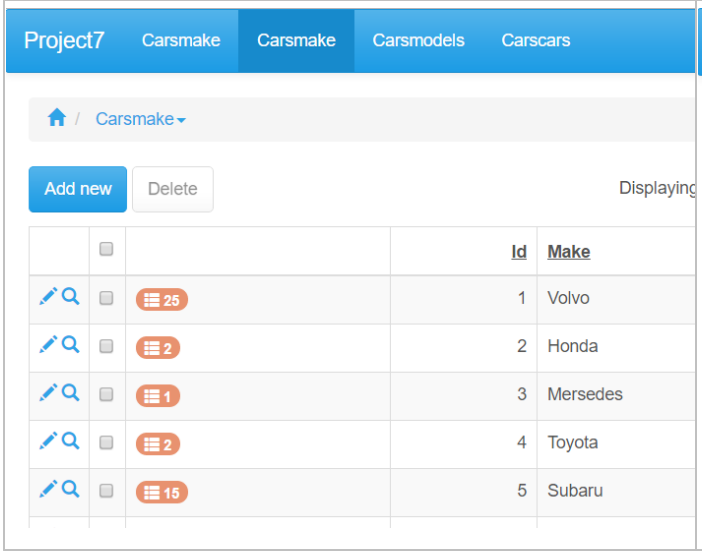
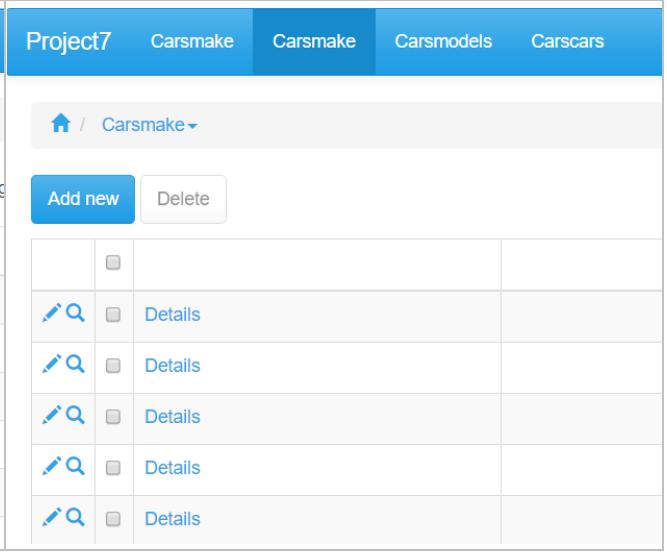
```
$("a[id^=details_]").attr('title', 'Do not show details');
```



Example 15. How to change the icon into the word "Details".

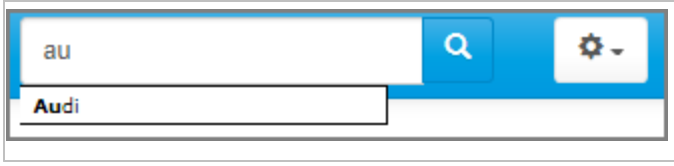
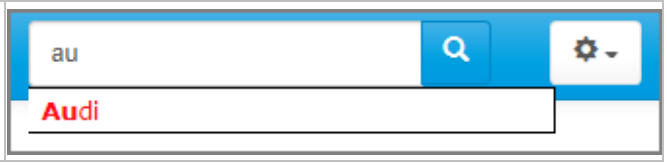
Add the following to JavaScript OnLoad event of the **List** page.

```
$("a[id^=details_]").find("span").attr("class","").text("Details");
```

Before custom CSS was applied	After custom CSS was applied
	

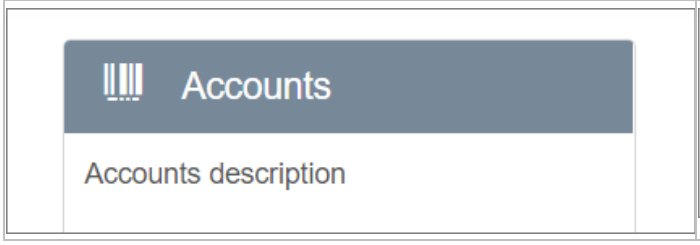
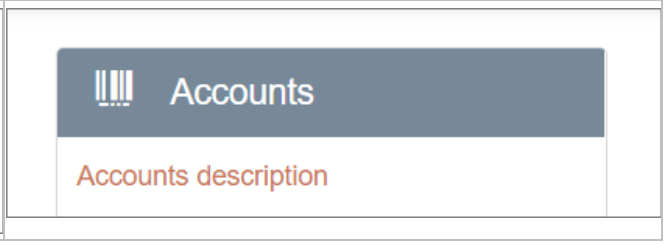
Example 16. How to change the width, font size and color of the search suggest window.

```
.suggest_link,  .suggest_link_over  {
  color: red;
  font-size: 14px;
}

.search_suggest {
 width: 300px;
}
```

Before custom CSS was applied	After custom CSS was applied
	

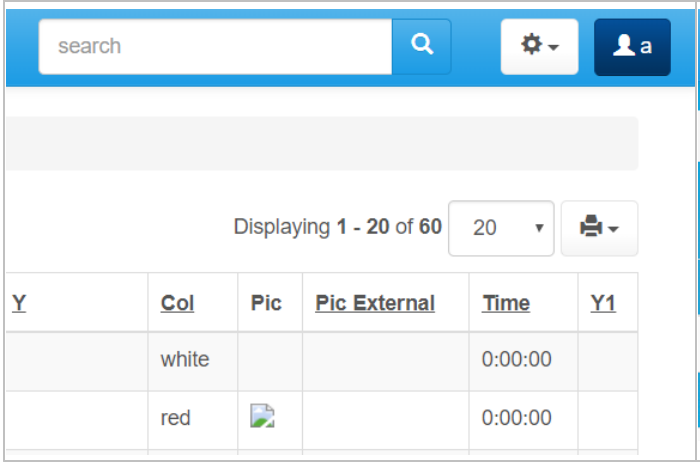
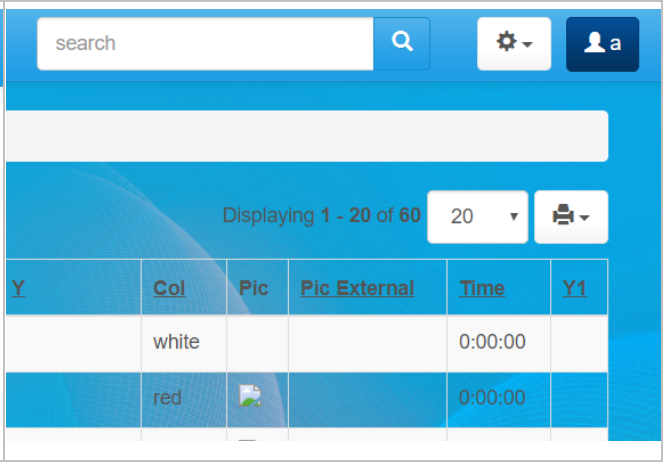
Example 17. How to change the color of the info on the Welcome page panels.

```
.bs-welcome-content {
 color: #CC7755;
}
```

Before custom CSS was applied	After custom CSS was applied
	

Example 18. How to set an image as the page background.

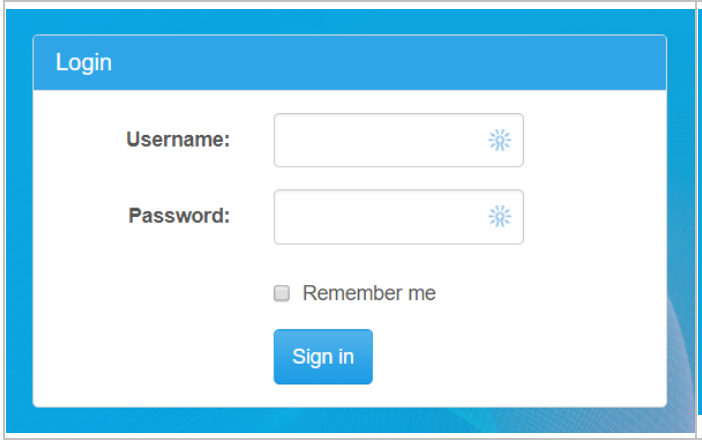
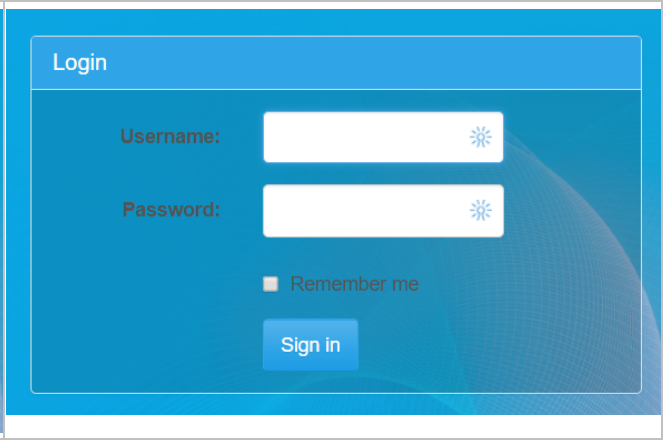
```
body {
height:100%;
background:transparent url("/images/some_image.jpg") no-repeat center center fixed;
background-size:cover;
}
```

Before custom CSS was applied	After custom CSS was applied
	

Example 19. How to make the login form semi-transparent.

```
.bs-pagepanel {
background: rgba(17,17,17,0.15) !important;
}d: rgba(17,17,17,0.15) !important;
}
```

Sample screenshot of the **Login** page with the background image set in the Example 19:

Before custom CSS was applied	After custom CSS was applied
	

See also:

- Building a nice looking login page with custom CSS
- Events: JavaScript OnLoad
- JavaScript API: Control object > add CSS Class
- About Editor
- Menu builder
- About Page Designer

2.17.3 Menu builder

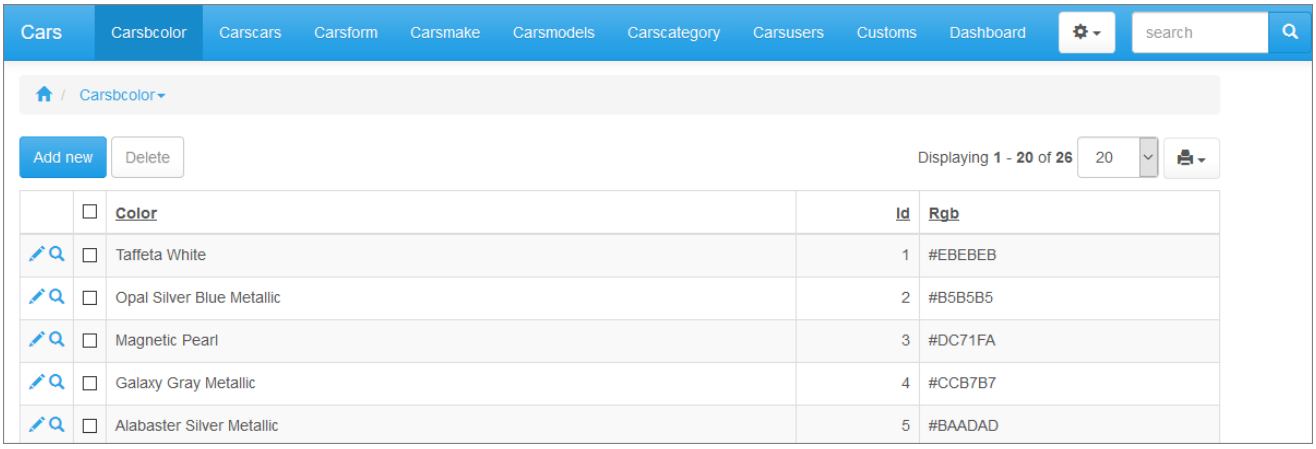
Quick jump

The **Menu Builder** lets you organize your tables and views into a multi-level cascading menu for quicker navigation. This feature is particularly useful when you have a large number of tables.

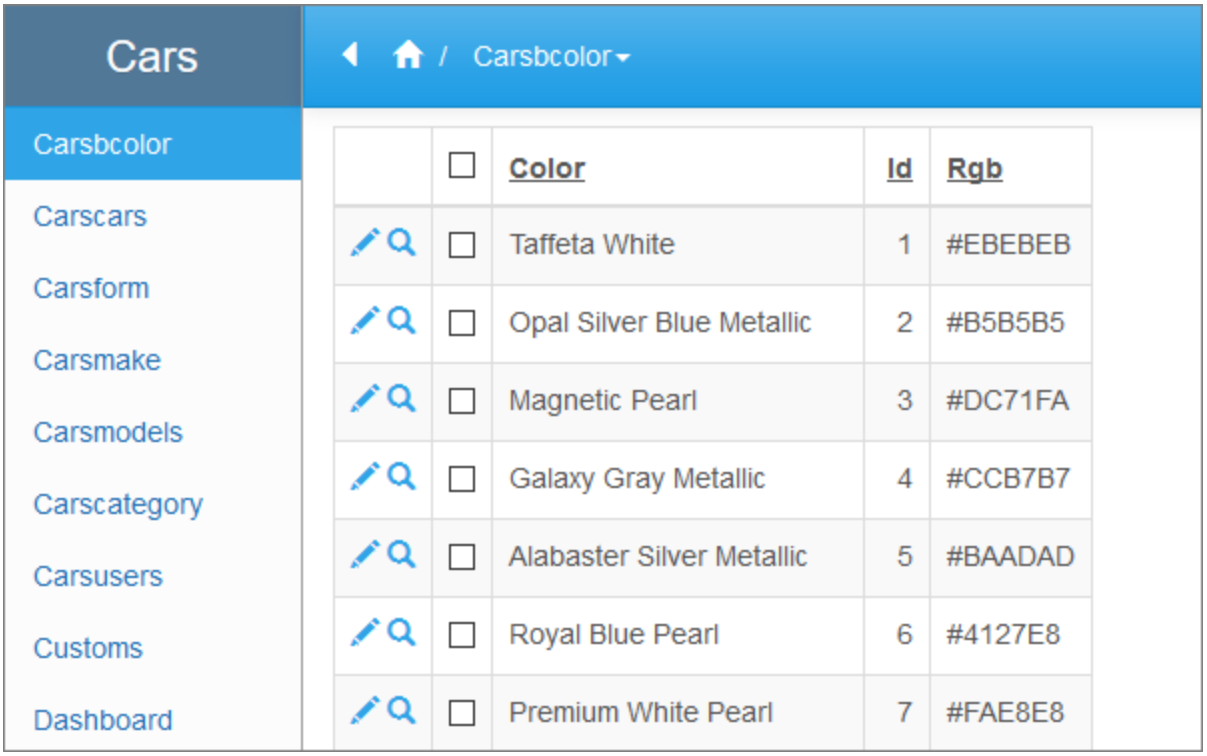
Once the project is created, the menu consists of links to the tables selected on the Datasource tables screen. Depending on the selected layout, the menu appears horizontally or vertically. The menu is single-level at this stage.

See Page layout and grid type to learn more.

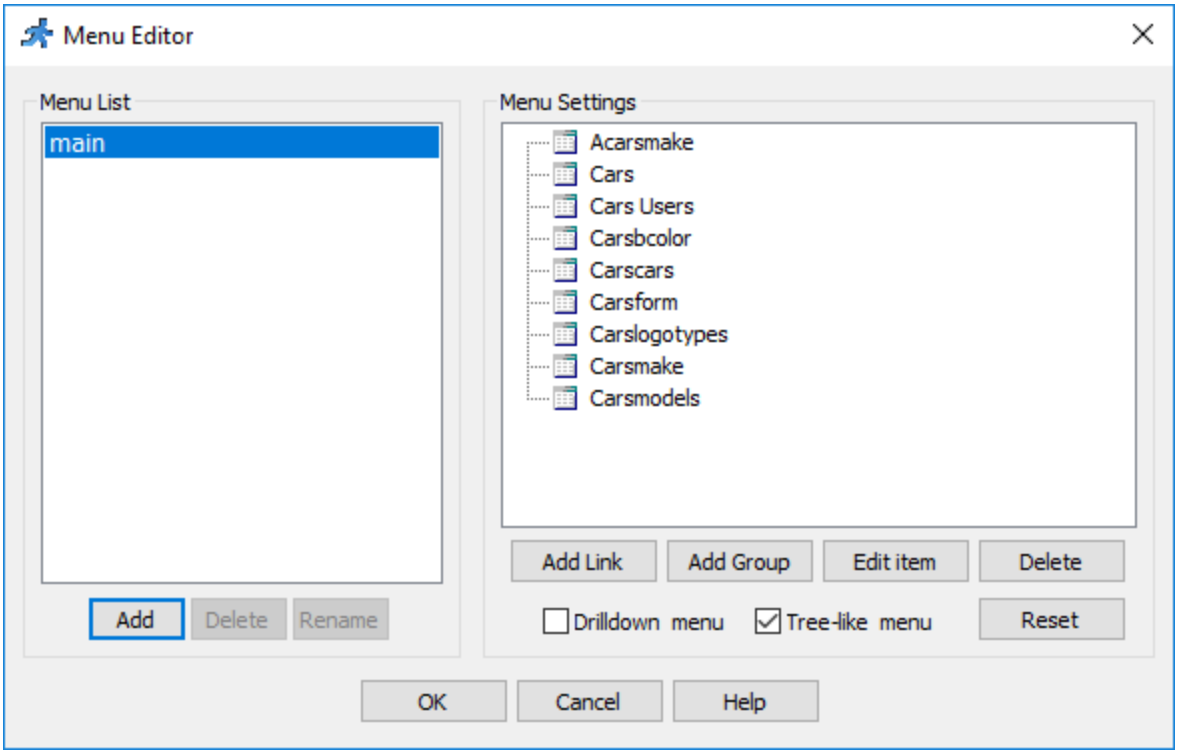
An example of a horizontal single-level menu:



An example of a vertical single-level menu:



To open the **Menu builder**, go to the **Editor screen** and click the **Menu builder** button. Alternatively, you can go to the Datasource tables screen and click the **Menu Editor** button.



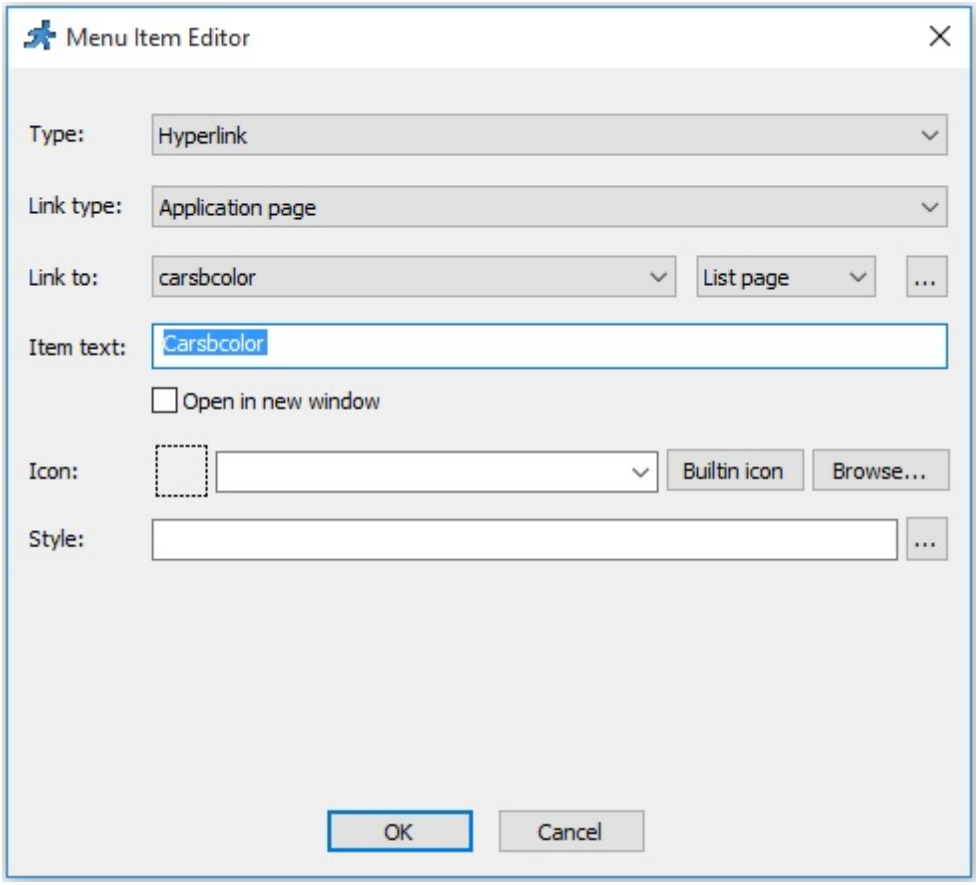
You can change the existing menu entries by double-clicking them, using the **Edit item** button, or right-clicking the entry and selecting **Properties**.

To add new groups and links, click the **Add Link** and **Add Group** buttons. Groups serve as folders for other groups and links.

> **Note**: you can always change the type of any menu entry into a group or a link.

The **Delete** button deletes the currently selected menu item. The **Reset** button resets the menu to its default state.

Link attributes



When adding or editing a link, you can configure the following link attributes:

1. **Type**. Select between a hyperlink or a group.

2. **Link type**. Select between an *PHPRunner* page or an external page.

3. **Link to**. If the **Link type** is set to external page, you can add a link to any web page. If the **Link type** is set to PHPRunner page, select the table within your project and then - one of the pages available for the selected table.

Usually, you can choose between:

- **List page** - displays the data from the table;

- **Add page** - allows to add new records to the table;
- **Search page** - provides an advanced search for the data in the table;
- **Print page** - prepares a printer-friendly page with the data from the table.

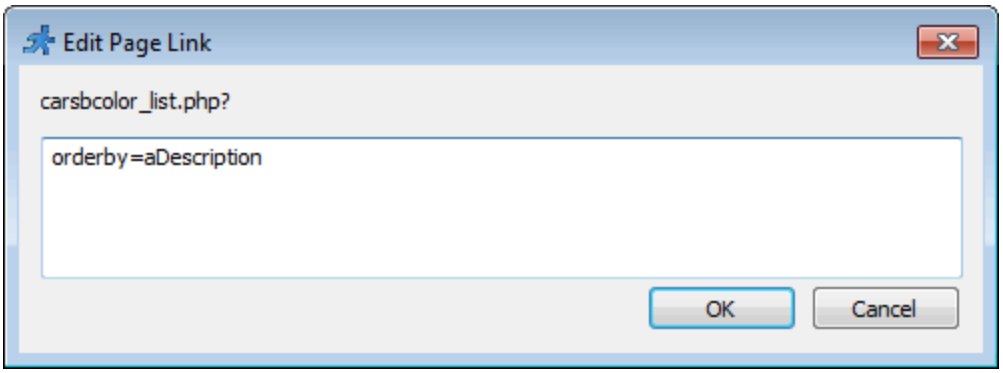
See Choose pages screen to learn more about the pages you can create/enable.

Click the '**...**' button next to the page type dropdown to set the link parameters.

If you, for example, want to display the page ordered by the *Sales* figures, type in the *orderby=dSales* parameter.

When working with complex parameters, the best way to proceed is to copy the parameter string from the application URL and paste it into the link parameter dialog.

Here is an example of a link parameter: *orderby=aDescription*. The resulting link looks like this: *carsbcolor_list.php?orderby=aDescription*.

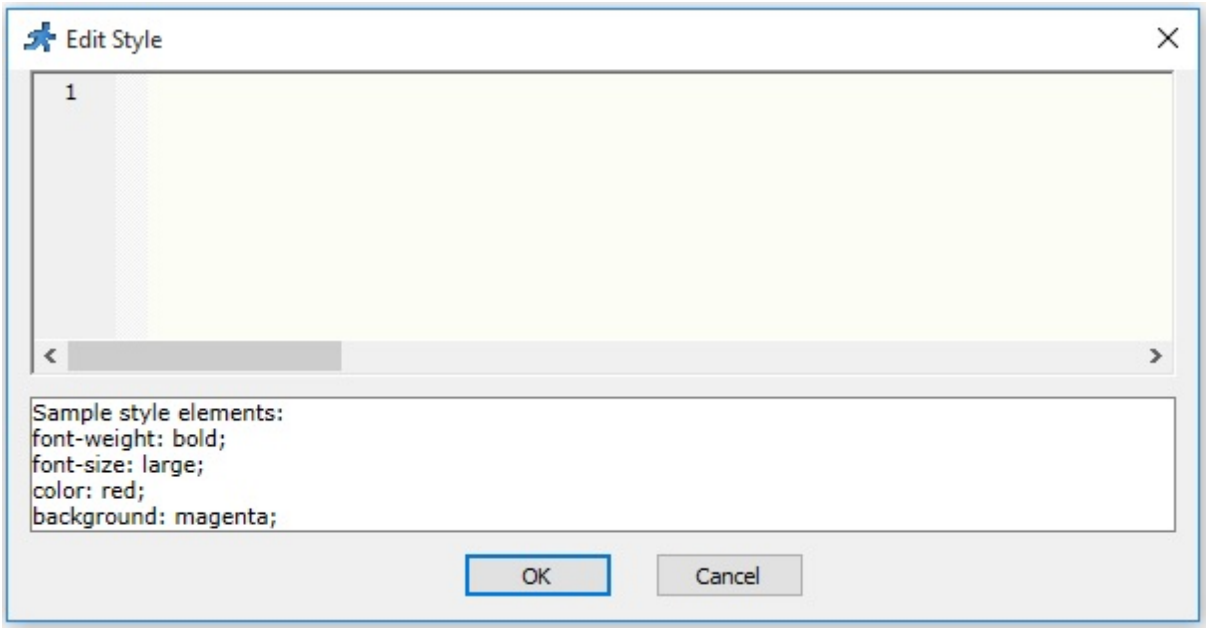


> **Note**: the user group permissions are only applied to the internal links (PHPRunner pages).

4. **Link text**. The **Link text** is displayed as the menu item title. If you turned on multilanguage support in your project, a **Multilanguage** button appears that allows translating the link text into several languages. See Miscellaneous settings to learn more about multilanguage support.

5. **Icon**. Select an image to be displayed next to the menu item title.

6. **Style**. Set the menu item text style by adding custom CSS properties. Use a semicolon to separate different properties.



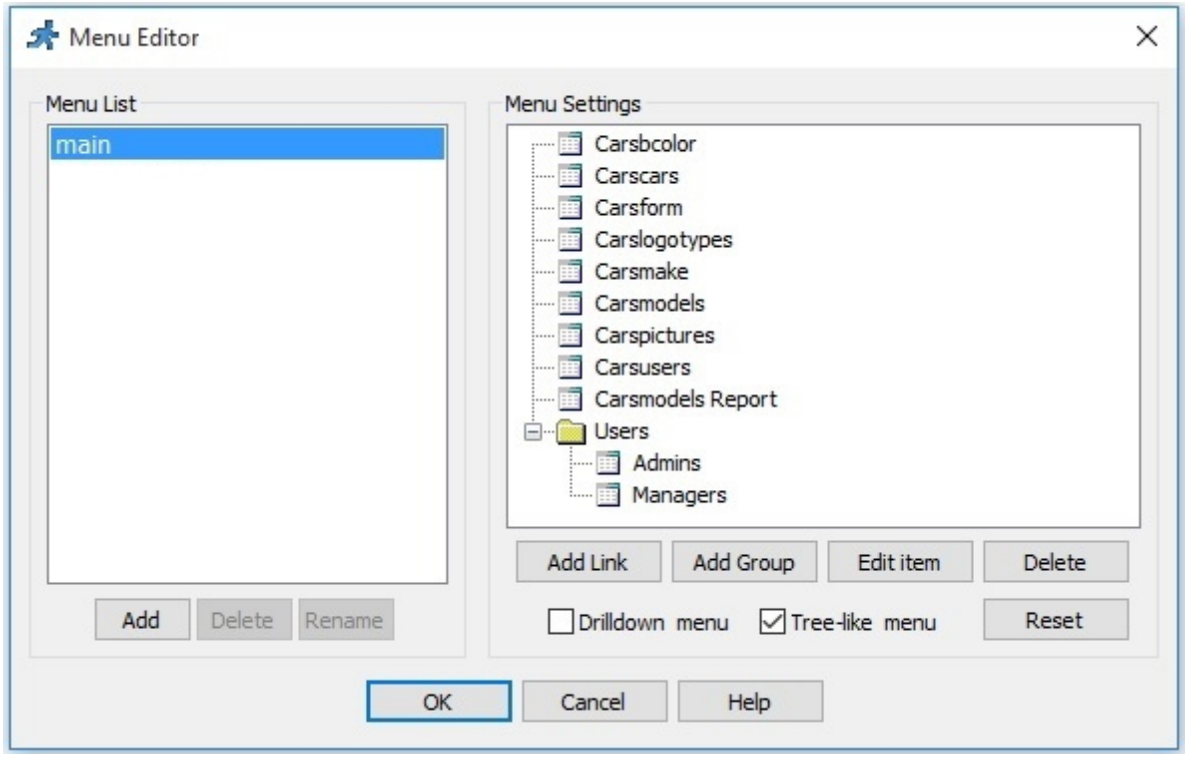
Here are several examples of how you can modify menu item text style:

- *font-size:12px; color:red*
- *font-size:200%; font-weight:bold*
- *font: 12px italic; border: solid*
- *color:rgb(255,0,0); background-color:black*
- *border: dotted red 2px*

The **Open in new window** option allows opening the links in a new browser window.

Tree-like menu

To create a cascading menu, create new groups, then drag-n-drop existing menu entries into the groups, or create new entries there.



For the vertical menu layout, the **Tree-like menu** checkbox allows displaying the expanding/collapsing cascade menu.

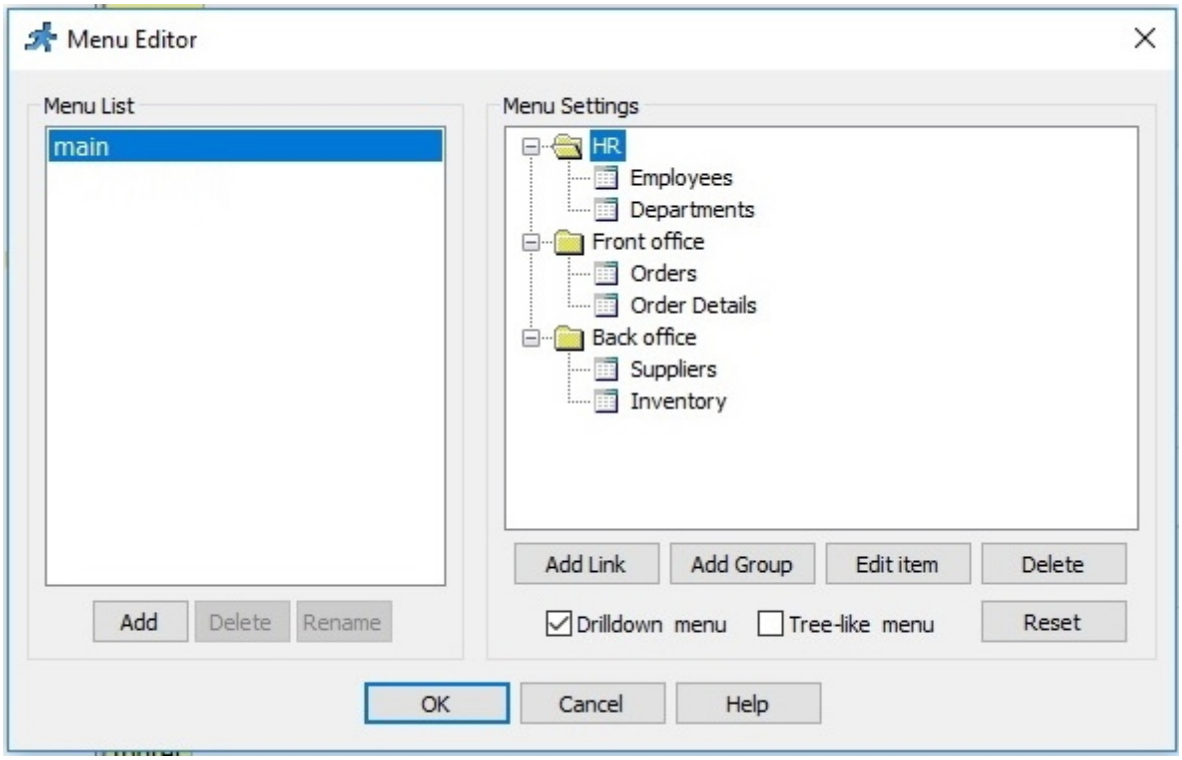
Drilldown menu

The **Drilldown menu** option might be useful for a multi-level hierarchical menu.

Instead of displaying the whole menu structure on each page, the drilldown menu shows only the current submenu and its children.

The **Breadcrumbs** control helps the user to determine the relative position of the page to the application hierarchy.

For example, you have the following menu structure:



- Without the **Drilldown menu** option, the same menu appears on each page:

- With the **Drilldown menu** option, this menu appears on the *Employees* page:

and this menu appears on the *Orders* page:

See also:

- Page layout and grid type
- Miscellaneous settings: Language settings
- Datasource tables
- About Editor
- Customizing CSS examples
- About Page Designer

2.17.4 Editing the <head> section

Starting with PHPRunner version 10.3, you can access and edit the *<head>* section of your HTML templates directly on the Editor screen.

What is a <head> section?

The *<head>* section is a container for metadata and is placed between the *<html>* tag and the *<body>* tag.

HTML metadata is data about the HTML document. Metadata is not displayed on the page.

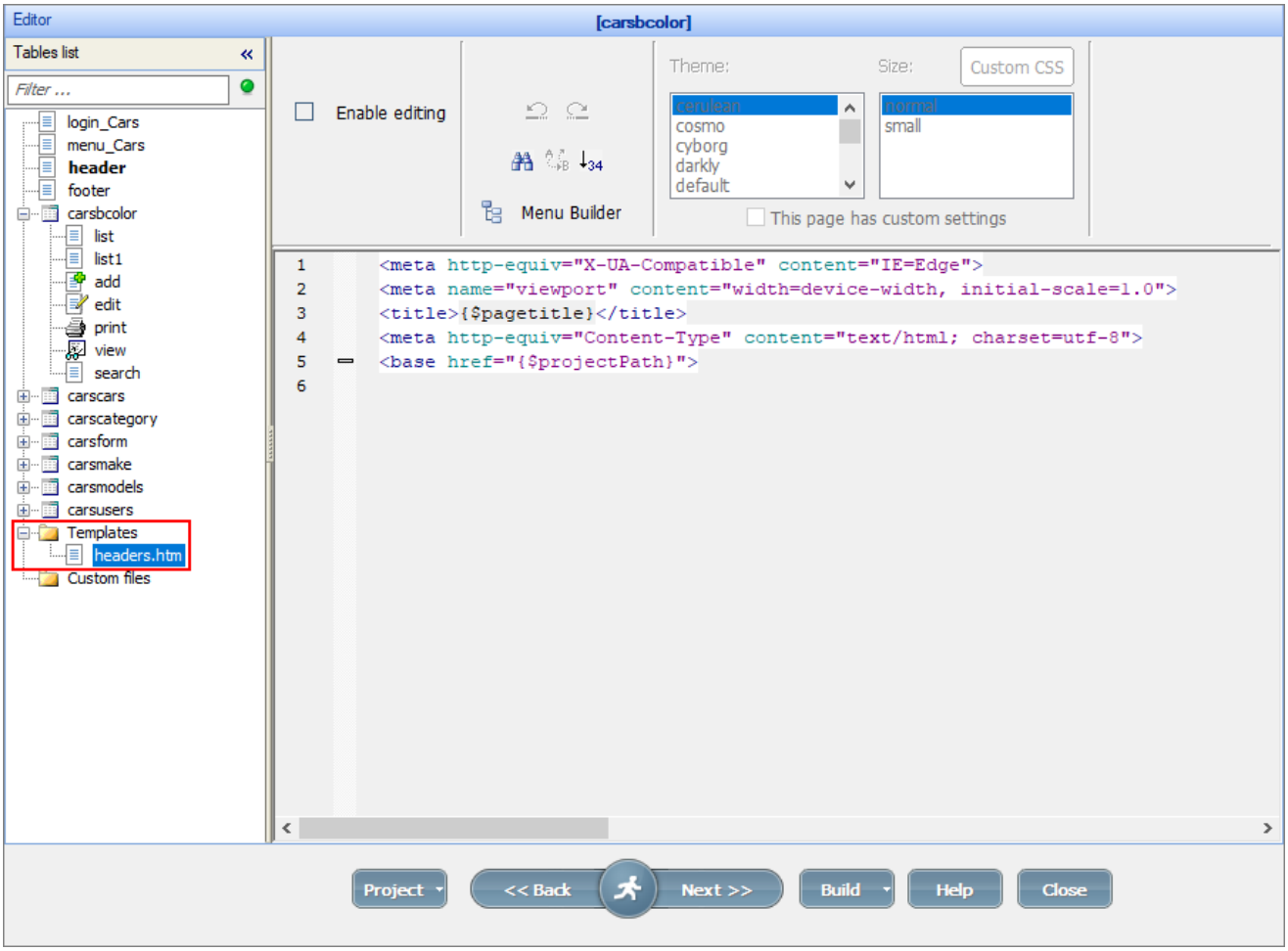
Metadata typically defines the document title, character set, styles, and other meta information.

The following tags describe metadata: *<title>*, *<base>*, *<link>*, *<meta>*, *<style>*, *<script>* and *<noscript>*.

To learn more about the metadata tags, visit https://www.w3.org/TR/html52/document-metadata.html

Accessing and editing the <head> section

You can access the *headers.htm* file responsible for the *<head>* section. Proceed to the **Editor** screen -> Templates -> *headers.htm* to do so.



To edit the *<head>* section of the project, click the **Enable editing** checkbox.

Example

If you want, for example, to add a favicon to the project, add the following line to the *headers.htm* file:

```
<LINK rel="shortcut icon" type="image/png" href="https://example.com/favicon.png"/>
```

Here is how a full *<head>* section may look like:

```
<!DOCTYPE HTML>
  <HTML lang="en">
  <HEAD>
    <META CHARSET="UTF-8">
    <BASE HREF="https://www.example.com/">
    <TITLE>An application title</TITLE>
    <LINK REL="STYLESHEET" HREF="default.css">
    <LINK REL="STYLESHEET ALTERNATE" HREF="big.css" TITLE="Big Text">
    <SCRIPT SRC="support.js"></SCRIPT>
    <META NAME="APPLICATION-NAME" CONTENT="Long headed application">

    <LINK rel="shortcut icon" type="image/png"
href="https://example.com/favicon.png"/>
  </HEAD>
  <BODY>
    ...
```

See also:

- About Editor
- Customizing CSS examples
- Menu Builder

2.18 Event editor

Quick jump

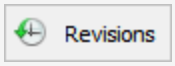
Events are fragments of PHP or JavaScript code that are executed automatically when certain conditions are met, for instance: a record was added to the database, or the user opened the **List** page. You can use sample event code snippets or write the code on your own.

Here are a few examples of events:

- send an email containing the data from a new record or showing changes to an existing record;
- save new data in another table;
- check if a specific record exists;
- limit the number of records a user can add.

There are two types of events: **Global** and **Table** events. Global events can be used with any table/view. Table events work with a specific table/view.

Event Editor toolbar description

Button	Description
	Allows adding one of the predefined actions.
	Erases the event code.
	Reverses the last operation.
	Reverses the last "*Undo*" operation.
	Checks the event code syntax.
	Opens the **Search and Replace** dialog. Note that you can search and replace content in all events as well as the selected one.
	Opens the **Find All** dialog.
	Displays the revision history panel.

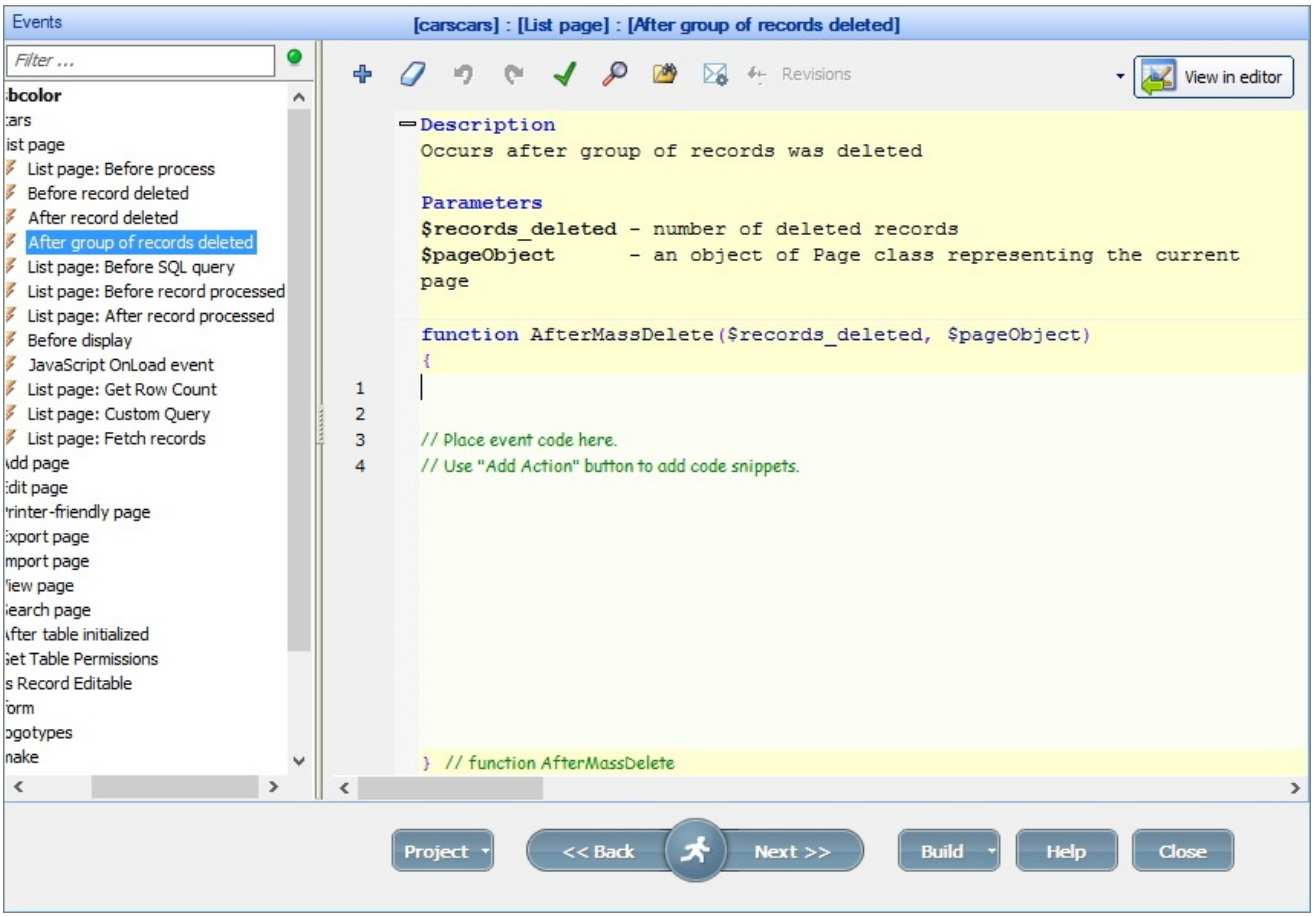
Working with events

To add an event:

1. Go to the **Events** screen.

2. Select an event from the tree in the left pane. To show only the events with code in them, click the  button. To filter the events by name, use the text field above the tree in the left pane.

Each event has a collapsible hint with the event description, function syntax, and a list of parameters. To hide the hint, click the button.



> **Note**: you can change the font size on the **Event** and Editor screens. Hold CTRL and scroll the mouse wheel to do so.

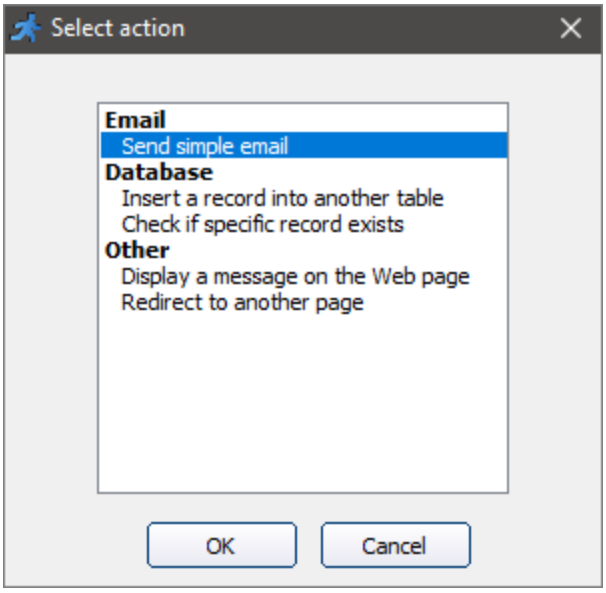
3. Add the code or click the  button to open the predefined actions popup.

To learn more, see these articles:

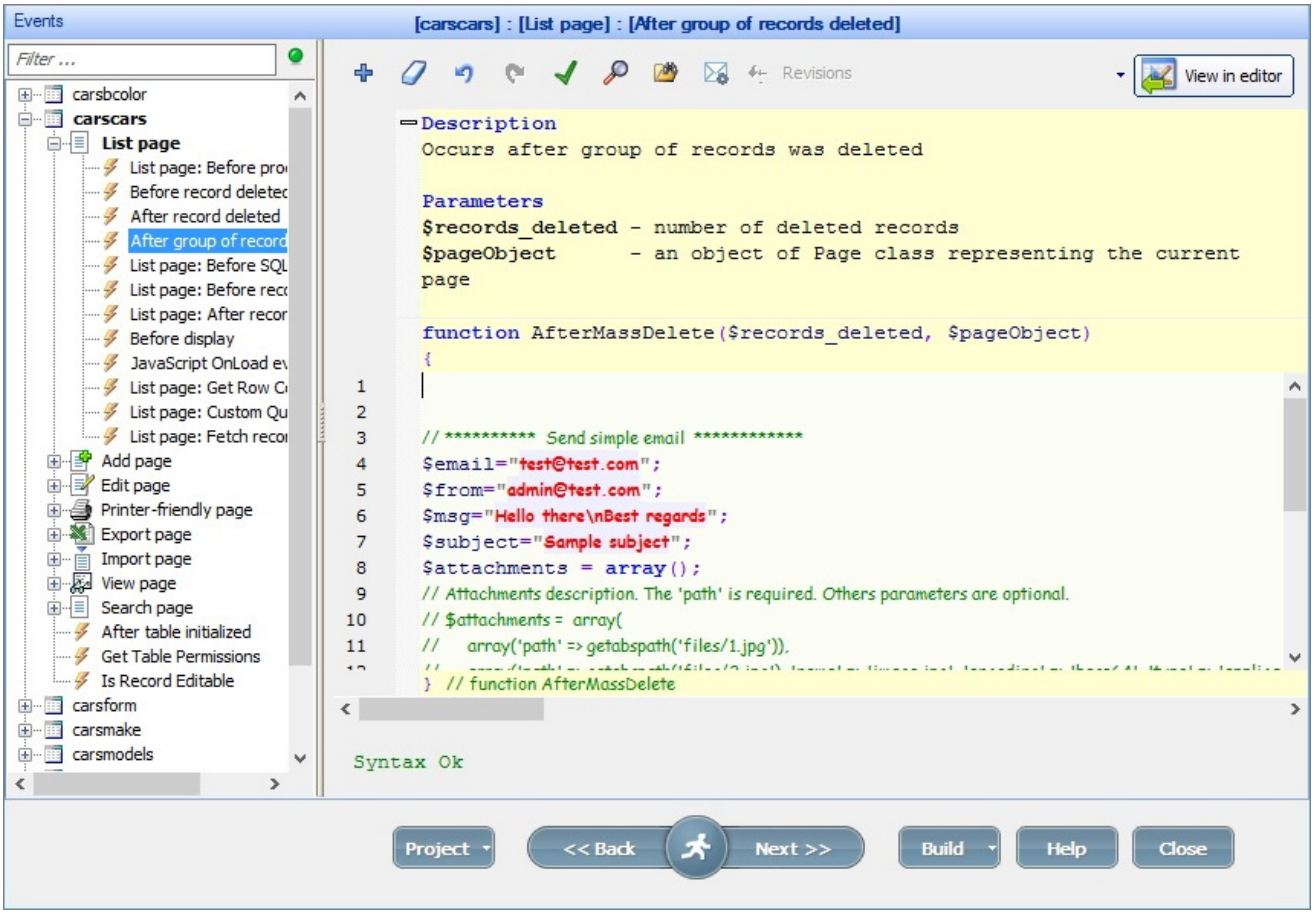
- Sample events - examples of commonly used events with code snippets.

- Common parameters you can use in the events.

4. Choose one of the predefined actions to add a code snippet to the event.



5. Modify the code snippet by inserting the necessary values instead of the ones colored in red.



In the example above, you need to insert:

- the **email address** - instead of "*test@test.com*" and "*admin@test.com*";
- the **email message** - instead of "*Hello there\nBest regards*";
- the **email subject** - instead of "*Sample subject*".

> **Note**: you can add more than one predefined action to an event.

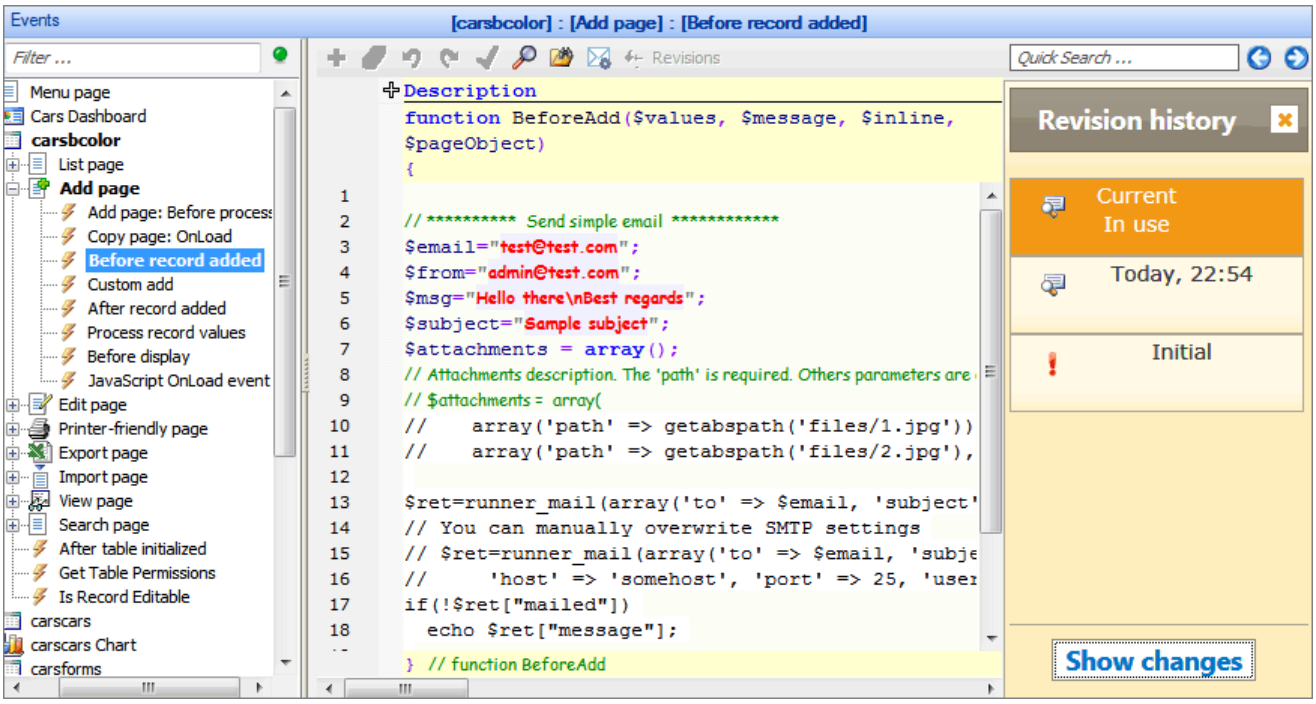
To edit the event, select an event and modify the code. To delete the event, select an event and click the  button.

Revisions

You can track the changes made to any event, review and restore any revision. To see the revision history, click the **Revisions** button.

Click **Show changes** at the bottom of the revision panel to see the changes between the current and previous revisions.

To restore any revision, select that revision and click **Restore revision**.

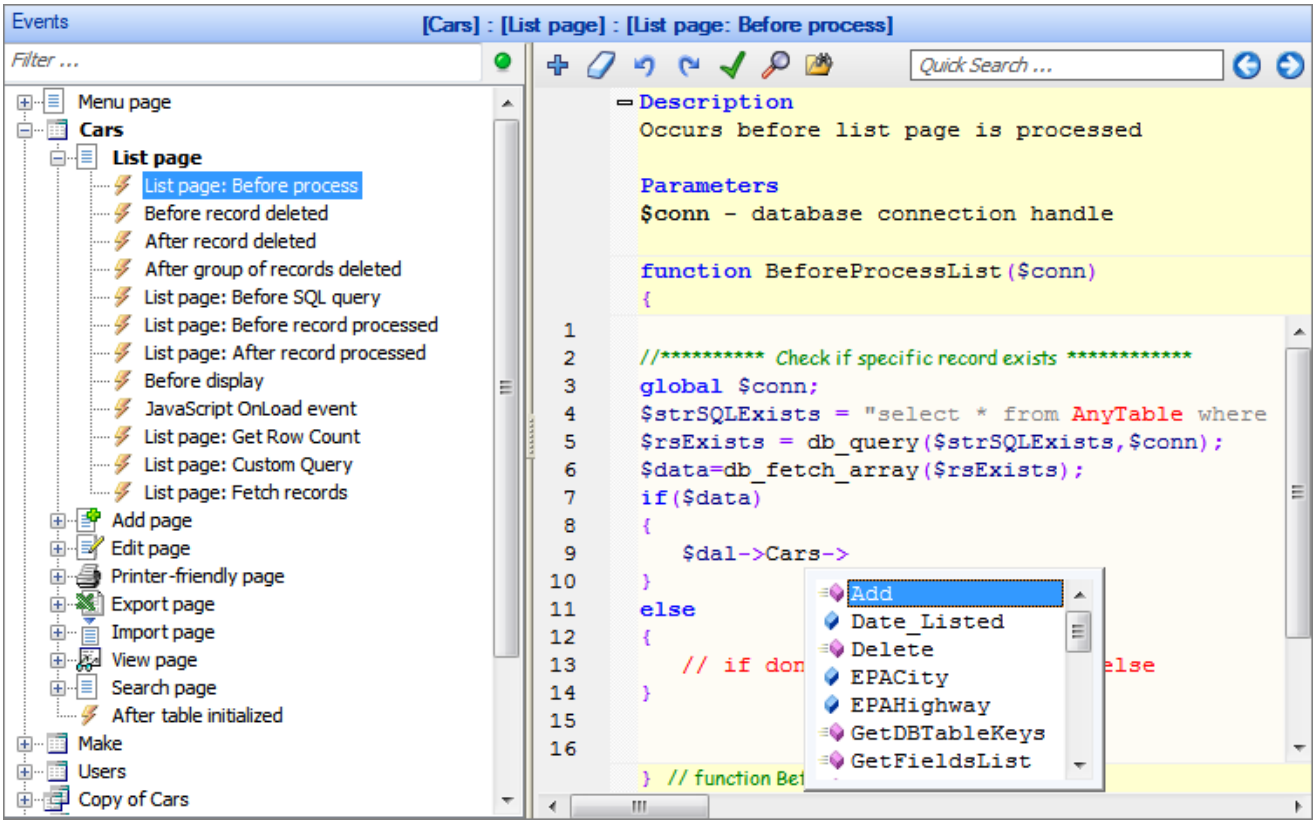


Intellisense

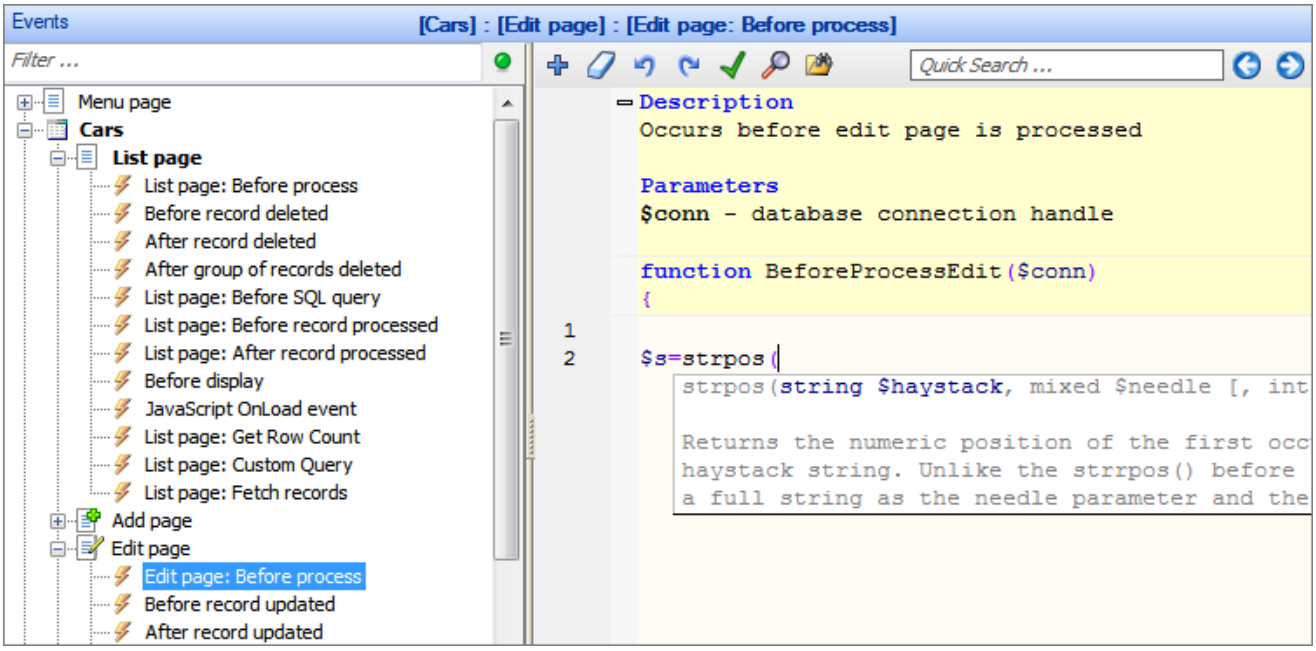
Intellisense is a convenient way to access the descriptions and variables of the functions. The **Event** editor recognizes the functions and variables and shows the function description or the list of available variables to choose from in a popup.

Here are a few examples:

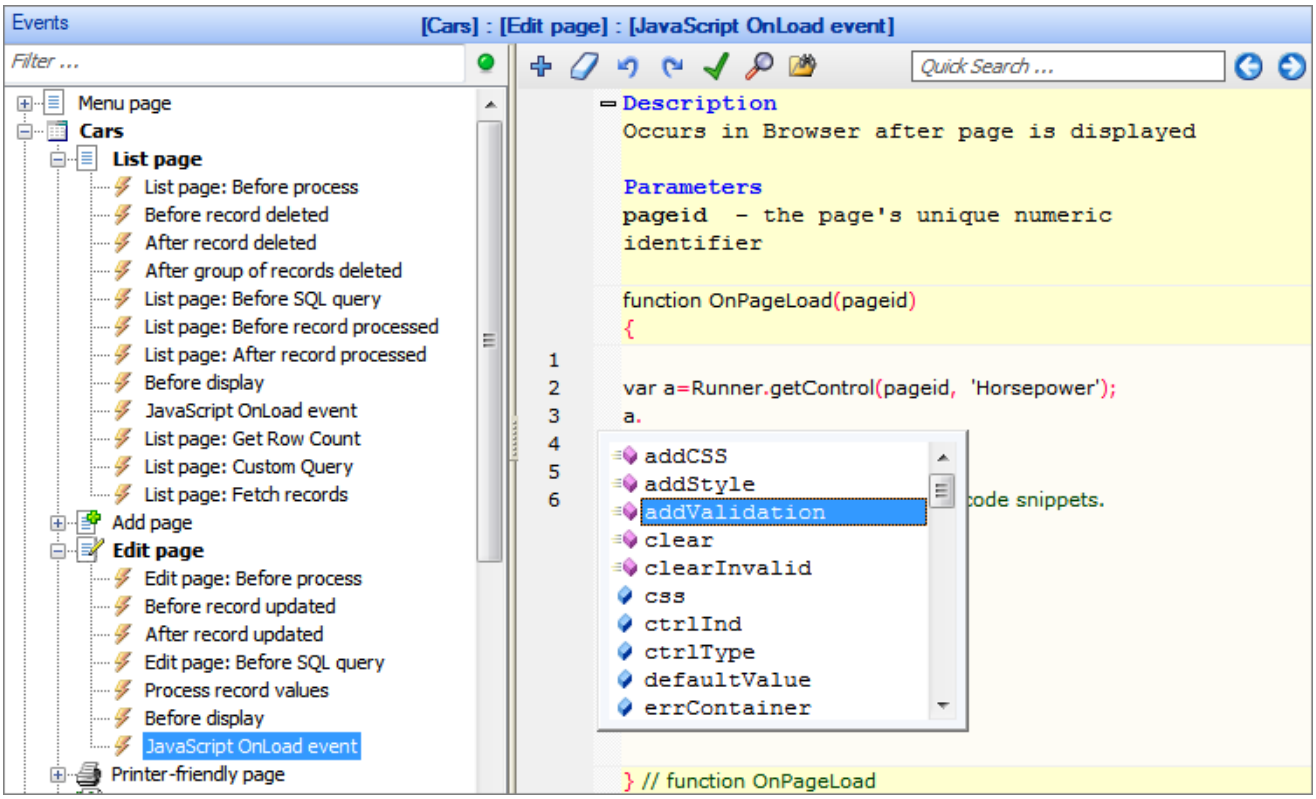
Data Access Layer, database field names



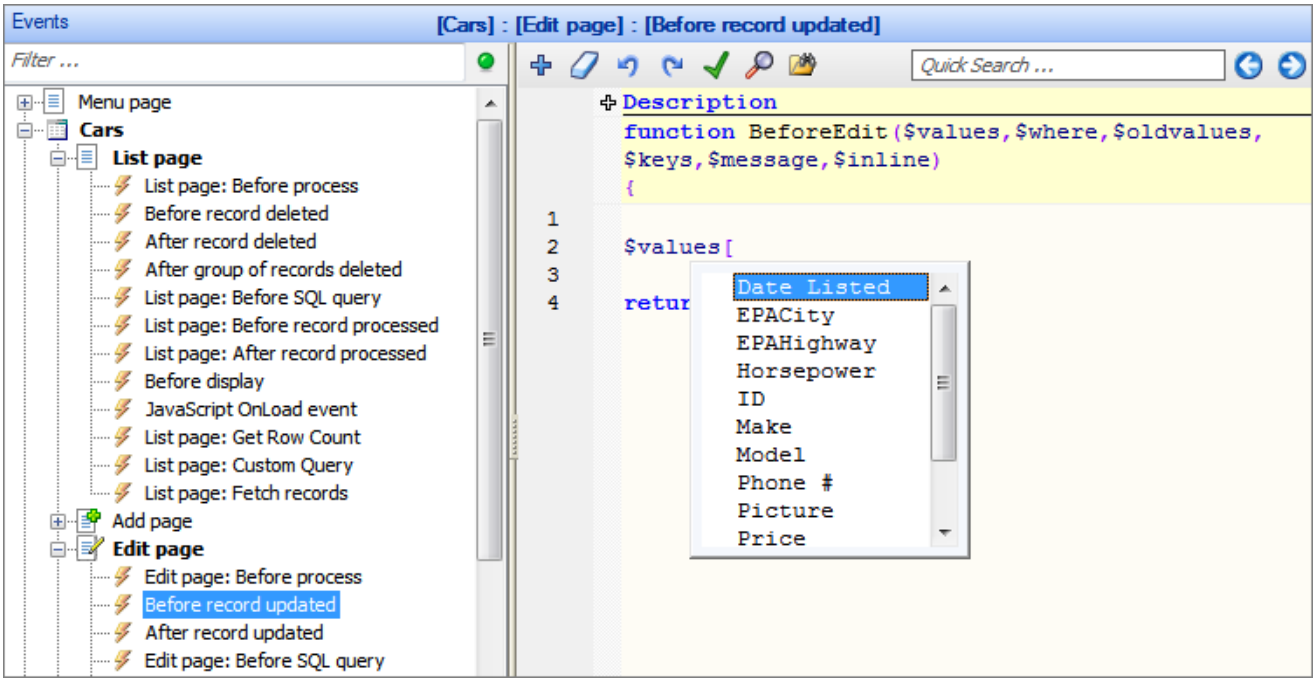
PHP functions



JavaScript API functions



Field names

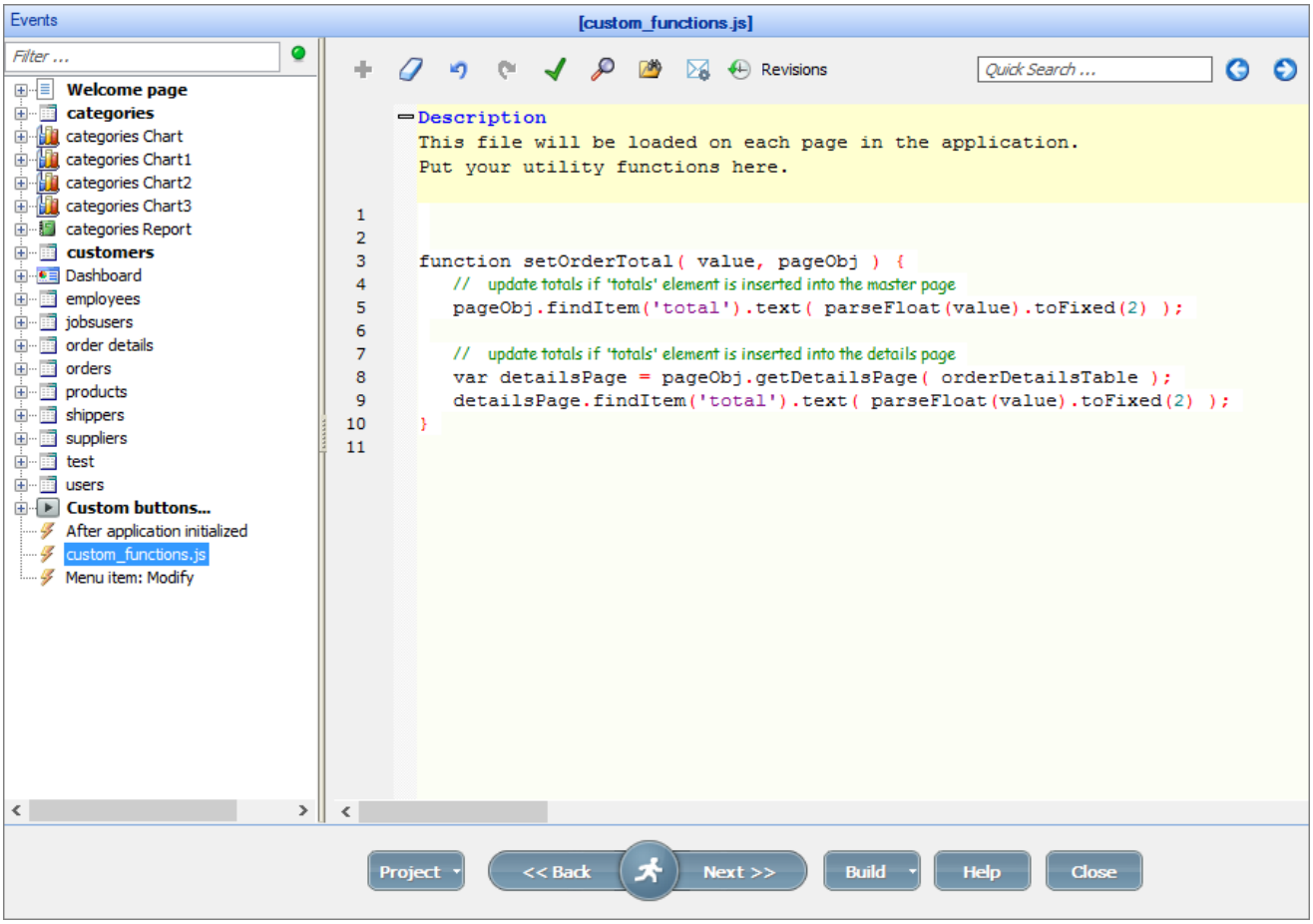


Custom_function.js

Starting with PHPRunner version 10.3, you can find a *custom_function.js* file in the event tree in the left pane. This file is loaded on each page in the generated application.

Custom_function.js is best suited for utility functions. For example, you can add an *OrderTotal* calculation to this file to Show order total on the Edit page as the details table is updated.

Here is how the *cuctom_function.js* window looks like on the **Events** screen:



See also:

- Predefined actions
- Sample events
- Global events
- Table events
- Page life cycle overview
- Common event parameters
- Field events
- Tri-part events

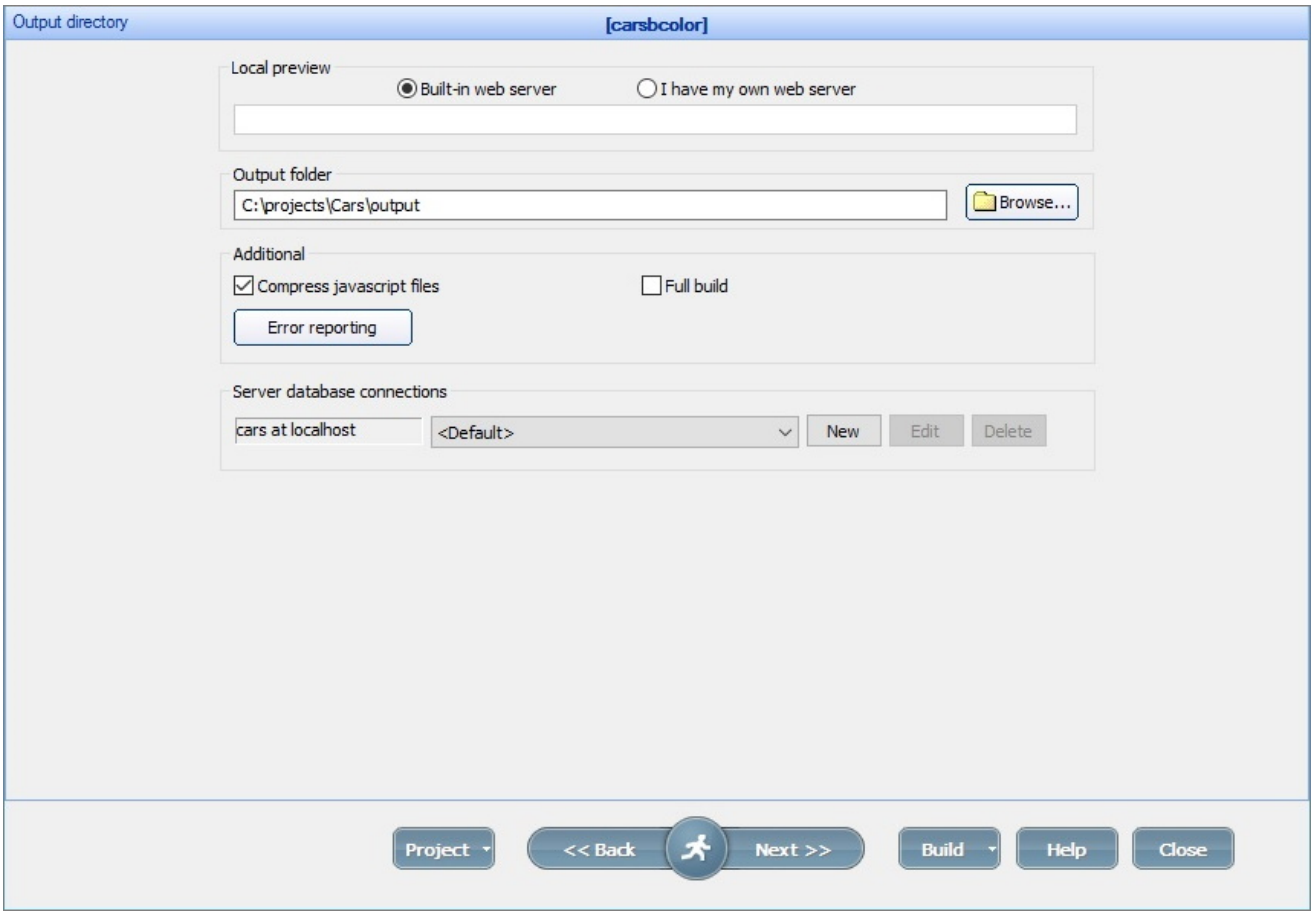
2.19 Output directory settings

The **Output directory** screen allows you to select the output directory and configure additional settings.

Preview

PHPRunner comes with a built-in web server (Apache).

In most cases, you can leave the default settings unchanged, click **Build** and proceed to **View in browser** on the next screen.



Note: the built-in web server doesn't interfere with your existing web server if you have one.

If you want to view the generated application using your web server, switch to **I have my own web server** option and enter the URL manually.

You should change the output directory as well to one of the web server subdirectories (i.e., *C:\xampp\htdocs\project1* if you use XAMPP).

> **Note**: if you don't have a web server and want to install one, see How to install local server.

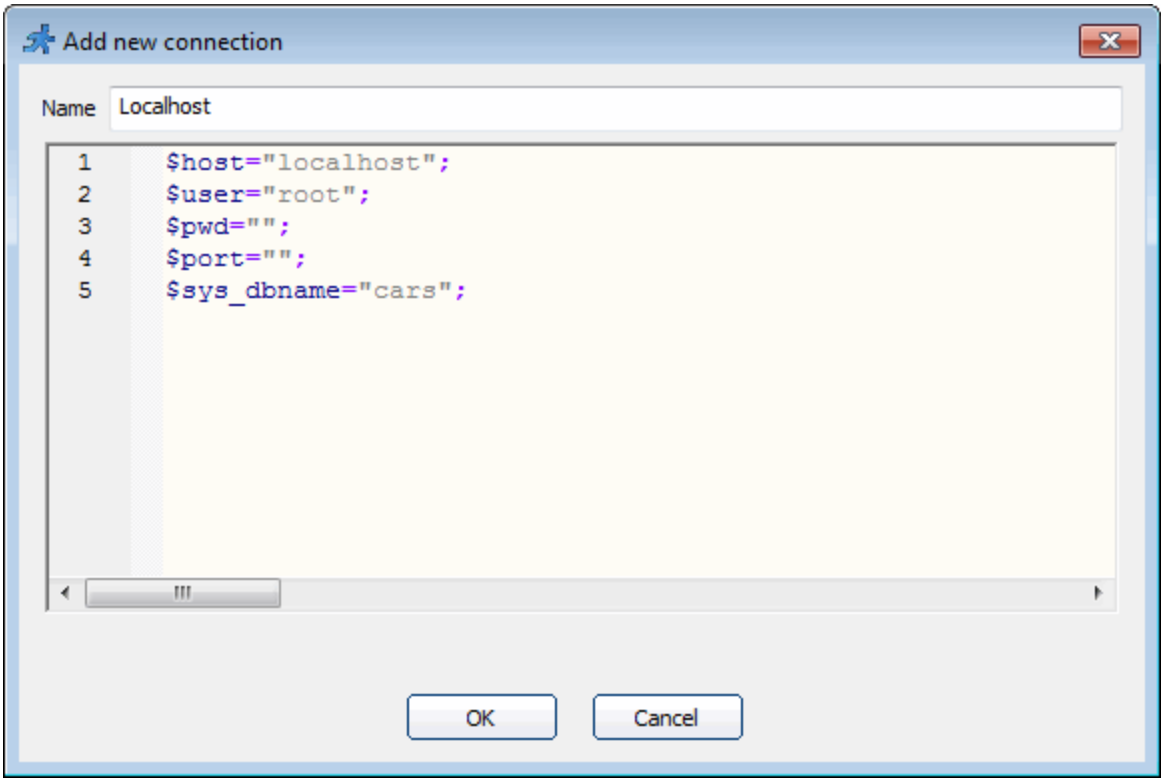
Select the **Full build** checkbox to perform a full build of the project. Otherwise, the partial (faster) build is performed.

The **Compress javascript files** option allows compressing the generated JavaScript files leading to faster load times. Unless you need to debug the JavaScript code, leave this option on.

Server database connections

You can create several database connections and select the required one before building the project. For example, you can use the *<Default>* database connection for local testing and create a new database connection for uploading files to the production server.

To create a new database connection, click the **New** button. Then specify the connection name and settings. Click **OK**.



If you are using MySQL locally, you can configure the connection settings like this:

```
$host="localhost";
$user="root";
$pwd="";
$port="";
$sys_dbname="cars";
```

When uploading files to the server, you may use the following settings:

```
$host="localhost";
$user="mike375";
$pwd="dcHd*eS2";
$port="";
$sys_dbname="mike375_cars";
```

Note: you can't use any code in your connection settings.

How to set local time on a built-in web server

To set the timezone on your local machine when using the built-in web server, you need to adjust the timezone parameter in C*:\Program Files\PHPRunner10.3\EmbeddedServer\php\php.ini.template* file.

1. Open the file *C:\Program Files\PHPRunner 10.3\EmbeddedServer\php\php.ini.template*.

2. Find the line.

```
date.timezone = 'UTC'
```

and modify the timezone, for example:

```
date.timezone = 'EDT'
```

3. Save the file and restart PHPRunner.

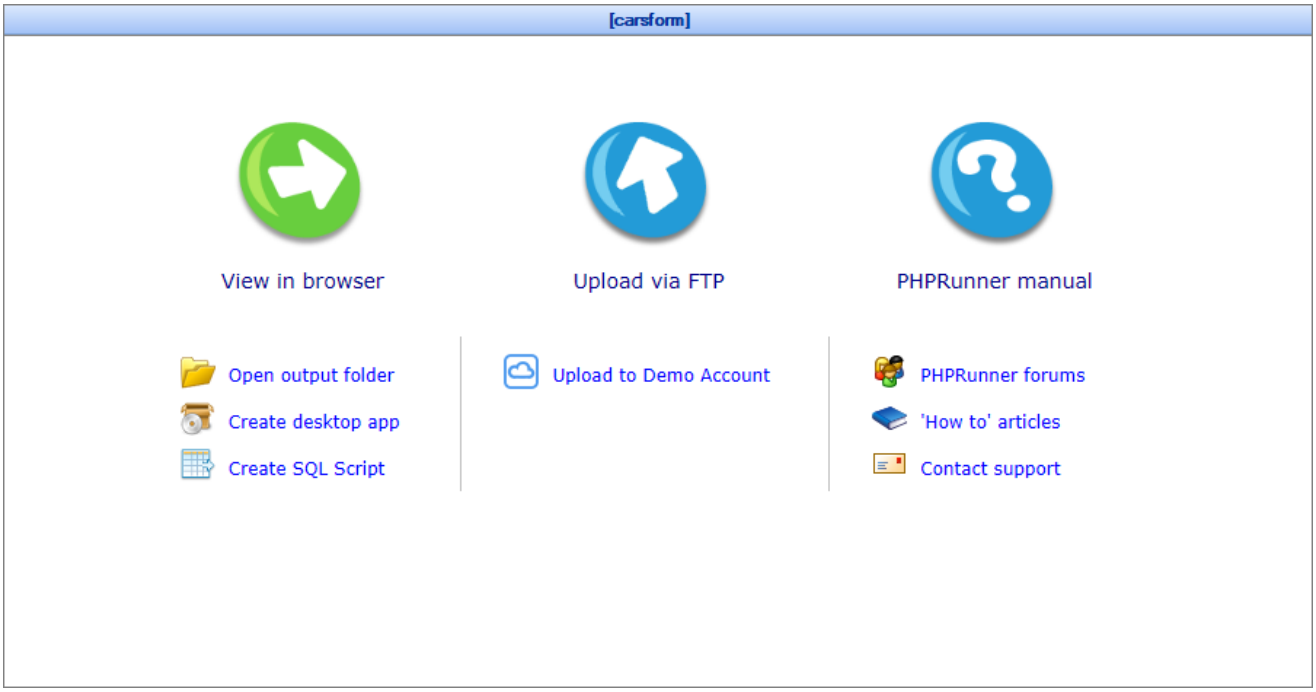
You can find the timezone abbreviations at http://www.timeanddate.com/library/abbreviations/timezones/.

See also:

- After you are done
- Connection to the Database
- FTP upload

2.20 After you are done

After successfully building the project, you have the following options:



Testing the application locally

- **View in browser** - run the generated application in the browser. This option will not work if you have enabled the **Connect using PHP** option on the Connecting to the database screen.

If for some reason the built-in web server doesn't start (i.e., nothing happens after you click the **View in browser** button), make sure your antivirus or firewall software doesn't block the web server. Turn it off and try **View in browser** one more time. The built-in web server uses ports 8085-8090, so you may want to open them with your firewall software.

- **Open output folder** - open Windows Explorer to browse the folder with the generated application files.

- **Create desktop application** - build and package the desktop application.

- **Create SQL script** - create an SQL script for the tables/data transfer to another server.

Testing the application on the remote Web server

- **Publish via FTP** - upload the files to the remote Web server with the built-in FTP client.

- **Demo Account** - you can open a free Demo Account and publish your project to our demo web server with a single click.

If you prefer to use third-party tools to upload the generated application, check the following topic:

- Uploading the generated application using a third-party FTP client.

If you have any problems using PHPRunner or wish to learn advanced techniques, you can check PHPRunner articles or ask your questions on the support forum.

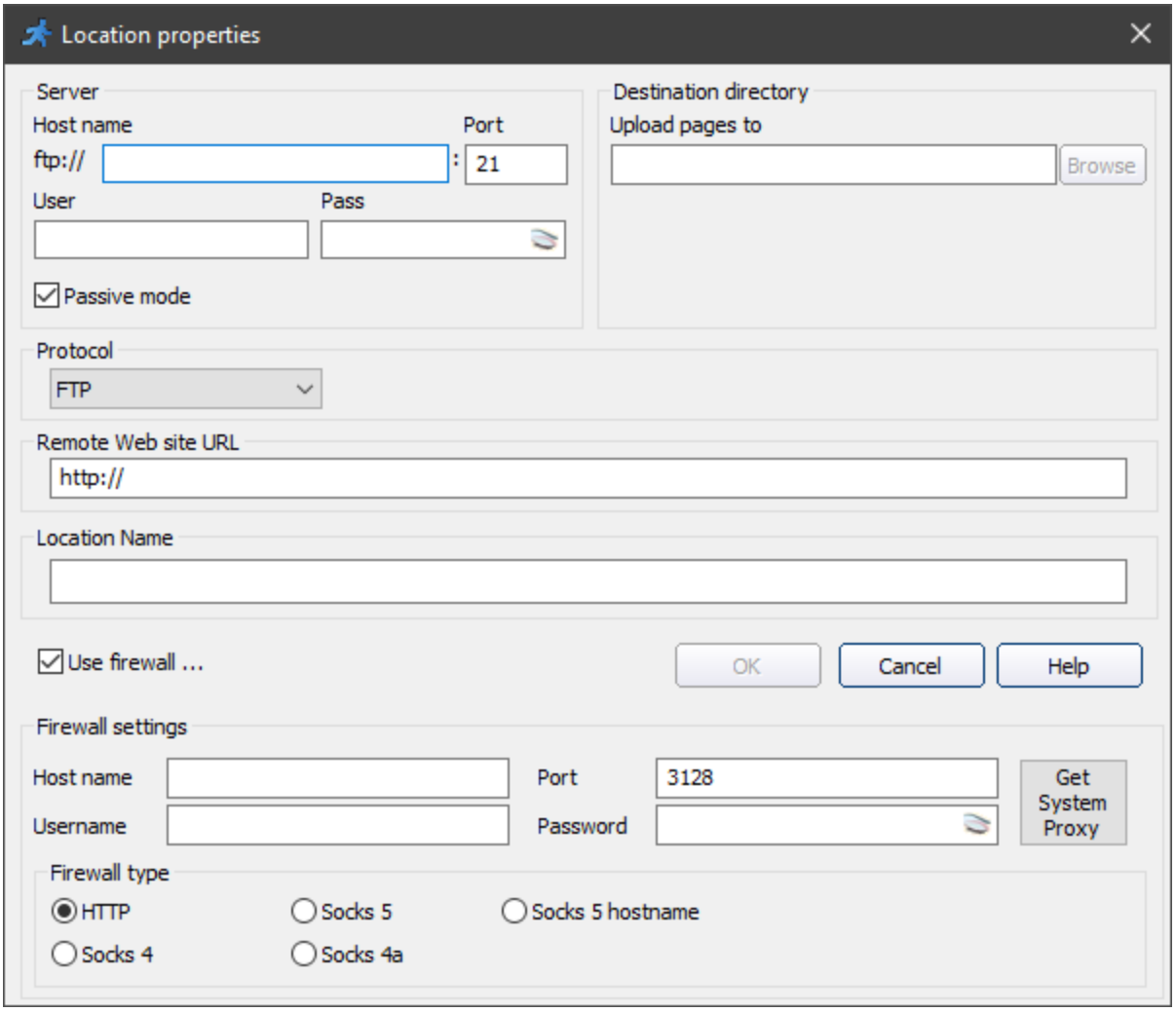
You can also contact our support team.

See also:

- Miscellaneous settings
- Security
- Demo account
- Testing the mobile version of your web app
- FTP upload
- Desktop applications

2.21 FTP upload

To upload files to an FTP server with a built-in FTP client, you need to configure the FTP location properties. Click the **Upload via FTP** button on the Your project was built successfully screen to open the FTP location properties window.



Enter the *Location name*, *Host name*, *Username*, and *Password* to enable the **Browse** button. Click **Browse** to choose the directory to upload generated files to. Choose between FTP, SFTP, and FTPS protocols.

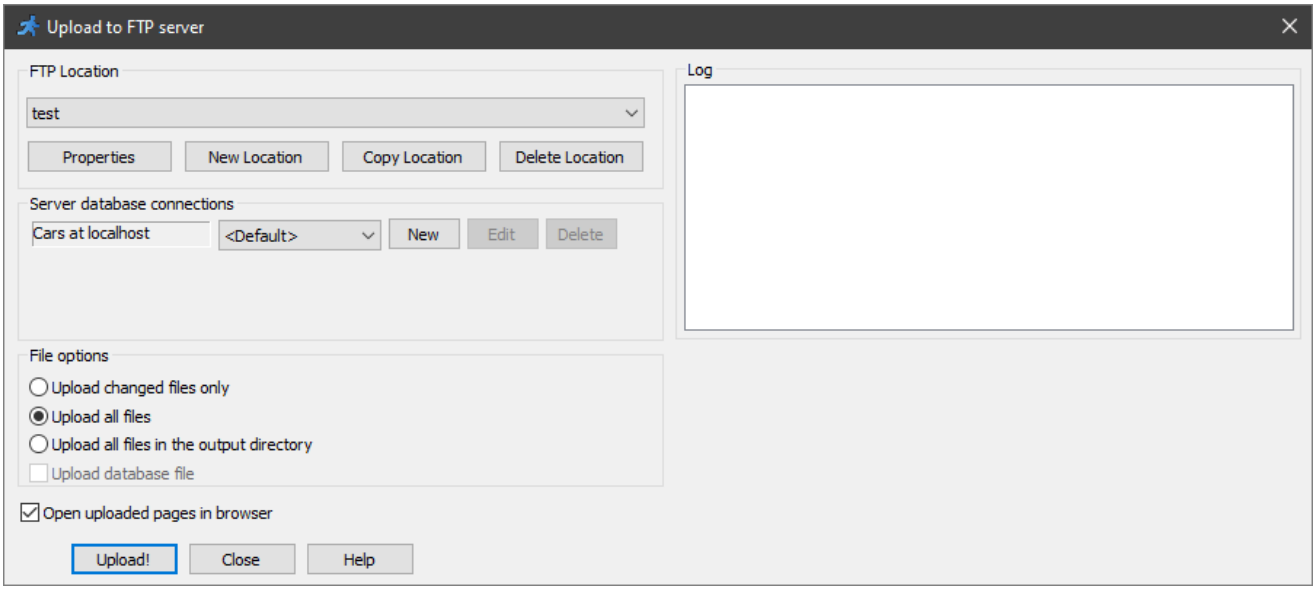
> **Note**: if your web hosting provider has a designated folder for databases, you need to move the database to that directory. The connection string is then updated to match the new database folder automatically.

If you have connected to the database successfully but can't create a folder or upload files, you can use the **Passive mode**.

Fill in *Remote Web site URL*, to open the application in the browser for test purposes.

Select the **Use firewall** checkbox to show the **Firewall settings**.

To upload the files, choose the FTP location and click the **Upload** button. You can stop the upload at any time by clicking the **Stop** button.



You can choose between the following **File options**:

- **Upload changed files only** - to upload the files that were changed since the previous upload.
- **Upload all files** - to upload all generated files.
- **Upload all files in the output directory** - to upload all files including the files that were not created by PHPRunner.

> **Note**: you can increase the upload speed by changing the number of FTP threads. Click Project -> Settings to open PHPRunner settings and change the number of FTP threads.

See also:

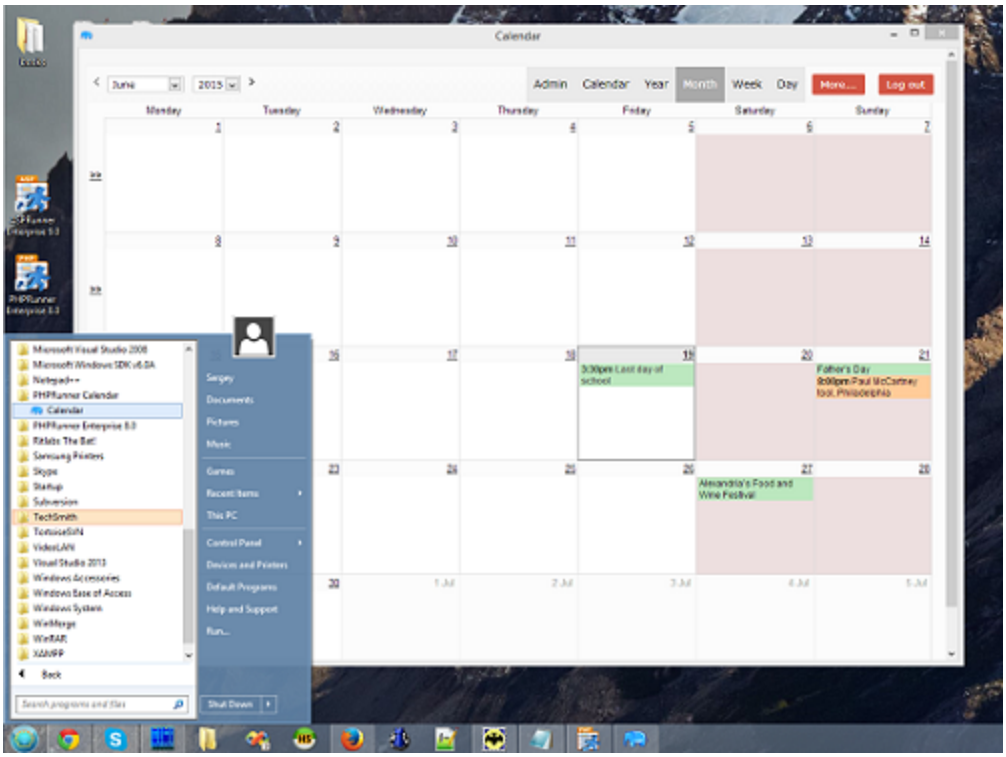
- After you are done
- Using a third-party FTP client to publish the application

- Demo account
- Testing the mobile version of your web app
- Desktop applications

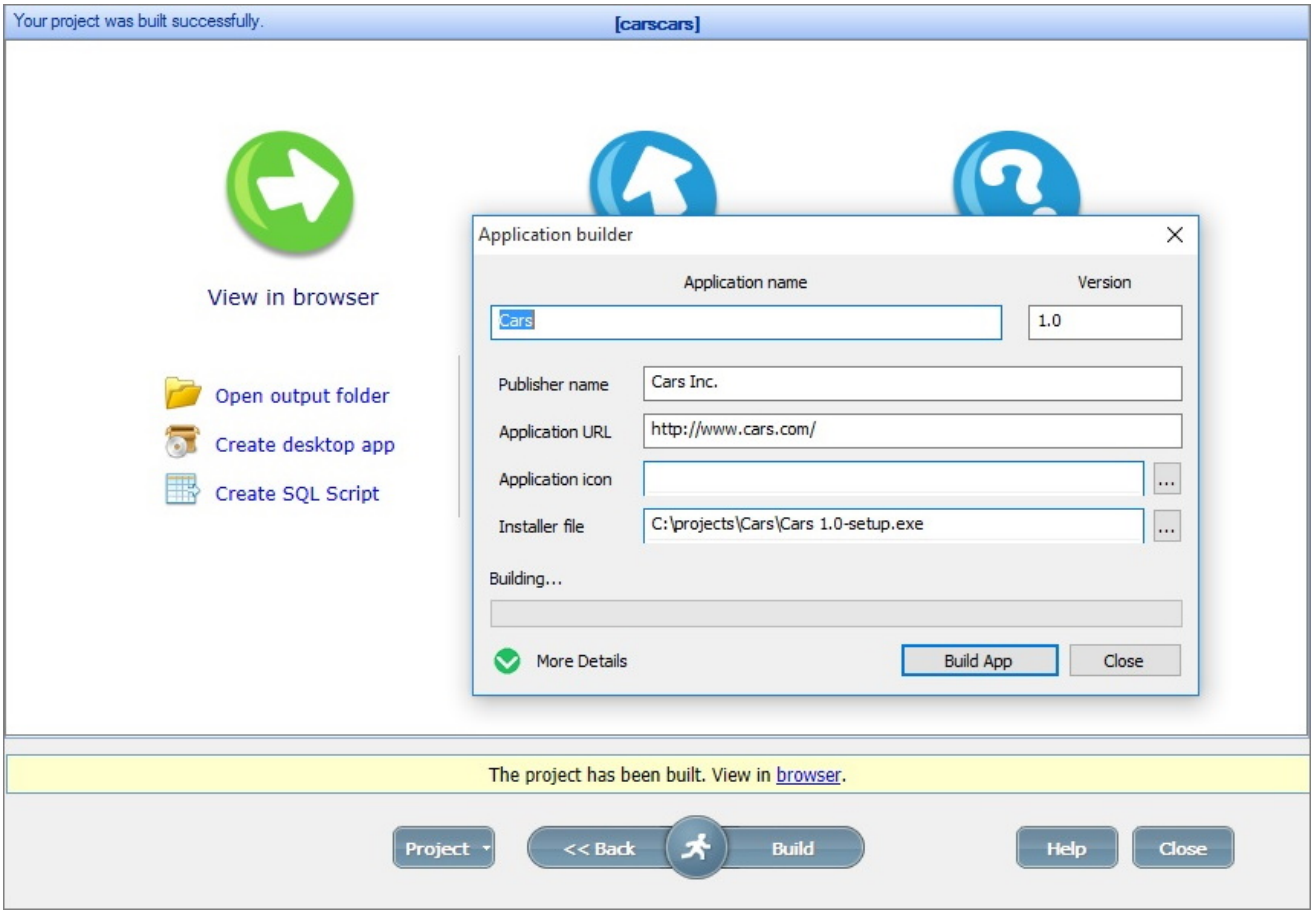
2.22 Desktop applications

PHPRunner can build and package desktop applications as well as web applications. These applications can be installed on any Windows machine and require no Internet connection to set up.

Here is an example of a PHPRunner desktop application:



To make a new application, click **Create desktop app** after building the project.



Note: the *Publisher name*, *Application URL*, and *Application icon* fields are optional.

If your desktop application displays an error message, check the PHP Desktop knowledge base for additional info. Look up the error message and take the suggested steps.

How it works

First of all, the **Desktop app** functionality is built on top of the open-source PHP Desktop project. PHP Desktop packages the following software components together:

- webserver (Mongoose);
- PHP;
- Chrome browser.

When you start the PHP Desktop application, it starts the webserver specifying the *www* directory as the website root folder. Then it starts the Chrome browser and points to the home page of the webserver. The browser is modified to hide the menus, navigation buttons, and address bar. You only see the application itself.

PHPRunner takes one extra step packaging the whole application into a single installation file using Inno Setup. The whole packaging process is described in the Create Your First Desktop Application With PHP And PHP Desktop tutorial.

Limitations

- Windows only;

PHP Desktop runs on Windows only.

- PHP only;

PHP Desktop works with PHP only.

- The installer is not signed;

If you want to make the desktop version of your app available worldwide, you need to sign the installer.

- Internet connectivity is required for remote database applications;

The desktop application is no different from the web application in terms of the database connection. If your application uses a remote database, you do need an Internet connection to work with the database.

Some typical use cases for the desktop applications

- You are a PHP developer who needs to create a desktop application. The best choice is to use SQLite or MS Access database in this case.
- You need to create an application that doesn't use the database at all. For example, the Euro 2016 desktop application. It connects to the football-data.org API, retrieves the data, and displays it. You don't need a website or a database in this scenario.
- You need access to the hardware or file system. Using a desktop app, you can access the devices connected to the end-user machine, work with the file system, use COM objects, etc. You can do everything the typical desktop application can.

- You can use your desktop application to connect to the remote database. When you do not have a website or do not want to maintain a website - all you have is a database like MySQL or Amazon RDS with remote access. In this case, the desktop application can be useful as well.

See also:

- After you are done
- FTP upload
- Using a third-party FTP client to publish the application

3 Advanced topics

3.1 Events

3.1.1 Predefined actions

3.1.1.1 About the Predefined actions

Quick jump

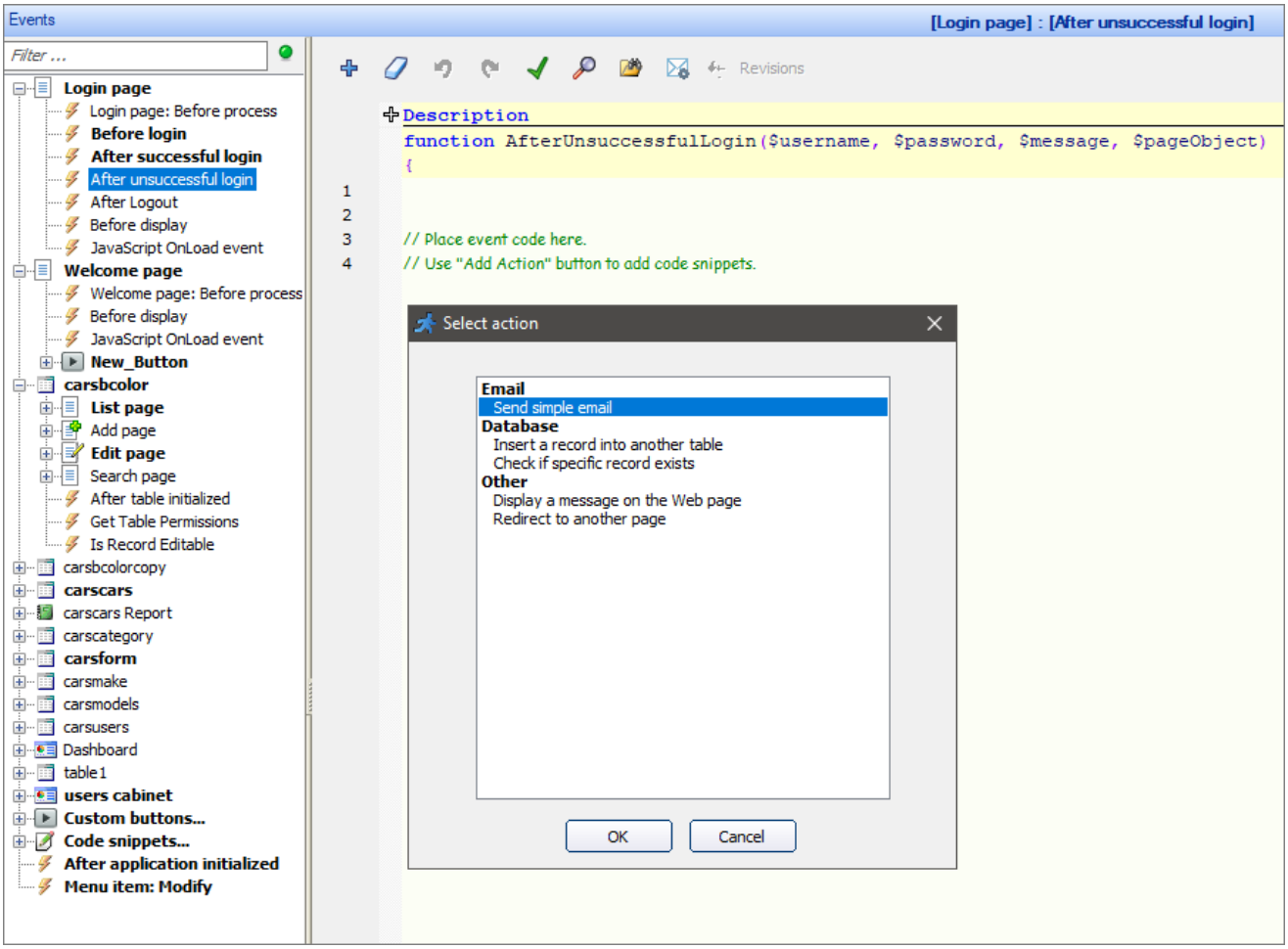
What are the Predefined actions?

Action in the PHPRunner is a piece of code that you can assign to Events. **Actions** determine what exactly occurs within the Event.

The PHPRunner comes with the set of the **Predefined actions** - pre-made code samples which you can customize.

> **Note**: You can write your code and combine it with the pre-made samples.

Add, edit, and modify **Predefined actions** on the Events screen.



Note: You cannot use **Predefined actions** with JavaScript onload events.

A list of the Predefined actions

Predefined actions are divided into three groups:

1. **Email** - these actions allow sending various emails in different conditions;
   - Send simple email
   - Send email with new data
   - Send email with old data

2. **Database** - these actions allow different operations in the Database;
   - Save new data in another table
   - Save old data in another table
   - Insert a record into another table
   - Check if specific record exists

3. **Other:**
   - Display a message on the Web page
   - Redirect to another page.

> **Note**: Each of the predefined actions works with specific events only. To learn more, see the respective articles.

See also:

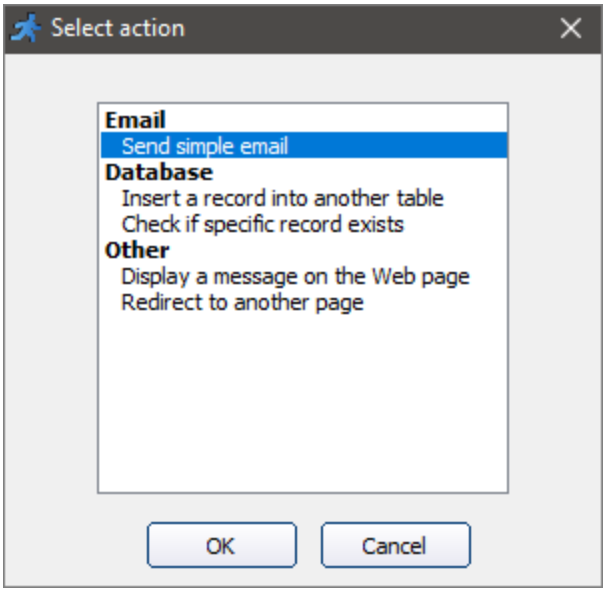
- Global events
- Table events
- Sample Events
- JavaScript Onload event
- Page life cycle overview

3.1.1.2 Send simple email

Send simple email action allows sending an email from one email address to another.

You can edit several parameters: the text and the subject of the message, the sender and recipient email addresses. You can also attach files of any format supported by your email service provider.

This action is available at any event except JavaScript onload events.



Note: To use this action, you have to set up the Email settings. Press this button on the toolbar or write the code manually.

Change red values to adjust the action code to your project.

```
// ********** Send simple email ************
$email="test@test.com";
$from="admin@test.com";
$msg="Hello there\nBest regards";
$subject="Sample subject";
$ret=runner_mail(array('to' => $email, 'subject' => $subject, 'body' => $msg,
'from'=>$from));
if(!$ret["mailed"])
   echo $ret["message"];
```

You can send HTML emails as well:

Note: Assign HTML code to your `$msg` variable and replace the 'body' key with `'htmlbody'` key.

```
// ********** Send HTML email ************
$email="test@test.com";
$from="admin@test.com";
$msg="<b>Hello there</b><br>Best regards";
$subject="Sample subject";
$attachments = array();
$ret=runner_mail(array('to' => $email, 'subject' => $subject, 'htmlbody' => $msg,
'from'=>$from, 'attachments' => $attachments));
```

To learn how to customize email templates, see Registration and passwords: Email templates.

See also:

- runner_mail function
- Send mass email to all users
- Send an email with attachment
- Send an email to selected users
- Send email with new data
- Send email with old data
- Email selected records
- How to email selected records as separate PDF files

3.1.1.3 Send email with new data

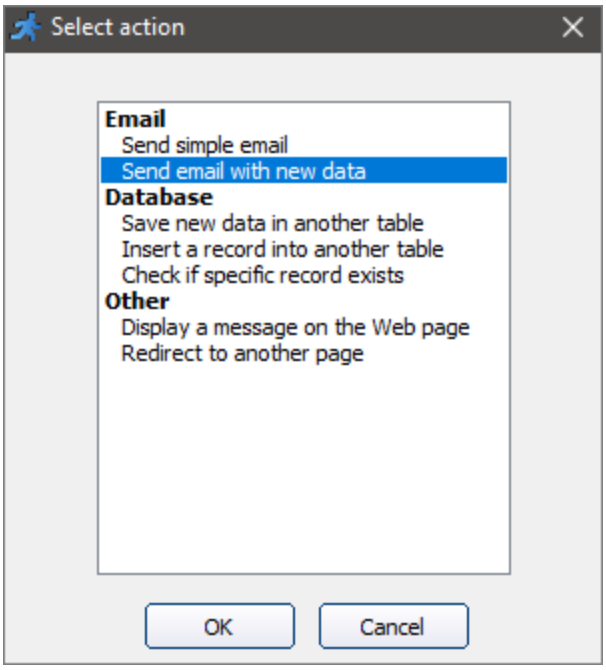
Send email with new data action allows sending an email containing information about new data in the table.

You can edit several parameters: the text and the subject of the message, the sender and recipient email addresses.

Available in the following events:

- Add page: Before record added
- Add page: After record added
- Edit page: Before record updated

- Edit page: After record updated



Note: To use this action, you have to set up the Email settings. Press button on the toolbar or write the code manually.

Change red values to adjust the action code to your project.

Note: To specify which table fields to email, edit the values assigned to `$msg` variable.

```
//**********  Send email with new data  ************
$email="test@test.com";
$from="admin@test.com";
$msg="";
$subject="New data record";

$msg.= "Name: ".$values["name"]."\r\n";
$msg.= "Email: ".$values["email"]."\r\n";
$msg.= "Age: ".$values["age"]."\r\n";

$ret=runner_mail(array('to' => $email, 'subject' => $subject, 'body' => $msg,
'from'=>$from));
```

```
if(!$ret["mailed"])
  echo $ret["message"];
```

See also:

- runner_mail function
- Send mass email to all users
- Send an email with attachment
- Send an email to selected users
- Send simple email
- Send email with old data
- Email selected records
- How to email selected records as separate PDF files

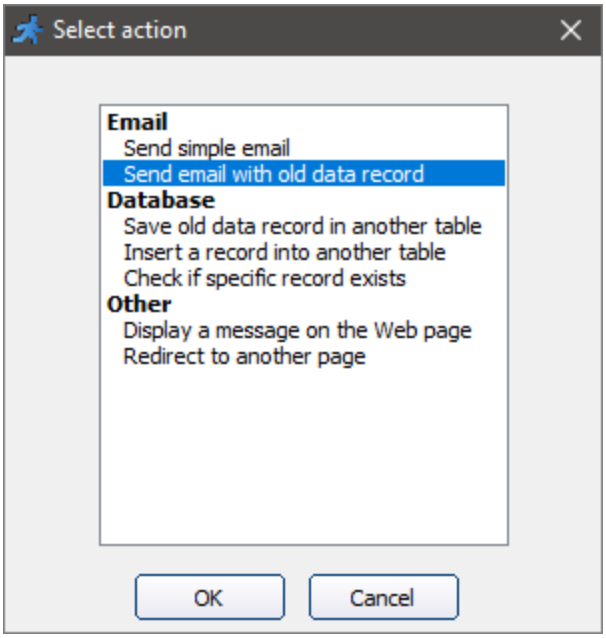
3.1.1.4 Send email with old data

Send email with old data record action allows, upon changing the record, emailing the previous data.

You can edit several parameters: the text and the subject of the message, the sender and recipient email addresses.

Available in the following events:

- Edit page: Before record updated
- Edit page: After record updated
- List page: Before record deleted



Note: To use this action, you have to set up the Email settings. Press button on the toolbar or write the code manually.

Change red values to adjust the action code to your project.

Note: To specify which table fields to email, edit the values assigned to `$msg` variable.

```
//**********  Send email with old data  ************
$email="test@test.com";
$from="admin@test.com";
$msg="";
$subject="New data record";

$msg.= "Name: ".$oldvalues["name"]."\r\n";
$msg.= "Email: ".$oldvalues["email"]."\r\n";
$msg.= "Age: ".$oldvalues["age"]."\r\n";

$ret=runner_mail(array('to' => $email, 'subject' => $subject, 'body' => $msg,
'from'=>$from));
if(!$ret["mailed"])
  echo $ret["message"];
```

See also:

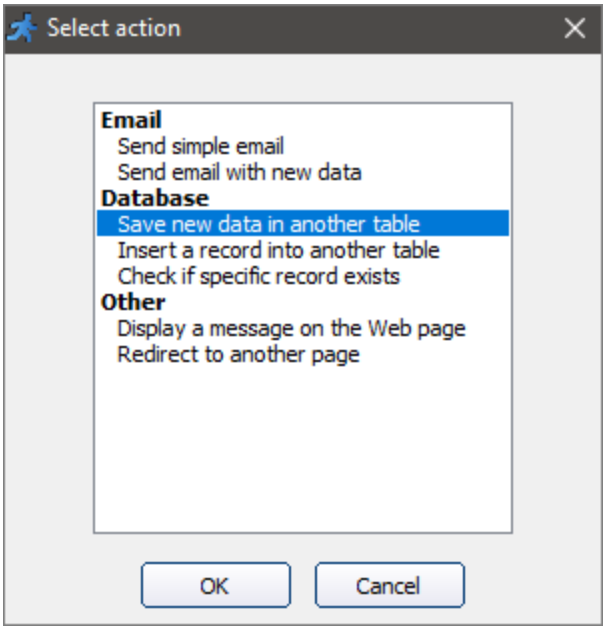
- runner_mail function
- Send mass email to all users
- Send an email with attachment
- Send an email to selected users
- Send email with new data
- Send simple email
- Email selected records
- How to email selected records as separate PDF files

3.1.1.5 Save new data in another table

Save new data in another table action allows copying added or modified data to another table. This table should be a part of the project.

Available in the following events:

- Add page: Before record added
- Edit page: Before record updated



Note: change `red` values to adjust the action code to your project.

The name `"copy_of_cars"` stands for the table where the new data is saved.

Example 1. Direct SQL query

```
//**********  Save new data in another table  ************
// note: text field values need to be wrapped by single quotes
$sql = "INSERT INTO copy_of_cars (make, model, price) values (";
$sql .=  "'".$values["make"]."',";
$sql .=  "'".$values["model"]."',";
$sql .=  $values["price"];
$sql .= ")";
DB::Query($sql);
```

Example 2. Database API

```
//**********  Save new data in another table  ************
$data = array();
$data["make"] = $values["make"];
$data["model"] = $values["model"];
```

```
$data["price"] = $values["price"];
DB::Insert("copy_of_cars", $data );
```

See also

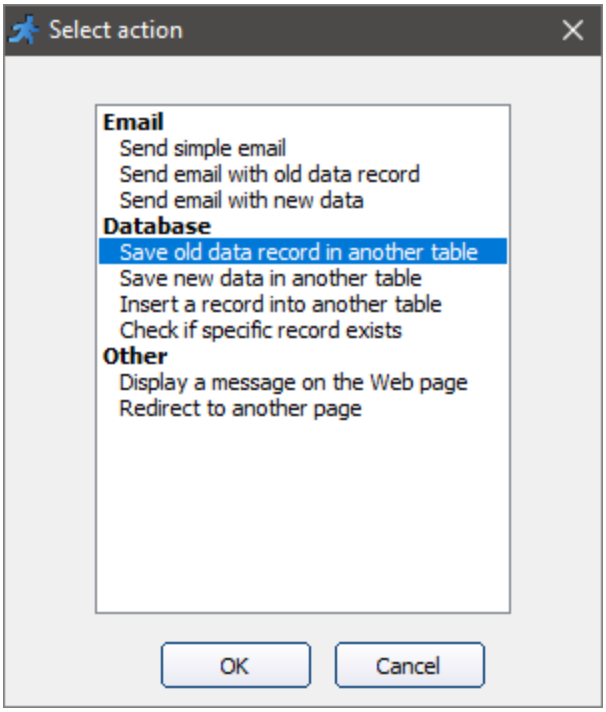
- Data_Access_Layer
- Database API
- Database API: Insert
- Save old data in another table
- Insert a record into another table

3.1.1.6 Save old data in another table

Save old data in another table action allows saving the old data in another table when the record is updated or deleted. This table should be a part of the project.

Available in following events:

- Edit page: Before record updated
- List page: Before record deleted



Note: change red values to adjust the action code to your project.

The name "copy_of_cars" stands for the table where the new data is saved.

Example 1. Direct SQL query

```
//**********  Save old data in another table  ************
$sql = "INSERT INTO copy_of_cars (make, model, price) values (";
$sql .=  "'".$oldvalues["make"]."',";
$sql .=  "'".$oldvalues["model"]."',";
$sql .=  $oldvalues["price"];
$sql .= ")";
DB::Query($sql);
```

Example 2. Database API

```
//**********  Save old data in another table  ************
$data = array();
$data["make"] = $oldvalues["make"];
$data["model"] = $oldvalues["model"];
$data["price"] = $oldvalues["price"];
DB::Insert("copy_of_cars", $data );
```

See also:

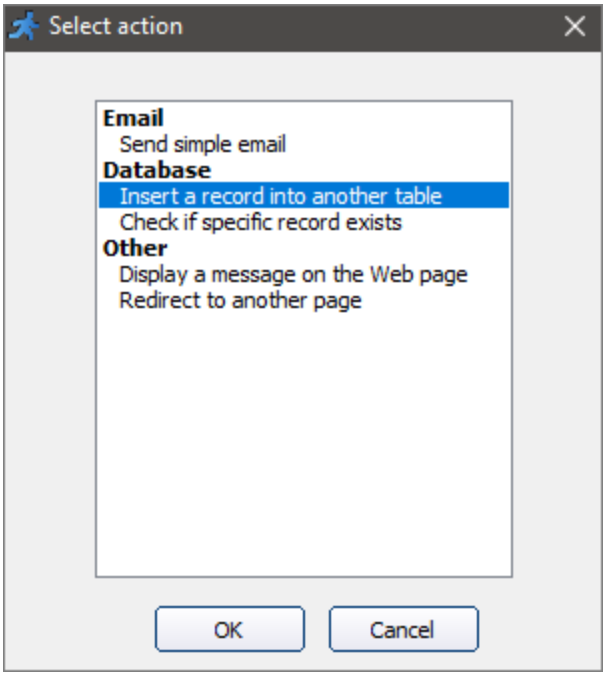
- Database API:Query()
- Database API
- Database API: Insert
- Save new data in another table
- Insert a record into another table

3.1.1.7 Insert a record into another table

Insert a record into another table action allows inserting the record into another table of the project.

Type the table name and the fields you want to insert the record into. You also need the data: type in the values manually or use the Database API methods.

This action is available at any event except JavaScript onload events.



Note: change `red` values to adjust the action code to your project.

Values `'Toyota'`, `'RAV4'`, `16000` are the inserted data. The name `"cars"` stands for the table where the record is inserted.

Example 1. Direct SQL query

```
//**********  Insert a record into another table  ************
$sql = "INSERT INTO cars (make, model, price) values
   ('Toyota', 'RAV4', 16000)";
DB::Query($sql);
```

Example 2. Database API

```
//**********  Insert a record into another table  ************
$data = array();
$data["make"] = "Toyota";
$data["model"]  = "RAV4";
$data["price"] = 16000;
DB::Insert("cars", $data );
```

See also

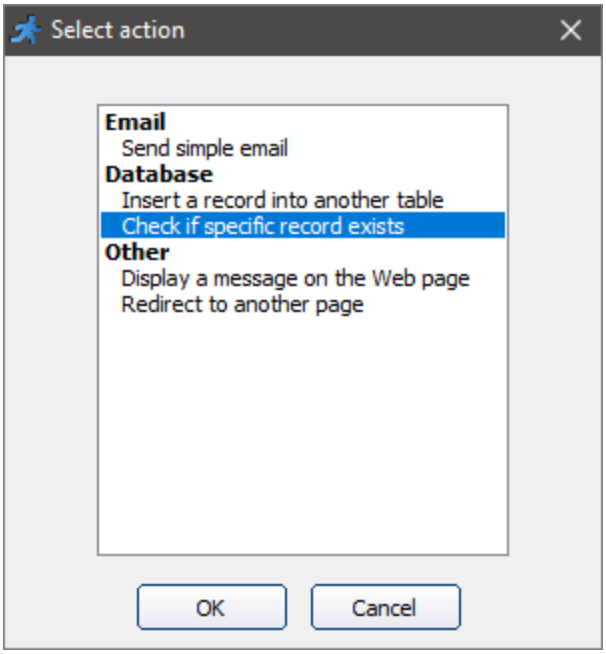
- DAL: CustomQuery()
- Database API
- Save old data in another table
- Save new data in another table

3.1.1.8 Check if specific record exists

Check if specific record exists action uses Database API to check if a *Specific Value* exists in the table.

Insert your code into the conditional statement instead of the *comments* to determine what happens if the *Specific Value* already exists or if it doesn't.

This action is available at any event except JavaScript onload events.



Change red values to adjust the action code to your project.

```
//**********  Check if specific record exists  ************
$rs = DB::Query("select * from AnyTable where SomeColumn='Specific Value'");
$data = $rs->fetchAssoc();
if($data)
{
   // if record exists do something
}
else
{
   // if dont exist do something else
}
```

See also:

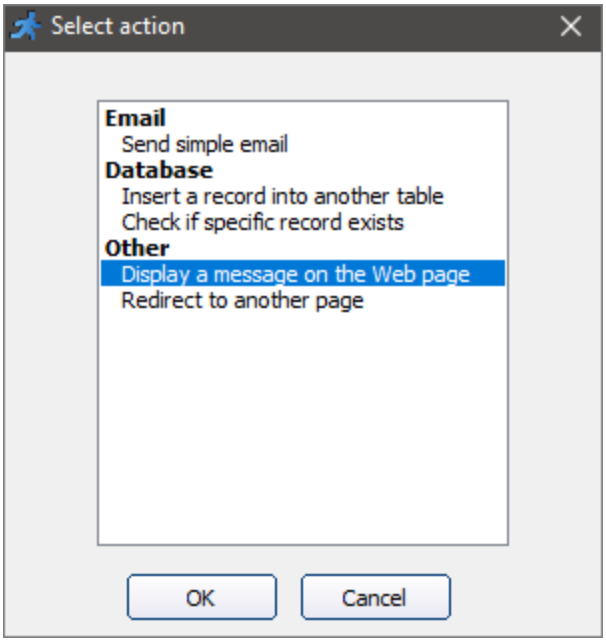
- Database API:Query()
- Database API: fetchAssoc()

- JavaScript API: getSelectedRecords()
- JavaScript API:getSelectedRecordKeys()
- Database API:Select()
- Update multiple records on the List page

3.1.1.9 Display a message on the Web page

To display a message on the Web page use **Display a message on the Web page** action.

This action is available at any event except JavaScript onload events.



Replace the text in the quotes with yours.

```
//**********  Display a message on the Web page  ************
echo "Your message here";
```

See also:

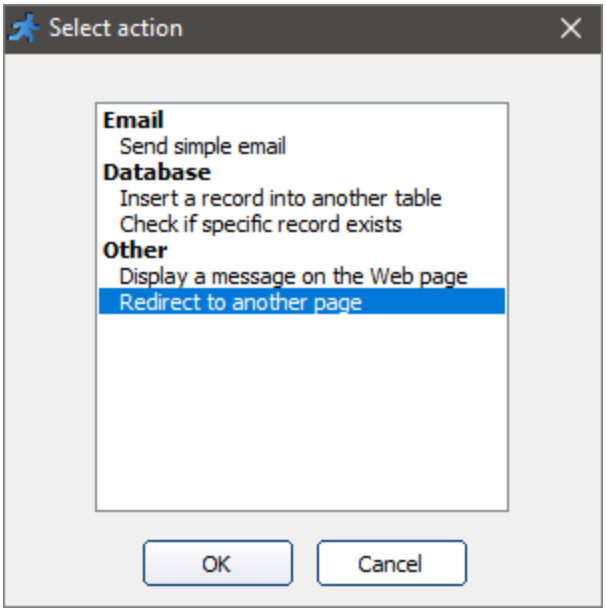
- Grid Row JavaScript API: row.setMessage()
- AJAX helper object: setMessage()
- How to display messages or tooltips for the fields
- Change the 'Logged on as' message

3.1.1.10 Redirect to another page

Use **Redirect to another page action** to redirect users to another web page using internal or external links.

> **Note**: An internal link is a link to any page within your web application. An external link is a link to another application or web site.

This action is available at any event except JavaScript onload events.



Replace the `"red"` value with an internal or external URL.

Note: To redirect to an external page, use an absolute link with the full URL. When redirecting to an internal link, you may omit the protocol, domain, and port.

```
//**********  Redirect to another page  ************
header("Location: anypage.php");
exit();
```

See also:

- Redirect to user info edit page
- Redirect to details page after adding master record

3.1.2 Sample events

3.1.2.1 Sample events

Sample events with code snippets

You can add sample code snippets to your events as custom code.

Here are a few examples of commonly used events with code snippets:

- Generate a list of customer orders
- Verify start date is earlier than end date
- Check for related records prior to deleting a record
- Speed up data entry using events
- Change the cell background color
- Change the row background color
- Select multiple values from checkboxes or a list field and have them appear as individual database entries
- Hide controls on Add/Edit pages, based on logged user name
- Send an email with updated fields only
- Show data from master table on details view/edit/add page
- Add dropdown list box with values for search

- Update multiple records
- Rename upload files
- Update multiple tables
- Redirect to details page after master record was added
- Email selected records
- Dynamic SQL Query
- Save user data in session variables
- Redirect to user info edit page
- Send email to email addresses from user table

You can find more examples in the **Advanced topics** -> **Events** -> **Sample events** category of this manual.

3.1.2.2 Appearance

Add a custom field to the form

To add a custom field to the form, follow the instructions below.

> **Note**: Change `red` values to match your project.

1. Proceed to the Editor page, switch to the HTML mode and add a custom input field to your form. Make sure you specify the field ID.

```
<INPUT id=test type=text>
```

2. Add the following code to the JavaScript OnLoad event of the page where the custom field was added:

```
this.on('beforeSave', function(formObj, fieldControlsArr, pageObj){
   var val = $("#test").val();
   formObj.baseParams['test'] = val;
});
```

3. In any event like Before record added or Before process use $_REQUEST["test"] to access the custom field value.

See also:

- ctrl.on
- Control object
- JavaScript API
- About Editor
- Event editor

Add a dropdown list box with values for the search

Add a dropdown list box with specific values.

For example, select a car make name from the dropdown list box, and make it so that only the data for the selected car make is displayed.

Insert PHP code snippet on the Page Designer screen to do so:

```
$str= "<select style='width: 150px; display: inline-block;' class='form-control'
onchange=\"window.location.href=this.options[this.".
  "selectedIndex].value;\"><option value=\"\">Please select</option>";
//select values from the database
$strSQL = "select Make from Cars";
$rs = db_query($strSQL);
while ($data = db_fetch_array($rs))
  $str.="<option value='cars_list.php?q=(Make~equals~".
  $data["Make"].")'>".$data["Make"]."</option>";
$str.="</select>";
echo $str;
```

> **Note**: Change `red` values to match your project.

See also:

- Insert code snippet
- About Page Designer
- Database API

Add a user profile link to the menu

Lets say you want to give each user a quick access to their data in the *users* table. You can do so by adding a *'My profile'* link to the main menu.

A link to the user profile page looks like this: *users_edit.php?editid1=XXXX*. We assume that the **Login** table name is *users* and *XXXX* is the value of the primary key field in *users* table.

> **Note**: Change `red` values to match your project.

1. Save the **ID** of the user account in a session variable. For this purpose, add the following code to AfterSuccessfulLogin event:

```
$_SESSION["user_id"]=$data["id"];
```

In this example `id` is the primary key column name in the **Login** table.

2. Create a new menu item via Menu Builder:

- Link type: *Application page*.
- Link to: *Users (login table) List page*.
- Link text: *My profile*.

3. Now add the following code to MenuItem: Modify event:

```
if ($menuItem->getTitle()=="My profile") {
   $menuItem->setUrl("users_edit.php?editid1=".$_SESSION["user_id"]);
}
return true;
```

See also:

- Event: ModifyMenuItem
- Global events
- Working with common pages
- Key columns

Add a new button to Add/Edit pages

Lets say we need to add a new button to the **Add** or **Edit** page that saves the record and redirects the user back to the **List** page.

> **Note**: Change `red` values to match your project.

1. Add a new button to the **Add** or **Edit** page via Insert button function in Page Designer.

2. Add the following code to the ClientBefore event of the button:

```
pageObj.on('beforeSave', function(formObj, fieldControlsArr, pageObj){
  formObj.baseParams['golist'] = "1";
});

$("#saveButton1").click();

return false;
```

3. Add the following code to the AfterAdd event of the **Add** page or AfterEdit event of the **Edit** page:

```
if($_REQUEST["golist"])
{
  header("Location: ..._list.php");
  exit();
}
```

See also:

- Tri-part events
- RunnerPage object
- Insert Standard button. Add page
- Insert Standard button. Edit page
- Troubleshooting custom buttons
- Insert custom button
- How to enable/disable a button
- Button object

Change the cell background color

To change any cell background color, use the following code in the After Record Processed event.

> **Note**: Change `red` values to match your project.

```
$record["FieldName_css"]='background:red;';
```

You can also change the background color of a cell when the mouse hovers over it:

```
$record["FieldName_hovercss"]='background:yellow;';
```

See also:

- Change the row background color
- Conditional formatting
- Customizing CSS
- Grid Row JavaScript API: row.fieldCell()
- About Grid Row JavaScript API
- After record updated event
- Field events

Change the font size in a text box

To change the font size in all text boxes placed on a page, use the following code in the JavaScript OnLoad event.

> **Note**: Change `red` values to match your project.

```
$("input[type=text]").css('fontSize', '120%');
```

See also:

- Customizing CSS
- Rich Text Editor plugins
- "Edit as" settings: Text field
- "Edit as" settings: Text area
- JavaScript API

Change the 'Logged on as' message

The *"Logged on as"* message can be changed in the AfterSuccessfulLogin event. Here is how you can display user's full name instead of *username*:

Note: Change `red` values to match your project.

```
$_SESSION["UserName"] = $data["FirstName"].' '.$data["LastName"]
```

See also:

- Save user data in session variables
- About Security API
- Security API: setDisplayName
- Security screen
- Grid Row JavaScript API: row.setMessage()

Change the message after the record was added or saved

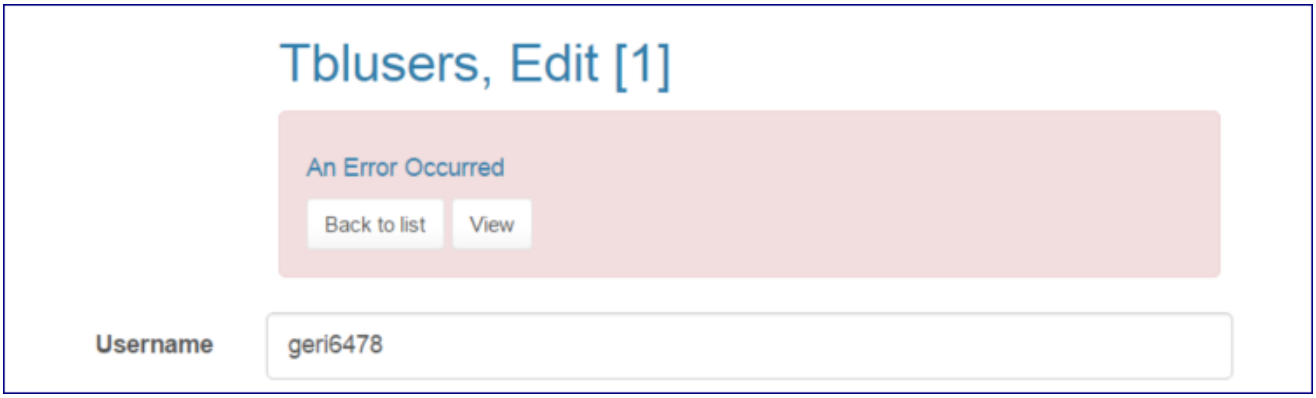
To change the message, use the following code in the AfterAdd or AfterEdit events:

Example 1

Change the error message:

```
$pageObject->setMessageType(MESSAGE_ERROR);
$pageObject->setMessage("An Error Occurred");
```

This is how the result looks like:



Example 2

Change the *'success'* message:

```
$pageObject->setMessageType(MESSAGE_INFO);
$pageObject->setMessage("The data was successfully saved");
```

Note: These code snippets always work on the **Edit** page. They do not work on the **Add** page if you choose to display the master data on the details page.

See also:

- Grid Row JavaScript API: row.setMessage()
- RunnerPage object
- Error reporting
- Display a message on the Web page
- Master-details relationship between tables

Change the row background color

To change any row background color, use the following code in the After Record Processed event.

Note: Change `red` values to match your project.

```
$record["css"]='background:blue;';
```

You can also change the background color of a row when the mouse hovers over it:

```
$record["hovercss"]='background:yellow;';
```

See also:

- Change the cell background color
- Conditional formatting
- Customizing CSS
- Field events
- Grid Row JavaScript API: row.fieldCell()

Change the width of an edit box with an AJAX popup

To change the width of an edit box with an AJAX popup, use the following code in the JavaScript OnLoad event.

> **Note**: Change `red` values to match your project.

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.getDispElem().css("width", "200px");
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > getDispElem()
- Customizing CSS
- AJAX-based Functionality

- AJAX helper object
- How to create a custom Edit control plugin
- JavaScript API

Change the width of a text field on the Quick Search panel

To change the width of a text field on the **Quick Search** panel, use the following code in the JavaScript OnLoad event.

> **Note**: Change `red` values to match your project.

```
$("input[name^='value_make'], select[name^='value_make']").width(150);
```

See also:

- Customizing CSS
- JavaScript API: SearchField object
- About Search API
- JavaScript API

How to hide the Edit link

To hide the **Edit** link when the record status is *"processed"*, use the following code in the List page: After record processed event.

> **Note**: Change `red` values to match your project.

```
if ($data["status"]=="processed")
{
  $pageObject->hideItem("grid_edit");
}
```

See also:

- About RunnerPage class
- JavaScript API: hide()
- Event: IsRecordEditable
- JavaScript API: makeReadonly()
- RunnerPage class: hideItem()

Hide controls on Add/Edit pages, based on the username

To hide controls on **Add**/Edit **pages**, based on the username, use the following code in the Add Page: BeforeDisplay and/or Edit Page: BeforeDisplay event.

> **Note**: Change `red` values to match your project.

Example 1

To show the `"Horsepower"` field edit control only if the username equals `"admin"`, use the following code:

```
if (Security::getUserName()!="admin")
   $pageObject->hideField("Horsepower");
```

Example 2

To show the `"Horsepower"` field edit control only if the current user belongs to the admin group, use the following code:

```
if (Security::isAdmin())
   $pageObject->hideField("Horsepower");
```

> **Note**: This code sample works only with static permissions.

3. To hide the "Horsepower" field edit control placed on tab or folding section on the **Edit**/**Add**/**View** page, use the following code:

```
$pageObject->hideField("Horsepower");
```

See also:

- Security API
- JavaScript API: hideField()
- Security API: isAdmin
- Security API: getUserName
- JavaScript API: hide()
- RunnerPage class: hideItem()
- JavaScript API: toggleItem()

Hide empty fields on the View page

Let's say you have a list of fields on the **View** page, and you want to hide those that are empty. For this purpose, use the following code in the View page: Before display event for each field you need to hide. This example uses field named *EPAHighway*.

Note: Change red values to match your project.

```
if (!$values["EPAHighway"])
   $pageObject->hideField("EPAHighway");
```

Note: There is an option to hide empty fields on the **View** page automatically. To enable it, go to the Choose pages screen, click the **Settings button** near the **View** record checkbox and then select **Hide empty fields** checkbox.

See also:

- JavaScript API: hideField()

- Before display event for all pages except View/Edit
- JavaScript API: hide()
- RunnerPage class: hideItem()
- JavaScript API: toggleItem()

Hide repeating values on the List page

Here is the typical **List** page with the list of cars sorted by make.

Id	Make ↑	Model	Year Of Make	Horsepower
4	Acura	NSX-T	2000	250
5	Acura	MDX	2000	180
6	Acura	RDX		
1	Audi	TT	2000	197
3	Audi	Q5		220
2	BMW	525i	2004	215
7	BMW	750iL	2001	300
8	BMW	Z4	2001	220
9	BMW	X6	2001	385
10	Mersedes	CL-500	2000	302

Sometimes you may need to make this screen less cluttered by removing the repeating values in the *Make* column. Something like this:

Id	Make ↑	Model	Year Of Make	Horsepower
4	Acura	NSX-T	2000	250
5		MDX	2000	180
6		RDX		
1	Audi	TT	2000	197
3		Q5		220
2	BMW	525i	2004	215
7		750iL	2001	300
8		Z4	2001	220
9		X6	2001	385
10	Mersedes	CL-500	2000	302

Here is how this can be done:

1. Add the following code to the List page: BeforeProcess event:

```
$_SESSION["Make"]="";
```

2. Set the 'View as' type of the *Make* field to 'Custom' and paste the following code there:

```
if ($value==$_SESSION["Make"])
$value="";
else
$_SESSION["Make"]=$value;
```

See also:

- "View as" settings
- AJAX-based sorting

Print search parameters on the List page

If you want to print **Advanced search** or **Search panel** parameters on the **List** page, add PHP code snippet with following code:

```
global $strTableName;
$srchObj = SearchClause::getSearchObject($strTableName);
$fields = $srchObj->getSearchFields();
if(count($fields))
   echo "Search was completed."."<br>";
foreach ($fields as $field=>$value ) {
   echo $field.": " .$srchObj->getSearchOption($field);
   if ($srchObj->getSearchOption($field)=="Between") {
      echo " AND ".$srchObj->getSecondFieldValue($field);
   }
   echo " ".$srchObj->getFieldValue($field);
   echo "<br>";
}
```

Printing basic search parameters:

```
global $strTableName;
$srchObj = SearchClause::getSearchObject($strTableName);
$fields = $srchObj->getSearchFields();
if(count($fields))
   echo "Search was completed."."<br>";
echo $srchObj->_where["_simpleSrch"];
```

Note: Change `red` values to match your project.

See also:

- getSearchObject
- getFieldValue

- getSecondFieldValue
- getSearchOption
- SearchAPI
- Insert code snippet

Redirect to the details page after adding the master record

Sometimes you need to redirect visitors to the details table (e.g., *Order details*) right after a new master record was added (*Orders* table).

The trick is to pass correct key field to the details table. Once user redirected there she can start adding details records using Inline or regular Add.

The following code needs to be added to AfterAdd event of the master table.

> **Note**: Change `red` values to match your project.

```
header("Location: order_details_list.php?mastertable=orders&masterkey1=" .
   $values["OrderID"]);
exit();
```

In this example:

`order_details_list.php`
: the URL of the details **List** page;

`orders`
: the master table name;

`OrderID`
: the name of the key field.

See also:

- Master-details relationship
- Key columns
- Event editor

Show data from a master table on the detail view/edit/add page

Show data from a master table on the details **View**/**Edit**/**Add** pages.

To add this event, insert PHP code snippet on the Page Designer screen:

```
global $pageObject;
echo "Master Info<br>";
if ($data = $pageObject->getMasterRecord())
  {
    echo "Field1: ".$data["Field1"]."<br>";
    echo "Field2: ".$data["Field2"]."<br>";
  }
```

For example, *"Orders"* is a master table and *"Order details"* is a details table.

```
global $pageObject;
echo "Order Info<br>";
if ($data = $pageObject->getMasterRecord())
  {
    echo "Customer: ".$data["CustomerID"]."<br>";
    echo "Employee: ".$data["EmployeeID"]."<br>";
  }
```

Note: Change `red` values to match your project.

See also:

- getMasterRecord
- RunnerPage class

Show dropdown list of US states if US was selected in country list

You can show the dropdown list of US states if US was selected in the *Country* list; hide it otherwise. Use the following code in JavaScript OnLoad event on **Add**/**Edit** pages in popup and inline.

> **Note**: Change `red` values to match your project.

```
var ctrlCountry = Runner.getControl(pageid, 'country');
var ctrlState = Runner.getControl(pageid, 'state');

ctrlCountry.on('change', function(e){
   if (this.getValue() == 'US'){
      ctrlState.show();
   }else{
      ctrlState.hide();
   }
});
```

You may want to hide the field label as well. Use the following code to hide or show the whole table row with the *State* edit control based on the *Country* field selection. This code works in popup and inline **Add**/**Edit** modes.

```
var ctrlCountry = Runner.getControl(pageid, 'country');

ctrlCountry.on('change', function(e) {
    if (this.getValue() == 'US') {
        pageObj.showField("state");
    } else {
        pageObj.hideField("state");
    }
});
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > getValue()
- JavaScript API: Control object > on()
- JavaScript API: Control object > show()
- JavaScript API: Control object > hide()
- JavaScript API: RunnerPage object > showField()

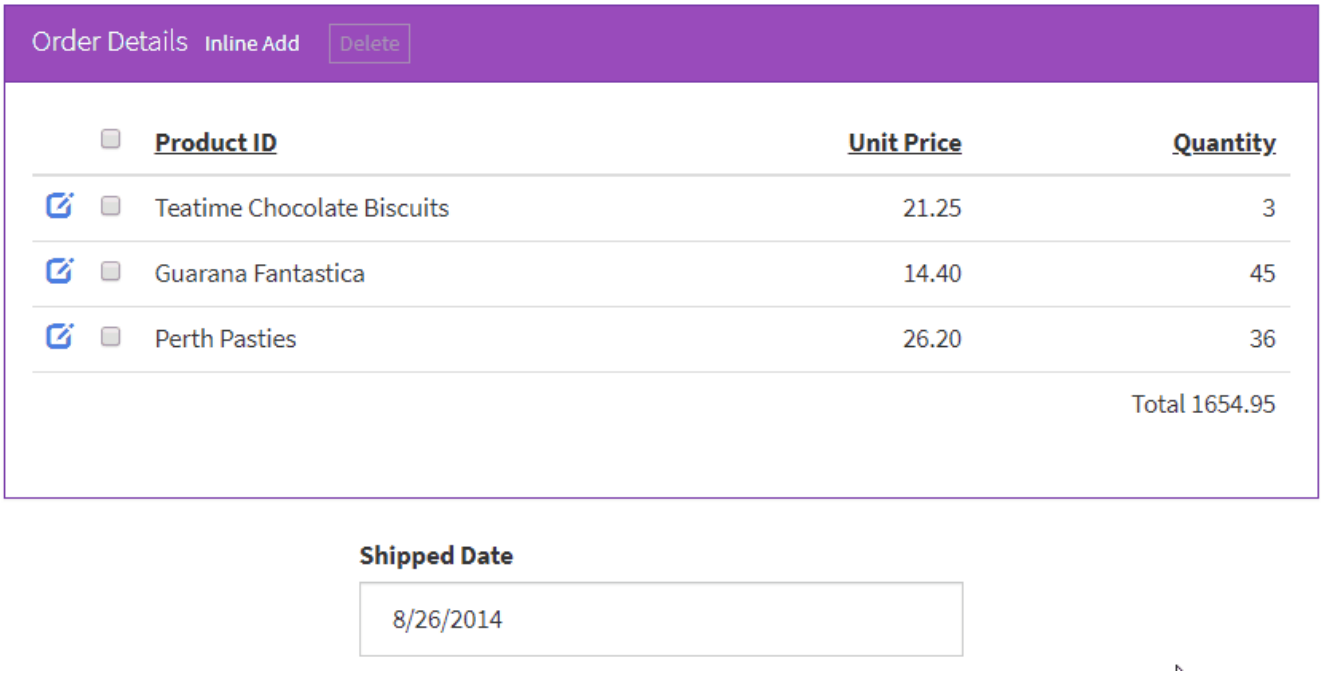
- JavaScript API: RunnerPage object > hideField()
- JavaScript API: RunnerPage object
- JavaScript API

Show order total on the Edit page as the details table is updated

We edit the order header and order details on the same page and want to show the order total on the master table **Edit** page as we add or edit order details.

In this example, the master table is *Orders* and the details table is *Order Details*.

This is how it looks like in action:



Instructions

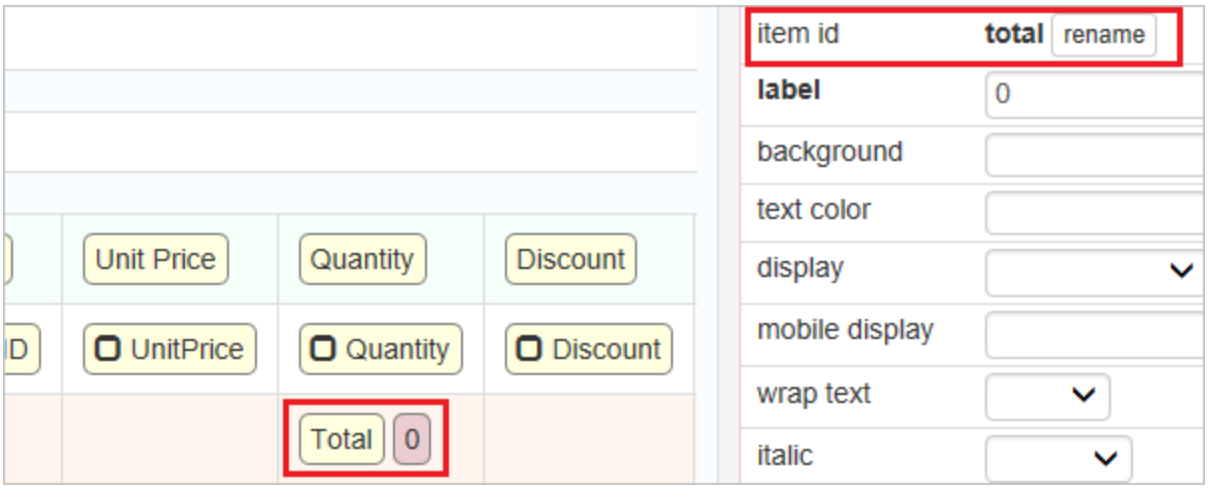
Link *Orders* and *Order Details* tables as master-details. Check off the **Display details table data on Edit page** option.

1. Insert a placeholder for the order total display

The Total is calculated as a sum of *UnitPrice* * *Quantity* for all *Order Details* records.

Proceed to the Page Designer and insert a text element (**Insert** -> **Text**). Enter *0* there. Rename its **Item ID** to *total*.

You can insert this element to either the master **Edit** page or details **List** page. It is only a question of appearance, as your design requires. You may also want to add another text element with the word *'Total'*.



Right-align the content of this cell so order total is aligned with the quantity.

2. custom_functions.js

This code needs to be added to custom_functions.js section under Event Editor.

> **Note**: the details table and the field names are in red. Replace them with the actual details table and field names. All table and field names are case-sensitive.
>
> In this example, we multiply *Quantity* by *UnitPrice*. Your formula might be different.

```
var orderDetailsTable = 'Order Details';
function setOrderTotal( value, pageObj ) {
```

```
    //    update totals if the 'totals' element is inserted into the master page
    pageObj.findItem('total').text( parseFloat(value).toFixed(2) );

    //    update totals if the 'totals' element is inserted into the details page
    var detailsPage = pageObj.getDetailsPage(orderDetailsTable);
    detailsPage.findItem('total').text( parseFloat(value).toFixed(2) );
}

function updateOrderTotal( pageObj ) {
    var detailsPage = pageObj.getDetailsPage(orderDetailsTable);
    if( !detailsPage )
        return;
    var records = detailsPage.getAllRecords();

    // actual calculation. Multiply Quantity by UnitPrice and sum the results.
    // total = sum( Quantity * UnitPrice )
    var total = 0;
    records.forEach( function( r ) {
        total +=
            ( parseFloat( r.getFieldValue('Quantity') ) || 0 ) *
            ( parseFloat( r.getFieldValue('UnitPrice') ) || 0 );
    });

    setOrderTotal( total, pageObj );
}
```

updateOrderTotal is the main calculation function. The rest of the code only calls it at the right moments.

3. Orders (master) - Edit page - Javascript OnLoad event

Calculate the totals on the **Edit** page loading.

```
updateOrderTotal( pageObj );
```

4. Order Details (details) - List page - Javascript OnLoad event

This code tells PHPRunner to recalculate the total on the page loading, and after adding or modifying records in the details table.

```
pageObj.updateTotals = function() {
    var masterPage = pageObj.getMasterPage();
    if( masterPage )
        updateOrderTotal( masterPage );
}
```

```
this.on('afterInlineEdit', pageObj.updateTotals );
this.on('afterInlineAdd', pageObj.updateTotals );

pageObj.updateTotals();
```

5. Order Details - Field events

The last step is to tell your application to recalculate the totals immediately on editing the *Quantity* and *UnitPrice* fields in an **Inline** mode.

Proceed to the Fields screen, open the *Order Details* page there. Then open the *Quantity* field properties and click Field events.

Select the **change** event and add a new event handler.

Add this code to the Client Before event:

```
if( pageObj.updateTotals ) {
    pageObj.updateTotals();
}
return false;
```

Then assign the same event handler to the **change** event of the second field, *UnitPrice*.

See also:

- getDetailsPage
- getMasterPage
- getAllRecords
- findItem
- ctrl.on
- Master-Details relationship
- Insert Text and Image
- Field events
- Tri-part events

3.1.2.3 Database

Check for related records before deleting the current one

Before deleting a record in the *Orders* table, check for related items in the *OrderDetails* table. Add the following code to the List page: Before record deleted event.

> **Note**: Change red values to match your project.

```
// Parameters:
// where - string with WHERE clause pointing to record to be deleted.
global $dal;
$tblOrder = $dal->Table("OrderDetails");
$rs = $tblOrder->Query("OrderID=".$deleted_values["OrderID"],"");
$data = db_fetch_array($rs);
if($data)
  return false;
else
  return true;
```

See also:

- Data Access Layer
- DAL method: TableName()
- DAL method: Query
- Database API
- Check if specific record exists
- Master-details relationship between tables

Dynamic SQL query

Sometimes you need to present different data to different groups of users. Let's say you run a classified ads board and wish to display the data added over the past 7 days to regular users and all of the ads to the admin.

Add the following code to the After table initialized event.

> **Note**: Change red values to match your project.

1. For MS SQL Server database:

```
if (Security::getUserName()!="admin")
   $query->addWhere("DATEDIFF(day,DateColumn, GETDATE()) <= 7");
```

2. For MySQL database:

```
if (Security::getUserName()!="admin")
  $query->addWhere("DATE_SUB(CURDATE(), INTERVAL 7 DAY) <= DateColumn");
```

3. For MS Access database:

```
if (Security::getUserName()!="admin")
   $query->addWhere("DATEDIFF('d',DateColumn, Now()) <= 7");
```

4. If you use Dynamic Permissions, the code should be different (MySQL example).

> **Note**: If the user doesn't belong to the admin group, show only the past 7 days worth of data.

```
if (!Security::isAdmin())
    $query->addWhere("DATE_SUB(CURDATE(), INTERVAL 7 DAY) <= DateColumn");
```

See also:

- Method: addWhere
- About SQLQuery class
- Security API: getUserName
- Security API: isAdmin
- SQL variables in SQL and Lookup wizards
- Dynamic Permissions

Limit the number of records users can add

Lets say you want to limit the number of records users can add to the certain table. For example, if you run a classified ads website, and you want free users to be able to post up to 3 ads, basic plan users to add up to 10 ads, etc.

Sample database structure

Table1: *users, userid, password, limit*.

Table2: *ads, id, userid, adtext.*

Tables are linked via the *userid* field. It would also make sense to turn on Advanced security mode '*Users can see all data; can edit their own data only*'.

Use the following code in the Add page: Before Record Added event.

Scenario 1. All users can add the same number of records

```
$limit = 3;

$rs = DB::Query("select count(*) as c from ads where userid = " .
$_SESSION["UserID"]);
$data = $rs->fetchAssoc();
$count = $data["c"];

if ($count>= $limit)
{
   echo "Limit reached: $count records added already";
   exit();
}
```

Scenario 2. Each user has it's own limit. Limits are stored in the userlimit field of users table

```
$rs = DB::Query("select count(*) as c from ads where userid = " .
$_SESSION["UserID"]);
$data = $rs.fetchAssoc();
$count = $data["c"];
```

```
$rs2 = DB::Query("select ".AddFieldWrappers( "userlimit" )." from users where userid
= " . $_SESSION["UserID"]);
$data2 = $rs2.fetchAssoc();
$limit = $data2["userlimit"];

if ($count>= $limit)
{
   echo "Limit reached: $count records added already";
   exit();
}
```

See also:

- QueryResult object fetchAssoc()
- Security screen
- Database API:Query()
- About SQLQuery class

Select multiple values from checkboxes or a list field and have them appear as individual database entrie

To select multiple values from checkboxes or a list field and have them appear as individual database entries, use the following code in the Add page: Before Record Added event.

Note: Change red values to match your project.

```
if ($values["fieldname"])
{
  $arr = explode(",",$values["fieldname"]);
  // This is the name of the multi check box or
  // list select field, its value becomes $arr
for ($i=0;$i<count($arr);$i++)
{
  $strInsert = "insert into TableName (field) values ('".$arr[$i]."')";
  // add more fields from the add page to be inserted into database

  db_exec($strInsert);
}
header("Location: MyPage_list.php");
  // Exit and Redirect to the list page after updating database
exit();
}
```

See also:

- JavaScript API:getSelectedRecordKeys()
- Database API: DB::Exec
- Redirect to another page

Show a list of customer orders

Show a list of all orders placed by the current customer on the **Order Edit** page.

Insert PHP code snippet on the Page Designer screen:

```
global $dal;
$tblOrders = $dal->Table("Orders");
$rs = $tblOrders->Query("OrderID=".$_REQUEST["editid1"],"");
$data = db_fetch_array($rs);
$CustomerID = $data["CustomerID"];
echo "Orders placed by " .$CustomerID. "<br>";
$rsOrders = $tblOrders->Query("customerid='".$CustomerID."'","");
while($data=db_fetch_array($rsOrders))
{
  echo "<a target=_blank href=Orders_edit.php?editid1=".$data["OrderID"].
  ">". $data["OrderID"]."</a> ".$data["OrderDate"]."<br>";
}
```

Note: Change red values to match your project.

See also:

- Data Access Layer
- QueryResult object fetchAssoc()
- Database API
- Master-details relationship between tables

Store the date and time when a record is modified

To keep track of the time/date when records were last modified and who made the modification, you have two options:

1. Set the default value under 'Edit as' settings

To store the **login name** of the user who made the change, set the default value to:

```
$_SESSION["UserID"]
```

To store the modification date and time, set the default value to:

```
now()
```

Note: Make sure to clear the AutoUpdate Value field.

2. Modify BeforeAdd/BeforeEdit events

Add the following code to **BeforeAdd**/**BeforeEdit** events:

```
$values["DateUpdated"]=now();
$values["UpdateBy"]=$_SESSION["UserID"];
```

See also:

- "Edit as" settings: Date
- About Date Control API
- Table events

Update multiple records on the List page

To update multiple records on the **List** page at the same time, you can create a new **Update selected** button, then choose records on the **List** page and click the **Update selected** button.

> **Note**: Change red values to match your project.

1. Proceed to the Page Designer screen.

2. Create the **Update selected** custom button and add the following code to the Server tab:

```
global $dal;

while ( $data = $button->getNextSelectedRecord() ) {
    // set ReportsTo field to 'Bob Smith'
    $sql = "Update employees set ReportsTo='Bob Smith' where
KeyField=".$data["KeyField"];
    CustomQuery($sql);
}
$result["txt"] = "Records were updated.";
```

and to the Client After tab:

```
var message = result["txt"];
ctrl.setMessage(message);
```

The Client Before tab should be blank (delete sample code there if you have any).

See also:

- Data Access Layer
- Database API
- Tri-part events
- Button object: getNextSelectedRecord()
- About SQLQuery class
- AJAX helper object: setMessage()

Update multiple tables

To update two joined tables, use the following code in the Add page: Before record added and/or Edit page: Before record updated events.

> **Note**: Change red values to match your project.

```
$sql = "update othertable set joinedfield=".$values["joinedfield"]." ... ";
db_exec($sql);
unset($values["joinedfield"]);
```

To update master and details tables, use the following code for the details table in the Add page: Before record added and/or Edit page: Before record updated events:

```
global $dal;
$tblDetail = $dal->Table("DetailTableName");
$tblDetail->Value["Field1"] = $values["Field1"];
$tblDetail->Param["OrderID"] = $values["OrderID"];
$tblDetail->Update();
unset($values["Field1"]);
```

`"Field1"` is a field in the details table and `"OrderID"` is the linked field in the master table.

See also:

- Data Access Layer
- Database API
- Database API: DB::Exec
- Master-details relationship between tables
- Update multiple records on the List page

Search Master and Details tables together

Here is how you can search master and details tables together.

For example, you have *Orders* and *OrderDetails* tables and you want to find the orders that contain a certain product.

1. Modify the *Orders* SQL Query to add a dummy field named 'product'. Make sure this field is searchable.

```
SELECT
        OrderID,
        CustomerID,
        EmployeeID,
        OrderDate,
        ShipAddress,
        ShipCity,
        ShipRegion,
        ShipPostalCode,
        ShipCountry,
        '' as product
FROM orders
```

2. *Orders* table, AfterTableInit event:

```
$srchObj = SearchClause::getSearchObject("orders");

$value = $srchObj->getFieldValue("product");

if( $value != null ) {
$srchObj->setSearchSQL("product", "OrderID in (select OrderID from OrderDetails where
ProductName like '%$value%')");
}
```

Note: In this event, we do a *subquery* to find all orders that contain the product in question.

See also:

- About Search API
- setSearchSQL
- About Search API
- Master-details relationship between tables

- SQL query screen

3.1.2.4 Email

Email selected records

Quick jump

How to send an email to a predefined address

To send an email with several selected records on the **List** page, you need to create a custom button.

1. Proceed to the Page Designer screen.

2. Create an **Update selected** custom button and add the following code snippets into it:

Server tab:

> **Note**: Change red values to match your project.

```
$body = "";
while( $data = $button->getNextSelectedRecord() )
{
    $body .= "OrderID: " . $data['OrderID'] . "\n";
    $body .= "Customer: " . $data['CustomerID'] . "\n";
    $body .= "Employee: " . $data['EmployeeID'] . "\n----------\n\n";
}

// send the email
$email = "test@test.com";
$subject = "Sample subject";
$arr = runner_mail(array('to' => $email, 'subject' => $subject,'body' => $body));

$result["txt"] = "Emails were sent.";

// if error happened print a message on the web page
if( !$arr["mailed"] )
```

```
{
    $errmsg = "Error happened: <br>";
    $errmsg.= "File: " . $arr["errors"][0]["file"] . "<br>";
    $errmsg.= "Line: " . $arr["errors"][0]["line"] . "<br>";
    $errmsg.= "Description: " . $arr["errors"][0]["description"] . "<br>";
    $result["txt"] = $errmsg;
}
```

Client After tab:

```
var message = result["txt"];
ctrl.setMessage(message);
```

Note: The Client Before tab should be blank (delete sample code there if any).

Sample email message:

OrderID: 10249

Customer: TRADH

Employee: 6

OrderID: 10251

Customer: VICTE

Employee: 3

OrderID: 10253

Customer: HANAR

Employee: 3

How to send an email to the currently authorized user

Instead of a hardcoded email address, you can send email to the current user.

1. The email address is used as the username

```
$email=$_SESSION["UserID"];
```

2. The email address is stored in an individual field

In this case, you need to save the email address to the session variable in the AfterSuccessfulLogin event:

```
$_SESSION["email"]=$data["email"];
```

BeforeProcess event code:

```
$email=$_SESSION["email"];
```

See also:

- Inserting custom button
- Update selected
- Tri-part events
- Send an email to all users
- Send an email to selected users
- Send simple email
- How to email selected records as separate PDF files
- Grid Row Javascript API: row.getKeys()
- JavaScript API:getSelectedRecordKeys()
- Button object: getNextSelectedRecord()
- runner_mail function
- AJAX helper object: setMessage()

Send an email to selected users

Sometimes you need to send an email to selected users. Let's say one of the fields on a page, ***EmailField*** contains email addresses. The key field on the page is **KeyColumn**, table name - *TableName*.

Let's consider two situations:

Subject and text of the email determined directly in the event

> **Note**: Change red values to match your project.

1. Proceed to the Page Designer screen.

2. Create a custom button (e.g., "**Email selected**") and add the following code to the **Server tab**:

```
$emails = array();

while( $data = $button->getNextSelectedRecord() )
{
   if( $data["EmailField"] )
       $emails[] = $data["EmailField"];
}

// send the email
$email = implode(", ", $emails);
$subject = "Sample subject";
$body = "Your email text here";
$arr = runner_mail(array('to' => $email, 'subject' => $subject, 'body' => $body));

$result["txt"] = "Emails were sent.";

// if error happened print a message on the web page
if( !$arr["mailed"] )
{
   $errmsg = "Error happened: <br>";
   $errmsg.= "File: " . $arr["errors"][0]["file"] . "<br>";
   $errmsg.= "Line: " . $arr["errors"][0]["line"] . "<br>";
   $errmsg.= "Description: " . $arr["errors"][0]["description"] . "<br>";
   $result["txt"] = $errmsg;
}
```

and to the **Client After tab**:

```
var message = result["txt"];
ctrl.setMessage(message);
```

Note: The **Client Before tab** should be blank (delete sample code there if any).

User enters the subject, email body, "from" email address

Note: Change red values to match your project.

1. Proceed to the **Page Designer** screen and select the **List** page.

2. Create a custom button (e.g., "**Email selected**").

Add the following code to the **Client Before tab**:

```
if ( proxy["emailUsers"] === true ) {
    params["emailFrom"] = proxy["emailFrom"];
    params["emailBody"] = proxy["emailBody"];
    params["emailSubject"] = proxy["emailSubject"];

    proxy["emailUsers"] = false;
    return true;
}

var selBoxes = pageObj.getSelBoxes( pageid ),
    args = {
        modal: true,
        header: "Input from, subject and body",
        html: '<div>'
           + '<div>From: <input type="text" id="emailFrom"s style="margin: 5px
0"></div>'
           + '<div>Subject: <input type="text" id="emailSubject"s style="margin: 5px
0"></div>'
           + '<div>Body: <textarea id="emailBody"></textarea></div>'
           + '</div>',
        footer: '<a href="#" id="emailUsersSave" class="btn btn-primary">' +
Runner.lang.constants.TEXT_SAVE + '</a>'
            + '<a href="#" id="emailUsersCancel" class="btn btn-default">' +
Runner.lang.constants.TEXT_CANCEL + '</a>',
        afterCreate: function( win ) {
            $("#emailUsersSave").on("click", function( e ) {
                var context = win.body();
```

```
                proxy["emailUsers"] = true;
                proxy["emailFrom"] = $("#emailFrom", context).val();
                proxy["emailBody"] = $("#emailBody", context).val();
                proxy["emailSubject"] = $("#emailSubject", context).val();
                $('[id="' + ctrl.id + '"]').click();

                e.preventDefault();
                win.destroy();
            });

            $("#emailUsersCancel").on("click", function( e ) {
                e.preventDefault();
                win.destroy();
            });
        }
    };

if ( selBoxes.length == 0 || !confirm('Do you really want to email these records?') )
{
    return false;
}

Runner.displayPopup( args );
return false;
```

Server tab:

```
if( $params["emailBody"] || $params["emailSubject"] || $params["emailFrom"] )
{
   $emails = array();
   while ( $data = $button->getNextSelectedRecord() )
   {
      if ( $data["EmailField"] )
         $emails[] = $data["EmailField"];
   }

   // send the email
   $from = $params["emailFrom"];
   $email = implode(",", $emails);
   $body = $params["emailBody"];
   $subject = $params["emailSubject"];
   $arr = runner_mail(array('to' => $email, 'subject' => $subject, 'body' => $body,
'from' => $from));

   $result["txt"] = "Emails were sent.";
   if (!$arr["mailed"])
   {
      $errmsg = "Error happened: <br>";
      $errmsg.= "File: " . $arr["errors"][0]["file"] . "<br>";
      $errmsg.= "Line: " . $arr["errors"][0]["line"] . "<br>";
      $errmsg.= "Description: " . $arr["errors"][0]["description"] . "<br>";
```

```
        $result["txt"] = $errmsg;
    }
}
```

Client After tab:

```
var message = result["txt"] + " !!!";
ctrl.setMessage(message);
```

Note: After selecting record(s) on the **List** page and clicking the "**Email selected**" button, a popup appears where you can enter `From`, `Subject` and `Body` parameters of your letter.

Click the "**Save**" button to send an email to selected user(s).

See also:

- Grid Row Javascript API: row.getKeys()
- JavaScript API:getSelectedRecordKeys()
- Button object: getNextSelectedRecord()
- Send an email to all users
- Send simple email
- runner_mail function
- Email selected records
- About Tabs/Sections API
- JavaScript API: RunnerPage object
- AJAX helper object: setMessage()
- About Dialog API

Send an email with updated fields only

To send an email with updated fields only, use the following code in Edit page: Before record updated event.

Note: Change red values to match your project.

```
$body = "Changed fields \n";
foreach ($values as $key => $value)
{
   if ($value!=$oldvalues[$key])
      $body .= $key . ": " . $value . "\n";
}
$email="test@test.com";
$subject="Sample subject";
runner_mail(array('to' => $email, 'subject' => $subject, 'body' => $body));
```

See also:

- Update selected
- Send an email to all users
- Send an email to selected users
- Send simple email
- runner_mail function
- Email selected records

Send mass email to all users

To send an email with new data to email addresses from the *user table*, use the following code in one of the following events:

- Add page: Before record added
- Add page: After record added
- Edit page: Before record updated
- Edit page: After record updated

Note: Change red values to match your project.

```
global $dal;
//select emails from Users table
$tblUsers = $dal->Table("UsersTableName");
$rs = $tblUsers->QueryAll();
while ($data = db_fetch_array($rs))
{
  $email=$data["EmailAddress"];
  $from="admin@test.com";
  $msg="Check what's hot this season";
  $subject="Monthly newsletter";
  $ret=runner_mail(array('to' => $email, 'subject' => $subject,
    'body' => $msg, 'from'=>$from));
  if(!$ret["mailed"])
    echo $ret["message"]."<br>";
}
```

See also:

- Data Access Layer
- About Database API
- Send simple email
- runner_mail function
- DAL method: QueryAll()

Send an email with attachment from the database

To send an email with attachments stored in a database field, use the following code in events like BeforeAdd/AfterAdd/BeforeEdit/AfterEdit.

Make sure to replace `"Field that stores attachments"` with the actual field name.

> **Note**: Attachments will only work when you selected "*Use custom mailer server settings*" under "Email settings".

> **Note**: Change red values to match your project.

```
$email= "test@gmail.com" ;
$msg="File Attached";
$subject="Attached some files";

$fileArray = my_json_decode($values["Field that stores attachments"]);
$attachments = array();
$msg = "";
foreach($fileArray as $f)
{
   $attachments[] = array("path" => $f["name"]);
}

$msg.= "Some field: ".$values["Some field"]."\r";
$msg.= "Some other field: ".$values["Some other field"]."\r";

$ret=runner_mail(array('to' => $email, 'subject' => $subject, 'body' => $msg,
"attachments" => $attachments));
if(!$ret["mailed"])
    echo $ret["message"];
```

See also:

- Send simple email
- runner_mail function
- How to email selected records as separate PDF files
- Email selected records
- Send an email to all users
- Send an email to selected users
- Send an email with updated fields only

3.1.2.5 Upload

Rename uploaded files

Each file uploaded to the disk has two names: *display name*, that you can see on the site, and *physical file name* on the disk.

Let's study how to change both *display* and *physical file* names. Use the following code in Add page: Before record added and/or Edit page: Before record updated events.

Note: Change red values to match your project.

1. Change the display name of the uploaded file

```
// get information about uploaded files
$fileArray = my_json_decode($values["fieldname"]);

// rename each file. In this example - convert to lowercase.
for($i = 0; $i < count($fileArray); $i++)
{
    $fileArray[$i]["usrName"] = strtolower($fileArray[$i]["usrName"]);
}

// update values of the field that stores file names
$values["fieldname"] = my_json_encode($fileArray);
```

Note: In this example, *fieldname* stores the names of uploaded files.

2. Change the physical name of the uploaded file

This code snippet changes the file names to the *LastName* field values.

```
// get information about uploaded files
$fileArray = my_json_decode($values["fieldname"]);

// rename files
for($i = 0; $i < count($fileArray); $i++)
{
    $fileName = $fileArray[$i]["name"];
    $newFileName = "files/".$values["LastName"].$i.".jpg";
    rename($fileName, getabspath($newFileName));
    $fileArray[$i]["name"] = $newFileName;
}

// update values of the field that stores file names
$values["fieldname"] = my_json_encode($fileArray);
```

Note: In this example, *fieldname* stores the names of uploaded files.

See also:

- Export/Import pages
- "View as" settings: File
- "Edit as" settings: File/Image

3.1.2.6 Misc

Check if the start date is ealier than the end date

Verify that the start date is earlier than the end date on the **Edit** page . Use the following code in the Edit page: Before record updated event.

> **Note**: Change red values to match your project.

```
if (toPHPTime($values["start_date"]) > toPHPTime($values["end_date"]))
{
   $message = "Start date can not be later than End date.";
   return false;
}
else
{
   return true;
}
```

See also:

- About Date Control API
- "Edit as" settings: Date
- AJAX helper object: setMessage()
- Store the date and time when a record is modified

Redirect to the profile edit page after successful login

To redirect to the profile edit page after successful login, use the following code in the Login page: After successful login event.

> **Note**: Change red values to match your project.

```
global $dal;
$tblUsers = $dal->Table("UserTableName");
$rs = $tblUsers->Query("UserNameField='".$username."' and
  PasswordField='".$password."'","");
$data = db_fetch_array($rs);
header("Location: tablename_edit.php?editid1=".$data["IDField"]);
exit();
```

See also:

- Data Access Layer
- About Date Control API
- Redirect to details page after master record was added
- Redirect to another page

Restrict access to PHPRunnerapplication by IP address

Restricting or allowing access by an **IP address** is an easy task. Here are a few examples. The code needs to be added to the beginning of the AfterAppInit event.

> **Note**: This tutorial uses IPv4 addresses.

> **Note**: Change red values to match your project.

Allow local access only:

```
if ($_SERVER['REMOTE_ADDR'] != '127.0.0.1') exit();
```

Allow access from the list of approved IP addresses:

```
$array = array('127.0.0.1', '96.24.12.122', '96.24.12.123');
if (!in_array($_SERVER['REMOTE_ADDR'], $array)) exit();
```

Restrict access from a certain IP address:

```
if ($_SERVER['REMOTE_ADDR'] == '123.123.123.123') exit();
```

Restrict access from the list of addresses:

```
$array = array('127.0.0.1', '96.24.12.122', '96.24.12.123');
if (in_array($_SERVER['REMOTE_ADDR'], $array)) exit();
```

See also:

- Security screen
- Advanced security settings
- Registration and passwords
- Active Directory

Save user data in session variables

Often you need to save data from the **login table** in *session* variables for later use. For this purpose, you can use the AfterSuccessfulLogin event.

> **Note**: Change red values to match your project.

```
function AfterSuccessfulLogin($username, $password, $data)
{
  $_SESSION["FirstName"] = $data["FirstName"];
}
```

Then you can use $_SESSION["FirstName"]as the default value of any field or in other events.

See also:

- Session keys
- PHPRunner session variables
- Session table variables in the WHERE clauses
- Speed up data entry using events
- Limit number of records users can add

Speed up data entry using events

When on the **Add** page, you might want to populate some edit boxes with previously used values. This can be done by using a combination of **Default** values and an AfterAdd Event.

Lat's say you need to enter several data records for each *Employee*. To populate the *Date* and *Employee* fields with the most recent values, you need to complete the following steps:

1. Set **Default** values of those fields to `$_SESSION["Date"]` and `$_SESSION["Employee"]` respectively. This can be done in the "Edit as" settings dialog.

2. Create an AfterAdd event and use the following code to save the *Date* and *Employee* values in Session variables:

```
$_SESSION["Date"] = $values["Date"];
$_SESSION["Employee"] = $values["Employee"];
```

See also:

- "Edit as" settings. Default values
- Session keys
- PHPRunner session variables
- Session table variables in the WHERE clauses

3.1.3 Global events

3.1.3.1 Global events

Global events can be used with any table/view.

A list of Global events:

- After application initialized
- After password changed
- After password reminder sent
- After successful login
- After successful registration
- After unsuccessful login
- After unsuccessful registration
- Before change password
- Before display
- Before login
- Before password reminder sent
- Before process
- Before registration
- JavaScript OnLoad
- Menu item: Modify

See also:

- Sample events
- Predefined actions
- Field events
- Tri-part events

3.1.3.2 Login page

Before process

Description

The event **BeforeProcess<PageName>** is executed before any processing takes place. Use this event to redirect user to another page, send an email or *log user action* in the database.

Syntax

```
BeforeProcess<PageName>($pageObject)
```

Arguments

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Recommended sample events:

- Add custom field to form
- Hide repeating values on View page
- Email selected records

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- Audit and record locking
- Event: OnAuditLog

Before login

The **BeforeLogin** event is executed on the **Login** page before verifying the *username* and *password*.

Syntax

```
BeforeLogin($username, $password, $message, $pageObject, $userdata)
```

Arguments

$username

a user-entered *login name*.

$password

a user-entered *password*.

$message

if the function returns **false**, place the message to be displayed into this variable.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

$userdata

an array with the user-entered data. Access fields by $userdata["FieldName"].

Return value

true: proceed with the *login process*. *Username* and *password* are validated against the Login table.

false: abort the *login procedure* and display the **"Invalid login**" message.

Applies to pages

Login.

Recommended predefined actions and sample events:

- Send a simple email

- Insert a record into another table
- Check to see if a specific record exists
- Display a message on the Web page
- Redirect to another page

See also:

- After successful login
- After unsuccessful login
- Security screen
- CAPTCHA on authentication pages
- Login form appearance
- Sign in with Google

After successful login

Description

The **AfterSuccessfulLogin** event is executed on the **Login** page after the username and password were successfully verified.

Syntax

```
AfterSuccessfulLogin($username,$password,$data)
```

Arguments

$username

a user-entered login name.

$password

a user-entered password.

$data

an array with existing user record in the *Login* table. Access fields by *$data["FieldName"]*.

> **Note**: field names are case-sensitive. If the field name is *PlayerId*, you should use *$data["PlayerId"]*. Note that *$data["playerid"]* or *$data["PlayerID"]* will not work.

In case of Active Directory based security, the data array contains attributes from the user record in AD. For instance, this is how you can retrieve the user's email and store it in a session variable:

```
$_SESSION["UserEmail"]=$data["email"];
```

Applies to pages

Login.

Recommended sample events:

- Add link to user profile to the menu
- Change 'Logged on as' message
- Redirect to user info edit page
- Save user data in session variables

See also:

- About Security API
- After unsuccessful login
- Before Login
- Security screen

After unsuccessful login

Description

The **AfterUnsuccessfulLogin** event is executed on the **Login** page if the user have failed authentication.

Syntax

```
AfterUnsuccessfulLogin($username,$password,$message,$pageObject)
```

Arguments

$username

a user-entered login name.

$password

a user-entered password.

$message

a place the message to be displayed into this variable.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

Login.

Recommended predefined actions and sample events:

- Send a simple email
- Insert a record into another table
- Check to see if a specific record exists
- Display a message on the Web page
- Redirect to another page

See also:

- Security API: checkUsernamePassword
- After successful login
- Before Login
- Audit and record locking

After Logout

Description

The **AfterLogout** event runs when the user clicks the **Logout button**.

The handler of this event is not called if the user closes the browser or the sessions times out.

Syntax

```
AfterLogout($username)
```

Arguments

`$username`

the *login name* of the user that logged out.

Return value

No return value.

Applies to pages

Login.

See also

- Redirect to another page
- After successful login
- Before Login
- After unsuccessful login
- Session keys
- Security screen
- Security API: logout

Before display

Description

The **Before Display** event is executed right before a page is displayed in the browser. Use this event to modify the value of any template variable or to define a new one.

Syntax

```
BeforeShow($xt, $pageObject)
```

Arguments

$xt

a template engine object. Use `$xt->assign($name, $val)` to assign a value `$val` to the variable `$name`.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Add, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Example

To display some text on the **List** page:

1. Proceed to the Editor page.

2. Switch to **HTML mode** and find the line *{END container_recordcontrols}*. Add the following code right before it:

```
<DIV>{$new_variable}</DIV>
```

3. Add the following code to the **BeforeDisplay** event.

Note: Change red values to match your project.

```
$new_variable = "test value";
$xt->assign("new_variable",$new_variable);
```

Recommended sample events:

- Hide controls on Add/Edit pages, based on logged user name

See also:

- Template language
- About templates
- PHPRunner templates
- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder

JavaScript OnLoad

Description

The **OnPageLoad** event occurs after the page is displayed in the browser. Use this event to work with the "*edit*" controls using the JavaScript API.

Syntax

```
OnPageLoad(pageObj,pageid,proxy,inlineRow)
```

Arguments

pageObj

an object representing the current page. For more information, see RunnerPage class.

pageid

the unique numeric identifier of the page.

proxy

data transferred from PHP code using $pageObject->setProxyValue function.

inlineRow

InlineRow object available in Add/Edit page events in the inline mode.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password.

Examples

List page behavior:

- How to calculate values (totals) on the fly
- How to control Inline Add/Edit functionality from script
- How to refresh List page after Edit in popup
- How to hide 'Edit selected'/'Delete selected' buttons
- How to display all Options on Search panel

Add/Edit lists behavior:

- How to ask for confirmation before saving the record
- How to control multi-step pages

Form and controls:

- Add custom field to form
- How to show dropdown list of US states if US was selected in country list
- How to enable/disable a button

Tweaking control appearance:

- Change width of edit box with AJAX popup

- Change width of text field on Quick Search panel
- Change font size in text box
- Change font in "edit" controls
- How to convert input into upper case

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- About Dialog API
- About Grid Row JavaScript API
- About JavaScript API
- JavaScript API: AJAX helper object
- AJAX-based features
- About Tabs/Sections API
- About PDF API
- Troubleshooting JavaScript errors

3.1.3.3 Menu page

Before process

Description

The event **BeforeProcess<PageName>** is executed before any processing takes place. Use this event to redirect user to another page, send an email or *log user action* in the database.

Syntax

```
BeforeProcess<PageName>($pageObject)
```

Arguments

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Recommended sample events:

- Add custom field to form
- Hide repeating values on View page
- Email selected records

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- Audit and record locking
- Event: OnAuditLog

Before display

Description

The **Before Display** event is executed right before a page is displayed in the browser. Use this event to modify the value of any template variable or to define a new one.

Syntax

```
BeforeShow($xt, $pageObject)
```

Arguments

`$xt`

a template engine object. Use `$xt->assign($name, $val)` to assign a value `$val` to the variable `$name`.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Add, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Example

To display some text on the **List** page:

1. Proceed to the Editor page.

2. Switch to **HTML mode** and find the line *{END container_recordcontrols}*. Add the following code right before it:

```
<DIV>{$new_variable}</DIV>
```

3. Add the following code to the **BeforeDisplay** event.

> **Note**: Change red values to match your project.

```
$new_variable = "test value";
$xt->assign("new_variable",$new_variable);
```

Recommended sample events:

- Hide controls on Add/Edit pages, based on logged user name

See also:

- Template language

- About templates
- PHPRunner templates
- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder

JavaScript OnLoad

Description

The **OnPageLoad** event occurs after the page is displayed in the browser. Use this event to work with the "*edit*" controls using the JavaScript API.

Syntax

```
OnPageLoad(pageObj,pageid,proxy,inlineRow)
```

Arguments

pageObj

an object representing the current page. For more information, see RunnerPage class.

pageid

the unique numeric identifier of the page.

proxy

data transferred from PHP code using $pageObject->setProxyValue function.

inlineRow

InlineRow object available in Add/Edit page events in the inline mode.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password.

Examples

List page behavior:

- How to calculate values (totals) on the fly
- How to control Inline Add/Edit functionality from script
- How to refresh List page after Edit in popup
- How to hide 'Edit selected'/'Delete selected' buttons
- How to display all Options on Search panel

Add/Edit lists behavior:

- How to ask for confirmation before saving the record
- How to control multi-step pages

Form and controls:

- Add custom field to form
- How to show dropdown list of US states if US was selected in country list
- How to enable/disable a button

Tweaking control appearance:

- Change width of edit box with AJAX popup
- Change width of text field on Quick Search panel
- Change font size in text box
- Change font in "edit" controls
- How to convert input into upper case

See also:

- Choose pages screen
- Creating charts
- Working with common pages

- Menu builder
- About Dialog API
- About Grid Row JavaScript API
- About JavaScript API
- JavaScript API: AJAX helper object
- AJAX-based features
- About Tabs/Sections API
- About PDF API
- Troubleshooting JavaScript errors

3.1.3.4 Register page

Before process

Description

The event **BeforeProcess<PageName>** is executed before any processing takes place. Use this event to redirect user to another page, send an email or *log user action* in the database.

Syntax

```
BeforeProcess<PageName>($pageObject)
```

Arguments

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Recommended sample events:

- Add custom field to form

- Hide repeating values on View page
- Email selected records

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- Audit and record locking
- Event: OnAuditLog

Before registration

Description

Function **BeforeRegister** is executed on the registration page.

Syntax

```
BeforeRegister($userdata,$message,$pageObject)
```

Arguments

$userdata

an array that stores values entered on the **Registration** page. To access a specific field value, use $userdata["FieldName"].

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $userdata["PlayerId"]. Note that $userdata["playerid"] or $userdata["PlayerID"] will not work.

$message

place the message to be displayed into this variable.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Return value

True: continue with registration.

False: prohibit registration.

Applies to pages

Registration.

Recommended predefined actions and sample events:

- Send a simple email
- Send an email with new data
- Save new data in another table
- Insert a record into another table
- Check to see if a specific record exists
- Display a message on the Web page
- Redirect to another page

See also:

- Grid Row JavaScript API: row.getFieldValue()
- Events.After successful registration
- Events.After unsuccessful registration
- Registration and passwords
- CAPTCHA on authentication pages
- Login form appearance
- Sign in with Google

After successful registration

Description

The **AfterSuccessfulRegistration** event is executed on the registration page after new user data was added to the database.

Syntax

```
AfterSuccessfulRegistration($userdata,$pageObject)
```

Arguments

`$userdata`

an array that stores values entered on the **Registration** page. To access a specific field value, use `$userdata["FieldName"]`.

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use `$userdata["PlayerId"]`. Note that `$userdata["playerid"]` or `$userdata["PlayerID"]` will not work.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

Registration.

Recommended predefined actions and sample events

- Send a simple email
- Insert a record into another table
- Check to see if a specific record exists
- Display a message on the Web page
- Redirect to another page

See also:

- Grid Row JavaScript API: row.getFieldValue()

- Events.After unsuccessful registration
- Security screen
- Before registration

After unsuccessful registration

Description

The **AfterUnsuccessfulRegistration** event is executed on the registration page if a new user record wasn't created.

Syntax

```
AfterUnsuccessfulRegistration($userdata,$message,$pageObject)
```

Arguments

$userdata

an array that stores values entered on the **Registration** page. To access a specific field value, use $userdata["FieldName"].

> **Note**: **Field names** are case-sensitive. If the field name is **PlayerId**, you should use $userdata["PlayerId"]. Note that $userdata["playerid"] or $userdata["PlayerID"] will not work.

$message

place the message to be displayed into this variable.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

Registration.

Recommended predefined actions and sample events

- Send a simple email

- Insert a record into another table
- Check to see if a specific record exists
- Display a message on the Web page
- Redirect to another page

See also:

- Grid Row JavaScript API: row.getFieldValue()
- Events.After successful registration
- Before registration

Before display

Description

The **Before Display** event is executed right before a page is displayed in the browser. Use this event to modify the value of any template variable or to define a new one.

Syntax

```
BeforeShow($xt, $pageObject)
```

Arguments

$xt

a template engine object. Use `$xt->assign($name, $val)` to assign a value `$val` to the variable `$name`.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Add, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Example

To display some text on the **List** page:

1. Proceed to the Editor page.

2. Switch to **HTML mode** and find the line *{END container_recordcontrols}*. Add the following code right before it:

```
<DIV>{$new_variable}</DIV>
```

3. Add the following code to the **BeforeDisplay** event.

> **Note**: Change red values to match your project.

```
$new_variable = "test value";
$xt->assign("new_variable",$new_variable);
```

Recommended sample events:

- Hide controls on Add/Edit pages, based on logged user name

See also:

- Template language
- About templates
- PHPRunner templates
- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder

JavaScript OnLoad

Description

The **OnPageLoad** event occurs after the page is displayed in the browser. Use this event to work with the "*edit*" controls using the JavaScript API.

Syntax

```
OnPageLoad(pageObj,pageid,proxy,inlineRow)
```

Arguments

`pageObj`

an object representing the current page. For more information, see RunnerPage class.

`pageid`

the unique numeric identifier of the page.

`proxy`

data transferred from PHP code using $pageObject->setProxyValue function.

`inlineRow`

InlineRow object available in Add/Edit page events in the inline mode.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password.

Examples

List page behavior:

- How to calculate values (totals) on the fly
- How to control Inline Add/Edit functionality from script
- How to refresh List page after Edit in popup
- How to hide 'Edit selected'/'Delete selected' buttons

- How to display all Options on Search panel

Add/Edit lists behavior:

- How to ask for confirmation before saving the record
- How to control multi-step pages

Form and controls:

- Add custom field to form
- How to show dropdown list of US states if US was selected in country list
- How to enable/disable a button

Tweaking control appearance:

- Change width of edit box with AJAX popup
- Change width of text field on Quick Search panel
- Change font size in text box
- Change font in "edit" controls
- How to convert input into upper case

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- About Dialog API
- About Grid Row JavaScript API
- About JavaScript API
- JavaScript API: AJAX helper object
- AJAX-based features
- About Tabs/Sections API
- About PDF API

- Troubleshooting JavaScript errors

3.1.3.5 Change password page

Before process

Description

The event **BeforeProcess<PageName>** is executed before any processing takes place. Use this event to redirect user to another page, send an email or *log user action* in the database.

Syntax

```
BeforeProcess<PageName>($pageObject)
```

Arguments

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Recommended sample events:

- Add custom field to form
- Hide repeating values on View page
- Email selected records

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder

- Audit and record locking
- Event: OnAuditLog

Before change password

Description

The **BeforeChangePassword** event is executed on the Change Password page before the password is changed.

Syntax

```
BeforeChangePassword($oldpassword,$newpassword,$pageObject)
```

Arguments

$oldpassword

an old password.

$newpassword

a new password.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Return value

True: allow the password to be changed.

False: do not allow the password to be changed.

Applies to pages

Change Password.

Recommended predefined actions and sample events

- Send a simple email

- Insert a record into another table
- Check to see if a specific record exists
- Display a message on the Web page
- Redirect to another page

See also:

- Registration and passwords
- How to hash your passwords manually
- After password change
- Working with common pages. Auxiliary pages

After password changed

Description

The **AfterChangePassword** event is executed on the Change Password page after the password was changed.

Syntax

```
AfterChangePassword($oldpassword,$newpassword,$pageObject)
```

Arguments

$oldpassword

an old password.

$newpassword

a new password.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

Change Password.

Recommended predefined actions and sample events

- Send a simple email
- Insert a record into another table
- Check to see if a specific record exists
- Display a message on the Web page
- Redirect to another page

See also:

- Registration and passwords
- How to hash your passwords manually
- Before change password
- Working with common pages. Auxiliary pages

Before display

Description

The **Before Display** event is executed right before a page is displayed in the browser. Use this event to modify the value of any template variable or to define a new one.

Syntax

```
BeforeShow($xt, $pageObject)
```

Arguments

`$xt`

a template engine object. Use `$xt->assign($name, $val)` to assign a value `$val` to the variable `$name`.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Add, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Example

To display some text on the **List** page:

1. Proceed to the Editor page.

2. Switch to **HTML mode** and find the line *{END container_recordcontrols}*. Add the following code right before it:

```
<DIV>{$new_variable}</DIV>
```

3. Add the following code to the **BeforeDisplay** event.

> **Note**: Change red values to match your project.

```
$new_variable = "test value";
$xt->assign("new_variable",$new_variable);
```

Recommended sample events:

- Hide controls on Add/Edit pages, based on logged user name

See also:

- Template language
- About templates
- PHPRunner templates
- Choose pages screen
- Creating charts

- Working with common pages
- Menu builder

JavaScript OnLoad

Description

The **OnPageLoad** event occurs after the page is displayed in the browser. Use this event to work with the "*edit*" controls using the JavaScript API.

Syntax

```
OnPageLoad(pageObj,pageid,proxy,inlineRow)
```

Arguments

pageObj

an object representing the current page. For more information, see RunnerPage class.

pageid

the unique numeric identifier of the page.

proxy

data transferred from PHP code using $pageObject->setProxyValue function.

inlineRow

InlineRow object available in Add/Edit page events in the inline mode.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password.

Examples

List page behavior:

- How to calculate values (totals) on the fly
- How to control Inline Add/Edit functionality from script
- How to refresh List page after Edit in popup
- How to hide 'Edit selected'/'Delete selected' buttons
- How to display all Options on Search panel

Add/Edit lists behavior:

- How to ask for confirmation before saving the record
- How to control multi-step pages

Form and controls:

- Add custom field to form
- How to show dropdown list of US states if US was selected in country list
- How to enable/disable a button

Tweaking control appearance:

- Change width of edit box with AJAX popup
- Change width of text field on Quick Search panel
- Change font size in text box
- Change font in "edit" controls
- How to convert input into upper case

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- About Dialog API
- About Grid Row JavaScript API
- About JavaScript API

- JavaScript API: AJAX helper object
- AJAX-based features
- About Tabs/Sections API
- About PDF API
- Troubleshooting JavaScript errors

3.1.3.6 Remind password page

Before process

Description

The event **BeforeProcess<PageName>** is executed before any processing takes place. Use this event to redirect user to another page, send an email or *log user action* in the database.

Syntax

```
BeforeProcess<PageName>($pageObject)
```

Arguments

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Recommended sample events:

- Add custom field to form
- Hide repeating values on View page
- Email selected records

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- Audit and record locking
- Event: OnAuditLog

Before password reminder sent

Description

The **BeforeRemindPassword** event is executed on the password reminder page before the password reminder is sent to the user's email.

Syntax

```
BeforeRemindPassword($username,$email,$pageObject)
```

Arguments

$username

the username entered by the user.

$email

the email entered by the user.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Return value

True: continue with the password reminder procedure.

False: cancel the password reminder procedure.

Applies to pages

Password Reminder

Recommended predefined actions and sample events:

- Send a simple email
- Insert a record into another table
- Check to see if a specific record exists
- Display a message on the Web page
- Redirect to another page

See also:

- Registration and passwords
- Security screen
- Working with common pages
- Before change password

After password reminder sent

Description

The **AfterRemindPassword** event is executed on the password reminder page after the password reminder is sent to the user's email.

Syntax

```
AfterRemindPassword($username,$email,$pageObject)
```

Arguments

$username

the username entered by the user.

$email

the email entered by the user.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

Password Reminder

Recommended predefined actions and sample events:

- Send a simple email
- Insert a record into another table
- Check to see if a specific record exists
- Display a message on the Web page
- Redirect to another page

See also:

- Registration and passwords
- Security screen
- Working with common pages
- Before change password

Before display

Description

The **Before Display** event is executed right before a page is displayed in the browser. Use this event to modify the value of any template variable or to define a new one.

Syntax

```
BeforeShow($xt, $pageObject)
```

Arguments

$xt

a template engine object. Use `$xt->assign($name, $val)` to assign a value `$val` to the variable `$name`.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Add, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Example

To display some text on the **List** page:

1. Proceed to the Editor page.

2. Switch to **HTML mode** and find the line *{END container_recordcontrols}*. Add the following code right before it:

```
<DIV>{$new_variable}</DIV>
```

3. Add the following code to the **BeforeDisplay** event.

> **Note**: Change red values to match your project.

```
$new_variable = "test value";
$xt->assign("new_variable",$new_variable);
```

Recommended sample events:

- Hide controls on Add/Edit pages, based on logged user name

See also:

- Template language

- About templates
- PHPRunner templates
- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder

JavaScript OnLoad

Description

The **OnPageLoad** event occurs after the page is displayed in the browser. Use this event to work with the "*edit*" controls using the JavaScript API.

Syntax

```
OnPageLoad(pageObj,pageid,proxy,inlineRow)
```

Arguments

`pageObj`

an object representing the current page. For more information, see RunnerPage class.

`pageid`

the unique numeric identifier of the page.

`proxy`

data transferred from PHP code using $pageObject->setProxyValue function.

`inlineRow`

InlineRow object available in Add/Edit page events in the inline mode.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password.

Examples

List page behavior:

- How to calculate values (totals) on the fly
- How to control Inline Add/Edit functionality from script
- How to refresh List page after Edit in popup
- How to hide 'Edit selected'/'Delete selected' buttons
- How to display all Options on Search panel

Add/Edit lists behavior:

- How to ask for confirmation before saving the record
- How to control multi-step pages

Form and controls:

- Add custom field to form
- How to show dropdown list of US states if US was selected in country list
- How to enable/disable a button

Tweaking control appearance:

- Change width of edit box with AJAX popup
- Change width of text field on Quick Search panel
- Change font size in text box
- Change font in "edit" controls
- How to convert input into upper case

See also:

- Choose pages screen
- Creating charts
- Working with common pages

- Menu builder
- About Dialog API
- About Grid Row JavaScript API
- About JavaScript API
- JavaScript API: AJAX helper object
- AJAX-based features
- About Tabs/Sections API
- About PDF API
- Troubleshooting JavaScript errors

3.1.3.7 After application initialized

Description

The **AfterAppInit** event is executed after the application is initialized.

Use this event to override any global PHPRunner variables. See the description area in the Event editor for the list of available global variables.

> **Note**: It is not recommended to use this event to display anything on the web page. Doing so may break your application.

Syntax

```
AfterAppInit()
```

Applies to pages

All pages.

Example

Let's say you need to troubleshoot your application by displaying an executed **SQL** query at the top of the page.

One way to do so is to proceed to the *include/appsettings.php* file and set the `$dDebug` variable to *true*. Though it might be tedious to change this variable back and forth.

By using the **AfterAppInit** event, you can display the debug info by adding *debug=true* to the URL.

AfterAppInit code:

```
if ($_REQUEST["debug"]=="true")
  $dDebug=true;
```

Sample URL: *categories_list.php?debug=true*.

Note: the Database connection is not yet open in this event. If you need to perform any database operations, open the database connection manually.

```
$conn = mysqli_connect("localhost:3306","root","root","dbname");
mysqli_query($conn,"INSERT INTO Log (LastAccesTime) Values(now())");
```

Note: in this event, you can also apply the Database API methods to interact with the database.

Recommended predefined actions and sample events:

- Send a simple email
- Display a message on the Web page
- Redirect to another page
- Restrict access to PHPRunner application by IP address

See also:

- Query Designer tab
- SQLQuery screen
- About SQLQuery API
- Troubleshooting tips

3.1.3.8 Menu item: Modify

Quick jump

Description

The **ModifyMenuItem** event is executed for each Menu item before the page is displayed in the browser. Use this event to modify or hide menu items.

Syntax

```
ModifyMenuItem($menuItem)
```

Arguments

`$menuItem`

a menu item object.

Return value

True: a menu item is shown.

False: a menu item is hidden.

Applies to pages

All pages with the Menu items.

Methods

`getLinkType()`

gets the link type.

> **Note**: The link types are: **Internal** (link to a page generated by PHPRunner, e.g. **List**, **Print** etc.), **External** (link to any external web page), **None** (if menu item is not a link: for example, a group or separator).

`getUrl()`

gets the URL of an external link.

`setUrl($url)`

sets the URL of the link and makes it external.

`getParams()`

gets the parameters of an internal link.

`setParams($params)`

sets the parameters of an internal link. These parameters may be also set on the Choose page screen using the '...' button next to the **List** page.

> **Note**: The parameters are a part of the link. E.g., if the parameters are *foo=bar&bar=foo*, the link is *...list.php?foo=bar&bar=foo*.

`setTitle($title)`

sets the title of the link.

`getTitle()`

gets the title of the link.

`getTable()`

gets the table name that an internal link points to.

`setTable($table)`

sets the table name.

`getPageType()`

gets the page type (**List**, **Add**, etc.).

`setPageType($pType)`

sets the page type (**List**, **Add**, **Search**, **Print**, **Report**, **Chart**).

Examples

Adding parameters to a menu item

```
if ($menuItem->getLinkType() == 'External')
   {
      $menuItem->setUrl('http://localhost/mn1/carsmodels_list.php');
   }
   else if($menuItem->getLinkType() == 'Internal')
   {
      $menuItem->setParams('id=30');
      if ($menuItem->getTable() == 'carsmake')
      {
         $menuItem->setTable('carsmodels');
      }
   }
   else
   {
      return false;
   }
   return true;
```

Hide some menu items based on the group of the authorized user

If the menu item is a link to an internal application page, you can assign the table permissions.

However, if the menu item is an external link, you need to set the permissions directly in the **Menu Item: Modify event**.

```
if ($_SESSION["GroupID"]!="manager")
{
   $title = $menuItem->getTitle();
   if ($title=="Yahoo Finance")
      return false;
}
return true;
```

Display a record counter next to each menu item

The code checks if the menu item is an internal table or view and adds the number of records to the menu item title.

```
if($menuItem->getLinkType() == 'Internal')
{
 global $tables_data;
 $table=$menuItem->getTable();
 include_once(getabspath("include/".$table."_settings.php"));
 $ps = new ProjectSettings($table);
 $table= $ps->getOriginalTableName();
 $rs=DB::Query("select count(*) as c from " . AddTableWrappers($table));
```

```
  $data = $rs.fetchAssoc();
  $menuItem->setTitle($menuItem->getTitle() . " (". $data["c"] . ")");
}
return true;
```

Hide some menu items in the Mobile mode

```
    if ($menuItem->getTable() == 'Cars' && MobileDetected()) {
        return false;
    }
    return true;
```

Recommended predefined actions and sample events:

- Check to see if a specific record exists

See also:

- Choose pages screen
- Menu builder
- Working with page elements
- RunnerPage class: hideItem()
- Datasource tables
- Session keys
- Save user data in session variables
- Database API: fetchAssoc()
- Database API: Query()
- Javascript API: toggleItem()
- Events. Redirect to another page

3.1.3.9 Before audit log

Description

The **Before audit log** event is executed before a record is added to the log.

Syntax

```
OnAuditLog($action,$params,$table,$keys,$newvalues,$oldvalues)
```

Arguments

$action

the occured action.

$params

$params[0]: the user's IP address, $params[1]: username.

$table

the modified table.

$keys

an array of key column values pointing to the current record.

$newvalues

an array of field values added to the database.

$oldvalues

an array of the previous field values. Applies to the **Edit** and **Delete** functions.

Return value

True: save the action in the log.

False: do not save the action in the log.

Applies to pages

All pages. Insert your code into the **Before audit log** event.

Example

If you do not want to record the actions done by the admin in the audit log, you can use the following code:

```
if ($params[1]=="admin")
   return false;

return true;
```

See also:

- Audit and record locking
- Security screen

3.1.4 Table events

3.1.4.1 Table events

Table events work with a specific table/view.

A list of Table events:

- After group of records deleted
- After import finished
- After record added
- After record deleted
- After record processed
- After record updated
- After table initialized
- Before display
- Before import started
- Before process
- Before record added
- Before record deleted
- Before record exported
- Before record inserted
- Before record processed

- Before record updated
- Copy page: OnLoad
- Custom Query
- Custom record fetch
- Get Row Count
- JavaScript OnLoad
- Process record values

See also:

- Sample events
- Predefined actions
- Field events
- Tri-part events
- Global events

3.1.4.2 Add page

Before process

Description

The event **BeforeProcess<PageName>** is executed before any processing takes place. Use this event to redirect user to another page, send an email or *log user action* in the database.

Syntax

```
BeforeProcess<PageName>($pageObject)
```

Arguments

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Recommended sample events:

- Add custom field to form
- Hide repeating values on View page
- Email selected records

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- Audit and record locking
- Event: OnAuditLog

Copy page: OnLoad

Description

The **CopyOnLoad** event is executed when an **Add** page is loaded in a Copy mode.

Syntax

```
CopyOnLoad($values,$where,$pageObject)
```

Arguments

$values

an array of values to be displayed on the page. You can modify the values of this array.

> **Note**: To access a specific field value use $values["FieldName"].

Example: to clear the *PassportNumber* field value before copying, use $values["PassportNumber"]="";.

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use `$values["PlayerId"]`. Note that `$values["playerid"]` or `$values["PlayerID"]` will not work.

> **Note**: If the field was assigned an alias in the **SQL query**, then the `$values` array will get the alias instead of the field name from the database.
>
> E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use `$values["Inv_Salesrep"]`.

`$where`

WHERE clause that points to the record to be copied. Example: ID=19.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

Add page (in Copy mode).

Recommended predefined actions and sample events:

- Send a simple email
- Send an email with old data
- Save old data in another table
- Insert a record into another table
- Check to see if a specific record exists
- Display a message on the Web page
- Redirect to another page

See also:

- Choose pages screen
- Export/Import pages
- About SQLQuery class
- About SQLQuery screen

- Additional WHERE tabs

Before record added

Description

The **BeforeAdd** event is executed before the record is physically added to the database. It works in all add modes: **Inline** Add, **Regular** Add and an **Add** page in **popup**.

Syntax

```
BeforeAdd($values,$message,$inline,$pageObject)
```

Arguments

$values

an array of values to be written to the database. To access a specific field value, use $values["FieldName"].

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $values["PlayerId"]. Note that $values["playerid"] or $values["PlayerID"] will not work.

> **Note**: If the field was assigned an alias in the **SQL query**, then the $values array will get the alias instead of the field name from the database.
>
> E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use $values["Inv_Salesrep"].

$message

place the message to be displayed into this variable.

$inline

equals to true for the **Inline Add**, false otherwise.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Return value

True: continue adding the record.

False: the record would not be added.

Applies to pages

Add, Inline Add.

Recommended sample events:

- Add custom field to form
- Limit number of records users can add
- Select multiple values from checkboxes or a list field and have them appear as individual database entries
- Update multiple tables
- Send mass email to al users
- Rename uploaded files
- Speed up data entry using events

See also:

- How to control Inline Add/Edit functionality from script
- Javascript API: InlineRow object
- After record added
- Choose pages screen
- About SQLQuery class
- About SQLQuery screen
- Additional WHERE tabs

Custom add

Description

The **CustomAdd** event is executed before the record is physically added to the database. It is designed to replace the standard Add procedure.

Use it when you do not want record to be added to the table in question.

Syntax

```
CustomAdd($values, $keys, $error, $inline, $pageObject)
```

Arguments

$values

an array of values to be written to the database. To access a specific field value, use $values["FieldName"].

Note: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $values["PlayerId"]. Note that $values["playerid"] or $values["PlayerID"] will not work.

Note: If the field was assigned an alias in the **SQL query**, then the $values array will get the alias instead of the field name from the database.

E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use $values["Inv_Salesrep"].

$keys

an array of key column values that point to the new record. To access a specific key column, use $keys["KeyFieldName"].

Note: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $keys["PlayerId"]. Note that $keys["playerid"] or $keys["PlayerID"] will not work.

$error

place the message to be displayed into this variable.

$inline

equals to true for the **Inline Add**, false otherwise.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Return value

True: if you want the application to handle adding the record.

False: if you have added the record with your code.

Applies to pages

Add, Inline Add.

Example

Lets consider the situation when records are never added directly to the main table (e.g., Cars).

Instead, records are added to the temporary *TempCars* table and then moved to the main *Cars* table once approved by the admin.

In this case, the following code in the **CustomAdd** event will do the job:

```
global $dal;
$tblTempCars = $dal->Table("TempCars");
$tblTempCars->Value["make"]=$values["make"];
$tblTempCars->Value["model"]=$values["model"];
$tblTempCars->Value["yearOfMake"]=$values["yearOfMake"];
$tblTempCars->Add();
return false;
```

Note: You may have noticed that the BeforeAdd event does the similar job. The main difference is that returning false in the **BeforeAdd** event is considered an error, and the user will see that something went wrong. Returning false in the **CustomAdd** event is perfectly legitimate, and application execution continues after that.

See also:

- How to control Inline Add/Edit functionality from script
- Javascript API: InlineRow object
- After record added

- Choose pages screen
- About SQLQuery class
- About SQLQuery screen
- Additional WHERE tabs
- DAL method: Add()

After record added

Description

The **AfterAdd** event is executed after the record is physically added to the database. It works in all add modes: **Inline** Add, **Regular** Add and an **Add** page in a **popup**.

Syntax

```
AfterAdd($values,$keys,$inline,$pageObject)
```

Arguments

$values

an array of values to be written to the database. To access a specific field value, use $values["FieldName"].

Note: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $values["PlayerId"]. Note that $values["playerid"] or $values["PlayerID"] will not work.

Note: If the field was assigned an alias in the **SQL query**, then the $values array will get the alias instead of the field name from the database.

E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use $values["Inv_Salesrep"].

$keys

an array of key column values that point to the edited record. To access a specific key column, use $keys["KeyFieldName"].

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use `$keys["PlayerId"]`. Note that `$keys["playerid"]` or `$keys["PlayerID"]` will not work.

`$inline`

equals to true for the **Inline Add**, false otherwise.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

> **Note**: If you need to display a message on the page or pass a variable value to **JavaScript**, you need to add the following to the end of your event code:

```
$pageObject->stopPRG = true;
```

Example

Passing `true` to *pageObject.stopPRG = true* variable named "*saved*":

```
$pageObject->stopPRG = true;
$pageObject->setProxyValue("saved", true);
```

Applies to pages

Add, Inline Add.

Recommended sample events:

- Add new button to Add/Edit pages
- Change message after record was added or saved
- Redirect to details page after master record was added
- Send mass email to al users
- Speed up data entry using events

See also:

- AJAX helper object: setMessage()
- About Javascript API
- setProxyValue

- Custom add
- Before record added
- Choose pages screen
- About SQLQuery class
- About SQLQuery screen
- Additional WHERE tabs

Process record values

Description

The **ProcessValues<PageName>** event is executed before the record is displayed. Use this event to modify the displayed field values.

Syntax

```
ProcessValues<PageName>($values,$pageObject)
```

Arguments

$values

an array of values to be written to the database. To access a specific field value, use $values["FieldName"].

Note: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $values["PlayerId"]. Note that $values["playerid"] or $values["PlayerID"] will not work.

Note: If the field was assigned an alias in the **SQL query**, then the $values array will get the alias instead of the field name from the database.

E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use $values["Inv_Salesrep"].

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Return value

No return value.

Applies to pages

View, Add, Edit.

Example

Display an empty "*Comment*" field when the user edits a record:

```
$values["Comment"]="";
```

See also:

- Choose pages screen
- About SQLQuery class
- About SQLQuery screen
- Additional WHERE tabs
- JavaScript API: RunnerPage object
- Send email with new data
- Hide empty fields on the View page
- Store the date and time when a record is modified
- Grid Row Javascript API: row.getFieldValue()
- Grid Row JavaScript API: row.setFieldValue()

Before display

Description

The **Before Display** event is executed right before a page is displayed in the browser. Use this event to modify the value of any template variable or to define a new one.

Syntax

```
BeforeShow($xt, $pageObject)
```

Arguments

$xt

a template engine object. Use $xt->assign($name, $val) to assign a value $val to the variable $name.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Add, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Example

To display some text on the **List** page:

1. Proceed to the Editor page.

2. Switch to **HTML mode** and find the line *{END container_recordcontrols}*. Add the following code right before it:

```
<DIV>{$new_variable}</DIV>
```

3. Add the following code to the **BeforeDisplay** event.

Note: Change red values to match your project.

```
$new_variable = "test value";
$xt->assign("new_variable",$new_variable);
```

Recommended sample events:

- Hide controls on Add/Edit pages, based on logged user name

See also:

- Template language
- About templates
- PHPRunner templates
- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder

JavaScript OnLoad

Description

The **OnPageLoad** event occurs after the page is displayed in the browser. Use this event to work with the "*edit*" controls using the JavaScript API.

Syntax

```
OnPageLoad(pageObj,pageid,proxy,inlineRow)
```

Arguments

pageObj

an object representing the current page. For more information, see RunnerPage class.

pageid

the unique numeric identifier of the page.

proxy

data transferred from PHP code using $pageObject->setProxyValue function.

inlineRow

InlineRow object available in Add/Edit page events in the inline mode.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password.

Examples

List page behavior:

- How to calculate values (totals) on the fly
- How to control Inline Add/Edit functionality from script
- How to refresh List page after Edit in popup
- How to hide 'Edit selected'/'Delete selected' buttons
- How to display all Options on Search panel

Add/Edit lists behavior:

- How to ask for confirmation before saving the record
- How to control multi-step pages

Form and controls:

- Add custom field to form
- How to show dropdown list of US states if US was selected in country list
- How to enable/disable a button

Tweaking control appearance:

- Change width of edit box with AJAX popup
- Change width of text field on Quick Search panel
- Change font size in text box
- Change font in "edit" controls
- How to convert input into upper case

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- About Dialog API
- About Grid Row JavaScript API
- About JavaScript API
- JavaScript API: AJAX helper object
- AJAX-based features
- About Tabs/Sections API
- About PDF API
- Troubleshooting JavaScript errors

3.1.4.3 Edit page

Before process

Description

The event **BeforeProcess<PageName>** is executed before any processing takes place. Use this event to redirect user to another page, send an email or *log user action* in the database.

Syntax

```
BeforeProcess<PageName>($pageObject)
```

Arguments

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Recommended sample events:

- Add custom field to form
- Hide repeating values on View page
- Email selected records

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- Audit and record locking
- Event: OnAuditLog

Before record updated

Description

The **BeforeEdit** event is executed before the data record is updated in the database. It works in all edit modes: **Inline** Edit, **Regular** Edit and an **Edit** page in a **popup**.

Syntax

```
BeforeEdit($values,$where,$oldvalues,$keys,$message,$inline,$pageObject)
```

Arguments

$values

an array of values to be written to the database. To access a specific field value, use $values["FieldName"].

Note: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $values["PlayerId"]. Note that $values["playerid"] or $values["PlayerID"] will not work.

> **Note**: If the field was assigned an alias in the **SQL query**, then the `$values` array will get the alias instead of the field name from the database.
>
> E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use `$values["Inv_Salesrep"]`.

`$where`

WHERE clause that points to the record to be copied. Example: ID=19.

`$oldvalues`

an array with existing field values. To access a specific column value, use `$oldvalues["FieldName"]`.

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use `$oldvalues["PlayerId"]`. Note that `$oldvalues["playerid"]` or `$oldvalues["PlayerID"]` will not work.

> **Note**: If the field was assigned an alias in the **SQL query**, then the `$values` array will get the alias instead of the field name from the database.
>
> E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use `$values["Inv_Salesrep"]`.

`$keys`

an array of key column values that point to the edited record. To access specific key column, use `$keys["KeyFieldName"]`.

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use `$keys["PlayerId"]`. Note that `$keys["playerid"]` or `$keys["PlayerID"]` will not work.

`$message`

place the message to be displayed into this variable.

`$inline`

equals to true for the **Inline Edit**, false otherwise.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Return value

True: the changes are saved.

False: the changes are not saved.

Applies to pages

Edit, Inline Edit.

Recommended sample events:

- Update multiple tables
- Send an email with updated fields only
- Send mass email to all users
- Rename uploaded files
- Check if start date is ealier than end date

See also:

- Choose pages screen
- About SQLQuery class
- About SQLQuery screen
- Additional WHERE tabs
- How to control Inline Add/Edit functionality from script
- Javascript API: InlineRow object
- Before record added
- Custom record update
- After record update updated

Custom record update

Description

The **CustomEdit** event is executed before the data record is updated in the database. It is designed to replace the standard Edit procedure.

Use it when you do not want record to be updated in the table in question.

Syntax

```
CustomEdit($values, $where, $oldvalues, $keys, $error, $inline, $pageObject)
```

Arguments

$values

an array of values to be written to the database. To access a specific field value, use $values["FieldName"].

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $values["PlayerId"]. Note that $values["playerid"] or $values["PlayerID"] will not work.

> **Note**: If the field was assigned an alias in the **SQL query**, then the $values array will get the alias instead of the field name from the database.
>
> E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use $values["Inv_Salesrep"].

$where

WHERE clause that points to the edited record. Example: ID=19.

$oldvalues

array with existing field values. To access a specific column value, use $oldvalues["FieldName"].

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $oldvalues["PlayerId"]. Note that $oldvalues["playerid"] or $oldvalues["PlayerID"] will not work.

> **Note**: If the field was assigned an alias in the **SQL query**, then the `$values` array will get the alias instead of the field name from the database.
>
> E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use `$values["Inv_Salesrep"]`.

`$keys`

an array of key column values that point to the edited record. To access a specific key column, use `$keys["KeyFieldName"]`.

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use `$keys["PlayerId"]`. Note that `$keys["playerid"]` or `$keys["PlayerID"]` will not work.

`$error`

place the message to be displayed into this variable.

`$inline`

equals to true for the **Inline Edit**, false otherwise.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Return value

True: if you want the application to handle updating the record.

False: if you have updated the record with your code.

Applies to pages

Edit, Inline Edit.

Example

Lets consider the situation when records are never edited directly in the main table (e.g. Cars).

Instead, records are added to the temporary *TempCars table* and then moved to the *main Cars table* once approved by admin.

In this case the following code in the **CustomAdd** event will do the job:

```
global $dal;
$tblTempCars = $dal->Table("TempCars");
$tblTempCars->Value["make"]=$values["make"];
$tblTempCars->Value["model"]=$values["model"];
$tblTempCars->Value["yearOfMake"]=$values["yearOfMake"];
$tblTempCars->Add();
return false;
```

See also:

- How to control Inline Add/Edit functionality from script
- Javascript API: InlineRow object
- Choose pages screen
- About SQLQuery class
- About SQLQuery screen
- Additional WHERE tabs
- User group permissions
- DAL method: Add()
- Before record update
- After record updated

Process record values

Description

The **ProcessValues<PageName>** event is executed before the record is displayed. Use this event to modify the displayed field values.

Syntax

```
ProcessValues<PageName>($values,$pageObject)
```

Arguments

`$values`

an array of values to be written to the database. To access a specific field value, use `$values["FieldName"]`.

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use `$values["PlayerId"]`. Note that `$values["playerid"]` or `$values["PlayerID"]` will not work.

> **Note**: If the field was assigned an alias in the **SQL query**, then the `$values` array will get the alias instead of the field name from the database.
>
> E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use `$values["Inv_Salesrep"]`.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Return value

No return value.

Applies to pages

View, Add, Edit.

Example

Display an empty "*Comment*" field when the user edits a record:

```
$values["Comment"]="";
```

See also:

- Choose pages screen
- About SQLQuery class
- About SQLQuery screen
- Additional WHERE tabs
- JavaScript API: RunnerPage object

- Send email with new data
- Hide empty fields on the View page
- Store the date and time when a record is modified
- Grid Row Javascript API: row.getFieldValue()
- Grid Row JavaScript API: row.setFieldValue()

After record updated

Description

The **AfterEdit** event is executed after the data record was updated in the database. It works in all edit modes: **Inline** Edit, **Regular** Edit and an **Edit** page in a **popup**.

Syntax

```
AfterEdit($values,$where,$oldvalues,$keys,$inline,$pageObject)
```

Arguments

`$values`

an array of values to be written to the database. To access a specific field value, use `$values["FieldName"]`.

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use `$values["PlayerId"]`. Note that `$values["playerid"]` or `$values["PlayerID"]` will not work.

> **Note**: If the field was assigned an alias in the **SQL query**, then the `$values` array will get the alias instead of the field name from the database.
>
> E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use `$values["Inv_Salesrep"]`.

`$where`

WHERE clause that points to the edited record. Example: ID=19.

`$oldvalues`

an array with existing field values. To access a specific column value, use `$oldvalues["FieldName"]`.

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use `$oldvalues["PlayerId"]`. Note that `$oldvalues["playerid"]` or `$oldvalues["PlayerID"]` will not work.

> **Note**: If the field was assigned an alias in the **SQL query**, then the `$values` array will get the alias instead of the field name from the database.
>
> E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use `$values["Inv_Salesrep"]`.

`$keys`

an array of key column values that point to the edited record. To access a specific key column, use `$keys["KeyFieldName"]`.

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use `$keys["PlayerId"]`. Note that `$keys["playerid"]` or `$keys["PlayerID"]` will not work.

`$inline`

equals to true for the **Inline Edit**, false otherwise.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

> **Note**: If you need to display a message on the page or pass a variable value to **JavaScript**, you need to add the following to the end of your event code:

```
$pageObject->stopPRG = true;
```

Example

Passing **`true`** to *pageObject.stopPRG = true* variable named "*saved*":

```
$pageObject->setProxyValue('saved', true);
$pageObject->stopPRG = true;
```

Applies to pages

Edit, Inline Edit.

Recommended sample events:

- Add new button to Add/Edit pages
- Change message after record was added or saved
- Send mass email to al users

See also:

- AJAX helper object: setMessage()
- About Javascript API
- setProxyValue
- Choose pages screen
- About SQLQuery class
- About SQLQuery screen
- Additional WHERE tabs
- Before record updated
- Custom record update

Before display

Description

The **Before Display** is executed right before a page is displayed in the browser. Use this event to modify the value of any template variable or to define a new one.

Syntax

```
BeforeShow($xt, $values, $pageObject)
```

Arguments

$xt

a template engine object. Use `$xt->assign($name, $val)` to assign a value `$val` to the variable `$name`. More info about template engine.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

$values

an array of values to be displayed on the page. To access a specific field value, use `$values["FieldName"]`.

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use `$values["PlayerId"]`. Note that `$values["playerid"]` or `$values["PlayerID"]` will not work.

> **Note**: If the field was assigned an alias in the **SQL query**, then the `$values` array will get the alias instead of the field name from the database.
>
> E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use `$values["Inv_Salesrep"]`.

Applies to pages

View, Edit.

Recommended sample events:

- Hide controls on Add/Edit pages, based on logged user name
- Hide empty fields on View page
- Before display event for all pages except View/Edit

See also:

- Template language
- About templates
- PHPRunner templates
- Choose pages screen

JavaScript OnLoad

Description

The **OnPageLoad** event occurs after the page is displayed in the browser. Use this event to work with the "*edit*" controls using the JavaScript API.

Syntax

```
OnPageLoad(pageObj,pageid,proxy,inlineRow)
```

Arguments

`pageObj`

an object representing the current page. For more information, see RunnerPage class.

`pageid`

the unique numeric identifier of the page.

`proxy`

data transferred from PHP code using $pageObject->setProxyValue function.

`inlineRow`

InlineRow object available in Add/Edit page events in the inline mode.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password.

Examples

List page behavior:

- How to calculate values (totals) on the fly
- How to control Inline Add/Edit functionality from script
- How to refresh List page after Edit in popup
- How to hide 'Edit selected'/'Delete selected' buttons

- How to display all Options on Search panel

Add/Edit lists behavior:

- How to ask for confirmation before saving the record
- How to control multi-step pages

Form and controls:

- Add custom field to form
- How to show dropdown list of US states if US was selected in country list
- How to enable/disable a button

Tweaking control appearance:

- Change width of edit box with AJAX popup
- Change width of text field on Quick Search panel
- Change font size in text box
- Change font in "edit" controls
- How to convert input into upper case

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- About Dialog API
- About Grid Row JavaScript API
- About JavaScript API
- JavaScript API: AJAX helper object
- AJAX-based features
- About Tabs/Sections API
- About PDF API

- Troubleshooting JavaScript errors

3.1.4.4 List page

Before process

Description

The event **BeforeProcess<PageName>** is executed before any processing takes place. Use this event to redirect user to another page, send an email or *log user action* in the database.

Syntax

```
BeforeProcess<PageName>($pageObject)
```

Arguments

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Recommended sample events:

- Add custom field to form
- Hide repeating values on View page
- Email selected records

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder

- Audit and record locking
- Event: OnAuditLog

Before record deleted

Description

The **BeforeDelete** event is executed before the record is deleted.

Syntax

```
BeforeDelete($where,$deleted_values,$message,$pageObject)
```

Arguments

$where

a **WHERE clause** that points to the record to be copied. Example: ID=19.

$deleted_values

an array with the field values from the record to be deleted. To access a specific field value, use $deleted_values["FieldName"].

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $deleted_values["PlayerId"]. Note that $deleted_values["playerid"] or $deleted_values["PlayerID"] will not work.

$message

place the message to be displayed into this variable.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Return value

True - the record gets deleted.

False - the record does not get deleted.

Applies to pages

List.

Recommended sample events:

- Before deleting a record check if related records exist
- Update multiple records on the List page

See also:

- After record deleted
- After group of records deleted
- Database API: Delete
- RunnerPage class:setTabWhere()

After record deleted

Description

The **AfterDelete** event is executed after the record was deleted.

Syntax

```
AfterDelete($where,$deleted_values,$message,$pageObject)
```

Arguments

$where

a **WHERE clause** that points to the record to be copied. Example: ID=19.

$deleted_values

an array with the field values from the record to be deleted. To access a specific field value, use $deleted_values["FieldName"].

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use `$deleted_values["PlayerId"]`. Note that `$deleted_values["playerid"]` or `$deleted_values["PlayerID"]` will not work.

`$message`

place the message to be displayed into this variable.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List.

Recommended predefined actions and sample events:

- Send a simple email
- Insert a record into another table
- Check to see if a specific record exists
- Display a message on the Web page
- Redirect to another page

See also:

- Before record deleted
- After group of records deleted
- Database API: Delete
- RunnerPage class:setTabWhere()

After group of records deleted

Description

The **AfterMassDelete** event is executed after the bulk delete operation.

Syntax

```
AfterMassDelete($records_deleted,$pageObject)
```

Arguments

$records_deleted

a number of records that were deleted.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List.

Recommended predefined actions and sample events

- Send a simple email
- Insert a record into another table
- Check to see if a specific record exists
- Display a message on the Web page
- Redirect to another page

See also:

- After record deleted
- Before record deleted
- Database API: Delete
- RunnerPage class:setTabWhere()

Before record processed

Description

The **BeforeProcessRow<PageName>** event is executed right after a database record was retrieved from the database and before the formatting is applied.

Syntax

```
BeforeProcessRow<PageName>($data,$pageObject)
```

Arguments

$data

an array of field values of the record being processed. To access a specific field value, use $data["FieldName"].

Note: You can read the $data array and also write to it changing the values before showing them on the page.

Note: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $data["PlayerId"]. Note that $data["playerid"] or $data["PlayerID"] will not work.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Return value

True - the record is shown on the page.

False - record is skipped.

Applies to pages

List, Print.

Recommended predefined actions and sample events:

- Send a simple email
- Insert a record into another table
- Check to see if a specific record exists

- Display a message on the Web page
- Redirect to another page

See also:

- Database API:Select()
- Printer-friendly/PDF view settings
- About Grid Row JavaScript API
- About RunnerPage class
- About Database API
- After record processed

After record processed

Description

The **BeforeMoveNext<PageName>** event is executed after a record was processed and the formatting applied.

Syntax

```
BeforeMoveNext<PageName>($data,$row,$record,$pageObject)
```

Arguments

$data

an array of field values of the record being processed. To access a specific field value, use `$data["FieldName"]`.

Note: You can read the `$data` array and also write to it changing the values before showing them on the page.

Note: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use `$data["PlayerId"]`. Note that `$data["playerid"]` or `$data["PlayerID"]` will not work.

$row

an array representing a row on the page.

`$record`

an array representing a table record on the page.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Print.

Recommended sample events:

- Change cell background color
- Change row background color
- Disable record editing

See also:

- Conditional formatting
- Grid Row JavaScript API: row.getFieldValue()
- ProcessValues<PageName>
- Before record processed
- About Grid Row JavaScript API
- Printer-friendly/PDF view settings

Before SQL query

Description

The **BeforeQuery<PageName>** event is executed before the *SELECT* SQL query is processed. Use this event if you like to modify the default SQL query, add a WHERE clause, etc.

Syntax

```
BeforeQuery<PageName>($strSQL,$strWhereClause,$strOrderBy,$pageObject)
```

Arguments

$strSQL

an SQL query to be executed.

$strWhereClause

a **WHERE clause** to apply to the SQL query.

$strOrderBy

an ORDER BY query to apply to the SQL query.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Print, Export, Chart.

See also:

- Choose page screen
- SQL query screen
- About SQLQuery class
- List query

Before display

Description

The **Before Display** event is executed right before a page is displayed in the browser. Use this event to modify the value of any template variable or to define a new one.

Syntax

```
BeforeShow($xt, $pageObject)
```

Arguments

$xt

a template engine object. Use `$xt->assign($name, $val)` to assign a value `$val` to the variable `$name`.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Add, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Example

To display some text on the **List** page:

1. Proceed to the Editor page.

2. Switch to **HTML mode** and find the line *{END container_recordcontrols}*. Add the following code right before it:

```
<DIV>{$new_variable}</DIV>
```

3. Add the following code to the **BeforeDisplay** event.

Note: Change red values to match your project.

```
$new_variable = "test value";
$xt->assign("new_variable",$new_variable);
```

Recommended sample events:

- Hide controls on Add/Edit pages, based on logged user name

See also:

- Template language
- About templates
- PHPRunner templates
- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder

JavaScript OnLoad

Description

The **OnPageLoad** event occurs after the page is displayed in the browser. Use this event to work with the "*edit*" controls using the JavaScript API.

Syntax

```
OnPageLoad(pageObj,pageid,proxy,inlineRow)
```

Arguments

`pageObj`

an object representing the current page. For more information, see RunnerPage class.

`pageid`

the unique numeric identifier of the page.

`proxy`

data transferred from PHP code using $pageObject->setProxyValue function.

`inlineRow`

InlineRow object available in Add/Edit page events in the inline mode.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password.

Examples

List page behavior:

- How to calculate values (totals) on the fly
- How to control Inline Add/Edit functionality from script
- How to refresh List page after Edit in popup
- How to hide 'Edit selected'/'Delete selected' buttons
- How to display all Options on Search panel

Add/Edit lists behavior:

- How to ask for confirmation before saving the record
- How to control multi-step pages

Form and controls:

- Add custom field to form
- How to show dropdown list of US states if US was selected in country list
- How to enable/disable a button

Tweaking control appearance:

- Change width of edit box with AJAX popup
- Change width of text field on Quick Search panel
- Change font size in text box
- Change font in "edit" controls
- How to convert input into upper case

See also:

- Choose pages screen
- Creating charts
- Working with common pages

- Menu builder
- About Dialog API
- About Grid Row JavaScript API
- About JavaScript API
- JavaScript API: AJAX helper object
- AJAX-based features
- About Tabs/Sections API
- About PDF API
- Troubleshooting JavaScript errors

Get Row Count

Description

The **ListGetRowCount** event is executed before the **List** page is processed. Use this function when your database doesn't support the record count.

You need to use this event together with ListFetchArray and ListQuery events. Read about it in the article How to display data returned by stored procedure.

Syntax

```
ListGetRowCount($searchObj,$masterTable,$masterKeysReq,$selectedRecords,$pageObject)
```

Arguments

$searchObj

an instance of the class which performs search.

$masterTable

a master table name.

$masterKeysReq

an array of keys.

$selectedRecords

an array of selected records on the **Print** page, null on the **List** page.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Return value

The function returns the number of records or **false**.

Applies to pages

List, Print.

Examples

1. Return a hardcoded number of records:

```
return 10;
```

2. Calculate and return the number of records

```
return DBLookup("select count(*) from test");
```

See also:

- DAL method: DBLookup
- About Database API
- Connecting to the database
- How to display data returned by stored procedure
- List query
- About Search API

Custom Query

Description

The **ListQuery** event is executed before the **List** page is processed. Use this event when the data set is more complicated than the result of an SQL query.

Here is the article that shows how to use ListFetchArray and **ListQuery** events together: How to display data returned by stored procedure

Syntax

```
ListQuery($searchObj
,$orderBy
,$howOrderBy
,$masterTable,$masterKeysReq,$selectedRecord,$pageSize,$myPage,$pageObject)
```

Arguments

$searchObj

an instance of the class which performs search.

$orderBy

an array with field order.

$howOrderBy

an array with sort type for **'orderBy'** array.

$masterTable

a master table name.

$masterKeysReq

an array of keys.

$selectedRecord

an array of selected records on the **Print** page, null on the **List** page.

$pageSize

a number of records per page.

$myPage

the current page number.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Return value

The function returns a data array or **false**.

Applies to pages

List, Print.

See also:

- Event: ListGetRowCount
- Connecting to the database
- How to display data returned by stored procedure
- About Search API
- SQL query screen
- About SQLQuery class

Custom record fetch

Description

The **ListFetchArray** event fetches records from the given array and returns them as array. Use this function in conjunction with the Custom query function.

Here is an article that shows how to use **ListFetchArray** and ListQuery events together: How to display data returned by stored procedure

Syntax

```
ListFetchArray($rs,$pageObject)
```

Arguments

`$rs`

an array of values returned from the Custom query event.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Print.

See also:

- Database API:Query()
- Database API: fetchNumeric()
- Database API: fetchAssoc()
- Events. Check if specific record exists
- Add a dropdown list box with values for the search
- Event: ListGetRowCount

3.1.4.5 Report page

Before process

Description

The event **BeforeProcess<PageName>** is executed before any processing takes place. Use this event to redirect user to another page, send an email or *log user action* in the database.

Syntax

```
BeforeProcess<PageName>($pageObject)
```

Arguments

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Recommended sample events:

- Add custom field to form

- Hide repeating values on View page
- Email selected records

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- Audit and record locking
- Event: OnAuditLog

Before display

Description

The **Before Display** event is executed right before a page is displayed in the browser. Use this event to modify the value of any template variable or to define a new one.

Syntax

```
BeforeShow($xt, $pageObject)
```

Arguments

$xt

a template engine object. Use `$xt->assign($name, $val)` to assign a value `$val` to the variable `$name`.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Add, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Example

To display some text on the **List** page:

1. Proceed to the Editor page.

2. Switch to **HTML mode** and find the line *{END container_recordcontrols}*. Add the following code right before it:

```
<DIV>{$new_variable}</DIV>
```

3. Add the following code to the **BeforeDisplay** event.

> **Note**: Change red values to match your project.

```
$new_variable = "test value";
$xt->assign("new_variable",$new_variable);
```

Recommended sample events:

- Hide controls on Add/Edit pages, based on logged user name

See also:

- Template language
- About templates
- PHPRunner templates
- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder

Before SQL query

Description

The **BeforeQuery<PageName>** event is executed before a *SELECT* SQL query is processed. Use this event if you like to modify default SQL query, add a WHERE clause, etc.

Syntax

```
BeforeQuery<PageName>($strSQL,$strWhereClause,$strOrderBy,$pageObject)
```

Arguments

$strSQL

an SQL query to be executed.

$strWhereClause

a **WHERE clause** to apply to the SQL query.

$strOrderBy

ORDER BY query to apply to the SQL query.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Print, Export, Chart.

See also:

- Choose page screen
- SQL query screen
- About SQLQuery class

JavaScript OnLoad

Description

The **OnPageLoad** event occurs after the page is displayed in the browser. Use this event to work with the "*edit*" controls using the JavaScript API.

Syntax

```
OnPageLoad(pageObj,pageid,proxy,inlineRow)
```

Arguments

pageObj

an object representing the current page. For more information, see RunnerPage class.

pageid

the unique numeric identifier of the page.

proxy

data transferred from PHP code using $pageObject->setProxyValue function.

inlineRow

InlineRow object available in Add/Edit page events in the inline mode.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password.

Examples

List page behavior:

- How to calculate values (totals) on the fly
- How to control Inline Add/Edit functionality from script
- How to refresh List page after Edit in popup
- How to hide 'Edit selected'/'Delete selected' buttons
- How to display all Options on Search panel

Add/Edit lists behavior:

- How to ask for confirmation before saving the record

- How to control multi-step pages

Form and controls:

- Add custom field to form
- How to show dropdown list of US states if US was selected in country list
- How to enable/disable a button

Tweaking control appearance:

- Change width of edit box with AJAX popup
- Change width of text field on Quick Search panel
- Change font size in text box
- Change font in "edit" controls
- How to convert input into upper case

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- About Dialog API
- About Grid Row JavaScript API
- About JavaScript API
- JavaScript API: AJAX helper object
- AJAX-based features
- About Tabs/Sections API
- About PDF API
- Troubleshooting JavaScript errors

3.1.4.6 Chart page

Before process

Description

The event **BeforeProcess<PageName>** is executed before any processing takes place. Use this event to redirect user to another page, send an email or *log user action* in the database.

Syntax

```
BeforeProcess<PageName>($pageObject)
```

Arguments

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Recommended sample events:

- Add custom field to form
- Hide repeating values on View page
- Email selected records

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- Audit and record locking
- Event: OnAuditLog

Before display

Description

The **Before Display** event is executed right before a page is displayed in the browser. Use this event to modify the value of any template variable or to define a new one.

Syntax

```
BeforeShow($xt, $pageObject)
```

Arguments

$xt

a template engine object. Use `$xt->assign($name, $val)` to assign a value `$val` to the variable `$name`.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Add, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Example

To display some text on the **List** page:

1. Proceed to the Editor page.

2. Switch to **HTML mode** and find the line *{END container_recordcontrols}*. Add the following code right before it:

```
<DIV>{$new_variable}</DIV>
```

3. Add the following code to the **BeforeDisplay** event.

Note: Change red values to match your project.

```
$new_variable = "test value";
$xt->assign("new_variable",$new_variable);
```

Recommended sample events:

- Hide controls on Add/Edit pages, based on logged user name

See also:

- Template language
- About templates
- PHPRunner templates
- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder

Before SQL query

Description

The **BeforeQuery<PageName>** event is executed before a *SELECT* SQL query is processed. Use this event if you like to modify default SQL query, add a WHERE clause, etc.

Syntax

```
BeforeQuery<PageName>($strSQL,$strWhereClause,$strOrderBy,$pageObject)
```

Arguments

`$strSQL`

an SQL query to be executed.

`$strWhereClause`

a **WHERE clause** to apply to the SQL query.

`$strOrderBy`

ORDER BY query to apply to the SQL query.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Print, Export, Chart.

See also:

- Choose page screen
- SQL query screen
- About SQLQuery class

ChartModify

Quick jump	
Add horizontal scroller	Customize chart title
Add horizontal scroller and set initial zoom to 50%	Values formatting
Change labels color and font size	Multiple Y-axis
Series appearance	Adding range markers
Change colors in pie/doughnut chart	Create two series chart with two Y-Axis

Description

The **ChartModify** event occurs before a chart is displayed. Use this event to modify the chart settings.

Syntax

```
ChartModify(chart, proxy, pageObj)
```

Arguments

chart

a chart object.

proxy

data transferred from PHP code using setProxyValue function.

pageObj

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

Chart.

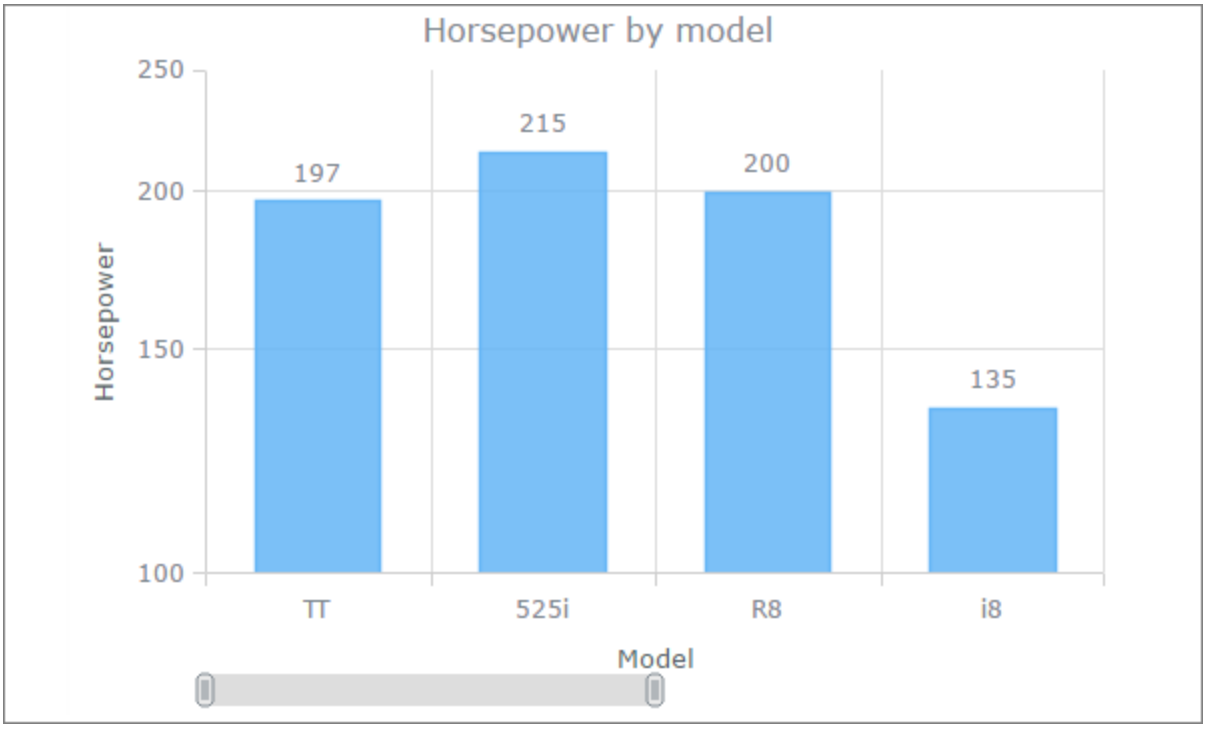
Official **AnyChart** documentation: https://api.anychart.com/.

Examples

Example 1. Add a horizontal scroller.

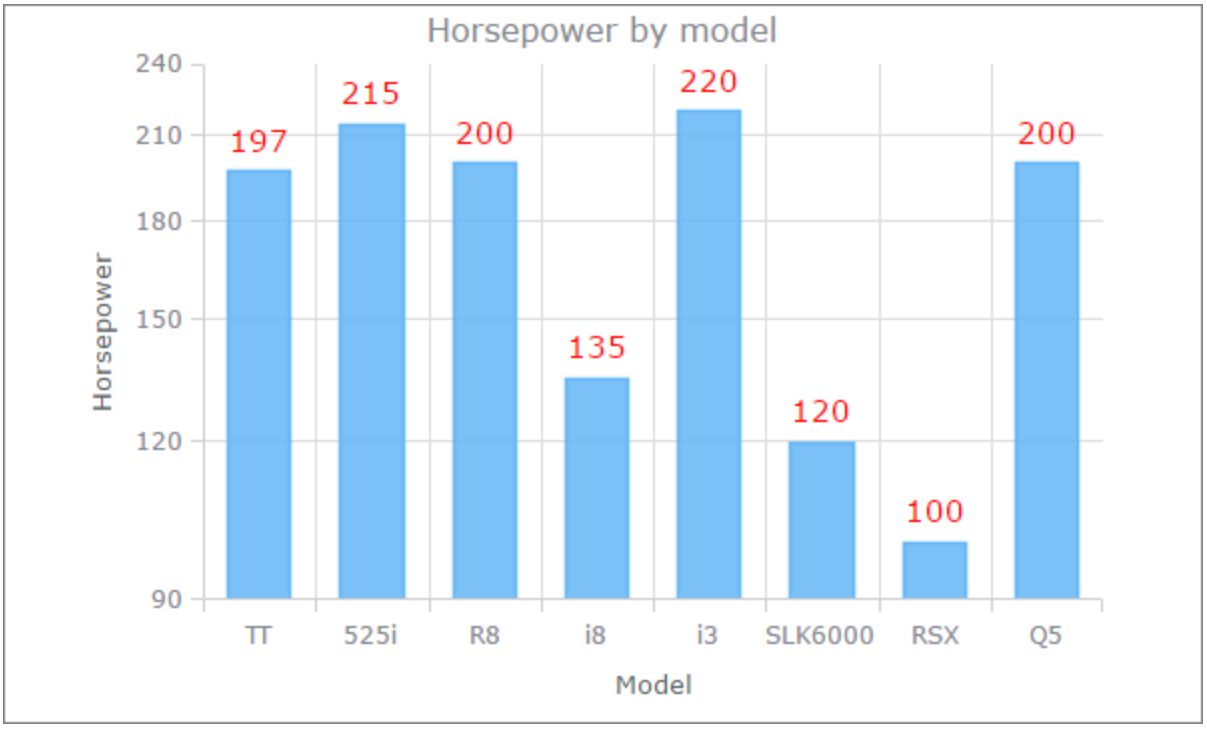
```
var currentScroller = chart.xScroller();
currentScroller.enabled(true);
```

Example 2. Add a horizontal scroller and set the initial zoom to 50%.




```
var zoom = chart.xZoom();
zoom.setTo(0, 0.5);

var currentScroller = chart.xScroller();
currentScroller.enabled(true);
```

Example 3. Change labels color and font size.



Using separate API calls to set the series color and font size:

```
// Gets the series by index, 0 - the first series, 1 - the second series, etc.
var series = chart.getSeriesAt(0);
series.labels(true);
series.labels().fontSize(15);
series.labels().fontColor("#ff0000");
```

Setting several label parameters in one go:

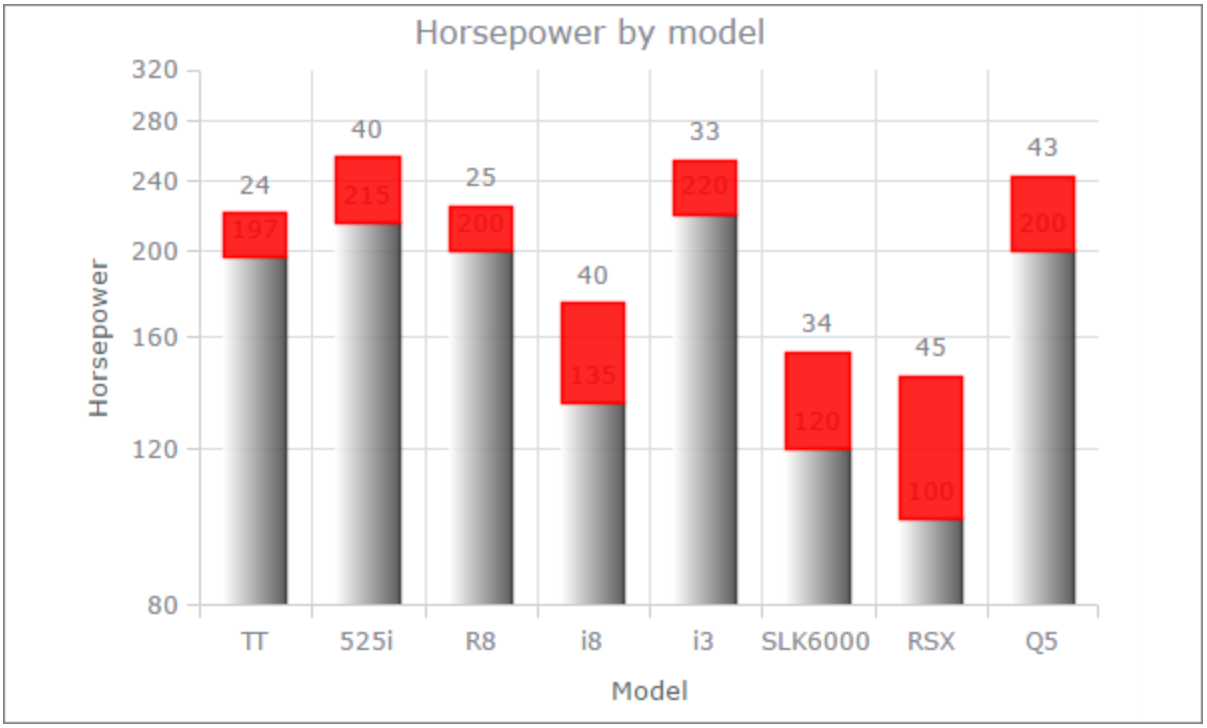
```
// Gets the series by index, 0 - the first series, 1 - the second series, etc.
var series = chart.getSeriesAt(0);
series.labels(true);
series.labels.{fontSize: 15, fontColor: "#ff0000"});
```

Example 4. The series appearance.

Single color:

```
// Gets the series by index, 0 - the first series, 1 - the second series, etc.
var series = chart.getSeriesAt(0);
series.color("#FF0000", 0.25);
```

Gradient fill:




```
// Gets the series by index, 0 - the first series, 1 - the second series, etc.
var series = chart.getSeriesAt(0);
series.color(["#FEFEFE", "#424242"], 0.69, 0.59);
```

Example 5. Customize the chart title.

Change the title color:

```
chart.title().fontColor("#FF0000");
```

Change the title and color:

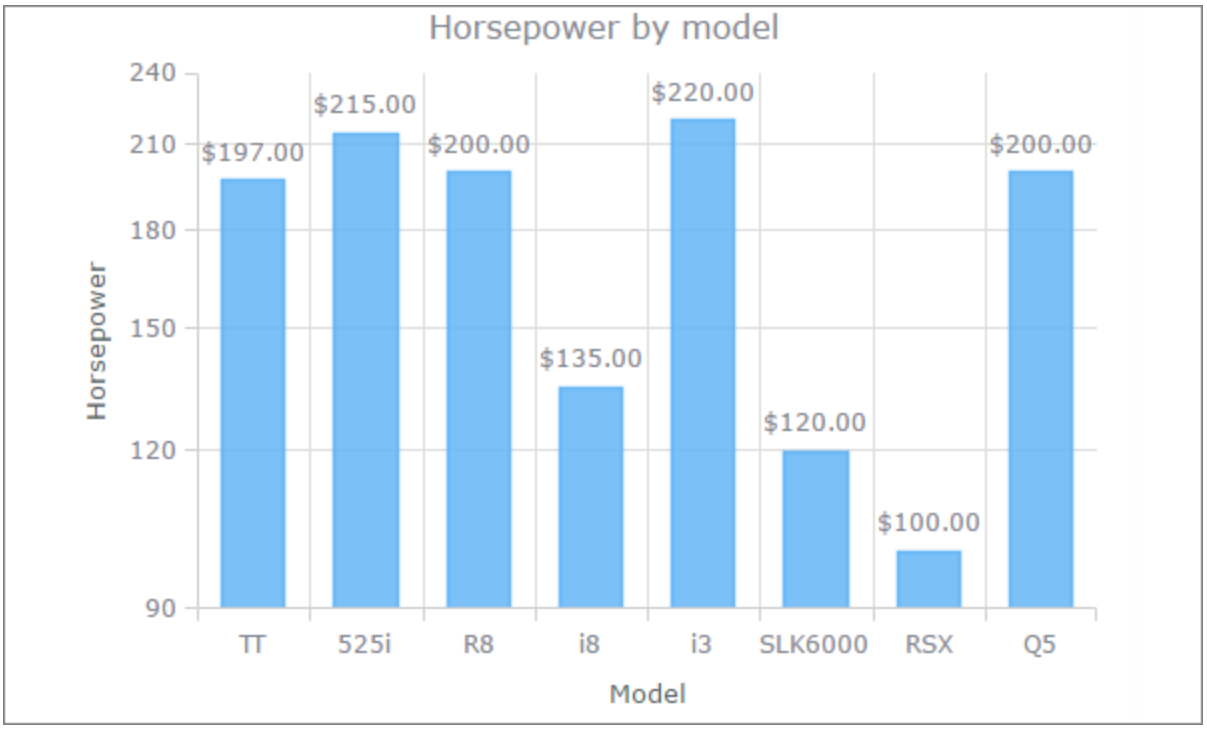
```
chart.title({text: "Custom title", fontColor: "#F44336"});
```

Disable the title:

```
chart.title(false);
```

Example 6. Values formatting.

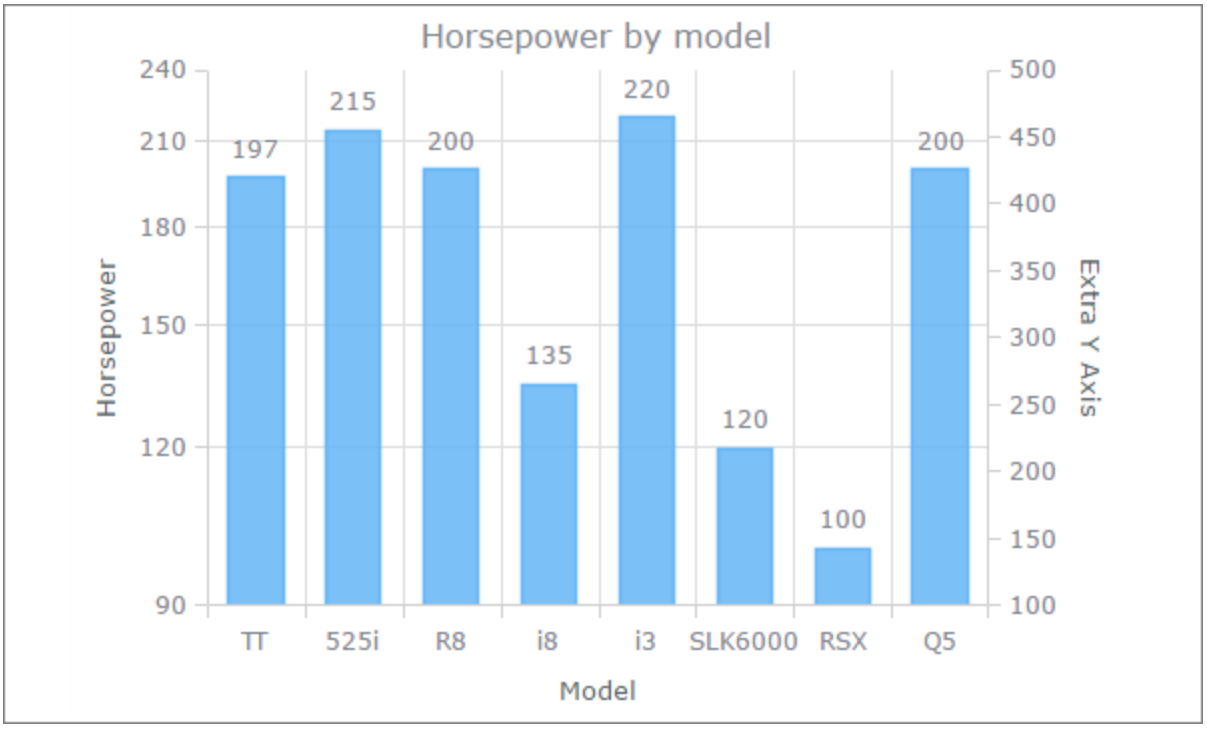
Formatting values as currency. You can use any JavaScript code to format values.




```
chart.labels().format (function(){
   var num = Number( this.value );
   return(("$"+ num.toFixed(2)));
});
```

Example 7. Multiple Y-axes.

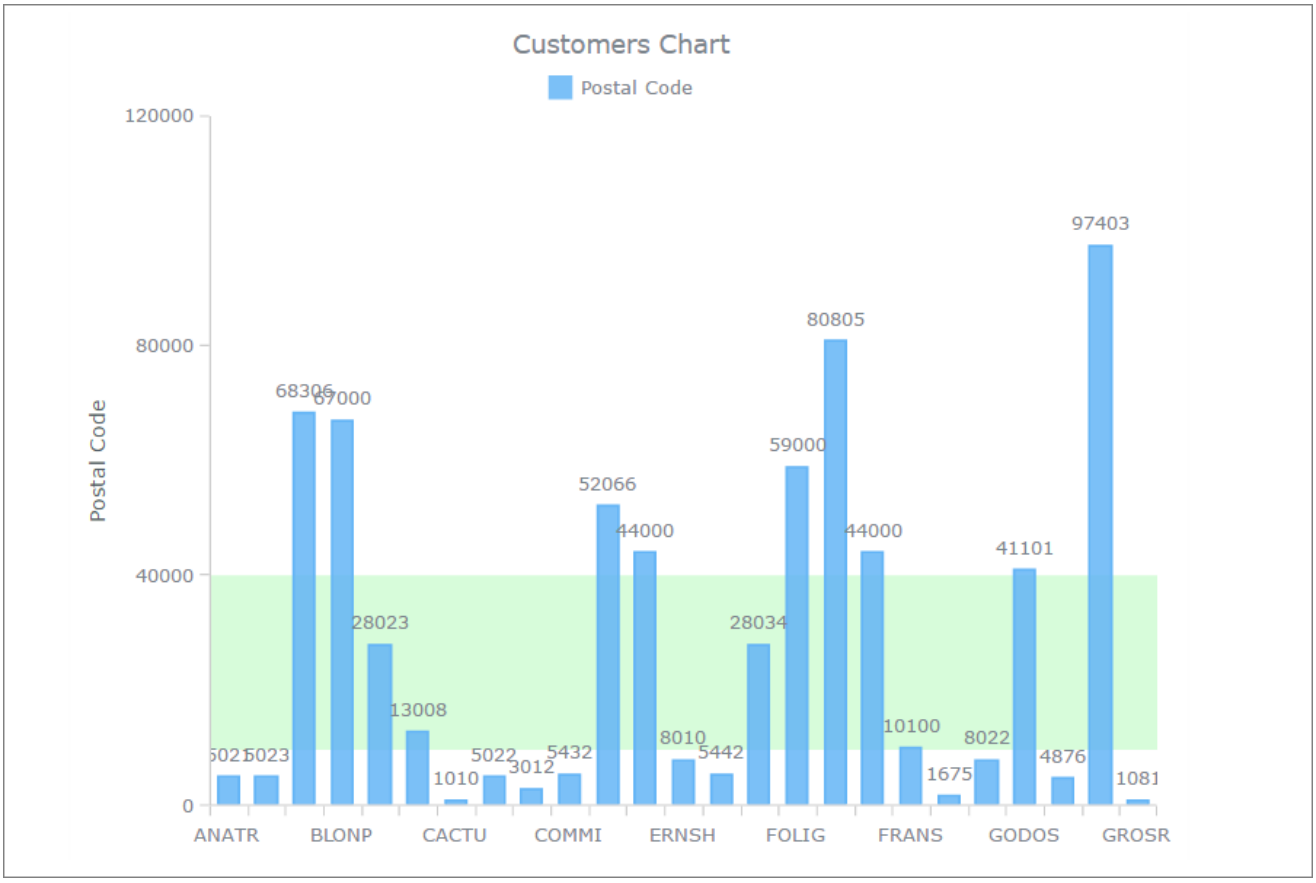
Here is how to add an extra Y-axis on the right ranging from 100 to 500 with ticks after each 50 points:




```
var extraYScale = anychart.scales.linear();
extraYScale.minimum(100);
extraYScale.maximum(500);
var extraTicks = extraYScale.ticks();
extraTicks.interval(50);

// Create and tune an additional y-axis
var extraYAxis = chart.yAxis(1);
extraYAxis.orientation("right");
extraYAxis.scale(extraYScale);
extraYAxis.title("Extra Y Axis");
```

Example 8. Adding range markers.

Here is how to add range markers:



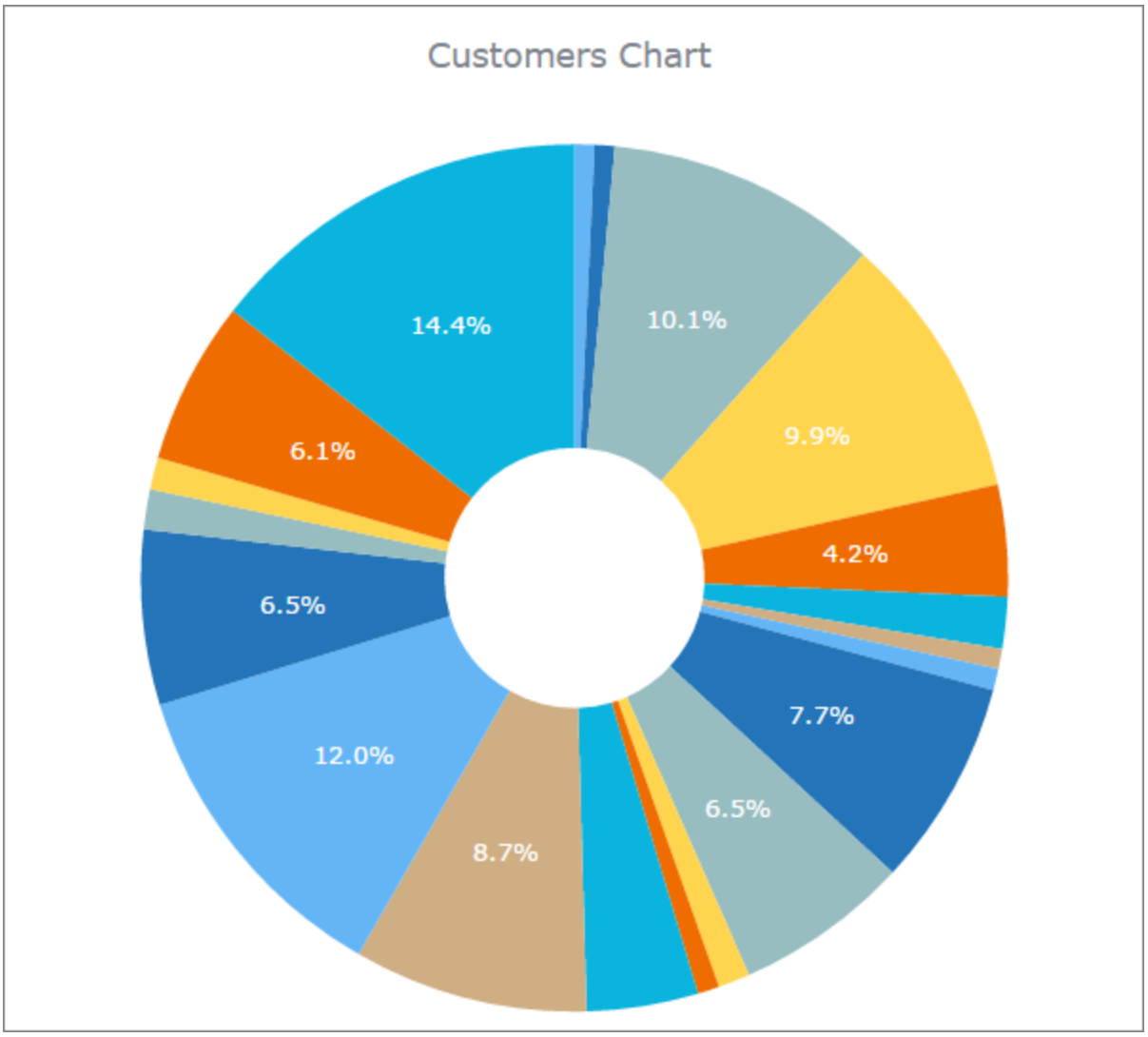

```
var vMarker = chart.rangeMarker(0);
vMarker.from("10000");
vMarker.to("40000");
vMarker.axis(chart.yAxis());
vMarker.fill("#d7fcda");
```

You can find more info at
http://playground.anychart.com/docs/7.13.0/samples/AGST_Range_Marker_04-plain.

Example 9. How to change colors of individual slices in a pie/doughnut chart.

```
var palette = anychart.palettes.distinctColors();
palette.items(["#64B5F6", "#2374B8", "#97BDC1", "#FFD54F", "#EF6C00", "#0AB4DE",
"#CFAE83"]);
chart.palette(palette);
```

If your pie chart has more slices than colors, you can specify which colors can be repeated.

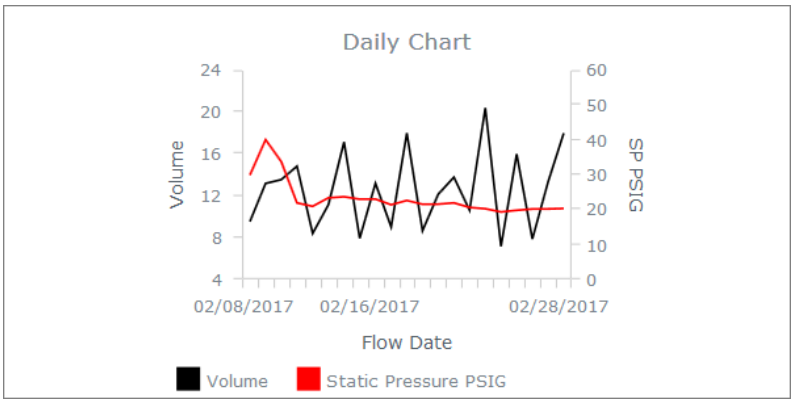


Example 10. How to create a two-series chart with two Y-Axis.

An example of a two-series chart with two Y-Axes. The idea is to choose such scales, so that it makes both series clearly visible.

```
var series2 = chart.getSeriesAt(1);

var extraYScale = anychart.scales.linear();
extraYScale.minimum(0);
extraYScale.maximum(60);
var extraTicks = extraYScale.ticks();
extraTicks.interval(10);
```

```
var extraYAxis = chart.yAxis(1);
extraYAxis.orientation("right");
extraYAxis.scale(extraYScale);
extraYAxis.title("Title right axis");

series2.yScale(extraYScale);
```



See also:

- Menu item: Modify
- Creating web charts
- Troubleshooting charts
- Creating charts
- Choose pages screen
- Customizing CSS examples
- Conditional formatting

JavaScript OnLoad

Description

The **OnPageLoad** event occurs after the page is displayed in the browser. Use this event to work with the "*edit*" controls using the JavaScript API.

Syntax

```
OnPageLoad(pageObj,pageid,proxy,inlineRow)
```

Arguments

pageObj

an object representing the current page. For more information, see RunnerPage class.

pageid

the unique numeric identifier of the page.

proxy

data transferred from PHP code using $pageObject->setProxyValue function.

inlineRow

InlineRow object available in Add/Edit page events in the inline mode.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password.

Examples

List page behavior:

- How to calculate values (totals) on the fly
- How to control Inline Add/Edit functionality from script
- How to refresh List page after Edit in popup
- How to hide 'Edit selected'/'Delete selected' buttons
- How to display all Options on Search panel

Add/Edit lists behavior:

- How to ask for confirmation before saving the record
- How to control multi-step pages

Form and controls:

- Add custom field to form
- How to show dropdown list of US states if US was selected in country list
- How to enable/disable a button

Tweaking control appearance:

- Change width of edit box with AJAX popup
- Change width of text field on Quick Search panel
- Change font size in text box
- Change font in "edit" controls
- How to convert input into upper case

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- About Dialog API
- About Grid Row JavaScript API
- About JavaScript API
- JavaScript API: AJAX helper object
- AJAX-based features
- About Tabs/Sections API
- About PDF API
- Troubleshooting JavaScript errors

3.1.4.7 Printer-friendly page

Before process

Description

The event **BeforeProcess<PageName>** is executed before any processing takes place. Use this event to redirect user to another page, send an email or *log user action* in the database.

Syntax

```
BeforeProcess<PageName>($pageObject)
```

Arguments

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Recommended sample events:

- Add custom field to form
- Hide repeating values on View page
- Email selected records

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- Audit and record locking
- Event: OnAuditLog

Before record processed

Description

The **BeforeProcessRow<PageName>** event is executed right after a database record was retrieved from the database and before the formatting is applied.

Syntax

```
BeforeProcessRow<PageName>($data,$pageObject)
```

Arguments

$data

an array of field values of the record being processed. To access a specific field value, use $data["FieldName"].

Note: You can read the $data array and also write to it changing the values before showing them on the page.

Note: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $data["PlayerId"]. Note that $data["playerid"] or $data["PlayerID"] will not work.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Return value

True - the record is shown on the page.

False - record is skipped.

Applies to pages

List, Print.

Recommended predefined actions and sample events:

- Send a simple email
- Insert a record into another table
- Check to see if a specific record exists

- Display a message on the Web page
- Redirect to another page

See also:

- Database API:Select()
- Printer-friendly/PDF view settings
- About Grid Row JavaScript API
- About RunnerPage class
- About Database API
- After record processed

After record processed

Description

The **BeforeMoveNext<PageName>** event is executed after a record was processed and the formatting applied.

Syntax

```
BeforeMoveNext<PageName>($data,$row,$record,$pageObject)
```

Arguments

$data

an array of field values of the record being processed. To access a specific field value, use $data["FieldName"].

Note: You can read the $data array and also write to it changing the values before showing them on the page.

Note: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $data["PlayerId"]. Note that $data["playerid"] or $data["PlayerID"] will not work.

$row

an array representing a row on the page.

`$record`

an array representing a table record on the page.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Print.

Recommended sample events:

- Change cell background color
- Change row background color
- Disable record editing

See also:

- Conditional formatting
- Grid Row JavaScript API: row.getFieldValue()
- ProcessValues<PageName>
- Before record processed
- About Grid Row JavaScript API
- Printer-friendly/PDF view settings

Before display

Description

The **Before Display** event is executed right before a page is displayed in the browser. Use this event to modify the value of any template variable or to define a new one.

Syntax

```
BeforeShow($xt, $pageObject)
```

Arguments

$xt

a template engine object. Use $xt->assign($name, $val) to assign a value $val to the variable $name.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Add, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Example

To display some text on the **List** page:

1. Proceed to the Editor page.

2. Switch to **HTML mode** and find the line *{END container_recordcontrols}*. Add the following code right before it:

```
<DIV>{$new_variable}</DIV>
```

3. Add the following code to the **BeforeDisplay** event.

> **Note**: Change red values to match your project.

```
$new_variable = "test value";
$xt->assign("new_variable",$new_variable);
```

Recommended sample events:

- Hide controls on Add/Edit pages, based on logged user name

See also:

- Template language
- About templates
- PHPRunner templates
- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder

Before SQL query

Description

The **BeforeQuery<PageName>** event is executed before a *SELECT SQL query* is processed. Use this event if you like to modify default SQL query, add a WHERE clause, etc.

Syntax

```
BeforeQuery<PageName>($strSQL,$strWhereClause,$strOrderBy,$pageObject)
```

Arguments

$strSQL

an SQL query to be executed.

$strWhereClause

a **WHERE clause** to apply to the SQL query.

$strOrderBy

ORDER BY query to apply to the SQL query.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Print, Export, Chart.

See also:

- Choose page screen
- SQL query screen
- About SQLQuery class

JavaScript OnLoad

Description

The **OnPageLoad** event occurs after the page is displayed in the browser. Use this event to work with the "*edit*" controls using the JavaScript API.

Syntax

```
OnPageLoad(pageObj,pageid,proxy,inlineRow)
```

Arguments

`pageObj`

an object representing the current page. For more information, see RunnerPage class.

`pageid`

the unique numeric identifier of the page.

`proxy`

data transferred from PHP code using $pageObject->setProxyValue function.

`inlineRow`

InlineRow object available in Add/Edit page events in the inline mode.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password.

Examples

List page behavior:

- How to calculate values (totals) on the fly
- How to control Inline Add/Edit functionality from script
- How to refresh List page after Edit in popup
- How to hide 'Edit selected'/'Delete selected' buttons
- How to display all Options on Search panel

Add/Edit lists behavior:

- How to ask for confirmation before saving the record
- How to control multi-step pages

Form and controls:

- Add custom field to form
- How to show dropdown list of US states if US was selected in country list
- How to enable/disable a button

Tweaking control appearance:

- Change width of edit box with AJAX popup
- Change width of text field on Quick Search panel
- Change font size in text box
- Change font in "edit" controls
- How to convert input into upper case

See also:

- Choose pages screen
- Creating charts
- Working with common pages

- Menu builder
- About Dialog API
- About Grid Row JavaScript API
- About JavaScript API
- JavaScript API: AJAX helper object
- AJAX-based features
- About Tabs/Sections API
- About PDF API
- Troubleshooting JavaScript errors

3.1.4.8 View page

Before process

Description

The event **BeforeProcess<PageName>** is executed before any processing takes place. Use this event to redirect user to another page, send an email or *log user action* in the database.

Syntax

```
BeforeProcess<PageName>($pageObject)
```

Arguments

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Recommended sample events:

- Add custom field to form

- Hide repeating values on View page
- Email selected records

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- Audit and record locking
- Event: OnAuditLog

Process record values

Description

The **ProcessValues<PageName>** event is executed before the record is displayed. Use this event to modify the displayed field values.

Syntax

```
ProcessValues<PageName>($values,$pageObject)
```

Arguments

$values

an array of values to be written to the database. To access a specific field value, use `$values["FieldName"]`.

Note: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use `$values["PlayerId"]`. Note that `$values["playerid"]` or `$values["PlayerID"]` will not work.

Note: If the field was assigned an alias in the **SQL query**, then the `$values` array will get the alias instead of the field name from the database.

E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use `$values["Inv_Salesrep"]`.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Return value

No return value.

Applies to pages

View, Add, Edit.

Example

Display an empty "*Comment*" field when the user edits a record:

```
$values["Comment"]="";
```

See also:

- Choose pages screen
- About SQLQuery class
- About SQLQuery screen
- Additional WHERE tabs
- JavaScript API: RunnerPage object
- Send email with new data
- Hide empty fields on the View page
- Store the date and time when a record is modified
- Grid Row Javascript API: row.getFieldValue()
- Grid Row JavaScript API: row.setFieldValue()

Before display

Description

The **Before Display** is executed right before a page is displayed in the browser. Use this event to modify the value of any template variable or to define a new one.

Syntax

```
BeforeShow($xt, $values, $pageObject)
```

Arguments

$xt

a template engine object. Use `$xt->assign($name, $val)` to assign a value `$val` to the variable `$name`. More info about template engine.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

$values

an array of values to be displayed on the page. To access a specific field value, use `$values["FieldName"]`.

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use `$values["PlayerId"]`. Note that `$values["playerid"]` or `$values["PlayerID"]` will not work.

> **Note**: If the field was assigned an alias in the **SQL query**, then the `$values` array will get the alias instead of the field name from the database.
>
> E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use `$values["Inv_Salesrep"]`.

Applies to pages

View, Edit.

Recommended sample events:

- Hide controls on Add/Edit pages, based on logged user name
- Hide empty fields on View page

- Before display event for all pages except View/Edit

See also:

- Template language
- About templates
- PHPRunner templates
- Choose pages screen

JavaScript OnLoad

Description

The **OnPageLoad** event occurs after the page is displayed in the browser. Use this event to work with the "*edit*" controls using the JavaScript API.

Syntax

```
OnPageLoad(pageObj,pageid,proxy,inlineRow)
```

Arguments

`pageObj`

an object representing the current page. For more information, see RunnerPage class.

`pageid`

the unique numeric identifier of the page.

`proxy`

data transferred from PHP code using $pageObject->setProxyValue function.

`inlineRow`

InlineRow object available in Add/Edit page events in the inline mode.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password.

Examples

List page behavior:

- How to calculate values (totals) on the fly
- How to control Inline Add/Edit functionality from script
- How to refresh List page after Edit in popup
- How to hide 'Edit selected'/'Delete selected' buttons
- How to display all Options on Search panel

Add/Edit lists behavior:

- How to ask for confirmation before saving the record
- How to control multi-step pages

Form and controls:

- Add custom field to form
- How to show dropdown list of US states if US was selected in country list
- How to enable/disable a button

Tweaking control appearance:

- Change width of edit box with AJAX popup
- Change width of text field on Quick Search panel
- Change font size in text box
- Change font in "edit" controls
- How to convert input into upper case

See also:

- Choose pages screen
- Creating charts
- Working with common pages

- Menu builder
- About Dialog API
- About Grid Row JavaScript API
- About JavaScript API
- JavaScript API: AJAX helper object
- AJAX-based features
- About Tabs/Sections API
- About PDF API
- Troubleshooting JavaScript errors

3.1.4.9 Search page

Before process

Description

The event **BeforeProcess<PageName>** is executed before any processing takes place. Use this event to redirect user to another page, send an email or *log user action* in the database.

Syntax

```
BeforeProcess<PageName>($pageObject)
```

Arguments

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Recommended sample events:

- Add custom field to form

- Hide repeating values on View page
- Email selected records

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- Audit and record locking
- Event: OnAuditLog

Before display

Description

The **Before Display** event is executed right before a page is displayed in the browser. Use this event to modify the value of any template variable or to define a new one.

Syntax

```
BeforeShow($xt, $pageObject)
```

Arguments

$xt

a template engine object. Use `$xt->assign($name, $val)` to assign a value `$val` to the variable `$name`.

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, Add, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Example

To display some text on the **List** page:

1. Proceed to the Editor page.

2. Switch to **HTML mode** and find the line *{END container_recordcontrols}*. Add the following code right before it:

```
<DIV>{$new_variable}</DIV>
```

3. Add the following code to the **BeforeDisplay** event.

> **Note**: Change red values to match your project.

```
$new_variable = "test value";
$xt->assign("new_variable",$new_variable);
```

Recommended sample events:

- Hide controls on Add/Edit pages, based on logged user name

See also:

- Template language
- About templates
- PHPRunner templates
- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder

JavaScript OnLoad

Description

The **OnPageLoad** event occurs after the page is displayed in the browser. Use this event to work with the "*edit*" controls using the JavaScript API.

Syntax

```
OnPageLoad(pageObj,pageid,proxy,inlineRow)
```

Arguments

`pageObj`

an object representing the current page. For more information, see RunnerPage class.

`pageid`

the unique numeric identifier of the page.

`proxy`

data transferred from PHP code using $pageObject->setProxyValue function.

`inlineRow`

InlineRow object available in Add/Edit page events in the inline mode.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password.

Examples

List page behavior:

- How to calculate values (totals) on the fly
- How to control Inline Add/Edit functionality from script
- How to refresh List page after Edit in popup
- How to hide 'Edit selected'/'Delete selected' buttons

- How to display all Options on Search panel

Add/Edit lists behavior:

- How to ask for confirmation before saving the record
- How to control multi-step pages

Form and controls:

- Add custom field to form
- How to show dropdown list of US states if US was selected in country list
- How to enable/disable a button

Tweaking control appearance:

- Change width of edit box with AJAX popup
- Change width of text field on Quick Search panel
- Change font size in text box
- Change font in "edit" controls
- How to convert input into upper case

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- About Dialog API
- About Grid Row JavaScript API
- About JavaScript API
- JavaScript API: AJAX helper object
- AJAX-based features
- About Tabs/Sections API
- About PDF API

- Troubleshooting JavaScript errors

3.1.4.10 Import page

Before import started

Description

The **BeforeImport** event is executed before the import is started.

Syntax

```
BeforeImport($pageObject, $message)
```

Arguments

$pageObject

an object representing the current page. For more information, see RunnerPage class.

$message

this message is displayed on the page if the import was canceled.

Applies to pages

Import.

Recommended predefined actions and sample events:

- Insert a record into another table
- Check to see if a specific record exists
- Display a message on the Web page

See also:

- Before record inserted
- After import finished
- Export/Import pages
- AJAX helper object: setMessage ()

Before record inserted

Description

The **BeforeInsert** event is executed before a record is inserted. Use this event to modify the record before it is inserted.

Syntax

```
BeforeInsert($rawvalues, $values, $pageObject, $message)
```

Arguments

$rawvalues

raw field values from the imported file.

$values

an array of values to be written to the database. To access a specific field value, use $values["FieldName"].

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $values["PlayerId"]. Note that $values["playerid"] or $values["PlayerID"] will not work.

> **Note**: If the field was assigned an alias in the **SQL query**, then the $values array will get the alias instead of the field name from the database.
>
> E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use $values["Inv_Salesrep"].

$pageObject

an object representing the current page. For more information, see RunnerPage class.

$message

this message will be added to the **Import Log** if the record wasn't imported

Return value

True - the record is inserted.

False - the record is not inserted.

Applies to pages

Import.

Recommended predefined actions and sample events:

- Insert a record into another table
- Check to see if a specific record exists
- Display a message on the Web page

See also:

- After import finished
- Before import started
- Export/Import pages
- AJAX helper object: setMessage ()

After import finished

Description

The **AfterImport** event is executed after the **import** is finished.

Syntax

```
AfterImport($count,$skipCount,$pageObject)
```

Arguments

$count

a number of records imported.

$skipCount

a number of lines not imported.

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

Import

Recommended predefined actions and sample events:

- Insert a record into another table
- Check to see if a specific record exists
- Display a message on the Web page

See also:

- Before record inserted
- Before import started
- Export/Import pages
- GetRowCount

3.1.4.11 Export page

Before process

Description

The event **BeforeProcess<PageName>** is executed before any processing takes place. Use this event to redirect user to another page, send an email or *log user action* in the database.

Syntax

```
BeforeProcess<PageName>($pageObject)
```

Arguments

`$pageObject`

an object representing the current page. For more information, see RunnerPage class.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password, Menu.

Recommended sample events:

- Add custom field to form
- Hide repeating values on View page
- Email selected records

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- Audit and record locking
- Event: OnAuditLog

JavaScript OnLoad

Description

The **OnPageLoad** event occurs after the page is displayed in the browser. Use this event to work with the "*edit*" controls using the JavaScript API.

Syntax

```
OnPageLoad(pageObj,pageid,proxy,inlineRow)
```

Arguments

pageObj

an object representing the current page. For more information, see RunnerPage class.

pageid

the unique numeric identifier of the page.

proxy

data transferred from PHP code using $pageObject->setProxyValue function.

inlineRow

InlineRow object available in Add/Edit page events in the inline mode.

Applies to pages

List, View, Add, Edit, Print, Search, Export, Report, Chart, Login, Register, Password reminder, Change password.

Examples

List page behavior:

- How to calculate values (totals) on the fly
- How to control Inline Add/Edit functionality from script
- How to refresh List page after Edit in popup
- How to hide 'Edit selected'/'Delete selected' buttons
- How to display all Options on Search panel

Add/Edit lists behavior:

- How to ask for confirmation before saving the record
- How to control multi-step pages

Form and controls:

- Add custom field to form
- How to show dropdown list of US states if US was selected in country list
- How to enable/disable a button

Tweaking control appearance:

- Change width of edit box with AJAX popup
- Change width of text field on Quick Search panel
- Change font size in text box
- Change font in "edit" controls
- How to convert input into upper case

See also:

- Choose pages screen
- Creating charts
- Working with common pages
- Menu builder
- About Dialog API
- About Grid Row JavaScript API
- About JavaScript API
- JavaScript API: AJAX helper object
- AJAX-based features
- About Tabs/Sections API
- About PDF API
- Troubleshooting JavaScript errors

Before record exported

Description

The **BeforeOut** event is executed before a record is exported. Use this event to modify the record before it is exported.

Syntax

```
BeforeOut($data,$values,$pageObject)
```

Arguments

$data

"In" Data. An array of values selected from the database table. To access a specific field value, use $data["FieldName"].

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $data["PlayerId"]. Note that $data["playerid"] or $data["PlayerID"] will not work.

$values

"Out" Data. An array of values to be written to the database. To access a specific field value, use $values["FieldName"].

> **Note**: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $values["PlayerId"]. Note that $values["playerid"] or $values["PlayerID"] will not work.

> **Note**: If the field was assigned an alias in the **SQL query**, then the $values array will get the alias instead of the field name from the database.
>
> E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use $values["Inv_Salesrep"].

$pageObject

an object representing the current page. For more information, see RunnerPage class.

Return value

True - the record is exported.

False - the record is not exported.

Applies to pages

Export.

Recommended predefined actions and sample events:

- Insert a record into another table

- Check to see if a specific record exists
- Display a message on the Web page

See also:

- SQL query screen
- About SQLQuery class
- About Database API
- Export/Import pages

3.1.4.12 After table initialized

Description

The **AfterTableInit** event is executed upon loading each page before any processing takes place, right after the AfterAppInit event. Use this event to override any table-specific PHPRunner variables.

Check the description area in the Event editor to find the list of available table variables.

> **Note**: It's not recommended to display anything on the web page from this event. This may break your application.

Syntax

```
AfterTableInit()
```

Applies to pages

All table specific pages like List, Print, Edit, Add, Export etc.

Example

To set the default records per page value to 10:

```
$tdata<tableName>[".pageSize"] = 10;
```

Recommended sample events

- Dynamic SQL query
- Add WHERE clause to the current SQL query
- Replace WHERE clause of the current SQL query
- Add a field name to the end of SELECT clause of the current SQL query
- Remove a field name from the SELECT clause of the current SQL query
- Replace field name in the SELECT clause of the current SQL query with new one

See also:

- Choose pages screen
- Datasource tables
- After application initialized
- About Database API

3.1.4.13 Get Table Permissions

Description

The **GetTablePermissions** event occurs after the table is initialized. Use this event to return an *access mask* for the table.

Syntax

```
GetTablePermissions($permissions)
```

Arguments

$permissions

a string containing permissions calculated for a given user and table.

Return value

`$permissions`

a string containing permissions calculated for a given user and table.

- **A** - Add;
- **D** - Delete;
- **E** - Edit;
- **S** - List/View/Search;
- **P** - Print/Export;
- **I** - Import;
- **M** - Admin option. A user can see all records in the table.

Applies to pages

All table specific pages like List, Print, Edit, List, Add, Export etc.

Example

Prohibit the editing of table data on the weekends:

```
$dw = date( "w" );
if ($dw==0 || $dw==6)
    $permissions="S";
return $permissions;
```

See also:

- Security API: setPermissions
- Security API :getPermissions
- User group permissions
- Choose Pages Screen
- Event: After table initialized
- Event: IsRecordEditable

3.1.4.14 Is Record Editable

Description

The **IsRecordEditable** event occurs after the AfterTableInit event. Use this event to implement custom edit permissions.

This event is executed once for each record on the **List** page as well as on the **Edit** page. This event is also called before the record is deleted.

Syntax

```
IsRecordEditable($values,$isEditable)
```

Arguments

$values

an array of values to be written to the database. To access a specific field value, use $values["FieldName"].

Note: **Field names** are case-sensitive. If the field name is *PlayerId*, you should use $values["PlayerId"]. Note that $values["playerid"] or $values["PlayerID"] will not work.

Note: If the field was assigned an alias in the **SQL query**, then the $values array will get the alias instead of the field name from the database.

E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use $values["Inv_Salesrep"].

$isEditable

an editing flag calculated for the current record.

Return value

True - the record is editable.

False - the record is not editable.

Applies to pages

List, Edit.

Example 1

Disable editing of data in a certain table on weekends:

```
$dw = date("w", time());
$isEditable = $dw!=6 && $dw!=0;
return $isEditable;
```

Example 2

Enable editing of only the records with odd **ID**s (1,3,5 ...):

```
if ($values["ID"] & 1)
  return false;
else
  return true;
```

See also:

- About Security API
- Get Table Permissions
- User group permissions
- Choose Pages Screen
- Event: After table initialized
- "Edit as" settings
- Audit and record locking
- SQL query screen
- About SQLQuery class

3.1.5 Page life cycle overview

Quick jump

When a generated app runs a page, the page goes through a life cycle in which it performs a series of processing steps. These include initialization, retrieving data, instantiating controls and rendering.

It is important to understand the page life cycle so that you can write code at the appropriate life-cycle stage for the effect you intend.

Global Events

AfterAppInit	Occurs upon loading each page before any processing takes place. Use this event to override any global PHPRunner variables. Check the description area in the Event editor to find the list of available global variables.
AfterTableInit	Occurs upon loading each page before any processing takes place, right after AfterAppInit. Use this event to override any table-specific PHPRunner variables. Check the description area in the Event editor to find the list of available global variables.
ModifyMenuItem	This event is executed for each Menu item before a page is displayed in the browser. Use this event to modify or hide menu items.

List/Print pages

BeforeProcessList BeforeProcessPrint	Occurs when the page processing starts and the database connection is established. Use this event for the following: • change the database connection to point to another database; • read request data and populate session variables; • redirect to another page.
BeforeDelete	This event is executed once for each record to be deleted. Use this event for the following: • prevent a record from being deleted; • save the deleted record in another table.
AfterDelete	Occurs once for each record after it was deleted.
AfterMassDelete	Occurs after a bulk delete operation.
BeforeProcessRowList BeforeProcessRowPrint	Occurs after a database record is retrieved from the database before formatting is applied. Use this event for the following: • modify the value of any column; • prevent certain records from being displayed on the page; • calculate/display your totals and subtotals.
BeforeMoveNextList BeforeMoveNextPrint	Occurs after a database record is retrieved from the database and formatting is applied. Use this event for the following: • display an empty row or a header between group of records; • apply additional formatting to any column.
BeforeShowList BeforeShowPrint	Occurs after the page is processed and ready to be displayed in the browser. Use this event for the following: • define a new template variable or change the value of an existing one; • display a different template.
OnPageLoad	Occurs after the page is displayed in the browser. Use this event to work with the "edit" controls using the JavaScript API.

Edit/Add pages

BeforeProcessEdit BeforeProcessAdd	Occurs when the page processing starts and the database connection is established. Use this event for the following: • change the database connection to point to another database; • read request data and populate session variables; • redirect to another page.
BeforeEdit BeforeAdd	Occurs before new data is written to the database. Works in all add/edit modes: **Inline Add**/**Edit**, **Regular Add**/**Edit** and an **Add**/**Edit** page in a popup. Use these events for the following: • prevent a data record from being added or edited; • send an email; • save old record in another table; • add a record to the log table.
AfterEdit AfterAdd	Occurs after data was written to the database. Works in all add/edit modes: **Inline Add**/**Edit**, **Regular Add**/**Edit** and an **Add**/**Edit** page in a popup. Use this event for the following: • send an email; • add a record to the log table.
ProcessValuesEdit	Occurs before the record is displayed (before the BeforeShowEdit event).
BeforeShowEdit BeforeShowAdd	Occurs after the page is processed and ready to be displayed in the browser. Use this event for the following: • define a new template variable or change the value of an existing one; • display a different template.
OnPageLoad	Occurs after the page is displayed in the browser. Use this event to work with the "edit" controls using the JavaScript API.

Login/Registration pages

BeforeProcessLogin BeforeProcessRegister	Occurs when the page processing starts and the database connection is established. Use this event for the following: • change the database connection to point to another database; • read request data and populate session variables; • redirect to another page.
BeforeLogin BeforeRegister	Occurs before the user logs in or the new user data is written to the database. Use this event for the following: • prevent the user from being logged in or registered; • send an email; • add a record to the log table.
AfterSuccessfulLogin AfterSuccessfulRegistration	Occurs after the user is logged in or registered successfully. Use this event for the following: • redirect user to another page; • send an email; • add a record to the log table.
AfterUnsuccessfulLogin AfterUnsuccessfulRegistration	Occurs if the user was not logged in or was not registered.
BeforeShowLogin BeforeShowRegister	Occurs after the page is processed and ready to be displayed in the browser. Use this event for the following: • define a new template variable or change the value of an existing one; • display a different template.
OnPageLoad	Occurs after the page is displayed in the browser. Use this event to work with the "edit" controls using the JavaScript API.

View/Search/Report/Chart pages

BeforeProcess<PageName>	Occurs when the page processing starts and the database connection is established. Use this event for the following: • change the database connection to point to another database; • read request data and populate session variables; • redirect to another page.
ProcessValuesView	Occurs before the record is displayed (before the BeforeShowView event).
BeforeShow<PageName>	Occurs after the page is processed and ready to be displayed in the browser. Use this event for the following: • define a new template variable or change the value of an existing one; • display a different template.
OnPageLoad	Occurs after the page is displayed in the browser. Use this event to work with the "edit" controls using the JavaScript API.

Import page

BeforeImport	This event is executed before the import is started.
BeforeInsert	Occurs before a record is inserted. Use this event to modify the record before it is inserted.
AfterImport	This event is executed after the import is finished.

See also:

- Menu builder
- Session keys
- PHPRunner session variables
- Template language
- About templates
- PHPRunner templates

- Choose pages screen
- Event editor
- Table Events
- Global Events
- Common event parameters
- Tri-part events
- Field events

3.1.6 Common event parameters

You can use the following common parameters in your event code:

pageObject

$pageObject

an RunnerPage class object that represents a current page. For more information, see RunnerPage class.

Example

Before record updated:

```
// Get the current record and display the Make and Model fields' values
$data = $pageObject->getCurrentRecord();
echo $data["Make"] ." ".$data["Model"];
```

strTableName and values

$strTableName

the name of the currently selected table.

$values

an array with the field values from an **Add/Edit** form.

Example

Before record updated:

```
echo $values["Field1"];
```

Note: If the field was assigned an alias in the SQL query, then the **$values** array gets the alias instead of the field name from the database.

E.g., if you have an **SQL query** *SELECT salesrep_id AS Inv_Salesrep ...*, you should use $values["Inv_Salesrep"].

keys

$keys

an array with the key columns.

Example

After record added:

```
echo $keys["ID"];
```

Other useful functions and parameters

$templatefile

the name of the template file being displayed.

$xt

a template engine object. Use $xt->assign($name, $val) to assign a value $val to the variable $name.

Example:

Before display event

```
$message = "This message";
$xt->assign("message",$message);
```

See also:

- Page life cycle overview
- Event editor
- Table Events
- Global Events
- SQL query screen
- About SQLQuery class

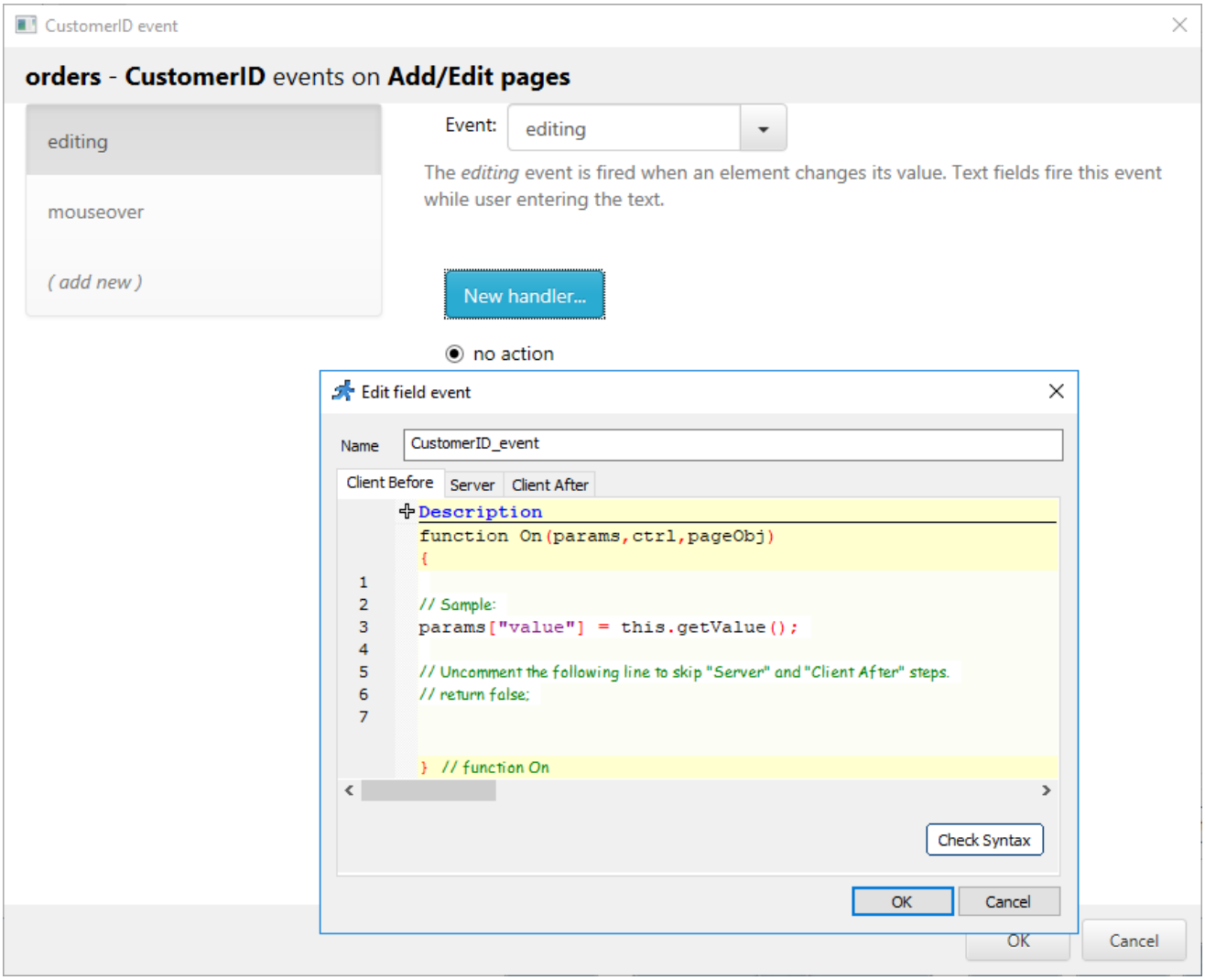
3.1.7 Field events

Quick jump

What are the Field events?

The **Field events** option allows performing an action when the cursor enters or leaves an edit field, or when the mouse is over a field. Perform any validation, make other fields hidden or required, etc. This option is designed to work on **Add**, **Edit**, and **Register** pages.

For example, the *editing* event occurs when an element changes its value. Text fields launch this event when the user enters the text.



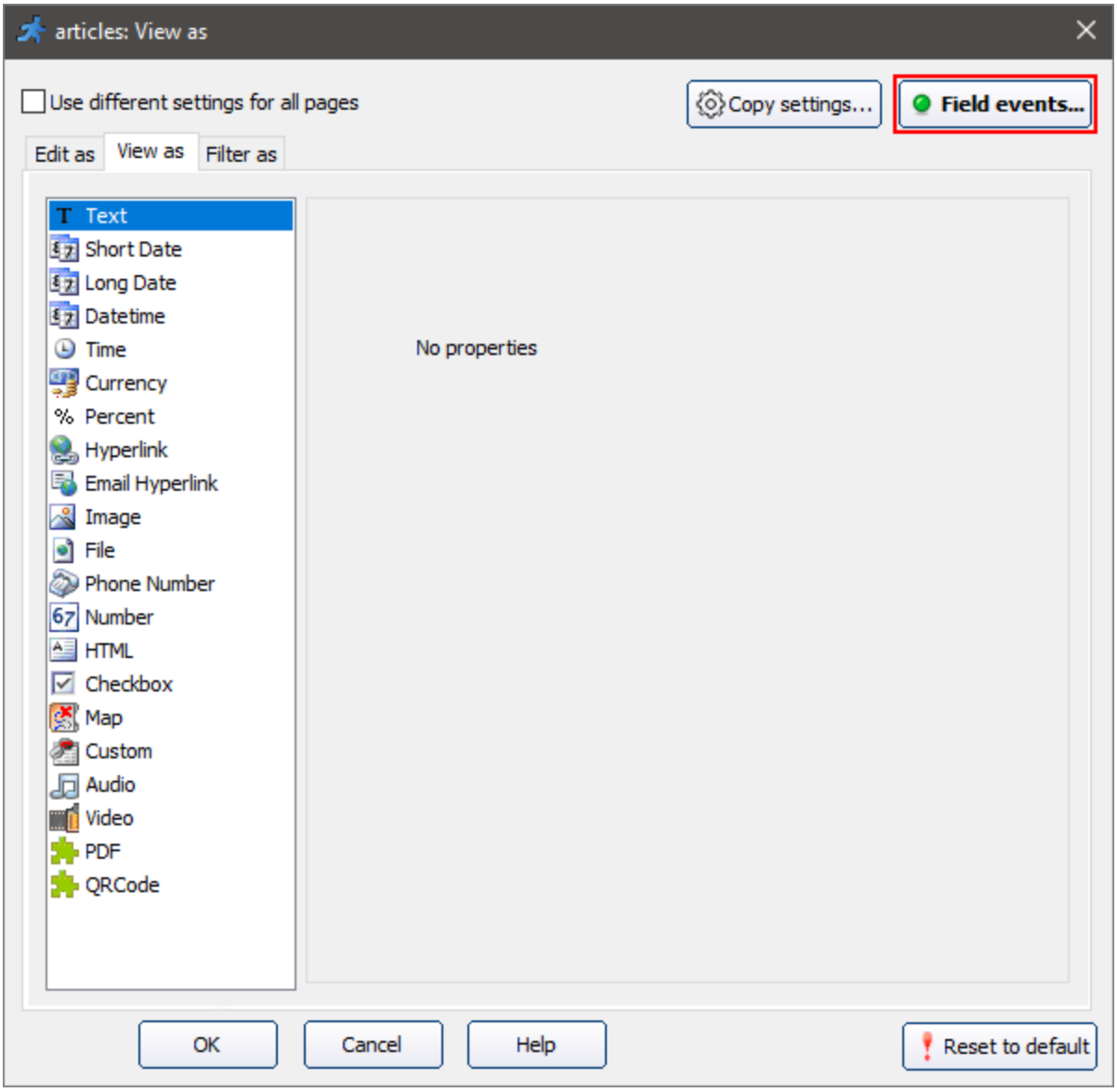
Note: Field events utilize the Tri-part event system, that consists of three parts: Client Before, Server, Client After.

The Client Before part runs JavaScript code in the browser, then passes parameters to the **Server** part that runs PHP code, and then back to the browser to run the JavaScript code of the Client After part.

To learn more about these types of events, see Tri-part events.

How to access Field events?

Open the View as / Edit as menu and click the **Field events** button in the top right corner of the window to start working with the **Field events**.



Once a field event was added, you can find it on the Events screen.

Using Field events

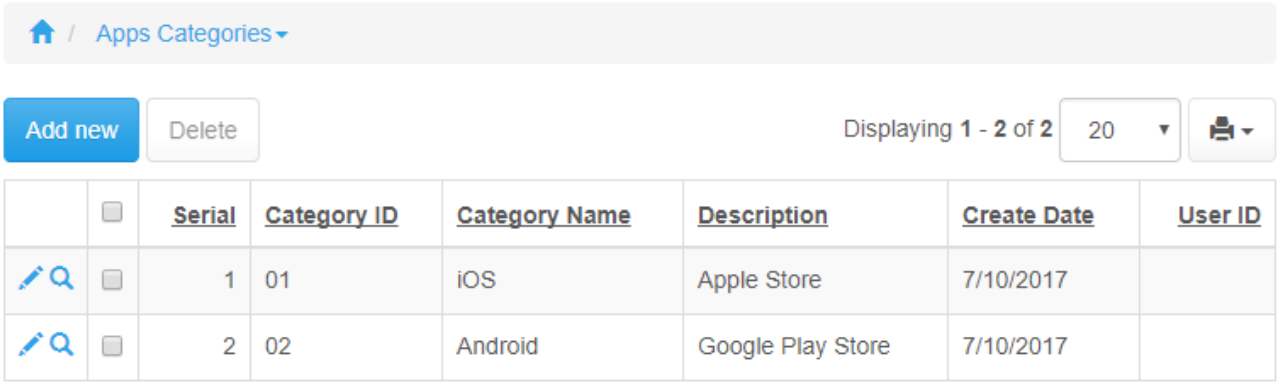
Example 1

This example shows how to use the **Field events** to make it easier to enter categories/subcategories into the table.

The user needs to enter only the numeric value (ID) into the text box to get other values automatically.

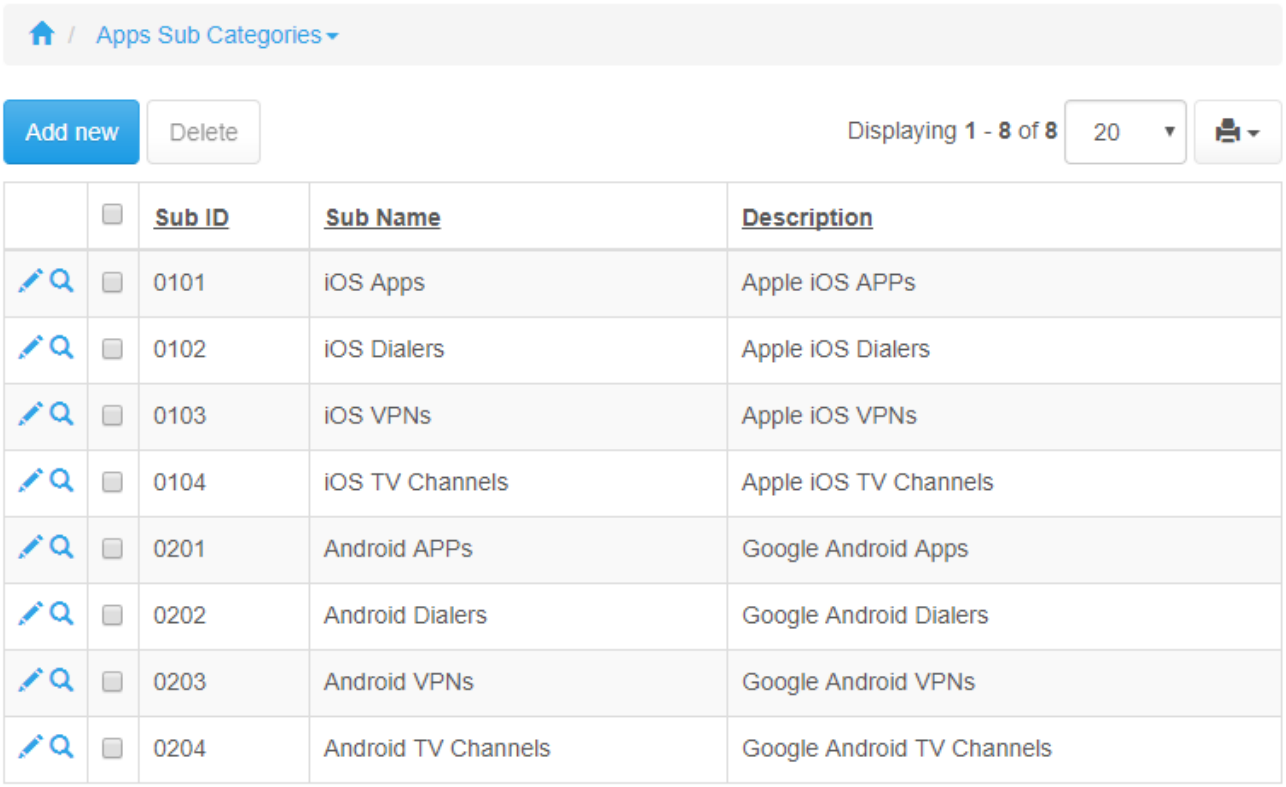
Let's take a detailed look.

We have two tables: *categories* and *sub_categories*. *CategoryID* is a two-digit category code.



	Serial	Category ID	Category Name	Description	Create Date	User ID
	1	01	iOS	Apple Store	7/10/2017	
	2	02	Android	Google Play Store	7/10/2017	

In the *sub_categories* table, *SubID* consists of a two-digit category code and a two-digit subcategory code.



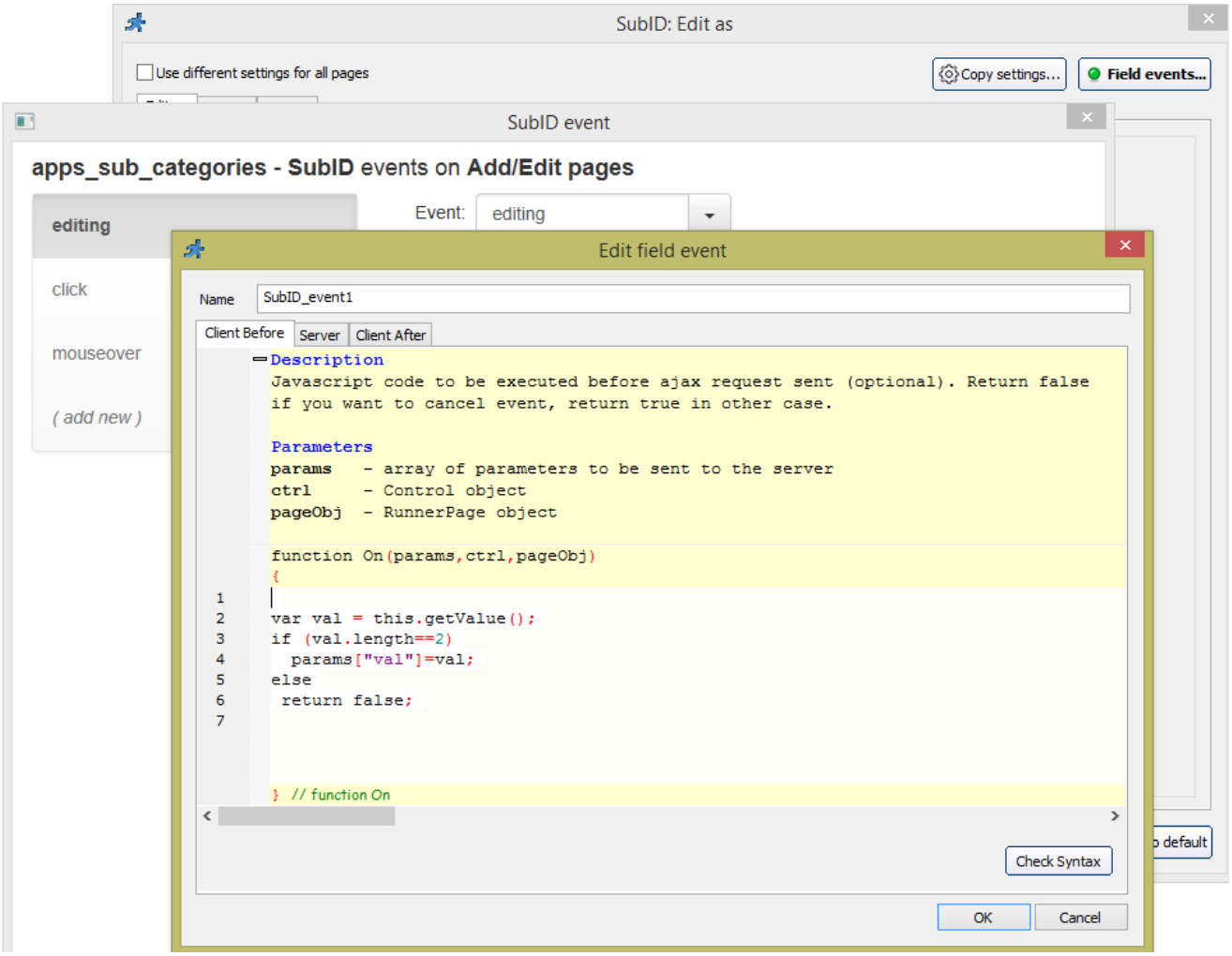
	Sub ID	Sub Name	Description
	0101	iOS Apps	Apple iOS APPs
	0102	iOS Dialers	Apple iOS Dialers
	0103	iOS VPNs	Apple iOS VPNs
	0104	iOS TV Channels	Apple iOS TV Channels
	0201	Android APPs	Google Android Apps
	0202	Android Dialers	Google Android Dialers
	0203	Android VPNs	Google Android VPNs
	0204	Android TV Channels	Google Android TV Channels

We need to simplify adding a large number of subcategories.

When the user types in the first two digits (category), the category name appears below the text box. The next subcategory number is calculated automatically.

Code

First, we need to create a **Field Event** for the *SubID* field. We are going to use the editing event here, which is called every time the content of the field is changed.



Client Before:

```
var val = this.getValue();
if (val.length==2)
  params["val"]=val;
else
 return false;
```

In the **Client Before** code, we check the length of the entered code and send it to the server side only when the code is two digits long.

Server:

```
$result["catName"] = DBLookup("select CategoryName from categories where
CategoryID='".$params["val"]."'");
$sub = DBLookup("select max(substring(SubID,3,2))+1 from sub_categories where
left(SubID,2)='".$params["val"]."'");
$result["newCat"] = $params["val"].str_pad($sub,2,"0",STR_PAD_LEFT);
```

On the **Server** side, we pull the *CategoryName* from the *categories* table and calculate the next subcategory ID.

We pass both the category name and new subcategory code back to the **Client After** event.

Client After:

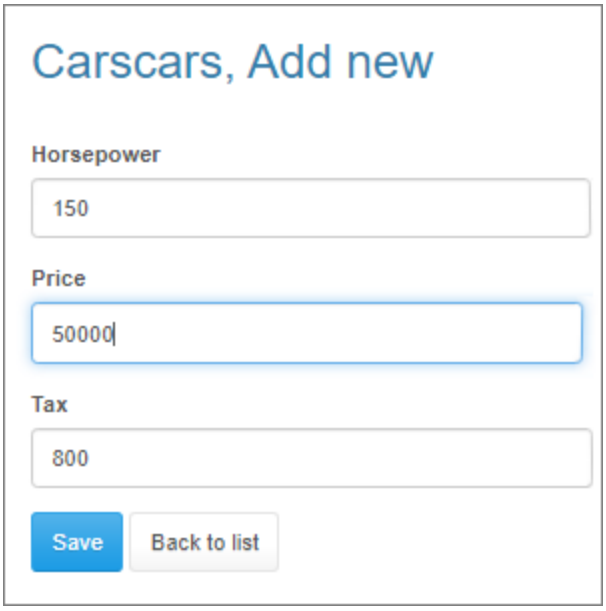
```
$("#sub_tip").remove();
$("input[id=value_SubID_1]").after("<div id='sub_tip' style='margin-top: 10px; color:
blue;'>"+result["catName"]+"</div>");
ctrl.setValue(result["newCat"]);
```

In this event, we replace the previous category name with a new one. We also make the text blue. Then we set the value of the *SubID* with a new subcategory code we received from the **Server** event.

Example 2

This example shows how to calculate a field value that depends on other field values on the fly.

For instance, in the Cars template, we need to get the *Tax value* that depends on the *Price* and *Horsepower*.



You need to create *editing* **Field events** for the *Horsepower* and the *Price* fields each. Insert the following code into the events.

> **Note**: insert the code into the **Client Before** event. Leave the **Server** and **Client After** events empty.

The code for the *Horsepower*:

```
var ctrlTax = Runner.getControl(pageid, 'Tax');
var ctrlPrice = Runner.getControl(pageid, 'Price');
ctrlTax.setValue(0.01*ctrlPrice.getValue()+2*this.getValue());

return false;
```

The code for the *Price*:

```
var ctrlTax = Runner.getControl(pageid, 'Tax');
var ctrlHorsepower = Runner.getControl(pageid, 'Horsepower');
ctrlTax.setValue(0.01*this.getValue())+2*(+ctrlHorsepower.getValue());

return false;
```

> **Note**: in this example, we get the *Tax* value using the formula: *Tax* = *Price**0.01 + *Horsepower**2.

Now, each time the *Horsepower* and *Price* values are changed, the *Tax* value is computed automatically without reloading the page.

See also:

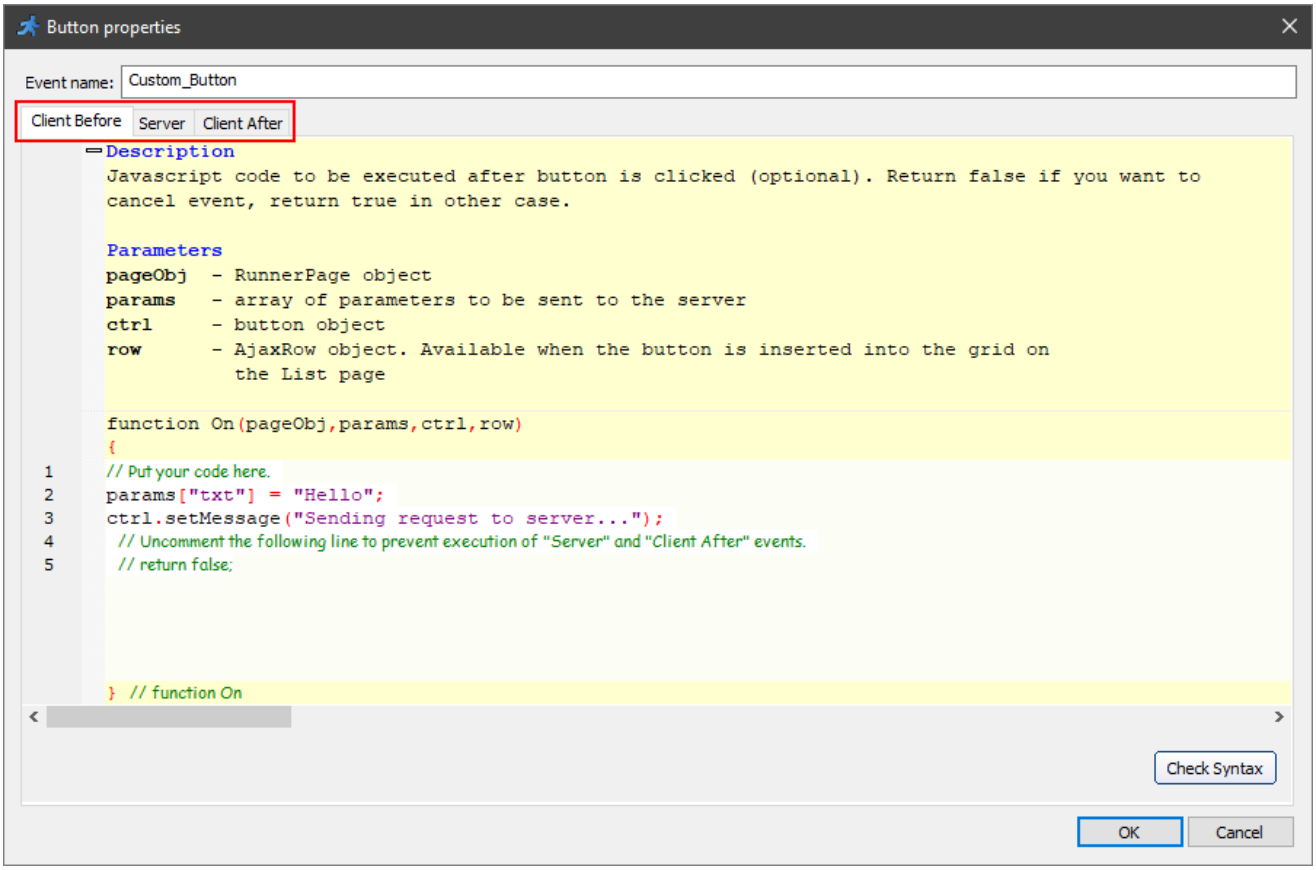
- How to access fields in the field events
- Passing data between events
- Tri-part events
- List page settings / Click actions

3.1.8 Tri-part events

Quick jump

Tri-part events are a system of interconnected events that provides a convenient way to design the interaction between the browser and web server.

Here is how the **Tri-part event** interface looks like when adding a Custom button:



Why three parts?

Any web application consists of two parts - **server** and **client**. The **client** part runs in the browser on the computer of the web site visitor. This part takes care of all user interface interactions, such as buttons, input controls, blocks of text, and images. The **client-side** code is written in JavaScript in most cases.

The **server** part runs the code on the web server itself and has direct access to the database, can send emails, read and write files to the disk. The PHPRunner applications, use PHP as the server code language.

Most real-life tasks require a joint action of both **client** and **server** parts. For example, the user clicks a button to change something in the database, or send an email.

Tri-part events provide a convenient way to create such integrated code snippets.

They consist of three parts running one after another:

- **Client Before** - this JavaScript code runs immediately after the user's action, for example, clicking the button.
- **Server** - this part runs on the server after the **Client Before** part has finished. You can only use the server-side code here (PHP).
- **Client After** - this part goes back to the web browser, runs another JavaScript code after the **Server** part has finished.

Passing data between events

It's essential that a functional **Tri-part event** be able to pass data between its parts. For example, the **Client Before** part receives input from the user, passes it to the **Server** event. The latter runs some database queries using the data and passes the results to **Client after**, which shows the results to the user.

Two objects serve as links between the three parts of the event:

- **params** object passes data from the **Client Before** to the **Server** event.
- **result** object passes data from the **Server** to the **Client After** event.

Mind the syntax difference between the **Client** and the **Server** events. The **Client** code is JavaScript, and the **Server** code is PHP.

The key names, users, and variables are up to you. You can choose any names here.

You can also pass arrays and objects this way:

ClientBefore:

```
params["data"] = {
    firstname: 'Luke',
    lastname: 'Skywalker'
};
```

Server:

```
do_something( $params["data"]["firstname"] );
```

Where in PHPRunner can you use these events?

Three features utilize **Tri-part events:**

- Custom buttons
- Field events
- Grid click actions

All these events work in the same way. The only difference between them is how users initiate the event - by clicking the button, interacting with an input control, or by clicking somewhere in the data grid.

Control flow

If you don't need the **Server** part, return false in the **Client Before** code:

```
//your event code here

return false;

// all done, skip the Server and Client after parts.
```

Asynchronous tasks

Some tasks in JavaScript code are asynchronous: they are not completed immediately, but at some indefinite moment in the future. Sometimes, it's necessary to wait for their conclusion before running the **Server** part of the event. In this case, return false in the **Client Before** event and call the `submit()` function on the task conclusion.

Example:

```
// run the Server event after a 5-seconds pause:
setTimeout( function() {
    submit();
}, 5000 );
return false;
```

Event parameters

PHPRunner passes the following parameters to the **ClientBefore** and **ClientAfter** events:

`pageObj`

a RunnerPage object representing the current page.

`ajax`

an Ajax helper object. It provides miscellaneous functions like showing dialogs or generating PDF files.

`row`

a GridRow object that represents a row in the grid. Buttons receive this parameter when positioned in the data grid on the **List** page. Field events receive it in the **Inline Add**/**Edit** mode.

`ctrl`

in the Field events, this parameter represents the input control itself. It is a RunnerControl object.

See also:

- Insert custom button
- Field events
- About Grid Row JavaScript API
- About Dialog API
- About PDF API
- About Control object
- About RunnerPage object

3.2 Programming topics

3.2.1 Buttons

3.2.1.1 Button object

Button object

The $button object is used in the Server event of the buttons inserted into the grid.

Methods

Method	Description
getCurrentRecord()	Gets the current record.
getNextSelectedRecord()	Returns associative arrays with the values of the records selected on the **List** page.

See also:

- Tri-part events
- Page Designer: Insert custom button
- Buttons: rowData object

Methods

getCurrentRecord

Gets the current record.

Syntax

```
getCurrentRecord()
```

Arguments

No arguments.

Return value

If the button is inserted in the grid on the **List** or **Edit**/**View** page, it returns an associative array (field name => value). Otherwise it returns false.

Example

Let's assume we have a button on the Classified Ad **View** page, which allows writing a letter to the person who posted the announcement. The page from which the message is sent is reply_add.php and we need to pass the email parameter to this page.

Server:

```
$record = $button->getCurrentRecord();
$result["email"]=$record["email"];
```

ClientAfter:

```
location.href='reply_add.phpasp?email='+result["email"];
```

See also:

- Buttons: getNextSelectedRecord
- About templates
- Tri-part events
- Page Designer: Insert custom button
- Buttons: Button object

getNextSelectedRecord

Returns associative arrays with values of the records selected on the **List** page (marked with checkboxes). If nothing is selected or all selected records are processed, returns false. It is available for any button on the **List** page.

Syntax

```
getNextSelectedRecord()
```

Arguments

No arguments.

Return value

Returns associative arrays with values of the records selected on the **List** page (marked with checkboxes). If nothing is selected or all selected records are processed, returns false.

Example

Send emails to the selected customers:

```
while($record = $button->getNextSelectedRecord())
{
   $message = "Dear ".$record["FirstName"];
   runner_mail(array('to' => $record["email"], 'subject' => "Greetings",
 'body' => $message));
}
```

See also:

- runner_mail function
- Buttons: getCurrentRecord()
- Tri-part events
- Page Designer: Insert custom button
- Buttons: Button object

3.2.1.2 rowData object

rowData object

After you inserted the button into a datagrid, you can use the **rowData object** to manipulate records.

rowData object is available in the Client Before and Client After events of the inserted button.

rowData.fields

rowData.fields is an object, where *index* is the name of the field, and *value* - a **jQuery container object** that contains the field.

Example 1

Make a red box around the value of the *ID* field:

```
rowData.fields['ID'].css('border', '1px solid red');
```

Example 2

Get the HTML code of the ID field

```
alert(rowData.fields['ID'].html());
```

Example3

Add a prefix to the value of the *ID* field:

```
rowData.fields['ID'].html("some prefix" + rowData.fields['ID'].html());
```

rowData.keys

rowData.keys is an array of values of key fields in the record.

Example

```
for (var i = 0; i < rowData.keys.length; i ++) {
   // Do something
}
```

rowData.id

rowData.id is the ID of the record. If you are running an **Inline Edit**, you can use the Runner.getControl function.

Example

```
for(var fName in rowData.fields){
   // get the control object, will work only if you open "inline"
   var ctrl = Runner.getControl(rowData.id, fName);
   if ( !ctrl ) {
      continue;
   }

   // get the control value
   var val = ctrl.getValue();
   if( val == "audi" )
      rowData.fields[fName].hide();
}
```

See also:

- JavaScript API: Date Control API > getValue
- Page Designer: Insert custom button
- Key columns
- Tri-part events
- Buttons: Button object

3.2.2 Data Access Layer (DAL)

3.2.2.1 About Data Access Layer

> **Deprecated**
>
> Data Access Layer (DAL) is deprecated. While the existing code keeps working, we recommend switching to Database API.

Each table in DAL is presented as a PHP class, all fields are PHP variables declared in this class.

Variables/Arrays

Variable/Array	Description

`Table("TableName")`	Provides access to the `TableName`. Example: ``` global $dal; $tblCars = $dal->Table("Cars"); ```
`Value["FieldName"]`	Provides access to the field values to be updated or added. Example: ``` global $dal; $tblCars = $dal->Table("Cars"); $tblCars->Value["Make"]="Volvo"; ```
`Param["FieldName"]`	Provides access to the field values. This function is used in the WHERE clause of the update query. This allows you to avoid the confusion when the same field appears in the field list and in WHERE clause. Example: ``` global $dal; $tblUsers = $dal->Table("UsersTable"); $tblUsers->Param["FirstName"]="Bob"; $tblUsers->Value["FirstName"]="Jim"; $tblUsers->Update(); ``` The corresponding SQL query: ``` Update UsersTable set 'FirstName'='Jim' where 'FirstName'='Bob' ```

Methods

Method	Description
Add()	Inserts a new record into the database.
CustomQuery()	Runs a custom SQL query.
Delete()	Deletes one or more records from the database.
DBLookup()	Executes an SQL query passed as a parameter and returns the first value of the first entry or null if nothing is found.
FetchByID()	Selects one or more records matching the condition.
Query()	Selects records from database and sorts the data by *orderby* field/fields and returns the recordset.
QueryAll()	Selects all records.
TableName()	Returns the table name. This function is used for complex queries with calculated fields or joined tables.
Update()	Updates one or more records in the database.
UsersTableName()	Returns a properly formatted login table name.
whereAdd()	Adds a new AND condition to the existing WHERE clause.

Examples

- Before deleting a record check if related records exist
- Dynamic SQL Query
- Redirect to user info edit page
- Show data from master table on details view/edit/add page
- Show list of customer orders
- Update multiple records on the List page
- Update multiple tables

See also:

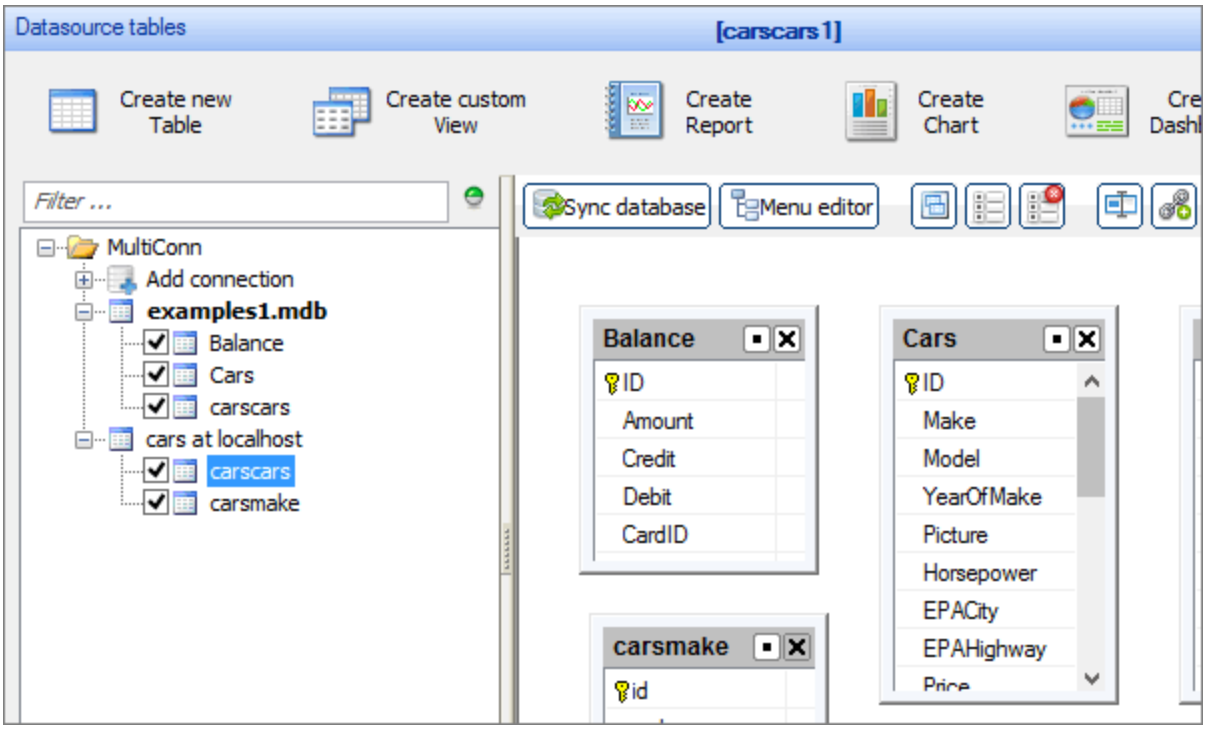
- Database API
- Using DAL functions in projects with multiple database connections

- Connecting to the database
- Datasource tables screen

3.2.2.2 Using DAL functions in projects with multiple database connections

> **Deprecated**
>
> Data Access Layer (DAL) is deprecated. While the existing code keeps working, we recommend switching to Database API.

In the Enterprise Edition of PHPRunner you can use multiple database connections in a single project. This article explains how you can access data from multiple databases in your events.



Method 1: using DAL functions

There are three options to refer to a table:

1. Finds the first matching table by its name in all connections, starting with the primary connection.

```
global $dal;
$dal->Table("table")
```

2. Finds the tables by the table name and schema name.

```
global $dal;
$dal->Table("table","schema")
```

3. Finds the table by the table name, schema name and connection name. The schema name can be left empty, and the last parameter is the connection name as it appears on Datasource tables screen.

```
global $dal;
$dal->Table("table","schema","connection")
```

A complete code example:

```
global $dal;
$table = $dal->Table("cars", "", "cars at localhost");
$rs = $table->QueryAll();
while ($data = db_fetch_array($rs)) {
    echo $data["Make"].
    ", ".$data["Model"].
    "<br>";
}
```

Method 2: using free form SQL Queries

1. Here is how to update all cars where *make* is *Audi* and the *YearOfMake* equals *2002*:

```
global $cman;
$cman->byName("cars at localhost")->exec("update carscars set yearofmake=2002 where
make='Audi'");
```

2. Example of a query returning data:

```
global $cman;
$connection = $cman->byName("cars at localhost");
$rs = $connection->query("select count(*) as c  from cars where make='Audi' ");
$data = $rs->fetchAssoc();
echo "Number of Audi cars listed: ".$data["c"];
```

See also:

- QueryResult object: fetchAssoc()
- Database API
- Using DAL functions in projects with multiple database connections
- Connecting to the database
- Datasource tables screen
- About Data Access Layer

3.2.2.3 Methods

Add

Deprecated

This function is deprecated, and we recommend using the DB:Insert() function from Database API.

Inserts a new record into the database.

Syntax

```
Add()
```

Arguments

No arguments.

Return value

No return value.

Example

Insert new record into the database:

```
global $dal;
$tblEvents = $dal->Table("EventsTable");
$tblEvents->Value["event"]="First event";
$tblEvents->Value["public"]="yes";
$tblEvents->Add();
```

The corresponding SQL query:

```
Insert into EventsTable (event,public) values ('First event','yes')
```

See also:

- DB API: Insert
- DAL Method: Delete
- Database API
- About Data Access Layer

CustomQuery

Deprecated

This function is deprecated, and we recommend using the DB:Query() function from Database API.

Runs a custom SQL query.

Syntax

```
CustomQuery($sql)
```

Arguments

$sql

a SELECT clause. Example: "select * from UsersTable where ID=32".

Return value

Returns the recordset.

Example 1

A query that doesn't return data:

```
$sql = "update Users set active=0 where id=32";
CustomQuery($sql);
```

Example 2

A query that returns data:

```
$sql = "select count(*) as c from orders group by customerid";
$rs = CustomQuery($sql);
$data = db_fetch_array($rs);
echo "Number of customers: " . $data["c"];
```

See also:

- DAL Method: Query
- DAL Method: QueryAll
- DB API: Query
- Database API
- Example: Update multiple records on the List page

- About Data Access Layer

Delete

> **Deprecated**
>
> This function is deprecated, and we recommend using the DB:Delete() function from Database API.

Deletes one or more records from the database.

Syntax

```
Delete()
```

Arguments

No arguments.

Return value

No return value.

Example 1

Delete all records where the ID is '32':

```
global $dal;
$tblUsers = $dal->Table("UsersTable");
$tblUsers->Param["ID"]=32;
$tblUsers->Delete();
```

The corresponding SQL query:

```
Delete from UsersTable where ID=32
```

Example 2

Delete all records where *FirstName* is 'Bob' and *Email* is 'test@test.com':

```
global $dal;
$tblUsers = $dal->Table("UsersTable");
$tblUsers->Param["FirstName"]="Bob";
$tblUsers->Param["Email"]="test@test.com";
$tblUsers->Delete();
```

The corresponding SQL query:

```
Delete from UsersTable where FirstName='Bob' and Email='test@test.com'
```

See also:

- DAL Method: Add
- DB API: Delete
- Database API
- About Data Access Layer

DBLookup

> **Deprecated**
>
> Data Access Layer (DAL) methods are deprecated. While the existing code keeps working, we recommend switching to Database API.

Executes an SQL query passed as a parameter. Returns the first value of the first entry or null if nothing is found.

Syntax

```
DBLookup($sql)
```

Arguments

$sql

a SELECT clause. Example: "select * from UsersTable where ID=32".

Return value

Returns the first value of the first entry or null if nothing is found.

Example

Returns the *zip* value where *userid* is "25":

```
$zip = DBLookup("select zip from users where userid=25");
```

See also:

- DAL Method: FetchByID
- Database API
- DB API: Exec
- About Data Access Layer

FetchByID

> **Deprecated**
>
> This function is deprecated, and we recommend using the DB:Select() function from Database API.

Selects one or more records from the database that matches the condition.

Syntax

```
FetchByID()
```

Arguments

No arguments.

Return value

Returns the recordset on success or FALSE on error.

Example

Select all records where ID is '32'. To fetch a returned row as an associative array, use the **db_fetch_array** function.

```
global $dal;
$tblUsers = $dal->Table("UsersTable");
$tblUsers->Param["ID"]=32;
$rs = $tblUsers->FetchByID();
while( $data = db_fetch_array($rs) )
{
   echo $data["UsersName"];
}
```

Note: the **db_fetch_array** function is deprecated, and we recommend using the fetchAssoc function instead.

The corresponding SQL query:

```
select * from UsersTable where ID=32
```

See also:

- QueryResult object: fetchAssoc
- DAL method: DBLookup
- DB API: Select
- DB API: LastId
- Database API
- About Data Access Layer

Query

> **Deprecated**
>
> This function is deprecated and we recommend using the DB:Query() function from Database API.

Selects records from the database, sorts the data by the *orderby* field(s), and returns the recordset.

Syntax

```
Query(where, orderby)
```

Arguments

where

a WHERE clause. Example: "ID=19".

orderby

one or more fields used to sort the recordset by.

Return value

Returns the recordset on success or FALSE on error.

Example 1

Select all records where name contains 'Jim'. To fetch a returned row as an associative array use db_fetch_array function.

```
global $dal;
$tblUsers = $dal->Table("UsersTable");
$rs = $tblUsers->Query("Name like '%Jim%'","Email DESC");
while( $data = db_fetch_array($rs) )
{
```

```
    echo $data["fieldName"]."<br>";
}
```

Note: the **db_fetch_array** function is deprecated, and we recommend using the fetchAssoc function instead.

The corresponding SQL query:

```
select * from UsersTable where Name like '%Jim%' order by Email DESC
```

Example 2

Select and print all orders for John Sample. To fetch a returned row as an associative array, use the **db_fetch_array** function.

```
global $dal;
$tblOrders = $dal->Table("OrdersTable");
$rs = $tblOrders->Query("Customer='John Sample'","OrderID DESC");
while ($data = db_fetch_array($rs))
{
    echo "Order ".$data["OrderID"]." was placed ".$data["OrderDate"]." by
".$data["Customer"]."<br>";
}
```

Note: the **db_fetch_array** function is deprecated, and we recommend using the fetchAssoc function instead.

The corresponding SQL query:

```
select * from OrdersTable where Customer like 'John Sample'
order by OrderID DESC
```

Examples:

- Example: Show data from master table on details view/edit/add page
- Example: Before deleting a record check if related records exist

- Example: Redirect to user info edit page
- Example: Show list of customer orders

See also:

- QueryResult object: fetchAssoc
- DAL Method: QueryAll
- DAL Method: CustomQuery
- DB API: Query
- Database API
- About Data Access Layer

QueryAll

> **Deprecated**
>
> This function is deprecated and we recommend using the DB:Query() function from Database API.

Selects all records from the table.

Syntax

```
QueryAll()
```

Arguments

No arguments.

Return value

Returns the recordset on success or FALSE on error.

Example

Send mass email to all users:

```
global $dal;
//select emails from Users table
$tblUsers = $dal->Table("UsersTableName");
$rs = $tblUsers->QueryAll();
while ($data = db_fetch_array($rs))
{
  $email.=$data["EmailAddress"].", ";
  $from="admin@test.com";
  $msg="Check what's hot this season";
  $subject="Monthly newsletter";
  $ret=runner_mail(array('to' => $email, 'subject' => $subject, 'body' => $msg,
'from'=>$from));
  if(!$ret["mailed"])
    echo $ret["message"]."<br>";
}
```

See also:

- runner_mail function
- DAL Method: Query
- DAL Method: CustomQuery
- DB API: Query
- Database API
- About Data Access Layer

TableName

> **Deprecated**
>
> This function is deprecated, and we recommend using the DB API: PrepareSQL function to prepare complex queries.

Returns the table name. **TableName()** is used for complex queries with calculated fields or joined tables.

Syntax

```
TableName()
```

Arguments

No arguments.

Return value

Returns the table name.

Example

Using a complex query:

```
global $dal;
$tblProducts = $dal->Table("Products");
$sql = "select sum(UnitsInStock) as total, concat(Category,' ',
  ProductName) as FullProductName from ";
$sql .= $tblProducts->TableName() . " group by country";
```

See also:

- DAL Method: UsersTableName
- DB API: PrepareSQL
- Database API
- About Data Access Layer

Update

Deprecated

This function is deprecated, and we recommend using the DB:Exec() function from Database API.

Updates one or more records in the database.

Syntax

```
Update()
```

Arguments

No arguments.

Return value

No return value.

Example

Update the record where ID is 32, making the *FirstName* field value 'Jim' and *LastName* field value 'Morrison':

```
global $dal;
$tblUsers = $dal->Table("UsersTable");
$tblUsers->Param["ID"]=32;
$tblUsers->Value["FirstName"]="Jim";
$tblUsers->Value["LastName"]="Morrison";
$tblUsers->Update();
```

The corresponding SQL query:

```
Update UsersTable set FirstName='Jim', LastName='Morrison' where ID=32
```

See also:

- DB API: Exec
- Database API
- Example: Update multiple tables
- About Data Access Layer

UsersTableName

> **Deprecated**
>
> This function is deprecated, and we recommend using Security API.

Returns a properly formatted login table name.

Syntax

```
UsersTableName()
```

Arguments

No arguments.

Return value

Returns properly formatted login table name.

Example

Print all email addresses from the *Users* table:

```
$sql = "select * from " . UsersTableName();
$rs = CustomQuery($sql);
while ($emailsData = db_fetch_array($rs))
   echo $emailsData["EmailAddress"]."<br>";
```

See also:

- DAL Method: Custom Query
- DAL Method: TableName
- About Security API
- Database API
- About Data Access Layer

whereAdd

> **Deprecated**
>
> This function is deprecated, and we recommend using Dynamic SQL query.

Adds a new AND condition to the existing WHERE clause.

Syntax

```
whereAdd($where, $condition)
```

Arguments

$where

a WHERE clause. Example: "ID=19".

$condition

any condition clause. Example: "CustomerID='123'".

Return value

Returns an updated WHERE clause.

See also:

- Example: Dynamic SQL Query
- Database API
- About Data Access Layer

3.2.3 Database API

3.2.3.1 About Database API

The Database API functions similarly to the Data Access Layer (DAL) but is more convenient to use and works with multiple connections. As DAL is deprecated, we recommend using the Database API.

Methods

Method	Description
SetConnection(name)	Sets the current database link when working with multiple database connections.
Exec(SQL)	Executes an SQL query using the current active connection.
Query(SQL)	Executes an SQL Query that returns data.
LastId	Returns the last inserted autoincremented field value in the current connection.
LastError	Returns the last error message in the current connection.
Insert	Adds records to the database.
Update	Updates records in the database.
Delete	Deletes records from the database.
Select	Retrieves data from the database.
PrepareSQL	Prepares an SQL query to use with SQL variables.

See also:

- QueryResult object: fetchNumeric()
- QueryResult object: value()
- QueryResult object: fetchAssoc()
- Using SQL variables

3.2.3.2 Methods

SetConnection

Sets the current database connection. Recommended for use when working with multiple database connections.

All consequent API calls made via DB API functions will be executed via this connection. When working with the default (primary) connection the call of this function is not required.

To switch back to primary connection call this function without arguments.

Syntax

```
DB::SetConnection($name)
```

Arguments

$name

the connection name, as it appears in wizard.

Return value

No return value.

Example

```
// switch connection to Northwind
DB::SetConnection("Northwind");

// execute query
DB::Exec("update orders set Status='shipped' where OrderDate=CURDATE()");

// switch back to primary connection
DB::SetConnection("");
```

See also:

- Database API: Exec()
- Database API: LastId()
- Database API: LastError()
- Connecting to the database
- About Database API

Exec

Executes an SQL query using the current active connection.

Use with SQL Queries that do not return data.

Syntax

```
DB::Exec($SQL)
```

Arguments

$SQL

an SQL query.

Return value

true if executed successfully;

false in all other cases.

Example

```
DB::Exec("insert into orders (Name, Amount, OrderDate) values ('Jonh Smith', 128.12,
CURDATE())");
```

See also:

- Database API: SetConnection()
- Database API: Query()
- About Database API

Query

Executes an SQL Query that returns data, i.e., `SELECT * FROM TableName`.

Syntax

```
DB::Query($SQL)
```

Arguments

$SQL

an SQL query.

Return value

A QueryResult object.

Example

```
$rs = DB::Query("select * from carsmake");

while( $data = $rs->fetchAssoc() )
{
echo $data["id"];
echo $data["make"];
}
```

See also:

- QueryResult object: fetchAssoc()
- Database API: Exec()
- Database API: Select()
- QueryResult object: fetchNumeric()
- QueryResult object: value()
- Using SQL variables
- About Database API

LastId

Returns the last inserted autoincremented field value in the current connection.

Syntax

```
DB::LastId()
```

Arguments

No arguments.

Return value

Returns the last inserted autoincremented field value in the current connection.

Example

You can print the ID of the previously inserted record. This code works in any server side event like AfterAdd, AfterEdit, or BeforeProcess.

```
$data = array();
$data["make"] = "Toyota";
$data["model"] = "RAV4";
$data["price"] = 16000;
DB::Insert("cars", $data );
// get the ID of the inserted record and print it on the page
echo DB::LastId();
```

See also:

- Database API: Insert()
- Database API: LastError()
- Database API: SetConnection()
- Connecting to the database
- About Database API

LastError

Returns the last error message in the current connection.

Syntax

```
DB::LastError()
```

Arguments

No arguments.

Return value

The last error message in the current connection.

See also:

- Database API: LastId()
- Database API: SetConnection()
- Connecting to the database
- About Database API

Insert

Inserts a record into a database.

Syntax

```
DB::Insert($table, $values)
```

Arguments

`$table`

the table into which the data is inserted.

`$values`

an array with values that you wish to insert.

Return value

No return value.

Example 1

You can insert a record into any table.

```
// Insert a record into the 'Cars' table

$data = array();
$data["make"] = "Toyota";
$data["model"]  = "RAV4";
$data["price"] = 16000;
DB::Insert("cars", $data );
```

Example 2

You can copy the added or edited record into another table. To do so, use this code in the AfterAdd or AfterEdit event:

```
// Copy the record into the 'Copy_of_cars' table

$data = array();
$data["make"] = $values["make"];
$data["model"] = $values["model"];
$data["price"] = $values["price"];
DB::Insert("copy_of_cars", $data );
```

See also:

- Database API: Update()
- Database API: Delete()
- Database API: Select()
- About Database API

Update

Updates records in the database. The Update method overwrites existing records in the database instead of adding new ones.

> **Note**: the third parameter must be specified for the method to be executed successfully.

Syntax

```
DB::Update($table, $data, $keyvalues)
// or
DB::Update($table, $data, $where)
```

Arguments

`$table`

the table where you want to update the data.

`$data`

an array of values that overwrites the existing record.

`$keyvalues`

an array of key values that define the condition for the Update query.

`$where`

the condition for the Update query.

Return value

No return value.

Example

```
// Update the record with id=50 in the 'Cars' table

$data = array();
$keyvalues = array();
$data["make"] = "Toyota";
$data["model"]  = "RAV4";
$data["price"] = 16000;
$keyvalues["id"] = 50;
DB::Update("cars", $data, $keyvalues );
```

Alternative syntax:

```php
// Update a record in the 'Cars' table where id=50

$data = array();
$data["make"] = "Toyota";
$data["model"]  = "RAV4";
$data["price"] = 16000;
DB::Update("cars", $data, "id=50" );
```

See also:

- Database API: Insert()
- Database API: Delete()
- Database API: Select()
- About Database API

Delete

Deletes records from a database. The second parameter must be specified for the method to be executed successfully.

Syntax

```php
DB::Delete($table, $values)
// or
DB::Delete($table, $where)
```

Arguments

$table

the table from which the data is deleted.

$values

an array with values that define the condition for the delete query.

$where

the condition for the query; if unspecified, no action will be done.

Return value

No return value.

Example

```
// delete from Cars where id=50

$data = array();
$data["id"] = 50;
DB::Delete("cars", $data );
```

Alternative syntax:

```
// delete from Cars where id=50

DB::Delete("cars", "id=50" );
```

See also:

- Database API: Insert()
- Database API: Update()
- Database API: Select()
- About Database API

Select

Retrieves data from the database.

Syntax

```
DB::Select($table, $values)
// or
DB::Select($table, $where)
```

Arguments

$table

the table from which the data is selected.

$values

an array with values that define the condition of the Select query.

$where

the condition for the Select query.

Return value

An array with the result of the Select query.

Example 1

Retrieve all data from the *Cars* table where the *make* is Toyota and the *model* is RAV4.

```
$data = array();
$data["make"] = "Toyota";
$data["model"]  = "RAV4";
$rs = DB::Select("Cars", $data );
while( $record = $rs->fetchAssoc() )
{
     echo $record["id"];
     echo $record["make"];
}
```

Example 2

Retrieve all data from the *Cars* table where the *price* is less than 20,000.

```
$rs = DB::Select("Cars", "Price<20000" );
while( $record = $rs->fetchAssoc() )
{
echo $record["id"];
echo $record["make"];
}
```

See also:

- QueryResult object: fetchAssoc()
- Database API: Exec()
- Database API: Query()
- Database API: Insert()
- Database API: Update()
- Database API: Delete()
- Database API: PrepareSQL()
- QueryResult object: fetchNumeric()
- QueryResult object: value()
- Using SQL variables
- About Database API

PrepareSQL

This function prepares an **SQL query** when you use **SQL variables** in it. **SQL variables** are case-insensitive.

Read more about using Using SQL variables.

Syntax

```
DB::prepareSQL($SQL)
```

Arguments

$SQL

a query with variables.

Return value

The prepared SQL query.

Example 1

In the AfterAdd event, you can use:

```
$sql = DB::prepareSQL("insert into log set lastid=:new.id");
DB::Exec( $sql );
```

Example 2

You can use this code in the Server event of a Custom Button or in Field events for the **View**/**Edit** pages:

```
$sql = DB::prepareSQL("insert into log set lastname=':name'");
DB::Exec( $sql );
```

Example 3

You can also use several variables within the function:

```
$sql = DB::PrepareSQL("select * from customers where username=':session.userid' and
age>:1 and last_name=':2'", 20, "smirnoff");
```

The resulting query will look like this:

```
select * from customers where username='jsmith' and age>20 and last_name='smirnoff'
```

Note: the '20' and 'smirnoff' in the example above can be replaced with any PHP function or variable.

See also:

- Database API: Exec()
- Database API: Query()
- Database API: Select()
- Using SQL variables
- Insert custom button

- Field events
- QueryResult object: fetchNumeric()
- QueryResult object: value()
- QueryResult object: fetchAssoc()
- Tri-part events
- About Database API

3.2.3.3 QueryResult object

fetchAssoc

The **fetchAssoc()** function returns the record as an associative array: "field" => value.

Syntax

```
fetchAssoc()
```

Arguments

No arguments.

Return value

Returns the record as an associative array: "field" => value.

Null, if there are no matching records.

Example

```
$rs = DB::Query("select * from carsmake");
while( $data = $rs->fetchAssoc() )
{
echo $data["id"];
echo $data["make"];
}
```

See also:

- Database API: Query()
- QueryResult object: fetchAssoc()
- QueryResult object: fetchNumeric()
- QueryResult object: value()
- Database API: Exec()
- Database API: Select()
- Database API: PrepareSQL()
- Using SQL variables
- About Database API

fetchNumeric

The **fetchNumeric()** function returns the record as an array with numeric keys: 0 => value, 1=>value.

Syntax

```
fetchNumeric()
```

Arguments

No arguments.

Return value

Returns the record as an array with numeric keys: 0 => value, 1=>value.

Null, if there are no more records.

Example

```
$rs = DB::Query("select * from carsmake");
while( $data = $rs->fetchNumeric() )
{
echo $data[0];
```

```
echo $data[1];
}
```

See also:

- Database API: Query()
- QueryResult object: fetchAssoc()
- QueryResult object: value()
- Database API: Exec()
- Database API: Select()
- Database API: PrepareSQL()
- Using SQL variables
- About Database API

value

This is an alternative way of working with the query. Convenient to use when the query returns a single record.

This function should not be used together with fetchAssoc or fetchNumeric.

Syntax

```
value( fieldname or index )
```

Arguments

A fieldname or index.

Return value

Value.

Example1

```
$rs = DB::Query("select * from Cars where id=20");
echo $rs->value("Make");
```

Example2

```
$rs = DB::Query("select count(*) from Cars");
echo $rs->value(0);
```

See also:

- Database API: Query()
- QueryResult object: fetchNumeric()
- QueryResult object: fetchAssoc()
- Database API: Exec()
- Database API: Select()
- Database API: PrepareSQL()
- Using SQL variables
- About Database API

3.2.3.4 Using SQL variables

This feature can be used anywhere where you use SQL Queries. With **SQL variables**, you can write cleaner code and easily implement custom dropdown boxes or advanced security.

> **Note**: **SQL variables** are case-insensitive.

A list of SQL variables

`:field`

the current field value on an **Add**, **Edit** or **Register** page.

`:master.field`

any field from the master record.

`:session.key`

any session variable.

`:user.field`

any field from the *login* table.

`:old.field`

an old field value (before the changes were applied).

`:new.field`

a new field value.

Where to use the SQL variables

1. In regular SQL queries that you enter on the SQL Query screen.

2. In WHERE Tabs on the **SQL Query screen**.

3. In a Lookup Wizard WHERE clause. It can be used in dropdowns that are dependent on any type of field, or a master dropdown. Alternatively, it can be used when dependent dropdowns follow a more complex rule than equality (i.e., age is more than the selected).

```
WHERE CustomerID= ':user.CustomerID'
WHERE CustomerID= ':session.UserID'
```

In events by using the PrepareSQL function. For example, in the After Add event you can use:

```
$sql = DB::prepareSQL("insert into log set lastid=:new.id");
DB::Exec( $sql );
```

In the Server code of a Custom Button or in Field events for **View**/**Edit** pages:

```
$sql = DB::prepareSQL("insert into log set lastname=':name'");
DB::Exec( $sql );
```

See also:

- Database API: PrepareSQL()
- Database API: Exec()
- Database API: Query()
- Database API: Select()
- Insert custom button
- Lookup wizard: WHERE expression
- Field events
- Tri-part events
- About Database API

3.2.4 Dialog API

3.2.4.1 About Dialog API

Quick jump

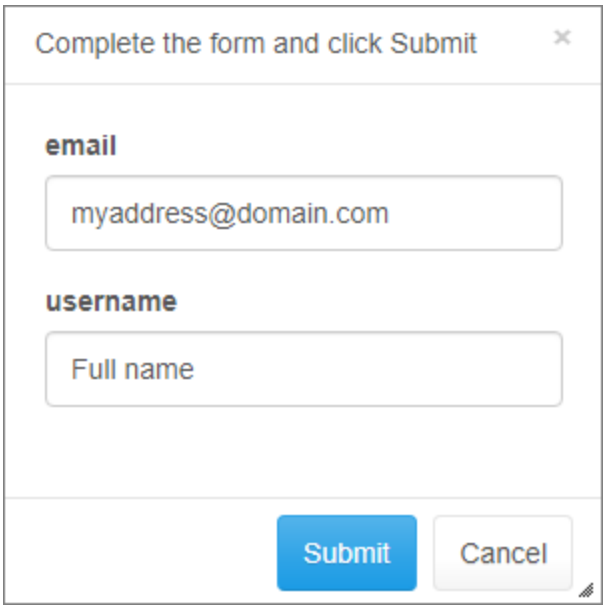
What is the Dialog API?

The **Dialog API** displays a dialog to the user and sends user-provided data to the Server event.

The dialog contains a title, text, input controls, **OK** and **Cancel** buttons. If the user clicks **OK**, the **Server** part of the event is called with the user-entered data passed to it. If the user clicks **Cancel**, the dialog closes, and nothing else happens.

This function is best suited for the Client Before section of the Custom button event or an AJAX code snippet.

Here is how a Dialog popup looks like:



Note: when you use the **Dialog API** in the **Client Before** event, you should always return false. Add this line to the end of your event code:

```
return false;
```

Alternatively, you can return the ctrl.dialog call itself, as it always returns false:

```
return ctrl.dialog( {... } );
```

Syntax

```
ctrl.dialog({settings})
```

Parameters

`title {string}`

sets the dialog title.

`header {string}`

sets the text displayed in the dialog above the input controls.

`ok {string}`

the label for the **OK** button. *'OK'* by default. If an empty string is provided, the **OK** button is not displayed.

`cancel {string}`

the label for the **Cancel** button. *'Cancel'* by default. If an empty string is provided, the **Cancel** button is not displayed.

`fields {array}`

sets an array of input fields and their descriptions.

Usage

In the description below **settings** is the dialog settings object.

Basic field description

The field description can be a simple string, for example, *'email'*. In this case, a text field with a label *email* is displayed. This input field is initially empty, and in the **Server part** of the event, its value is stored in the `$params["email"]` variable.

Here is an example of a settings.fields parameter with a basic description:

```
settings.fields = ['email', 'subject'];
```

Extended field description

An extended field description is an object with the following properties:

`name {string}`

the name of the variable in the **Server** event.

`label {string}`

the label of the field in the dialog.

`type {string}`

the type of an input control. It can be either *text* or *textarea*.

`value {string}`

the initial value of the field control.

Here is an example of a settings.fields parameter with both basic and extended descriptions:

```
settings.fields = [
'email',     // basic description

// extended description:
{
name: 'body',
label: 'Message body',
type: 'textarea',
value: 'Hello,\n'
}];
```

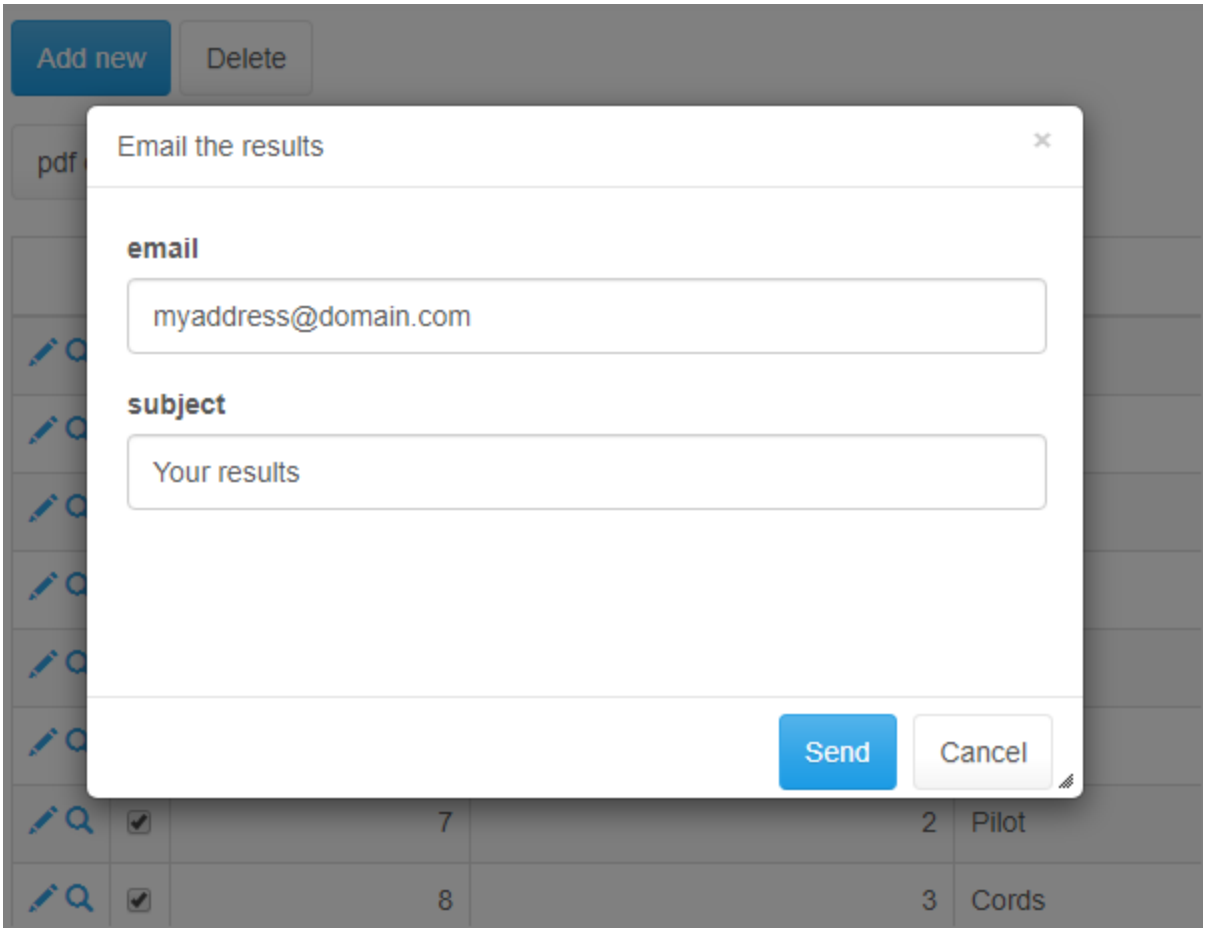
Example

Let's say you have a page that provides the results of a test where you want to implement a **Email the results** button.

To ask the user for the email and its subject, add the following code to the **Client before** section of the Custom button:

```
return ctrl.dialog( {
        title: 'Email the results',
        fields: [{
            name: 'email',
            value: 'myaddress@domain.com'
        },
        {
            name: 'subject',
            value: 'Your results'
        }],
        ok: 'Send',
        cancel: 'Cancel'
    });
```

This dialog appears after the user clicks the button.



You can access *email* and *subject* fields in the **Server** part of the button the same way you use any other parameter, i.e., using `$params["email"]` and `$params["subject"]` respectively.

Here is the complete code of the **Server** event that sends the email using the entered data.

```
$email=$params["email"] ;
$message="Hello there\nBest regards";
$subject=$params["subject"];
runner_mail(array('to' => $email, 'subject' => $subject,'body' => $message));
```

Examples:

- How to create a PDF file and save it to the output folder
- How to save the View page as PDF from the List page

- How to email the current page as PDF
- How to email selected records as separate PDF files

See also:

- runner_mail function
- Tri-part events
- About PDF API
- JavaScript API: getAllRecords()

3.2.5 Grid Row Javascript API

3.2.5.1 About Grid Row Javascript API

The **GridRow** object represents a row in the grid on the **List** page. It provides useful functions such as reading and modifying field values.

It is available as a row parameter in:

- AJAX snippets used in the **Click actions**;
- Custom buttons inserted into the data grid on the **List** page;
- In the **Inline Add**/**Edit** mode of the Field events.

You can also obtain **GridRow** objects for the grid records from any JavaScript event on the **List** page using these functions:

- RunnerPage.getAllRecords
- RunnerPage.getSelectedRecords

Methods

Method	Description

row.fieldCell()	Returns the jQuery object of the table cell with this field.
row.getFieldValue()	Returns a raw field value without formatting applied.
row.getFieldText()	Returns the field value with formatting applied. In other words, returns the HTML code of this table cell.
row.setFieldText()	Sets the field's HTML as it appears in grid.
row.setFieldValue()	The field value set this way does not appear in the grid and only applies to the consequent getFieldValue calls.
row.record()	Returns the jQuery object representing the table row.
row.setMessage()	Prints a message in one of the cells in the row. Useful to debug/troubleshoot your click action code.
row.getMessage()	Gets the previously added message.
row.getKeys()	Returns an array of key field values of the record.
row.recordId()	Returns the record ID.

See also

- Page Designer: Grid type
- Page Designer: Insert button into datagrid
- Tri-part events
- Buttons: rowData object
- Event: ListGetRowCount
- Change the row background color

3.2.5.2 Methods

row.fieldCell

Returns the **jQuery** object of the table cell with this field.

Syntax

```
row.fieldCell(field);
```

Arguments

`field`

the name of the field.

Return value

Returns the **jQuery** object of the table cell with this field.

Example

Change the "*make*" field background to red:

```
row.fieldCell("make").css('background', 'red');
```

See also:

- Customizing CSS examples
- Changing the background of a cell
- Working with cells
- Change the cell background color
- Click actions
- About Grid Row JavaScript API

row.getFieldValue

row.getFieldValue() gets the current field value on the page. If you used the row.setFieldValue function to adjust the value of the field, **row.getFieldValue()** will return the adjusted value, and not the one in the database.

Syntax

```
row.getFieldValue(field);
```

Arguments

`field`

the name of the field.

Return value

Returns a raw field value without formatting applied.

Example

Click the *OrderID* field to retrieve the current order total and display it in the *OrderID* field.

Client Before:

```
// pass OrderID to Server event
params["OrderID"] = row.getFieldValue("OrderID");
```

Server:

```
// run SQL Query to retrieve order total
$result["total"] = DBLookup("select sum(Quantity*UnitPrice) from `Order Details`
where OrderID=".$params["OrderID"]);
```

Client After:

```
// change cell background
row.fieldCell("OrderID").css('background', 'yellow' );
// add order total to OrderID cell content
row.fieldCell("OrderID").html(
   row.fieldCell("OrderID").html()+"<br>Total: "+result["total"]);
```

See also:

- Grid Row JavaScript API: row.fieldCell()
- DAL method: DBLookup()

- Grid Row JavaScript API: row.setFieldValue()
- Search API: getFieldValue()
- JavaScriptAPI: addField()
- RunnerPage object: hideField()
- RunnerPage object: showField()
- Tri-part events
- About SQL query API
- Field events
- About Grid Row JavaScript API

row.setFieldValue

Sets the field value on the current page.

The **row.setFieldValue()** function doesn't change the value in the database, so the adjusted value doesn't appear in the grid. The value set this way only applies to the consequent row.getFieldValue calls.

Syntax

```
row.setFieldValue(field, value);
```

Arguments

`field`

the field name.

`value`

the field value.

Return value

No return value.

Remarks

The field value set this way does not appear in the grid and only applies to consequent **getFieldValue** calls.

Example

Let's say we added a Custom button that changes the task status (the *'Status'* field) to *'Complete'* during the Server event.

In order to avoid reloading the page to get the correct status of the task, we can use the **row.SetFieldValue()** function in the Client After event for the button:

```
row.setFieldValue('Status', 'Complete');
```

See also:

- Grid Row JavaScript API: row.getFieldValue()
- Insert custom button
- Tri-part events
- Click actions
- About Grid Row JavaScript API

row.getFieldText

Returns the field value with formatting applied. In other words, returns the HTML code of this table cell.

Syntax

```
row.getFieldText(field);
```

Arguments

`field`

the name of the field.

Return value

Returns the field value with formatting applied.

For instance, we have a Lookup Wizard field that has different **Link** and **Display** fields selected. **getFieldText(field)** returns the **Display** field value. **getFieldValue(field)** returns the **Link** field value.

Example

This example shows how to emphasize some information in the field by underlining the text.

Create a custom button and insert it into the data grid containing the field you want to modify.

Use the code below and replace the `field` with the field name that corresponds to your project.

Client Before:

```
var txt = row.getFieldText("field");
txt = "<u>"+txt+"</u>";
row.setFieldText("field",txt);
```

See also:

- Grid Row JavaScript API: row.setFieldText()
- Displaying font in different colors
- Customizing CSS examples
- Insert Text/HTML code
- Change font size in text box
- About Grid Row JavaScript API

row.setFieldText

Sets the HTML of the field as it appears in grid.

Syntax

```
row.setFieldText(field, text);
```

Arguments

`field`

the field name.

`text`

the HTML of the field.

Return value

No return value

Example

This example shows how to emphasize some information in the field by making the font bold.

Create a custom button and insert it into the data grid containing the field you want to modify.

Use the code below and replace the `field` with the field name that corresponds to your project.

Client Before:

```
var txt = row.getFieldText("field");
txt = "<strong>"+txt+"</strong>";
row.setFieldText("field",txt);
```

See also:

- Grid Row JavaScript API: row.getFieldText()

- Displaying font in different colors
- Customizing CSS examples
- Insert Text/HTML code
- Change font size in text box
- About Grid Row JavaScript API

row.record

Returns the Jquery object representing the table row.

Syntax

```
row.record()
```

Arguments

No arguments.

Return value

Returns the Jquery object representing the table row.

Example

Change the row background to red:

```
row.record().css('background', 'red');
```

See also:

- Grid Row JavaScript API: row.recordId()
- Grid Row JavaScript API: row.setMessage()
- Customizing CSS examples
- Change the row background color

- Highlighting an entire row of a table
- About Grid Row JavaScript API

row.setMessage

Prints a message in one of the row cells. Useful to debug/troubleshoot your click action code.

Syntax

```
row.setMessage(str)
```

Arguments

`str`

the string to be printed in the row cell.

Return value

No return value.

Example

Add a message to the first cell in the row:

```
row.setMessage('test message', row.record()[0]);
```

See also:

- Grid Row JavaScript API: row.getMessage()
- Troubleshooting tips
- Grid Row JavaScript API: row.record()
- Change message after record was added or saved
- Display a message on the Web page
- About Grid Row JavaScript API

row.getMessage

Gets the previously added message.

Syntax

```
row.getMessage()
```

Arguments

No arguments.

Return value

No return value.

Example

Update the previously added message on click with some additional text:

```
row.setMessage(row.getMessage() + 'some additional text');
```

See also

- Grid Row JavaScript API: row.setMessage()
- Troubleshooting tips
- Change message after record was added or saved
- Display a message on the Web page
- About Grid Row JavaScript API

row.getKeys()

Returns an array of key field values of the record.

Syntax

```
row.getKeys()
```

Arguments

No arguments.

Return value

Returns an array of key field values of the record.

Example

```
console.log( row.getKeys() )
```

Output

```
[1]
//or
[3, 'Beverages']
```

See also

- Key columns
- JavaScript API: getSelectedRecordKeys()
- Buttons: rowData object
- Grid Row JavaScript API: row.recordId()
- About Grid Row JavaScript API

row.recordId()

Returns the record ID.

Note: **Record ID** is a numeric identifier of a record in the grid. It is used to control elements like buttons or Inline edit controls in a specific row.

Syntax

```
row.recordId()
```

Arguments

No arguments.

Return value

Returns the record ID.

Examples

```
// show/hide edit button in the current row

pageObj.toggleItem( 'grid_edit', false, row.recordId() );
```

```
// set value of the 'name' field control in the Inline Edit mode

var ctrl = Runner.getControl( row.recordId(), 'name' ');
ctrl.setValue('Luke Skywalker');
```

```
// imitate Edit button click

RunnerPage.getItemButton( 'grid_edit', row.recordId() ).click();
```

See also:

- JavaScript API: RunnerPage object > toggleItem()
- JavaScript API: SearchField object > getControl()
- JavaScript API: RunnerPage object > getItemButton(itemId, recordId)
- JavaScript API: Control object > setValue()
- JavaScript API: Date Control API > setValue()
- About Grid Row JavaScript API

3.2.6 JavaScript API

3.2.6.1 About JavaScript API

JavaScript API allows you to work with the "edit" controls, manage search panel. JavaScript API objects are available in the JavaScript OnLoad event of the appropriate page.

Objects

Object	Description
AJAX helper	Provides asynchronous functions for JavaScript events.
Control	Allows to work with the "edit" controls.
Date Control	Allows working in-depth with the date values.
InlineRow	Allows to process clicking the **Cancel** button in the **Inline** mode on the **Add**/**Edit** pages.
RunnerPage	Represents the current page object.
SearchController (deprecated)	Allows to manage the search panel.
SearchField	Allows to manage search fields on the search panel.

The JavaScript API works with **Add**/**Edit**/**View**/**Register** pages.

For example, use the following code to make the field red on the **View** page.

```
var ctrl = Runner.getControl(pageid,'action');
ctrl.addStyle('color: red');
```

Examples:

- Ask for confirmation before saving a record
- Change font size in text box
- Change font in "edit" controls

- Change width of edit box with AJAX popup
- Change width of text field on Quick Search panel
- How to calculate values (totals) on the fly
- How to control Inline Add/Edit functionality from script
- How to enable/disable a button
- How to hide 'Edit selected'/'Delete selected' buttons
- How to refresh List page after Edit in popup
- How to control Inline Add/Edit functionality from script
- Show dropdown list of US states if US was selected in country list

See also:

- JavaScript API: Control object > addStyle()
- JavaScript API: Control object > getControl()
- Events: JavaScript OnLoad

3.2.6.2 AJAX helper object

About AJAX helper object

The **AJAX helper** object provides several useful asynchronous functions for JavaScript events.

The **AJAX helper** object is available as an **ajax** parameter in:

- Click action AJAX snippets;
- Custom buttons;
- Field events.

Methods

Method	Description
setMessage()	Displays a message next to the button.

removeMessage()	Removes the previously set message.
setDisabled()	Makes the button disabled, so it can't be clicked.
setEnabled()	Enables the previously disabled button.
dialog()	Displays a dialog and sends user-provided data to the Server event. See Dialog API to learn more.
addPDF()	Creates a PDF file and sends in to the Server event. See PDF API to learn more.
submit()	Starts the Server part of the event when the ClientBefore part runs an asynchronous task.

See also:

- Dialog API
- Tri-part events
- PDF API
- About JavaScript API

Methods

setMessage

The **setMessage()** method displays a message next to the button.

Syntax

```
ajax.setMessage( text );
```

Arguments

text

text for the message as a string.

Return value

No return value.

Example

Let's say you have added a Custom button that creates a PDF file and saves it in the output folder of the project. The file opens after it is downloaded, and you want to display a message that says so.

Add the following code to the Client Before event:

```
ajax.setMessage("The PDF file will open automatically after it is downloaded");
```

See also:

- JavaScript API: AJAX helper object >removeMessage()
- Tri-part events
- How to create a PDF file and save it to the output folder
- Dialog API
- PDF API
- JavaScript API: AJAX helper object
- About JavaScript API

removeMessage

The **removeMessage()** method removes the previously set message next to the button.

Syntax

```
ajax.removeMessage();
```

Arguments

No arguments.

Return value

No return value.

Example

Let's say you have added a Custom button that creates a PDF file and saves it in the output folder of the project. The file opens after it is downloaded, and you have displayed a message that says so.

Now you want to remove the message as the file has already been opened. Add the following code to the Client After event:

```
ajax.removeMessage();
```

See also:

- JavaScript API: AJAX helper object > setMessage()
- Tri-part events
- How to create a PDF file and save it to the output folder
- Dialog API
- PDF API
- JavaScript API: AJAX helper object
- About JavaScript API

setDisabled

The **setDisabled()** method makes the button disabled, so it can't be clicked.

Syntax

```
ajax.setDisabled();
```

Arguments

No arguments.

Return value

No return value.

Example

Let's say you want to add a Custom button where the Server event needs to be delayed by 5 seconds. To prevent users from clicking the button while the delay takes place, you can disable the button.

Add the following code before the *setTimeout()* function:

```
ctrl.setDisabled();
```

See also:

- JavaScript API: AJAX helper object > setEnabled()
- Tri-part events: Asynchronous tasks
- Dialog API
- PDF API
- JavaScript API: AJAX helper object
- About JavaScript API

setEnabled

The **setEnabled()** method enables the previously disabled button, so it can be clicked.

Syntax

```
ajax.setEnabled();
```

Arguments

No arguments.

Return value

No return value.

Example

Let's say you have added a Custom button where the Server event needs to be delayed by 5 seconds. And to prevent users from clicking the button while the delay takes place, you have disabled the button.

Add the following code to the Client After event to re-enable the button:

```
ctrl.setEnabled();
```

See also:

- JavaScript API: AJAX helper object > setDisabled()
- Tri-part events
- Dialog API
- PDF API
- JavaScript API: AJAX helper object
- About JavaScript API

submit

The **submit()** method starts the Server part of the event, when the basic routine can not be used.

When the Client Before part of the Custom button event runs an asynchronous task (for example, displaying a dialog and waiting for the user's answer, or creating a PDF file), the event must return false. Otherwise the **Server** part runs immediately, before the asynchronous task ends.

Call the **submit()** function after the asynchronous task has finished to start the **Server** event.

Syntax

```
ajax.submit();

//or

submit();

//you can use either one
```

Arguments

No arguments.

Return value

No return value.

Example

```
// postpone the Server event by 5 seconds
setTimeout( function() {
submit();
}, 5000 );
return false;
```

See also:

- Tri-part events: Asynchronous tasks
- Dialog API
- PDF API
- JavaScript API: AJAX helper object
- About JavaScript API

addPDF

Description

The **ajax.addPDF** method creates a PDF file and sends it to the Server event. The file can be emailed, saved to the disk or into the database.

Syntax

```
ajax.addPDF( name, settings, callback )
```

Arguments

name {string}

the name of the variable in which the PDF file contents are stored in the *$params* server array, so that it is available to use in the Server event.

settings {object}

the PDF settings. See PDF Parameters for details. Use an empty object {} for default settings.

callback

defines which callback function is called after the PDF file is created. You should call either the submit() or dialog() function in the callback to start the **Server** part of the event.

Note: the *submit()* function is applied to start the **Server** part of the event when the basic routine cannot be used. It is necessary, when the Client Before event runs an asynchronous task, e.g., creating a PDF.

See submit() to learn more.

Return value

No return value.

Examples

Example 1

This example shows the use of the name argument.

Client before:

```
ajax.addPDF( 'pdf', {} );
return false;
```

Note: when using the **addPDF** fucntion in the **Client Before** event, you should always return *false*. Just add the following line to the end of your event code:

```
return false;
```

Server:

```
file_put_contents( "file.pdf", $params["pdf"] );
```

Note: see Tri-part events to learn more about **Client before** and **Server** events.

Example 2

This example shows the use of the callback argument in the **ClientBefore** event.

```
ajax.addPDF( "pdf", {}, function() {
   ajax.submit();
});
return false;
```

See also:

- Example: How to create a PDF and save it to the disk
- Example: How to save the View page as PDF from the List page
- Example: How to email the current page as PDF
- Example: How to email selected records as separate PDF files
- Tri-part events
- About Dialog API
- About AJAX helper object
- About PDF API

3.2.6.3 Control object

About Control object

About Control object

The **Control object** allows you to work with the *edit* controls. **Control object** is available in the JavaScript OnLoad event of the appropriate page.

Before you start working with the *edit* controls, you need to get those controls.

getControl()

Use the **getControl()** method to get the controls by the field name and page ID:

Syntax

```
var ctrl = Runner.getControl(pageid, fieldname);
```

Example

```
var ctrl = Runner.getControl(pageid, 'Make');
```

controls.ControlManager.getAt(tableName)

To get all the controls of the table, you need to pass only the table name as the argument:

Syntax

```
var ctrlArr = Runner.controls.ControlManager.getAt(tableName);
```

Example

```
// Get all the controls for the 'Cars' table
var recCtrlsArr = Runner.controls.ControlManager.getAt('Cars');
// loop through all controls on the page making them all required
for(var i=0;i<recCtrlsArr.length;i++)
{
    var ctrl = recCtrlsArr[i];
```

```
    ctrl.addValidation("IsRequired");
}
```

Methods

Method	Description
addClass()	Adds the class to the control with a value.
addStyle()	Adds the style to the control.
addValidation()	Adds validation to the control.
clear()	Sets "" as the control value. If the control was not previously validated, removes the message that it was not validated.
clearEvent()	Deletes events.
getDispElem()	Returns a **jQuery object** - the element (input, textarea, select, etc.) that displays the value of the selected field.
getValue()	Reads the control value.
hide()	Hides the control.
invalid()	Gets the control status after the last validation.
isReadonly()	Gets the control readonly status.
makeReadonly()	Makes control readonly.
makeReadWrite()	Makes control writable.
on()	Adds an event to the control and transfers an array of arguments to the handler.
removeClass()	Removes the from the control.
removeValidation()	Removes validation from the control.
reset()	Sets the control value to the original one.
setDisabled()	Makes the control "disabled".
setEnabled()	Makes the control "enabled".
setFocus()	Sets focus to the control.
setValue()	Sets the control value.

show()	Shows the control.
validate()	Validates the control.
validateAs()	Validates the control value for some validation type.

Events

Event name	Description
blur	An element loses focus.
change	The user changes the content of a field.
click	Mouse clicks an object.
dblclick	Mouse double-clicks an object.
focus	An element gets focus.
keydown	A keyboard key is pressed.
keypress	A keyboard key is pressed or held down.
keyup	A keyboard key is released.
mousedown	A mouse button is pressed.
mousemove	The mouse is moved.
mouseout	The mouse is moved off the element.
mouseover	The mouse is moved over an element.
mouseup	A mouse button is released.
resize	A window or frame is resized.
select	Text is selected.

Control object examples:

- Ask for confirmation before saving a record
- Change font in dropdown list
- Change font in "edit" controls
- Change width of edit box with AJAX popup

- Change width of text field on Quick Search panel
- How to calculate values (totals) on the fly
- How to control Inline Add/Edit functionality from script
- How to hide 'Edit selected'/'Delete selected' buttons
- Show dropdown list of US states if US was selected in country list

See also:

- JavaScript API: Control object > addValidation()
- About JavaScript API

Methods

add CSS Class

Adds the CSS class to the control with a value.

Syntax

```
ctrl.addClass(className);
```

Arguments

`className`

the CSS class name. Example: `'highlight'`.

Return value

No return value.

Example

Add a CSS class highlight to the control using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.addClass('highlight');
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > remove CSS Class
- JavaScript API: Control object >addStyle()
- Customizing CSS examples
- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

addStyle

Adds a style to the control.

Syntax

```
ctrl.addStyle(style);
```

Arguments

`style`

a CSS string with style definition. Example: `'width: 200px;'`.

Return value

No return value.

Example

Add the style `'display: none;'` to the control using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.addStyle('display: none;');
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > add CSS class
- JavaScript API: Control object > remove CSS Class
- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

addValidation

Adds validation to the control.

Syntax

```
ctrl.addValidation(validation_type);
```

Arguments

validation_type

one of the available validation types:

Validation type	Description
IsRequired	Makes the field required.
IsNumeric	A number.
IsPassword	The password cannot be blank, cannot be 'Password' and should be at least 4 characters long.
IsEmail	A valid email address.
IsMoney	A numeric value. Decimal point is allowed. Examples: 13, 24.95.
IsZipcode	Five or ten digit number. Valid formats: 12345, 12345-6789 or 123456789.

IsPhonenumber	Numbers, spaces, hyphens, and parentheses are allowed. Examples: (123) 456-7890, 123 456 7890, 123 4567.
IsState	A two-letter US state abbreviation. Examples: AK, AL, CA, MN.
IsSSN	A nine-digit US social security number. Valid formats: 123-45-6789 or 123 45 6789.
IsCC	A valid credit card number.
IsTime	Any valid time format that matches regional settings.
IsDate	Any valid date format that matches regional settings.

{
regex: "regular_expression",
message: "warning_message",
messagetype: "message_type"
}
	A regular expression (regexp).
	
	Note that **regexp validation** overwrites the previous validation, if there is any.

Return value

No return value.

Example 1

Make all controls on the page required using the JavaScript OnLoad event:

```
// Get all the controls for the 'Cars' table
var recCtrlsArr = Runner.controls.ControlManager.getAt('Cars');
// loop through all controls on the page making them all required
for(var i=0;i<recCtrlsArr.length;i++)
{
    var ctrl = recCtrlsArr[i];
    ctrl.addValidation("IsRequired");
}
```

Example 2

Add a regular expression validation to the control using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Number');
ctrl.addValidation(/[0-9]/);
ctrl.customValidationFailedMessages[ "RegExp" ] = {
```

```
    message: "The field should be a number from 0 to 9",
    messageType: "Text"
};
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > validate()
- JavaScript API: Control object > validateAs()
- JavaScript API: Control object > removeValidation()
- JavaScript API: Control object > invalid()
- "Edit as" settings : Validation types
- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

clear

Sets "" as the control value, and if the control was not previously validated, removes the message that it was not validated.

Syntax

```
ctrl.clear();
```

Arguments

No arguments.

Return value

No return value.

Example

Clear the control value using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Model');
ctrl.clear();
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > clearEvent()
- JavaScript API: Control object > hide()
- JavaScript API: Control object > reset()
- JavaScript API: Control object > setDisabled()
- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

clearEvent

Deletes events.

Syntax

```
ctrl.clearEvent(event);
```

Arguments

event

one of the available event values.

Return value

No return value.

Example

Delete the 'click' event using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.clearEvent('click');
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > on()
- JavaScript API: Control object > clear()
- JavaScript API: Control object > hide()
- JavaScript API: Control object > reset()
- JavaScript API: Control object > setDisabled()
- Event: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

getDispElem

Returns a **jQuery object** - the element (input, textarea, select, etc.) that displays the value of the selected field with the prefix value_ (e.g., value_Make for the field *Make*).

Syntax

```
ctrl.getDispElem();
```

Arguments

No arguments.

Return value

Returns a jQuery object.

Example

Change width of edit box with AJAX popup using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.getDispElem().css("width", "200px");
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > getValue()
- JavaScript API: Control object > add CSS class
- JavaScript API: Control object > remove CSS Class
- JavaScript API: Control object > addStyle()
- Event: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

getValue

Reads the control value.

Syntax

```
var value = ctrl.getValue();
```

Arguments

No arguments.

Return value

Returns the current value of the control.

Example1

Read the control value using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
var value = ctrl.getValue();
```

Example2

Show an alert if the checkbox control is checked using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Discounted');
if (ctrl.getValue()=='on')
   alert('Checked');

// you can use the setValue function to clear the checkbox:
ctrl.setValue('');
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > setValue()
- JavaScript API: Control object > reset()
- Example: Show dropdown list of US states if US was selected in country list
- Event: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

hide

Hides the control.

Syntax

```
ctrl.hide();
```

Arguments

No arguments.

Return value

No return value.

Example

Hide the control (not the label) using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.hide();
```

If you need to hide the control and the label, use the pageObj.hideField() method.

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > show()
- JavaScript API: Control object > setDisabled()
- JavaScript API: Control object > clear()
- JavaScript API: RunnerPage object > hideField()
- Example: Show dropdown list of US states if US was selected in country list
- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

invalid

Gets the control status after the last validation.

Syntax

```
var isInvalid = ctrl.invalid();
```

Arguments

No arguments.

Return value

True if the control is Invalid.

False if the control is Valid.

Example

Get the control status using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'YearOfMake');
var isInvalid = ctrl.invalid();
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > addValidation()
- JavaScript API: Control object > removeValidation()
- JavaScript API: Control object > validate()
- JavaScript API: Control object > validateAs()
- "Edit as" settings: Validation types
- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

isReadonly

Gets the readonly status of the control.

Syntax

```
var readonly = ctrl.isReadonly();
```

Arguments

No arguments.

Return value

True if the control is readonly.

False if the control is writable.

Example

Check if the control is readonly using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
var readonly = ctrl.isReadonly();
```

Remarks

Readonly controls are submitted with the form and saved in the database. **Readonly** controls are filled by the Autofill feature. You can apply default values to **Readonly** controls.

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > makeReadonly()
- JavaScript API: Control object > makeReadWrite()
- JavaScript API: Control object > reset()
- JavaScript API: Control object > getValue()

- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

makeReadonly

Makes the control readonly.

Syntax

```
ctrl.makeReadonly();
```

Arguments

No arguments.

Return value

No return value.

Example

Make the control readonly using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.makeReadonly();
```

Remarks

Readonly controls are submitted with the form and saved in the database. **Readonly** controls will be filled by the Autofill feature. You can apply default values to **Readonly** controls.

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > isReadonly()

- JavaScript API: Control object > makeReadWrite()
- JavaScript API: Control object > reset()
- JavaScript API: Control object > getValue()
- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

makeReadWrite

Makes the control writable.

Syntax

```
ctrl.makeReadWrite();
```

Arguments

No arguments.

Return value

No return value.

Example

Make the control writable using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.makeReadWrite();
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > isReadonly()

- JavaScript API: Control object > makeReadonly()
- JavaScript API: Control object > reset()
- JavaScript API: Control object > getValue()
- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

on

Adds an event to the control and transfer the array of arguments to the handler.

Syntax

```
ctrl.on(event,handler,arguments);
```

Arguments

`event`

one of the available event values.

`handler`

the function performed after the event was fired.

`arguments`

the parameters passed to the handler function.

Return value

No return value.

Example

Add the 'click' event to the control using the JavaScript OnLoad event:

```
ctrl.on('click', function() {
// call the 'setValue' method to set a new value
```

```
    this.setValue('newValue');
});
```

See also:

- JavaScript API: Control object > setValue()
- JavaScript API: Control object > clearEvent()
- Example: Show dropdown list of US states if US was selected in country list
- Example: Ask for confirmation before saving a record
- Example: Show order total on the Edit page as the details table is updated
- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

remove CSS Class

Removes the CSS class from the control.

Syntax

```
ctrl.removeClass(className);
```

Arguments

`className`

a CSS class name. Example: `'highlight'`.

Return value

No return value.

Example

Remove the `'highlight'` class from the control using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.removeClass('highlight');
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > add CSS Class
- JavaScript API: Control object > addStyle()
- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

removeValidation

Removes validation from the control.

Syntax

```
ctrl.removeValidation(validation_type);
```

Arguments

validation_type

one of the available validation types:

Validation constant	Description
IsRequired	Makes the field required.
IsNumeric	A number.
IsPassword	The password cannot be blank, cannot be 'Password' and should be at least 4 characters long.
IsEmail	A valid email address.

IsMoney	A numeric value. Decimal point is allowed. Examples: 13, 24.95.
IsZipcode	Five or ten digit number. Valid formats: 12345, 12345-6789 or 123456789.
IsPhonenumber	Numbers, spaces, hyphens, and parentheses are allowed. Examples: (123) 456-7890, 123 456 7890, 123 4567.
IsState	A two-letter US state abbreviation. Examples: AK, AL, CA, MN.
IsSSN	A nine-digit US social security number. Valid formats: 123-45-6789 or 123 45 6789.
IsCC	A valid credit card number.
IsTime	Any valid time format that matches regional settings.
IsDate	Any valid date format that matches regional settings.
RegExp	A regular expression (regexp).

Return value

No return value.

Example

Remove the *IsRequired* validation from the control using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.removeValidation("IsRequired");
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > addValidation()
- JavaScript API: Control object > validate()
- JavaScript API: Control object > validateAs()
- JavaScript API: Control object > invalid()
- "Edit as" settings: Validation types

- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

reset

Sets the control value to the original one.

Syntax

```
ctrl.reset();
```

Arguments

No arguments.

Return value

No return value.

Example

Set the control value to the original one using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.reset();
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > clear()
- JavaScript API: Control object > hide()
- JavaScript API: Control object > setDisabled()
- Events: JavaScript OnLoad

- JavaScript API: Control object
- JavaScript API

setDisabled

Makes the control disabled.

Syntax

```
ctrl.setDisabled();
```

Arguments

No arguments.

Return value

No return value.

Example

Make the control "disabled" using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.setDisabled();
```

Remarks

Disabled controls are NOT submitted with the form and are not saved in the database. Disabled controls will be filled by the Autofill feature. You can apply default values to Disabled controls.

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > setEnabled()
- JavaScript API: Control object > clear()

- JavaScript API: Control object > hide()
- JavaScript API: Control object > reset()
- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

setEnabled

Makes the control enabled.

Syntax

```
ctrl.setEnabled();
```

Arguments

No arguments.

Return value

No return value.

Example

Make the control "enabled" using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.setEnabled();
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > setDisabled()
- JavaScript API: Control object > clear()

- JavaScript API: Control object > hide()
- JavaScript API: Control object > reset()
- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

setFocus

Sets focus to the control and, depending on the triggerEvent argument, raises/does not raise "focus" event.

Syntax

```
ctrl.setFocus(triggerEvent);
```

Arguments

triggerEvent

if **true**, the app runs the "focus" event, previously added to the control using the on function. If **false** or not specified, the app does not run the event.

Return value

No return value.

Example

Set focus to the control using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.setFocus();
```

See also:

- JavaScript API: Control object > getControl()

- JavaScript API: Control object > on()
- JavaScript API: Control object > show()
- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

setValue

Sets the control value.

Syntax

```
ctrl.setValue(value);
```

Arguments

value

the value of the control.

Return value

No return value.

Example 1

Calculate the total value on the fly using the JavaScript OnLoad event:

```
var ctrlPrice = Runner.getControl(pageid, 'Price');
var ctrlQuantity = Runner.getControl(pageid, 'Quantity');
var ctrlTotal = Runner.getControl(pageid, 'Total');

function func() {
  ctrlTotal.setValue(Number(ctrlPrice.getValue()) * Number(ctrlQuantity.getValue()));
};

ctrlPrice.on('keyup', func);
ctrlQuantity.on('keyup', func);
```

Example 2

Show an alert if the checkbox control is checked using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Discounted');
if (ctrl.getValue()=='on')
   alert('Checked');

// you can use the setValue function to clear the checkbox:
ctrl.setValue('');
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > getValue()
- JavaScript API: Control object > on()
- Example: Show dropdown list of US states if US was selected in country list
- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

show

Shows the control.

Syntax

```
ctrl.show();
```

Arguments

No arguments.

Return value

No return value.

Example

Show the control using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.show();
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > hide()
- JavaScript API: Control object > setEnabled()
- Example: Show dropdown list of US states if US was selected in country list
- Events: JavaScript OnLoad
- JavaScript API: RunnerPage object > showField()
- JavaScript API: Control object
- JavaScript API

validate

Validates the control value against all previously added validation types.

Syntax

```
var vRes = ctrl.validate();
```

Arguments

No arguments.

Return value

Returns an array with two values:

- **result** (the validation result, values: *true* or *false*);
- **messageArr** (the array of validation errors). If the result is *true*, then the **messageArr** array is blank.

Example

Validate the control and display validation errors if there are any using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'make');
var vRes = ctrl.validate();
if ( !vRes.result ) {
    var message = "";
    for (var i = 0; i < ctrl.validationArr.length ; i++) {
        if ( vRes.messagesData[ ctrl.validationArr[i] ] ) {
            message += vRes.messagesData[ ctrl.validationArr[i] ].join(" ");
        }
    }

    alert( message );
}
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > addValidation()
- JavaScript API: Control object > removeValidation()
- JavaScript API: Control object > validateAs()
- JavaScript API: Control object > invalid()
- "Edit as" settings: Validation types
- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

validateAs

Validates the control value for a certain validation type. The control is not marked as unvalidated.

Syntax

```
var vRes = ctrl.validateAs(validation_type);
```

Arguments

validation_type

one of the available validation types:

Validation constant	Description
IsRequired	Makes the field required.
IsNumeric	A number.
IsPassword	The password cannot be blank, cannot be 'Password' and should be at least 4 characters long.
IsEmail	A valid email address.
IsMoney	A numeric value. Decimal point is allowed. Examples: 13, 24.95.
IsZipcode	Five or ten digit number. Valid formats: 12345, 12345-6789 or 123456789.
IsPhonenumber	Numbers, spaces, hyphens, and parentheses are allowed. Examples: (123) 456-7890, 123 456 7890, 123 4567.
IsState	A two-letter US state abbreviation. Examples: AK, AL, CA, MN.
IsSSN	A nine-digit US social security number. Valid formats: 123-45-6789 or 123 45 6789.
IsCC	A valid credit card number.
IsTime	Any valid time format that matches regional settings.
IsDate	Any valid date format that matches regional settings.

Return value

Returns *true* or a validation error.

Example

Validate the control value for the "IsRequired" validation using the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'make');
Runner.validation.control = ctrl;
var vRes = ctrl.validateAs("IsRequired");
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > addValidation()
- JavaScript API: Control object > removeValidation()
- JavaScript API: Control object > validate()
- JavaScript API: Control object > invalid()
- "Edit as" settings: Validation types
- Events: JavaScript OnLoad
- JavaScript API: Control object
- JavaScript API

3.2.6.4 Date Control API

About Date Control API

The **Date Control API** allows working in-depth with the date values. It can also limit the weekdays the user can choose or set the available date interval.

Date Control API can be used in the JavaScript OnLoad event, as well as any other JavaScript event.

Date Control API Methods

Method	Description
setValue	Sets the Date control value.
getValue	Gets the Date control value as a Date object.

getMomentValue	Get the Date control value as a Moment.js object.
toggleWeekDay	Sets the allowed days of the week.
setAllowedInterval	Sets the allowed dates interval.
getAllowedInterval	Gets the allowed dates interval, returns the JavaScript Date array.

See also:

- Event: GetTablePermissions
- "Edit as" settings: Date
- About JavaScript API

Methods

setValue

This method sets the Date control value.

Syntax

```
ctrl.setValue(value)
```

Arguments

value

a datetime string "2001-05-12 12:22:33", a JavaScript Date or a Moment.js object.

Return value

No return value.

Examples

setValue() can be used in the JavaScript OnLoad event, as well as any other JavaScript event.

1. Set the date control value to 12/24/19 as a string date.

```
//string date
var ctrl = Runner.getControl(pageid, 'datefield');
ctrl.setValue('2019-12-24 12:15:30');
```

2. Set the date control value as a JavaScript Date object.

```
//javascript date object
var ctrl = Runner.getControl(pageid, 'datefield');
var js_date = new Date('2019-12-24T03:24:00');
ctrl.setValue(js_date);
```

3. Set the date control value as a current date with a Moment.js object.

```
//Moment.js date object
var ctrl = Runner.getControl(pageid, 'datefield');
var moment_date = moment();
ctrl.setValue(moment_date);
```

Note: replace `"datefield"` with the actual field name in the function.

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Date Control API > getValue()
- JavaScript API: Date Control API > getMomentValue()
- Events: JavaScript OnLoad
- Event editor
- Date control API
- JavaScript API

getValue

This method gets the Date control value as a JavaScript Date object.

Syntax

```
ctrl.getValue()
```

Arguments

No arguments.

Return value

The date control value as a JavaScript Date object.

Example

getValue() can be used in the JavaScript OnLoad event, as well as any other JavaScript event.

Show the date, entered into the selected control:

```
var ctrl = Runner.getControl(pageid, 'datefield');
var str_date = ctrl.getStringValue();
alert(str_date);
```

Note: replace `"datefield"` with the actual field name in the function.

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Date Control API > setValue()
- JavaScript API: Date Control API > getMomentValue()
- Date control API
- JavaScript API

getMomentValue

This method gets the Date control value as a Moment.js object.

Syntax

```
ctrl.getMomentValue()
```

Arguments

No arguments.

Return value

The date control value as a Moment.js object.

Example

getMomentValue() can be used in the JavaScript OnLoad event, as well as any other JavaScript event.

Show the date, entered into the selected control:

```
var ctrl = Runner.getControl(pageid, 'datefield');
var moment_date = ctrl.getMomentValue();
alert(moment_date);
```

Note: replace `"datefield"` with the actual field name in the function.

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Date Control API > setValue()
- JavaScript API: Date Control API > getValue()

- Date control API
- JavaScript API

toggleWeekDay

This method sets the allowed days of the week, that can be entered into the Date control. To restrict the manual user input, create a custom validation plugin for the Edit as: Date fields.

Alternatively, you can set up the allowed weekdays and the validation with the **Edit as: Date** options.

> **Note**: the "allowed" status and the message for each day are stored separately, so you can use this method several times to define individual messages for each set of days.

Syntax

```
ctrl.toggleWeekDay( day, enable: boolean, message: string )
```

Arguments

day

a number or an array of numbers, where 0 - is a Sunday, 1 - is a Monday, etc.;

enable

a boolean variable, enables or disables the allowed days of the week;

message

an error message that is displayed when a date outside of the selected interval is selected. If no message is specified, a default one is displayed.

Return value

No return value.

Example

toggleWeekDay() can be used in the JavaScript OnLoad event, as well as any other JavaScript event.

Restrict selecting Friday and Sunday/Monday/Tuesday with 2 different messages:

```
var ctrl = Runner.getControl(pageid, 'datefield');
//restrict selecting Friday, show error message 1 when selected
ctrl.toggleWeekDay( 5, false, 'error message 1' );
//restrict selecting Sunday, Monday, Tuesday, show error message 2 when selected
ctrl.toggleWeekDay( [0,1,2], false, 'error message 2');
```

Note: replace "datefield" and each of the "error message" with the actual field name and error message in the function.

See also:

- JavaScript API: Control object > getControl()
- "Edit as" settings: Validation types
- "Edit as" settings: Date
- "Edit as" settings
- Date control API
- JavaScript API

setAllowedInterval

This method sets the allowed interval, that can be entered into the Date control.

Syntax

```
ctrl.setAllowedInterval( start, end, message: string )
```

Arguments

start/end

a datetime string "2001-05-12 12:22:33", a JavaScript Date or a Moment.js object. If the start/end argument is undefined, the interval has no start/end;

`message`

a string that appears when a date outside of the selected interval is entered into the control. If the message is undefined, a default one is displayed.

Return value

No return value.

Examples

setAllowedInterval() can be used in the JavaScript OnLoad event, as well as any other JavaScript event.

1. Allow selecting dates starting from 12/15/19 and on with a string date:

```
//string date
var ctrl = Runner.getControl(pageid, 'datefield');
ctrl.setAllowedInterval( '2019-12-15', null, 'error message' );
```

2. Allow selecting only the dates from 12/10/19 to 12/25/19 with a JS Date object:

```
//javascript date object
var ctrl = Runner.getControl(pageid, 'datefield');
var js_start = new Date('2019-12-10');
var js_end = new Date('2019-12-25');
ctrl.setAllowedInterval( js_start, js_end, 'error message' );
```

3. Allow selecting only the past dates (prior to the current one) with a Moment.js object:

```
//Moment.js date object
var ctrl = Runner.getControl(pageid, 'datefield');
var moment_date = moment().add(-1, 'days');
ctrl.setAllowedInterval( null, moment_date, 'error message' );
```

4. Allow selecting only the future dates:

```
//javascript date object
var ctrl = Runner.getControl(pageid, 'datefield');
var js_start = new Date(new Date().getTime() + 24 * 60 * 60 * 1000);
ctrl.setAllowedInterval( js_start, null, 'error message' );
```

5. Allow selecting ten days, starting with tomorrow's date:

```
//Moment.js date object
var ctrl = Runner.getControl(pageid, 'datefield');
var start_moment_date = moment().add(1,'days');
var end_moment_date = moment().add(11,'days');
ctrl.setAllowedInterval( start_moment_date, end_moment_date, 'error message' );
```

Note: replace "datefield" and "error message" with the actual field name and error message in the function.

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Date Control API > getAllowedInterval()
- "Edit as" settings: Date
- "Edit as" settings
- Date control API
- JavaScript API

getAllowedInterval

This method returns the allowed interval for the selected Date control.

Syntax

```
ctrl.getAllowedInterval()
```

Arguments

No arguments.

Return value

Returns the *[start, end]* array as a JavaScript Date object.

Example

getAllowedInterval() can be used in the JavaScript OnLoad event, as well as any other JavaScript event.

Set the allowed interval and then make the interval have an open upper boundry:

```
var ctrl = Runner.getControl(pageid, 'datefield');
//set the allowed interval
ctrl.setAllowedInterval( '2019-12-10', '2019-12-24', 'error message' );
//.......................
//get the allowed interval
var interval = ctrl.getAllowedInterval();
//get the lower boundry of the interval
start_date = interval[0];
//set the new allowed interval with an open upper boundry
ctrl.setAllowedInterval( start_date, null );
```

Note: replace `"datefield"` and `"error message"` with the actual field name and error message in the function.

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Date Control API > setAllowedInterval()
- "Edit as" settings: Date
- "Edit as" settings
- Date control API
- JavaScript API

3.2.6.5 InlineRow object

About InlineRow object

Description

The **InlineRow object** allows to process clicking the **Cancel** button in the Inline mode on **Add**/**Edit** pages.

inlineRow.data object contains values of the fields as they were before editing. Example:

```
inlineRow.data["Price"]
```

Methods

Method	Description
onBeforeCancel	The method is invoked by clicking the **Cancel** button during **inline add**/**edit**.
onCancel	The method is invoked after the cancellation of **inline add**/**edit**.
revokeCancel	Revokes the cancellation of **inline add**/**edit** (the record stays in edit mode).

Examples

```
if(inlineRow){
    inlineRow.onCancel = function(){
        console.log('edit cancelled');
    };
}
```

```
if(inlineRow){
 inlineRow.onBeforeCancel = function(){
            inlineRow.revokeCancel = true;
    };
}
```

See also:

- About GridRow JavaScript API
- How to control Inline Add/Edit functionality from script
- JavaScript API

3.2.6.6 RunnerPage object

About RunnerPage object

RunnerPage object represents the current page object and is called *pageObj*. It is available in the JavaScript OnLoad event of the appropriate page.

Methods

Method	Description
getSearchController()	Gets the object of the search panel (SearchController object).
hideField()	Hides the field and its label.
showField()	Shows the previously hidden field and its label.
getTabs()	Returns the RunnerTabs object, which represents a single tab group.
toggleItem()	Shows or hides a page element in JavaScript events.
getAllRecords()	Returns an array of GridRow objects for all the records in the data grid on the **List** page.
getSelectedRecords()	Returns an array of GridRow objects for all selected records in the data grid on the **List** page.
getSelectedRecordKeys()	Returns an array of key field values for all the selected records.
getDetailsPage(name, recordId)	Returns the **RunnerPage object** of the details preview displayed on the page.

getDetailsPages(recordId)	Returns the **RunnerPage objects** for all the details previews on the current page.

getItemButton(itemId, recordId)	Returns the jQuery object that represents the button object.
findItem(itemId, recordId)	Returns the jQuery object representing the page element. Use this function when you need to apply code to one of the page elements.
getCurrentTabId()	Returns the id of the current tab. This function can be used on any **List**, **Print** and **Export** page events.
setCurrentTabId(tabId)	Selects a tab to display. This event can be used in the Before process events on the **List**, **Print** and **Export** pages.

See also:

- Tri-part events
- AJAX helper object
- Fiild events
- Click actions
- Global events
- Table events
- JavaScript API

Methods

getSearchController

Gets the object of the search panel (SearchController object). The **getSearchController()** method is available in the JavaScript OnLoad event of the **List**/Chart/Report/**Advanced Search** pages.

> **Note**: the **SearchController object** is deprecated.

Syntax

```
var srch = pageObj.getSearchController();
```

Arguments

No arguments.

Return value

Returns the **SearchController object**.

See also:

- About SearchController object
- About Search API
- JavaScript API: RunnerPage object
- JavaScript API

hideField

Hides the field and its label. The **hideField()** method is available in the JavaScript OnLoad event of the **Add**/**Edit**/**View**/**Register** pages.

Syntax

```
pageObj.hideField(field);
```

Arguments

`field`

the field name. Example: "*Make*".

Return value

No return value.

Example 1

Hide the *Make* field. Add the following code to the JavaScript OnLoad event:

```
pageObj.hideField("Make");
```

Example 2

Show the *State* field, if the *Country* is *USA*; **hide** the field otherwise. Add the following code to the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, "Country");
if(ctrl.getValue() != "USA")
pageObj.hideField("State");
ctrl.on('change', function(){
 if (this.getValue() == "USA"){
    pageObj.showField("State");
 }
 else {
    pageObj.hideField("State");
 }
});
```

See also:

- JavaScript API: Control object > getValue()
- JavaScript API: Control object > getControl()
- JavaScript API: RunnerPage object > showField()
- RunnerPage class: showField()
- RunnerPage class: hideField()
- Show dropdown list of US states if US was selected in country list
- JavaScript API: RunnerPage object
- JavaScript API

showField

Shows the previously hidden field and its label. The **showField()** method is available in the JavaScript OnLoad event of the **Add**/**Edit**/**View**/**Register** pages.

Syntax

```
pageObj.showField(field);
```

Arguments

`field`

the field name. Example: "*Make*".

Return value

No return value.

Example 1

Show the *Make* field. Add the following code to the JavaScript OnLoad event:

```
pageObj.showField("Make");
```

Example 2

Show the *State* field, if the *Country* is *USA*; **hide** the field otherwise. Add the following code to the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, "Country");
if(ctrl.getValue() != "USA")
pageObj.hideField("State");
ctrl.on('change', function(){
 if (this.getValue() == "USA"){
    pageObj.showField("State");
 }
 else {
    pageObj.hideField("State");
 }
});
```

See also:

- JavaScript API: Control object > getValue()
- JavaScript API: Control object > getControl()

- JavaScript API: RunnerPage object > hideField()
- RunnerPage class: showField()
- RunnerPage class: hideField()
- Show dropdown list of US states if US was selected in country list
- JavaScript API: RunnerPage object
- JavaScript API

getTabs

Returns the **RunnerTabs object**, which represents a single tab group.

Syntax

```
getTabs( [N] );
```

Arguments

N

a zero-based index of a tab group. If no argument specified, returns first tab group of the page.

Return value

Returns the **RunnerTabs object**.

If **N** is more than the number of tab groups, the method returns *null*.

Example

Activate the third tab in the first tab group.

```
var tabs = pageObj.getTabs(0);
tabs.activate(2);
```

See also:

- Tabs API: activate
- Sections API: getSection
- Grid Row JavaScript API: row.getKeys()
- RunnerPage object
- Tabs/Sections API
- JavaScript API: RunnerPage object
- JavaScript API

toggleItem()

Shows or hides a page element in JavaScript events.

Syntax

```
toggleItem( itemId, show, recordId );
```

Arguments

`itemId`

the element ID. You can use this method only with the items (elements) that have an ID. Learn how to get it in the Inserting button article.

`show`

use *true* to show the item, and *false* to hide it.

`recordId`

an optional {*integer*}. A unique identifier of a record in a data grid on the **List** page. With this parameter, you can show/hide elements in specific grid rows.

> **Note**: use the recordId() function in the GridRow object to obtain this parameter.

Return value

No return value.

Example

```
//show the login form
pageObj.toggleItem("loginform_login", true );

//or

//hide the login form
pageObj.toggleItem("loginform_login", false );
```

See also:

- RunnerPage class: hideItem()
- RunnerPage class: showItem()
- JavaScript API: RunnerPage object > hideField()
- JavaScript API: RunnerPage object > showField()
- AJAX helper object
- Click action
- Field events
- JavaScript API: RunnerPage object
- JavaScript API

getAllRecords()

Returns an array of GridRow objects for all the records in the data grid on the **List** page.

Syntax

```
getAllRecords();
```

Arguments

No arguments

Return value

Returns an array of GridRow objects for all the records in the data grid on the **List** page.

Example

Let's say you need to email the data stored in your Database as a PDF file.

You can create a Custom button to email all table records on the page as a PDF file.

This example shows how to do it.

Client before:

```
var pdfParams = {},
  allRecords = pageObj.getAllRecords();
if( allRecords.length == 0 )
  return false;
params.recordCount = allRecords.length;
params.filenames = [];
var createOnePdf = function( idx ) {
  ajax.addPDF( 'pdf'+idx, pdfParams[idx], function() {
     delete pdfParams[idx];
    if( Object.keys( pdfParams ).length == 0 ) {
        showDialog();
     }
  });
}
var showDialog = function() {
 ctrl.dialog( {
  title: 'Email ' + params.recordCount + ' PDF files',
  fields: [
     {
        name: 'email',
        value: 'email@address.com'
     },
     {
        name: 'subject',
        value: 'Check out this data'
     },
     {
        name: 'body',
        value: 'This email is generated by Runner-created application',
        type: 'textarea'
     },
     ]
  });
}
allRecords.forEach( function( ajaxRow, idx ) {
  pdfParams[ idx ] = {
     pageType: 'view',
```

```
      records: ajaxRow.getKeys()
    };
    params.filenames.push( '_viewpage_id'+ajaxRow.getKeys()+'.pdf');
    createOnePdf( idx );
  });
  return false;
```

For the Server and the Client after events, use the code from the "How to email selected records as separate PDF files" article.

See also:

- AJAX helper object: ajax.addPDF
- Dialog API
- PDF API
- PDF Parameters
- Events: JavaScript onload
- Click actions
- AJAX-based functionality
- AJAX helper object
- JavaScript API: RunnerPage object > getSelectedRecords()
- JavaScript API: RunnerPage object > getSelectedRecordKeys()
- Show order total on the Edit page as the details table is updated
- JavaScript API: RunnerPage object
- JavaScript API

getSelectedRecords()

Returns an array of GridRow objects for all selected records in the data grid on the **List** page.

> **Note**: The selected records are the ones that have the checkbox next to them selected.

Syntax

```
getSelectedRecords();
```

Arguments

No arguments

Return value

Returns an array of GridRow objects for all selected records in the data grid on the **List** page.

Example

Let us assume you would like to highlight some important information on the web page.

Changing the look of the records may come in handy in this scenario.

In this example, we create a Custom button that allows us to change the background color of the selected records.

Client before:

```
var selectedRecords = pageObj.getSelectedRecords();
if( selectedRecords.length == 0 )
  return false;
selectedRecords.forEach( function( ajaxRow, idx ) {
   $("[data-record-id="+ajaxRow.row["id"]+"]").css("background-color","red");
});

return false;
```

See also:

- JavaScript API: RunnerPage object > getSelectedRecordKeys()
- Button object: getNextSelectedRecord()
- Select multiple values from checkboxes or a list field and have them appear as individual database entries
- Click actions
- AJAX-based functionality
- AJAX helper object

- Events: JavaScript onload
- Customizing CSS examples
- JavaScript API: RunnerPage object
- JavaScript API

getSelectedRecordKeys()

Returns an array of key field values for all the selected records.

> **Note**: The selected records are the ones that have the checkbox next to them selected.

Syntax

```
getSelectedRecordKeys();
```

Arguments

No arguments

Return value

The return value is an array of arrays even if there is only one key field in the table.

> **Note**: The key field values obtained this way can be used as is in the selection parameter of Runner.PDF.open or Runner.PDF.download functions.

Output

```
[ [1], [2], [10] ]
```

Example

You may need to clean all the **Details table** records corresponding to one of the **Master table** record.

For this purpose, you can insert a Custom button into the Master table **List** page.

Client before:

```
params["keys"] = pageObj.getSelectedRecordKeys();
```

Server:

```
foreach($params["keys"] as $key){
   DB::Exec("delete from carsmodels where make=".$key["id"]);
```

Note: In this example, `carsmodels` is a **Details table** name; `make` is the name of the **Master-details** link field.

Client after:

```
location.reload();
```

See also:

- Database API: Exec()
- JavaScript API: RunnerPage object > getSelectedRecords()
- JavaScript API: RunnerPage object > getAllRecords()
- Grid Row JavaScript API: row.getKeys()
- Button object: getNextSelectedRecord()
- Select multiple values from checkboxes or a list field and have them appear as individual database entries
- About PDF API
- How to create PDF from all selected records except the first one
- Key columns

- Events: JavaScript onload
- JavaScript API: RunnerPage object
- JavaScript API

getMasterPage()

Returns the RunnerPage object representing the master page.

When the *Details preview* is shown on the masters **List**, **Add**, **Edit** or **View** page, this function can be called from the Details **List** Page JavaScript events to access the master page.

Returns *null* when there is no master page available, i.e., when the Details Page is displayed in the standalone mode.

Syntax

```
getMasterPage();
```

Arguments

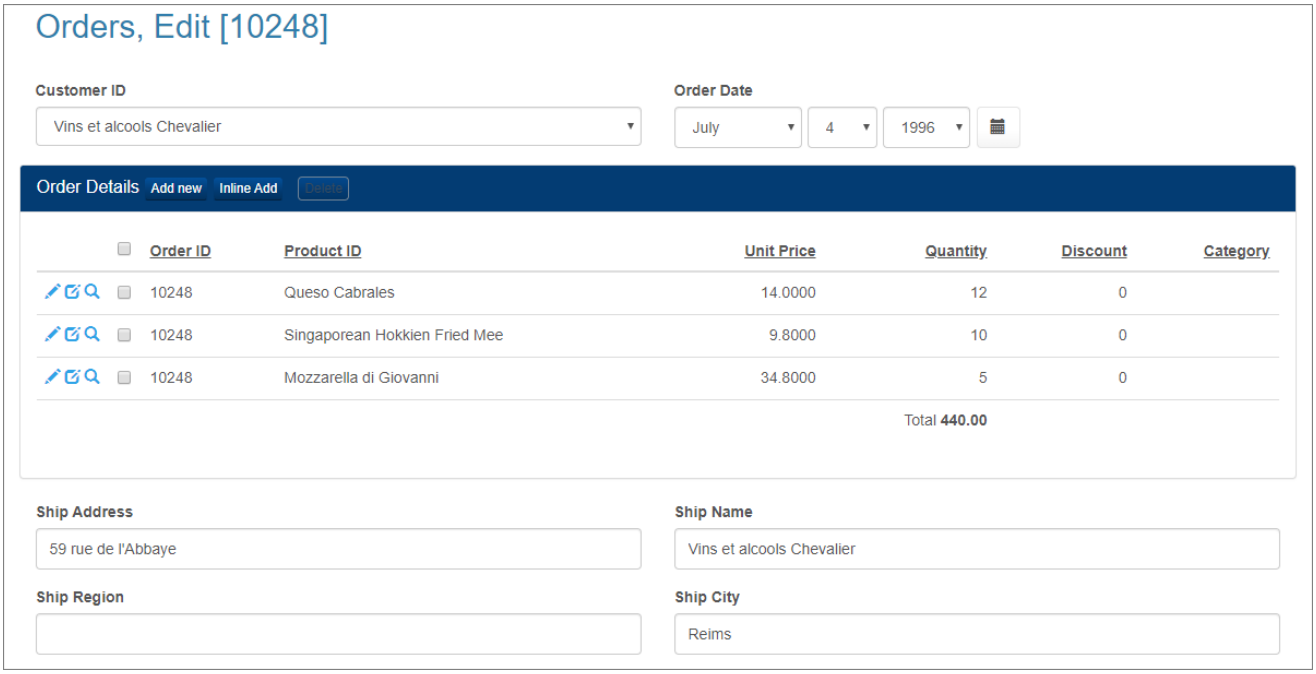
No arguments.

Return value

Returns the RunnerPage object representing the master page. Returns *null* when there is no master page available, i.e., when the Details Page is displayed in the standalone mode.

Example

Let's say we have *Orders* and *Order Details* tables. The *Order Details* is shown on the *Orders* - **Edit** page.



To hide the **Inline Edit** control for the master table when viewing the details table from the master page, add the following code to the *Order Details* -> **List** page -> Javascript: OnLoad event:

```
var master = pageObj.getMasterPage();
if( master ) {
  master.hideItem( 'integrated_edit_field' );
}
```

See also:

- RunnerPage class: hideItem()
- RunnerPage class: getMasterRecord()
- Setting up the master-details relationship between tables
- Master-details relationship between tables: Display details data on View/Add/Edit pages
- List page settings / Click actions
- Show data from a master table on the details view/edit/add page
- Show order total on the Edit page as the details table is updated
- AJAX-based functionality. Details page records preview
- Events: JavaScript onload

- JavaScript API: RunnerPage object
- JavaScript API

getDetailsPage(name, recordId)

Returns the **RunnerPage object** of the Details preview displayed on the page.

Syntax

```
getDetailsPage( name, recordId );
```

Arguments

`name {string}`

the exact name of the Details table. Case sensitive.

`recordId {integer}`

an optional {*integer*}. A unique identifier of a record in a data grid on the **List** page.

> **Note**: use the recordId() function in the GridRow object to obtain this parameter.

Return value

Returns the **RunnerPage object** of the Details preview displayed on the page.

Example

Hide the **Add** button for the *order details* table preview.

```
var details = pageObj.getDetailsPage( 'order details' );
details.hideItem( 'add' );
```

See also:

- RunnerPage class: hideItem()
- RunnerPage class: getMasterRecord()

- Setting up the master-details relationship between tables
- List page settings / Click actions
- Show data from a master table on the details view/edit/add page
- Show order total on the Edit page as the details table is updated
- AJAX-based functionality: Details page records preview
- JavaScript onload events
- JavaScript API: RunnerPage object
- JavaScript API

getDetailsPages(recordId)

Returns the **RunnerPage objects** for all the Details previews on the current page.

Syntax

```
getDetailsPages(recordId);
```

Arguments

`recordId {integer}`

an optional *{integer}*. A unique identifier of a record in a data grid on the **List** page.

> **Note**: use the recordId() function in the GridRow object to obtain this parameter.

Return value

The return value is an object with table names as keys and **RunnerPage objects** as values.

Example

Hide the **Add** button for every Details preview.

```
var details = pageObj.getDetailsPages();
details[ 'order details' ].hideItem( 'add' );
```

See also:

- RunnerPage class: hideItem()
- RunnerPage class: getMasterRecord()
- Setting up the master-details relationship between tables
- List page settings / Click actions
- Show data from a master table on the details view/edit/add page
- AJAX-based functionality: Details page records preview
- JavaScript onload events
- JavaScript API: RunnerPage object
- JavaScript API

getItemButton(itemId, recordId)

Returns the jQuery object that represents the **Button object**. Can be used with the standard and custom buttons the same way.

Syntax

```
getItemButton( itemId, recordId );
```

Arguments

`itemId`

the element ID. You can use this method only with the items (elements) that have an ID. Learn how to get it in the Inserting button article.

`recordId {integer}`

an optional {*integer*}. A unique identifier of a record in a data grid on the **List** page.

> **Note**: use the recordId() function in the GridRow object to obtain this parameter.

Return value

Returns the jQuery object that represents the **Button object**.

Examples

```
// disable the button
pageObj.getItemButton('custom_button').addClass( 'disabled' );
```

```
// re-enable the button
pageObj.getItemButton('custom_button').removeClass( 'disabled' );
```

```
// imitate clicking the button
pageObj.getItemButton('custom_button').click();
```

See also:

- Page Designer: Insert custom button
- Button object
- JavaScript API: AJAX helper object > setDisabled()
- JavaScript API: AJAX helper object > setEnabled()
- Events: JavaScript onload
- Troubleshooting custom buttons
- JavaScript API: RunnerPage object
- JavaScript API

findItem(itemId, recordId)

Returns the jQuery object representing the page element. Use this function when you need to apply code to one of the page elements.

Syntax

```
findItem( itemId, recordId );
```

Arguments

itemId

the element ID. You can use this method only with the items (elements) that have an ID. Learn how to get it in the Inserting button article.

`recordId`

an optional {*integer*}. A unique identifier of a record in a data grid on the **List** page.

> **Note**: use the recordId() function in the GridRow object to obtain this parameter.

Return value

Returns the jQuery object representing the page element.

Examples

Example 1

```
// set the text of a 'total' element
pageObj.findItem( 'total' ).text( '42' );
```

Example 2

```
// set the background of a  'simple_grid_field' element
pageObj.findItem( 'simple_grid_field' ).css( 'background', 'red' );
```

> **Note**: use the `recordId` parameter for the data grid elements on the **List** page.

Example 3

For this example, let's say you want a field in the grid to change its background when a user clicks the row.

Proceed to the Choose Pages screen -> Click actions -> **Run AJAX snippet** and add this code to the Client before event:

```
pageObj.findItem( 'simple_grid_field', row.id() ).css( 'background', 'yellow' );
```

> **Note**: If you omit the `recordId` parameter, the background is applied to the field value in all grid rows instead.

Example 4

```
// change the background of the first record's custom button
var allRecords = pageObj.getAllRecords();
pageObj.findItem( 'custom_button',
allRecords[0].recordId() ).find('a').css( 'background', 'red' );
```

See also:

- JavaScript API: RunnerPage object > getAllRecords()
- Grid Row JavaScript API: recordId()
- Tri-part events
- Page designer: Insert custom button
- Page Designer
- Choose pages screen
- Click actions
- Show order total on the Edit page as the details table is updated
- JavaScript API: RunnerPage object
- JavaScript API

getCurrentTabId()

Returns the id of the current tab. This function can be used on any **List**, **Print** and **Export** page events.

Syntax

```
getCurrentTabId()
```

Arguments

No arguments.

Return value

Returns the id of the current tab.

Example

You can hide the **Delete** button on the "main" tab using this code:

```
if( $pageObject->getCurrentTabId() == "main" ) {
$pageObject->hideItem("delete");
}
```

See also:

- RunnerPage class: hideItem()
- JavaScript API: RunnerPage object > setCurrentTabId(tabId)
- Additional WHERE tabs
- Additional WHERE tabs API
- JavaScript API: RunnerPage object
- JavaScript API

setCurrentTabId(tabId)

Selects a tab to display. This event can be used in the Before process events on the **List**, **Print** and **Export** pages.

Syntax

```
setCurrentTabId($tabId);
```

Arguments

`$tabId`

the id of the tab.

Return value

No return value.

Example

Select the "*main*" tab to be displayed:

```
$pageObject->setCurrentTabId("main");
```

See also:

- JavaScript API: RunnerPage object > getCurrentTabId()
- Additional WHERE tabs
- Additional WHERE tabs API
- JavaScript API: RunnerPage object
- JavaScript API

3.2.6.7 SearchController object

About SearchController object (deprecated)

> **Deprecated**
>
> SearchController object is deprecated, we recommend using Search API instead.

SearchController object allows you to manage the search panel. It is available in the JavaScript OnLoad event for the **List**/**Report**/**Chart** pages (pages with a Search panel).

Use the **getSearchController()** method to get the search panel object:

```
var srch = pageObj.getSearchController();
```

Methods

Method	Description
addField()	Adds a field to the search panel and calls the callback function.
clear()	Deletes all fields from the search panel.
deleteField()	Deletes a field from the search panel.
display()	Hides or shows the search panel, shows the search panel as a popup.
getSearchFields()	Returns search fields related to all fields or only the specified one.
toggleCriteria()	Hides or shows the criteria block on the search panel, sets values of the search criteria.
toggleOptions()	Hides or shows options on the search panel.

See also:

- About Search API
- JavaScript API: SearchField object
- JavaScript API

Methods

addField

> **Deprecated**
>
> SearchController object methods are deprecated, we recommend using Search API instead.

Adds a field to the search panel. Then calls the callback function, passing the **SearchField object** of an added field as the argument. In the callback function, you can use all the functionality available for the SearchField object.

Syntax

```
srch.addField(fieldName, callback(field))
```

Arguments

`fieldName`

the name of the field. Example: *'Make'*.

`callback(field)`

a call to a callback function. **`field`** is the **SearchField object** of an added field. This argument may be omitted.

Return value

No return value.

Example 1

Add the *Make* field to the search panel using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();
srch.addField('Make');
```

Example 2

Add the *Price* field to the search panel and set search parameters using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();

srch.addField('Price', function( field ) {
    field.getControl().setValue("20000");
    field.setOption(Runner.search.options.LESS_THAN);
});
```

Example 3

Add five *Make* fields to the search panel using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();
var field = srch.getSearchFields('Make');
var count = field.length;

for (i=count; i<5; i++)
    srch.addField('Make');
```

See also:

- JavaScript API: Control object > setValue()
- JavaScript API: SearchField object > setOption()
- JavaScript API: SearchField object
- About Search API
- Events: JavaScript OnLoad
- JavaScript API: SearchController object (deprecated)
- JavaScript API

clear

Deprecated

SearchController object methods are deprecated, we recommend using Search API instead.

Deletes all search fields from the search panel.

Syntax

```
srch.clear();
```

Arguments

No arguments.

Return value

No return value.

See also:

- About Search API
- JavaScript API: SearchField object
- Events: JavaScript OnLoad
- JavaScript API: SearchController object (deprecated)
- JavaScript API

deleteField

> **Deprecated**
>
> SearchController object methods are deprecated, we recommend using Search API instead.

Deletes all entries of specified field from the search panel.

Syntax

```
srch.deleteField(fieldName);
```

Arguments

`fieldName`

the name of the field. Example: *'Make'*.

Return value

No return value.

Example

Removes the *Make* field from the search panel using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();
srch.deleteField('Make');
```

See also:

- JavaScript API: SearchField object
- About Search API
- Events: JavaScript OnLoad
- JavaScript API: SearchController object (deprecated)
- JavaScript API

display

> **Deprecated**
>
> SearchController object methods are deprecated, we recommend using Search API instead.

Hides or shows the search panel, shows search panel as popup.

Syntax

```
srch.display(value);
```

Arguments

value - one of the values listed below:

Value	Description
hide	Hides the search panel.
popup	Shows the search panel as a popup.
show	Shows the search panel.

Return value

No return value.

Example

Show the search panel using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();
srch.display("show");
```

See also:

- About Search API
- JavaScript API: SearchField object
- Events: JavaScript OnLoad
- JavaScript API: SearchController object (deprecated)
- JavaScript API

getSearchFields

> **Deprecated**
>
> SearchController object methods are deprecated, we recommend using Search API instead.

Returns search fields related to all fields or only the specified one.

Syntax

```
var fields = srch.getSearchFields(fieldName);
```

Arguments

`fieldName`

the name of the field. Example: *'Make'*. This argument may be omitted.

Return value

If the argument is specified: returns the array of search fields related to the specified field.

If the argument is not specified: returns the array of search fields related to all fields.

Example

Get an array of SearchField objects for the *Make* field using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();
var field = srch.getSearchFields('Make');
for(var i=0; i<field.length; i++){
  // do something with SearchField objects
}
```

See also:

- JavaScript API: SearchField object
- About Search API
- Events: JavaScript OnLoad
- JavaScript API: SearchController object (deprecated)
- JavaScript API

toggleCriteria

> **Deprecated**
>
> SearchController object methods are deprecated, we recommend using Search API instead.

Hides or shows the criteria block on the search panel, sets the values of the search criteria.

Syntax

```
srch.toggleCriteria(value);
```

Arguments

value

one of the values listed below:

Value	Description
all	Sets the search criteria to the "all" value.
any	Sets the search criteria to the "any" value.
hide	Hides the criteria block.
show	Shows the criteria block.

Return value

No return value.

Example

Show the criteria block using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();
srch.toggleCriteria('show');
```

See also:

- About Search API
- JavaScript API: SearchField object
- Events: JavaScript OnLoad
- JavaScript API: SearchController object (deprecated)
- JavaScript API

toggleOptions

> **Deprecated**
>
> SearchController object methods are deprecated, we recommend using Search API instead.

Hides or shows options on the search panel.

Syntax

```
srch.toggleOptions(value);
```

Arguments

`value`

one of the values listed below:

Value	Description
hide	Hides the options on the search panel.
show	Shows the options on the search panel.

Return value

No return value.

Example

Show options on the search panel using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();
srch.toggleOptions('show');
```

See also:

- About Search API
- JavaScript API: SearchField object
- Events: JavaScript OnLoad
- JavaScript API: SearchController object (deprecated)
- JavaScript API

3.2.6.8 SearchField object

About SearchField object

The **SearchField object** allows managing search fields on the search panel. It is used along with SearchController object.

SearchField object is available in the JavaScript OnLoad event for the **List**/**Report**/**Chart** pages (pages with a Search panel) and an Advanced Search page.

Use the **getSearchFields()** method to get the array of search fields:

```
var srch = pageObj.getSearchController();
var fields = srch.getSearchFields();
```

Methods

Method	Description
addOption()	Adds the option to the list of search options.
getControl()	Returns the control object of the search field.
getName()	Returns the field name.
getOption()	Returns the option for the search field that is currently selected.
getOptions()	Returns the list of all options that are currently available in the search panel.
getSecondControl()	Returns the second control object of the search field.
remove()	Removes the field from the search panel.
removeOption()	Removes the option from the list of search options.
setOption()	Selects the option for the search field.

Search options

- BETWEEN
- CONTAINS
- EMPTY
- EQUALS
- LESS_THAN
- MORE_THAN
- STARTS_WITH
- NOT_BETWEEN
- NOT_CONTAINS
- NOT_EMPTY
- NOT_EQUALS
- NOT_LESS_THAN
- NOT_MORE_THAN
- NOT_STARTS_WITH

See also:

- JavaScript API: SearchController object (deprecated)
- About Search API
- Events: JavaScript OnLoad
- JavaScript API

Methods

addOption

Adds an option to the list of search options in the search panel (CONTAINS, EQUALS, etc.).

Syntax

```
field.addOption(option);
```

Arguments

`option`

one of the available search options with the `PHPRunner.search.options` prefix. Example: *PHPRunner.search.options.MORE_THAN, PHPRunner.search.options.NOT_EMPTY.*

Return value

No return value.

Example

Add the EMPTY option to the list of search options using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();
var field = srch.getSearchFields()[0];
field.addOption(PHPRunner.search.options.EMPTY);
```

See also:

- JavaScript API: SearchController object (deprecated)
- About Search API
- Events: JavaScript OnLoad
- JavaScript API: SearchField object
- JavaScript API

getControl

Returns the control object of the search field.

Syntax

```
var ctrl = field.getControl();
```

Arguments

No arguments.

Return value

Returns the control object.

Example

Get the control object of the search field using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();
var field = srch.getSearchFields()[0];
var ctrl = field.getControl();
```

See also:

- JavaScript API: SearchController object (deprecated)
- About Search API
- JavaScript API: SearchField object

- Events: JavaScript OnLoad
- JavaScript API

getName

Returns the name of the field.

Syntax

```
var fName = field.getName();
```

Arguments

No arguments.

Return value

Returns name of the current field.

Example

Get the field name of the search field using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();
var field = srch.getSearchFields()[0];
var fName = field.getName();
```

See also:

- JavaScript API: SearchController object (deprecated)
- About Search API
- Events: JavaScript OnLoad
- JavaScript API: SearchField object
- JavaScript API

getOption

Returns the option that is currently selected in the search panel (CONTAINS, EQUALS, etc.).

Syntax

```
var option = field.getOption();
```

Arguments

No arguments.

Return value

Returns the option that is currently selected in the search panel. See the search options list to learn more.

Example

Get the search option of the search field using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();
var field = srch.getSearchFields()[0];
var option = field.getOption();
```

See also:

- JavaScript API: SearchController object (deprecated)
- About Search API
- Events: JavaScript OnLoad
- JavaScript API: SearchField object
- JavaScript API

getOptions

Returns the list of search options that are currently available in the search panel (CONTAINS, EQUALS, etc.).

Syntax

```
var options = field.getOptions();
```

Arguments

No arguments.

Return value

Returns the array of available search options. See the search options list to learn more.

Example

Get the list of search options using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();
var field = srch.getSearchFields()[0];
var options = field.getOptions();
```

See also:

- JavaScript API: SearchController object (deprecated)
- About Search API
- Events: JavaScript OnLoad
- JavaScript API: SearchField object
- JavaScript API

getSecondControl

Returns the second control object of the search field (for the BETWEEN option).

Syntax

```
var ctrl = field.getSecondControl();
```

Arguments

No arguments.

Return value

Returns the second control object of the search field.

Example

Get the second control of the search field in the search panel using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();
var field = srch.getSearchFields()[0];
var ctrl = field.getSecondControl();
```

See also:

- JavaScript API: SearchController object (deprecated)
- About Search API
- Events: JavaScript OnLoad
- JavaScript API: SearchField object
- JavaScript API

remove

Removes a field from the search panel.

Syntax

```
field.remove();
```

Arguments

No arguments.

Return value

No return value.

Example

Remove the search field from the search panel using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();
var field = srch.getSearchFields()[0];
field.remove();
```

See also:

- JavaScript API: SearchController object (deprecated)
- About Search API
- Events: JavaScript OnLoad
- JavaScript API: SearchField object
- JavaScript API

removeOption

Removes an option from the list of search options in the search panel (CONTAINS, EQUALS, etc.).

Syntax

```
field.removeOption(option);
```

Arguments

`option`

one of the available search options with the *PHPRunner.search.options* prefix. Example: *PHPRunner.search.options.MORE_THAN, PHPRunner.search.options.NOT_EMPTY*.

Return value

No return value.

Example

Remove the EQUALS option from the list of search options using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();
var field = srch.getSearchFields()[0];
field.removeOption(Runner.search.options.EQUALS);
```

See also:

- JavaScript API: SearchController object (deprecated)
- About Search API
- Events: JavaScript OnLoad
- JavaScript API: SearchField object
- JavaScript API

setOption

Selects an option for a search field in the search panel (CONTAINS, EQUALS, etc.).

Syntax

```
field.setOption(option);
```

Arguments

`option`

one of the available search options with the *PHPRunner.search.options* prefix . Example: *PHPRunner.search.options.MORE_THAN, PHPRunner.search.options.NOT_EMPTY*.

Return value

No return value.

Example

Select the BETWEEN option for a search field using the JavaScript OnLoad event:

```
var srch = pageObj.getSearchController();
var field = srch.getSearchFields()[0];
field.setOption(Runner.search.options.BETWEEN);
```

See also:

- JavaScript API: SearchController object (deprecated)
- About Search API
- Events: JavaScript OnLoad
- JavaScript API: SearchField object
- JavaScript API

3.2.6.9 Examples

How to ask for confirmation before saving record

To ask a user for a confirmation before saving a record, use the following code in the JavaScript OnLoad event for the **Edit** or **Add** page.

> **Note**: Change `red` values to match your project.

```
this.on('beforeSave', function(formObj, fieldControlsArr, pageObj){
if (confirm('Save record?')){
return true;
}else{
Runner.delDisabledClass ( pageObj.saveButton );
return false;
```

```
}
});
```

See also:

- JavaScript API: RunnerPage object
- About Dialog API
- Event: Before record added
- Events: Before record updated
- Events: JavaScript OnLoad
- JavaScript API

How to calculate values on the fly

Let's say there are three fields on the **Add**/**Edit** page: *Price*, *Quantity* and *Total*. To calculate the total value on the fly, use the following JavaScript code (add it to the Add page: JavaScript OnLoad event or Edit page: JavaScript OnLoad event on the Events tab).

Note: Change `red` values to match your project.

```
var ctrlPrice = Runner.getControl(pageid, 'Price');
var ctrlQuantity = Runner.getControl(pageid, 'Quantity');
var ctrlTotal = Runner.getControl(pageid, 'Total');

function func() {
   ctrlTotal.setValue(Number(ctrlPrice.getValue()) *
Number(ctrlQuantity.getValue()));
};

ctrlPrice.on('keyup', func);
ctrlQuantity.on('keyup', func);
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > setValue()
- JavaScript API: Control object > getValue()

- JavaScript API: Control object
- Field events
- JavaScript API

How to change font size in text box

To change the font size in all text boxes placed on a page, use the following code in the JavaScript OnLoad event.

> **Note**: Change `red` values to match your project.

```
$("input[type=text]").css('fontSize', '120%');
```

See also:

- Customizing CSS
- Rich Text Editor plugins
- "Edit as" settings: Text field
- "Edit as" settings: Text area
- JavaScript API

How to change font in "edit" controls

You can change the font in the "edit" controls in the following ways:

> **Note**: Change `red` values to match your project.

1. Change the font in one "edit" control

Use the following code in the JavaScript OnLoad event:

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.addStyle('font-size: 12px; color: red;');
```

To get more information about using JavaScript API, see JavaScript API.

2. Change the font for all "edit" controls on one page

Proceed to the Editor screen, open the page you need to modify, click on the **Custom CSS** button and add the following code there:

```
input,textarea {
color:#003333;
font-family:Verdana,Arial,SunSans-Regular,Sans-Serif;
font-size:12pt;
}
```

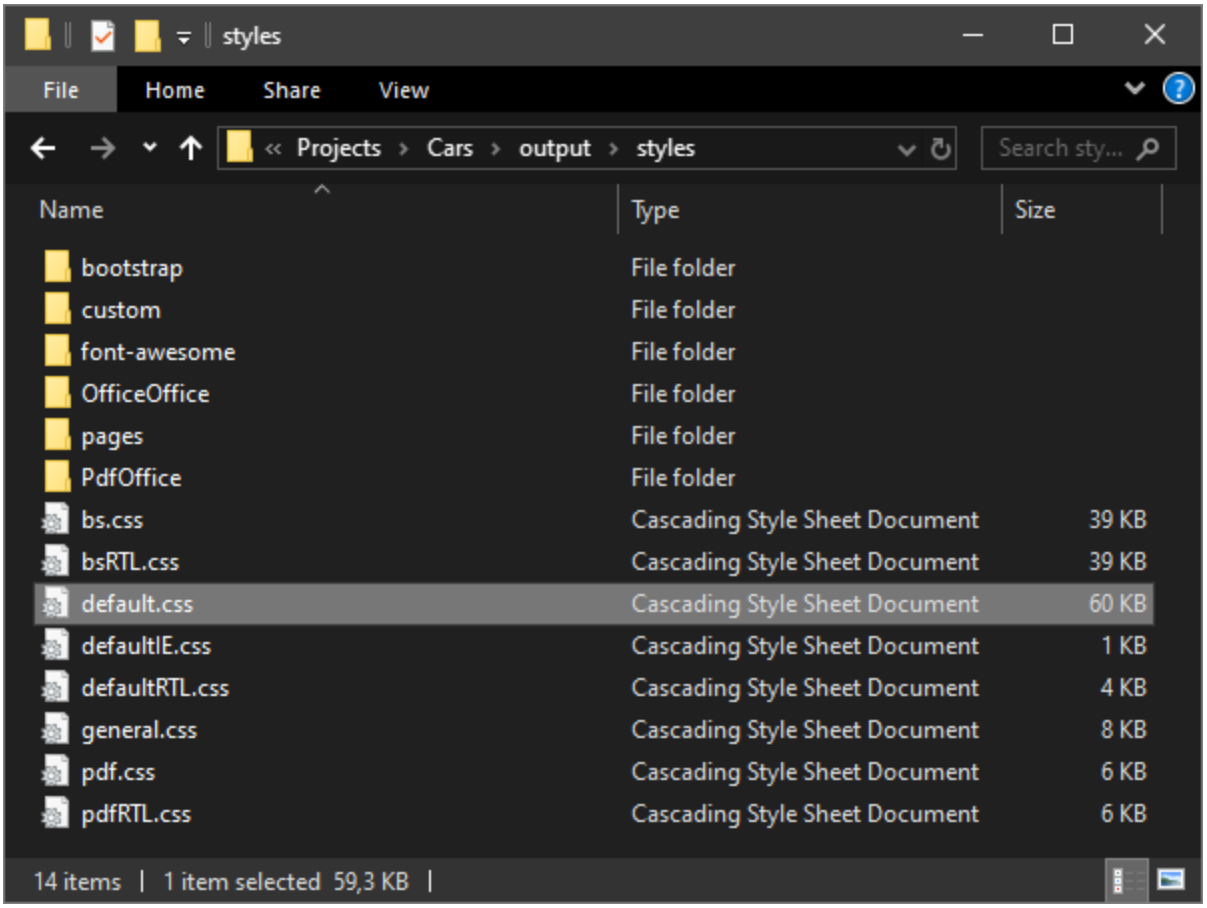
Note: You can also add this code on the Page Designer screen. Click the **Custom CSS** button to do so.

or use the following code in the JavaScript OnLoad event:

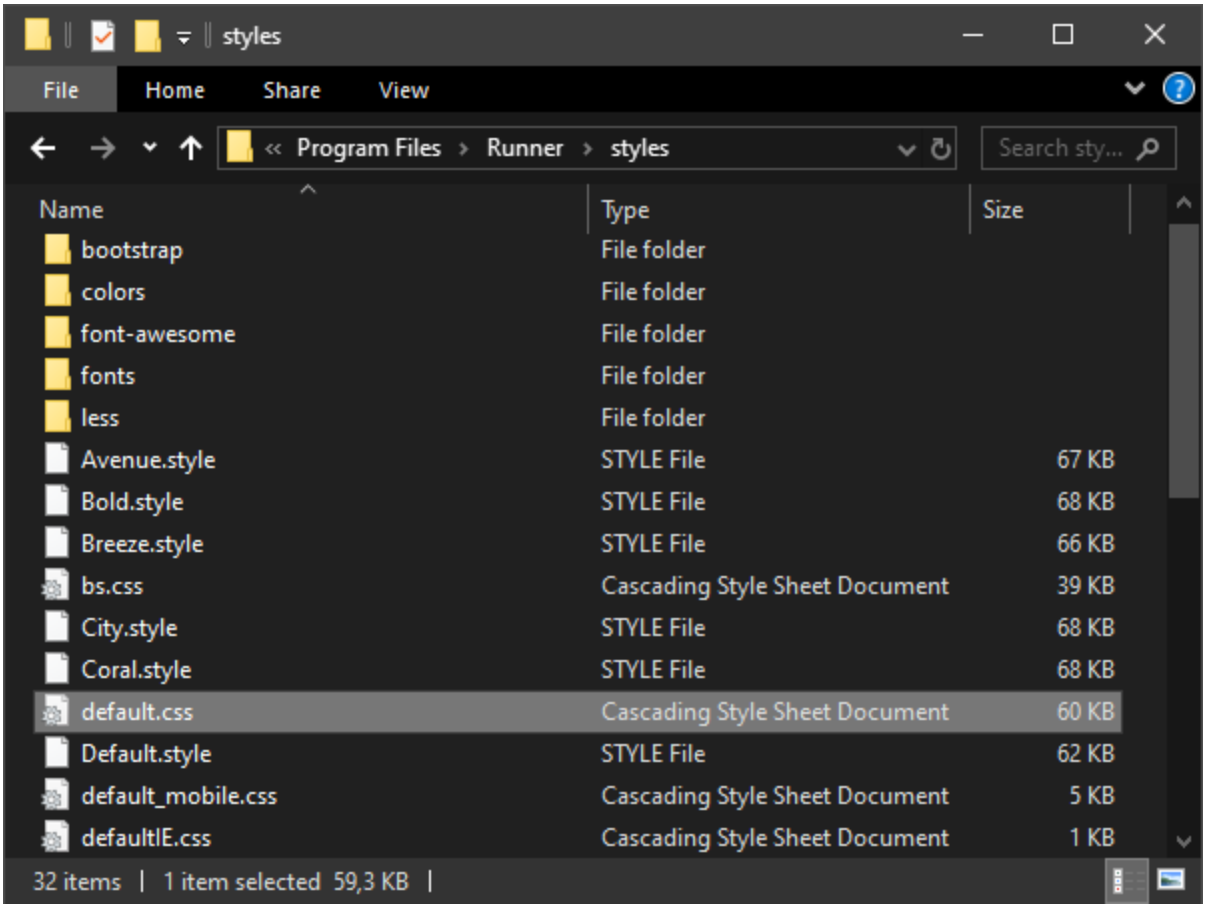
```
$("input, textarea").css( {fontSize: '120%', color: 'red'} );
```

3. Change the font for all "edit" controls in the project

Change the description of the control style (INPUT, TEXTAREA) in the *styles\default.css* file in the folder of the generated project:



Alternatively, you can change the description of the control style directly in the PHPRunner install folder - the *\PHPRunner10.3\styles\default.css*:



See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > add CSS Class
- Customizing CSS examples
- Conditional formatting
- JavaScript API: Control object
- JavaScript API

How to change width of edit box with AJAX popup

To change the width of an edit box with an AJAX popup, use the following code in the JavaScript OnLoad event.

> **Note**: Change `red` values to match your project.

```
var ctrl = Runner.getControl(pageid, 'Make');
ctrl.getDispElem().css("width", "200px");
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > getDispElem()
- Customizing CSS
- AJAX-based Functionality
- AJAX helper object
- How to create a custom Edit control plugin
- JavaScript API

How to change width of text field on Quick Search panel

To change the width of a text field on the **Quick Search** panel, use the following code in the JavaScript OnLoad event.

> **Note**: Change `red` values to match your project.

```
$("input[name^='value_make'], select[name^='value_make']").width(150);
```

See also:

- Customizing CSS
- JavaScript API: SearchField object
- About Search API
- JavaScript API

How to control the Inline Add/Edit functionality from script

Controlling the Inline Add/Edit functionality from script

To control the **Inline Add**/**Edit** functionality from script, add one of the following code snippets to JavaScript OnLoad event of the **List** page:

> **Note**: Change `red` values to match your project.

1. Simulate an "inline add" click:

```
pageObj.inlineAdd.inlineAdd();
```

2. Inline edit all records:

```
pageObj.inlineEdit.editAllRecs();
```

3. Inline edit the record number X, where X ranges from 1 to the number of records on the page:

```
// get the list of record IDs
var recsId = pageObj.inlineEdit.getRecsId();

// click the inline edit link
pageObj.inlineEdit.editRecById(recsId[X]);
```

4. Open the details table inline preview:

```
// get the list of record IDs
var recsId = pageObj.inlineEdit.getRecsId();

// use the actual details table name
var dTableName = 'DetailsTableName';
var dp = this.dpObjs[dTableName];
var row = dp.getRowById( recsId[X] );
dp.openDetailsTabs( row, $(null) );
```

5. Edit the record number X in a popup (when the 'Edit in popup' mode is turned on):

```
// get the list of record IDs
var recsId = pageObj.inlineEdit.getRecsId();

// click the Edit link
$("#editLink"+recsId[X]).click();
```

6. View the record number X in a popup (when the 'View in popup' mode is turned on):

```
// get the list of record IDs
var recsId = pageObj.inlineEdit.getRecsId();

// click the View link
$("#viewLink"+recsId[X]).click();
```

7. Open the Add page in a popup:

```
$("#addButton"+pageid).click();
```

8. Make new records appear at the end of the list (when an Inline or 'Add in popup' mode is enabled):

```
pageObj.addNewRecordsToBottom = true;
```

How to execute JavaScript code after the Inline Add or Edit

To refresh the **List** page when inline adding or editing is finished, use the following code in the JavaScript OnLoad event of the **List** page. This code can also work when **Add** or **Edit** pages are displayed in a popup.

1. Inline Add:

```
this.on('afterInlineAdd', function( fieldsData ) {
 location.reload();
} )
```

2. Inline Edit:

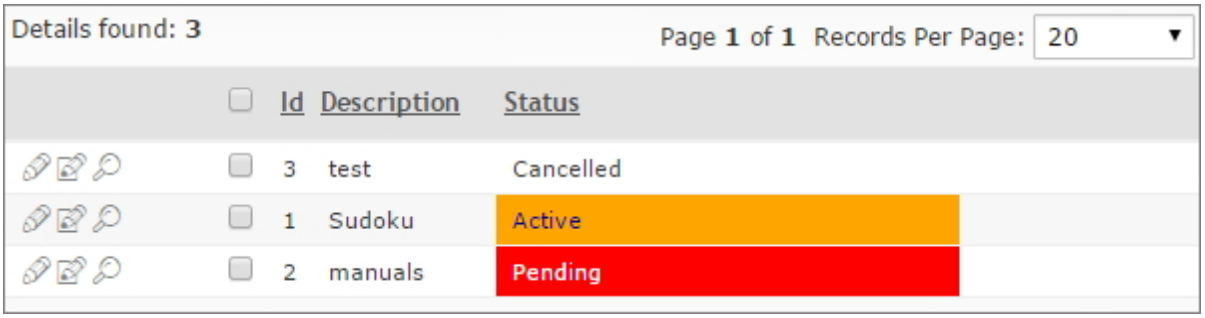
```
this.on('afterInlineEdit', function( fieldsData ) {
        location.reload();
} )
```

Note: here is a description of the **fieldsData** structure:

```
fieldsData = {
 name: field name,
 value: raw field value as it appears in the database,
 text: field value with formatting applied if any,
 container:  jQuery object, a container, where field value will be inserted
}
```

3. How to change the text color and background of a field after the Inline Add or Edit:

```
function funcAfter(fieldsData) {
    for (f in fieldsData) {
        var field = fieldsData[f];
        if (field.name == 'status') {
            if (field.value == 'Pending') {
                field.container.closest('td').css('background', 'red');
                field.container.closest('td').css('color', 'white');
            } else if (field.value == 'Active') {
                field.container.closest('td').css('background', 'orange');
                field.container.closest('td').css('color', 'darkblue');
            }
        }
    }

}

this.on('afterInlineEdit', funcAfter);
this.on('afterInlineAdd', funcAfter);
```



4. How to redirect a user to the View page after adding a new record in a popup:

Do not forget to replace "*documents*" with your table name.

```
function funcAfter(fieldsData) {
    for (f in fieldsData) {
        var field = fieldsData[f];
        if (field.name == 'id') {
            window.location.href = "documents_view.PHP?editid1=" + field.value;
        }
    }

}

this.on('afterInlineAdd', funcAfter);
```

See also:

- JavaScript API: Control object > on()
- JavaScript API: InlineRow object
- Grid Row JavaScript API: row.fieldCell()
- Predefined actions: Redirect to another page
- JavaScript API: RunnerPage object
- Choose pages screen
- Change width of edit box with AJAX popup
- How to display any page in a popup window
- JavaScript API

How to convert input into upper case

To convert user entered data into upper case, use the following code in the JavaScript OnLoad event.

> **Note**: Change red values to match your project.

```
var ctrl = Runner.getControl(pageid, 'FieldName');
ctrl.addStyle('text-transform: uppercase;');
```

See also:

- JavaScript API: Control object > addStyle()
- JavaScript API: Control object
- JavaScript API

How to display all Options on Search panel

To display all Options on the Search panel, use the following code in the List page: JavaScript OnLoad event.

> **Note**: Change red values to match your project.

```
$("[id^='showHideSearchType']").click();
```

See also:

- About Search API
- Events: JavaScript OnLoad
- JavaScript API

How to display any page in a popup window

You can use the **Runner.displayPopup()** function to display any page in a popup window

Description

Runner.displayPopup() is a JavaScript function. You can use it with any JavaScript event: e.g., the JavaScript OnLoad or ClientBefore event of a custom button.

The **Runner.displayPopup()** function has only one mandatory parameter: the URL of the page to be displayed.

Syntax

```
Runner.displayPopup();
```

Arguments

`url`

the URL of the page to be displayed. This is a mandatory parameter.

`html`

instead of specifying the URL, you can supply an HTML code to be displayed in a popup window.

`header`

to be displayed in a popup header section.

`footer`

to be displayed in a popup footer section.

`afterCreate`

the function to be called after the popup window is created.

`beforeClose`

the function to be called before the popup window is closed.

Return `false` to prevent popup from being closed. Return `true` to proceed with closing.

`width`

the popup width in pixels.

`height`

the popup height in pixels.

Return value

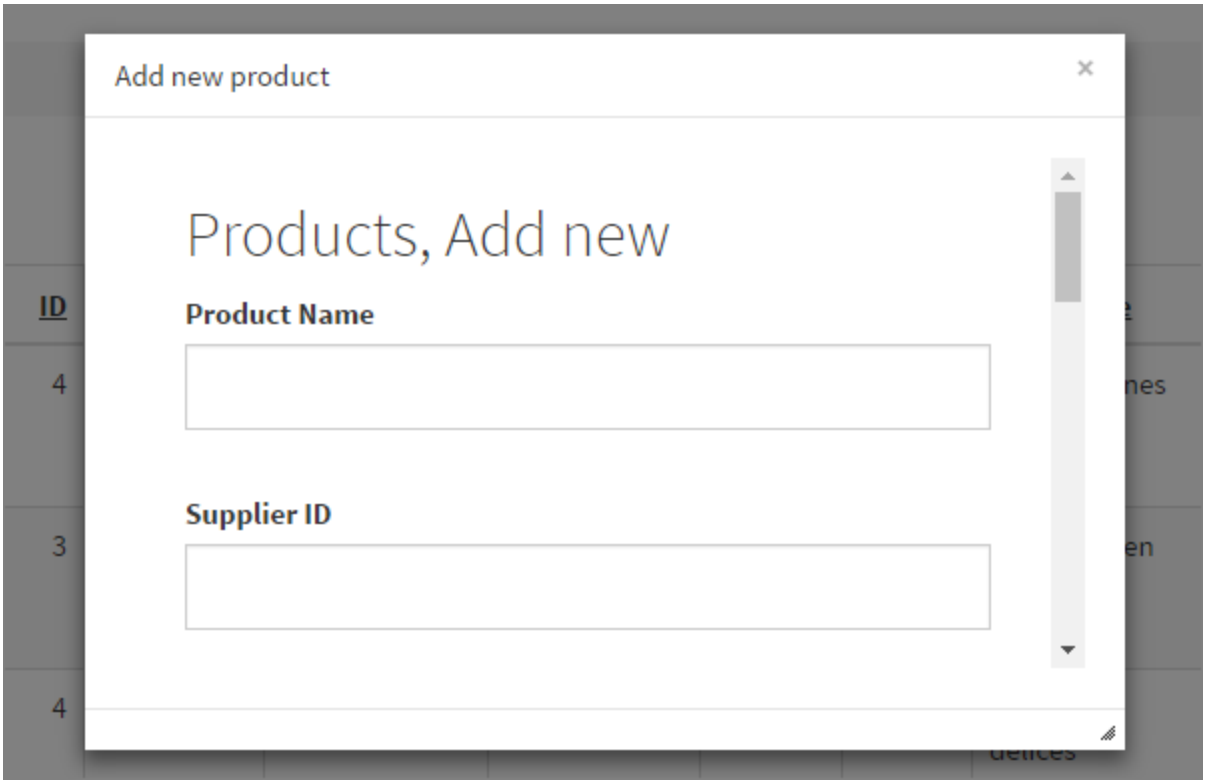
No return value.

Examples

Example 1

Here is how you can display the **Add** page of the *Products* table in a popup:

```
Runner.displayPopup( {
   url: "products_add.php"
});
```



Example 2

A popup window with its height and width defined:

```
var win = Runner.displayPopup( {
   url: "products_add.php",
   width: 700,
   height: 500,
   header: 'Add new product'
});
```

Example 3

Using HTML instead of URL:

```
var win = Runner.displayPopup( {
      html: "<h1>Hello world!</h1><p>It works</p>",
      header: 'Custom popup text'
});
```

Example 4

Using *close()*:

```
var win = Runner.displayPopup( {
   url: "products_add.php",
   width: 500,
   height: 300,
   header: 'Add new product'
});

alert('That was an example of popup window');

win.close();
```

Example 5

Using the *afterCreate()* function:

```
var win = Runner.displayPopup( {
   url: "products_add.php",
   header: 'Add new product',
   afterCreate: function(win) {
      win.setWidth(700);
```

```
      win.setHeight(500);
   }
});
```

Example 6

Add a *'Close window'* link to the footer:

```
var win = Runner.displayPopup( {
   url: "products_add.php",
   header: 'Add new product',
   footer: '<a href="#" onclick="window.win.close();return false;">Close window</a>',
   afterCreate: function(win) {
      window.win = win;
   }
});
```

Example 7

Using the *beforeClose()* function:

In this function, you can return false to prevent the window from being closed.

Do not allow closing the window if the *'Product Name'* field is empty:

```
var win = Runner.displayPopup( {
   url: "products_add.php",
   header: 'Add new product',
   footer: '<a href="#" onclick="window.win.close();return false;">Close window</a>',
   afterCreate: function(win) {
      window.win = win;
   },
   beforeClose: function(win) {
      if ($('iframe').contents().find("input#value_ProductName_1").val()=="")
         return false;
      else
         return true;
   }
});
```

Example 8

Show a **'View customer'** button on each row of the *Orders* **List** page.

Insert a button into the *Orders* **List** page grid.

Server

```
$record = $button->getCurrentRecord();
$result["CustomerID"] = $record["CustomerID"];
```

ClientAfter

```
var win = Runner.displayPopup( {
   url: "customers_view.php?editid1="+result["CustomerID"],
   width: 700,
   height: 500,
   header: 'View customer'
});
```

Example 9

Show the **Add** page in a popup, close popup on clicking **'Save'**, and then refresh the **List** page.

There is an added button to the **List** page that displays the **Add** page in a popup. Once the record is saved, we close the popup and refresh the **List** page to show the new record.

The **ClientBefore** code of the button:

```
window.popup = Runner.displayPopup({
    url: Runner.pages.getUrl("carsmake","add"),
    width: 700,
    height: 700,
    header: 'Add Category'
});
```

AfterAdd event

```
$pageObject->setProxyValue('saved', true);
$pageObject->stopPRG = true;
```

JavaScript OnLoad event of the **Add** page:

```
if ((proxy['saved']) && window.parent && window.parent.popup) {
    window.parent.location.reload();
    window.parent.popup.close();
}
```

If you need to close the popup without refreshing the page, comment the line:

```
window.parent.location.reload();
```

See also:

- Button object: getCurrentRecord()
- RunnerPage class: setProxyValue()
- Page Designer: Insert custom button
- Events: JavaScript OnLoad
- Events: AfterAdd
- Tri-part events
- JavaScript API

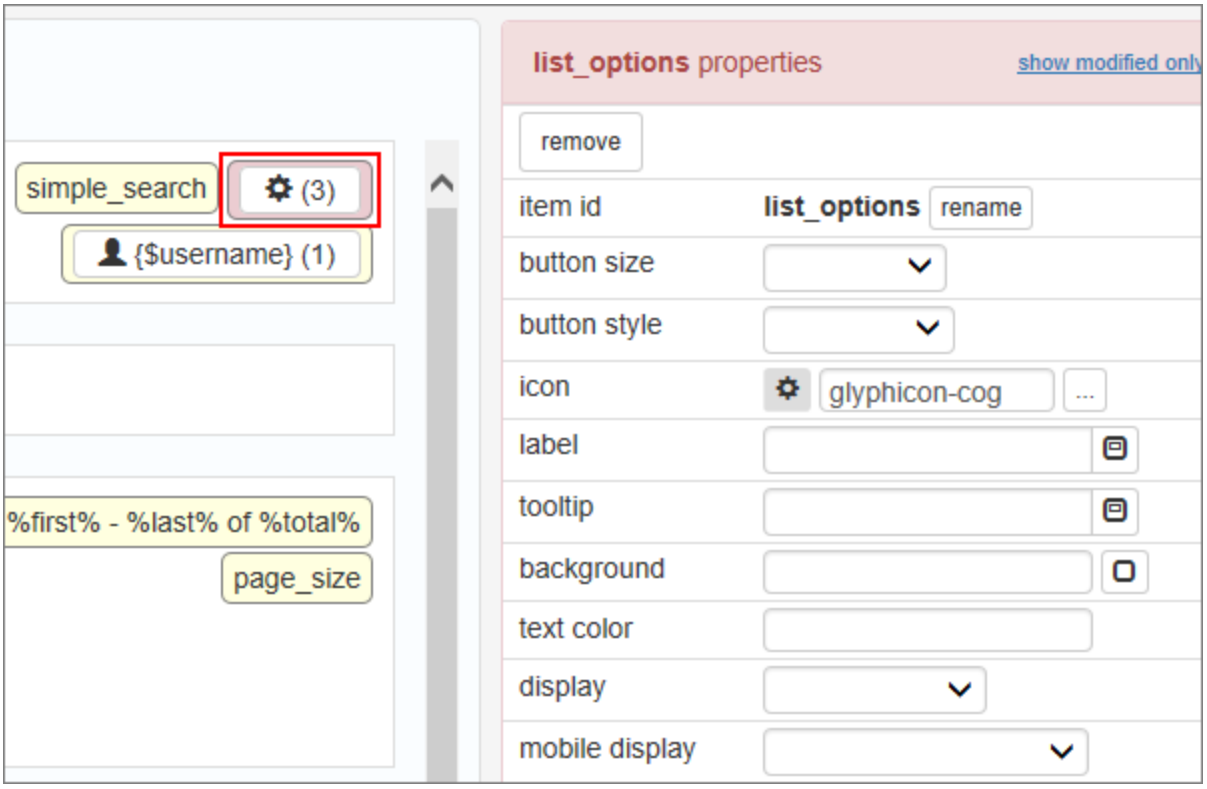
How to enable/disable a button

To enable/disable standard buttons or custom ones, you need to complete the following steps. This method does not work for buttons inserted directly into the grid.

> **Note**: Change `red` values to match your project.

Get the ID of the button

You can get the ID of the button (or any page element) in the properties section of the Page Desinger screen. Click the button to select it and check the **item id** field in its properties:



> Note: **item id** is case-sensitive.

Add the code

Add the following code to any JavaScript OnLoad event or the ClientAfter/ClientBefore event of the button.

To disable a button:

```
var id = "My_Button";
var button = $("[id^=" + id + "]");
Runner.addDisabledClass(button);
```

To enable a button:

```
var id = "My_Button";
var button = $("[id^=" + id + "]");
Runner.delDisabledClass(button);
```

See also:

- JavaScript API: RunnerPage object > getItemButton(itemId, recordId)
- Insert custom button
- Insert standard button
- Events: JavaScript OnLoad
- Tri-part events
- Page Designer
- Page Designer: Working with page elements
- JavaScript API

How to hide 'Edit selected'/'Delete selected' buttons

Sometimes you need to show the **Edit selected** and **Delete selected** buttons only when users select records on the **List** page. For this purpose, add the following code to the List page: JavaScript OnLoad event:

```
var $editSelected = $('#edit_selected' + pageid),
    $deleteSelected = $('#delete_selected' + pageid);
$editSelected.hide();
$deleteSelected.hide();

$('input[type=checkbox][name=^selection], #select_all' + pageid +', #chooseAll_' +
pageid).change( function(e) {
    var $target = $(e.target),
        selBoxes;

    if ( $target.attr('name') == 'selection[]' || $target.attr('id') == 'chooseAll_'
+ pageid ) {
     selBoxes = pageObj.getSelBoxes( pageid );
     $editSelected.toggle( selBoxes.length > 0 );
        $deleteSelected.toggle( selBoxes.length > 0 );
  }
});
```

See also:

- JavaScript API: Control object > hide()
- JavaScript API: Control object
- Events: JavaScript OnLoad
- JavaScript API

How to pass values from PHP to Javascript

To pass values from PHP to JavaScript, use the setProxyValue() method.

See also:

- RunnerPage class: setProxyValue()
- Tri-part events: passing data between events
- JavaScript API

How to refresh the List page after editing in a popup

By default, PHPRunner updates the **List** page automatically with new data. However, if you perform some actions behind the scene like adding a new record in the AfterEdit event, you might need to refresh the **List** page manually.

To refresh the **List** page manually, add the following code to the **List** page: JavaScript OnLoad event:

```
this.on('afterInlineEdit', function( fieldsData ) {
location.reload();
});
```

To refresh the grid on the **List** page without reloading the page, use the following code in the JavaScript OnLoad event:

> **Note**: the **AJAX search, pagination and sorting** option should be enabled (Choose pages -> List page settings).

```
Runner.runnerAJAX(Runner.pages.getUrl(pageObj.tName, pageObj.pageType)+"?a=return",
     pageObj.ajaxBaseParams,
    function(respObj){
      pageObj.pageReloadHn.call(pageObj, respObj)
     });
```

If you need to use both snippets at the same time, add the following code to the **List** page JavaScript OnLoad event:

> **Note**: the **AJAX search, pagination and sorting** option should be enabled (Choose pages -> List page settings).

```
window.listPage = pageObj;
```

And add the following code to the **Edit** page JavaScript OnLoad:

```
this.on('afterSave', function() {
var pageObj = window.listPage;
Runner.runnerAJAX(Runner.pages.getUrl(pageObj.tName, pageObj.pageType)+"?a=return",
pageObj.ajaxBaseParams,
function(respObj){
pageObj.pageReloadHn.call(pageObj, respObj)
});
});
```

See also:

- JavaScript API: Control object > on()
- JavaScript API: Control object
- JavaScript API: RunnerPage object
- Events: JavaScript OnLoad
- AJAX-based Functionality

- Choose pages screen
- List page settings
- JavaScript API

How to reload a page

To reload a page, use the following code in the JavaScript OnLoad event.

> **Note**: the **AJAX search, pagination and sorting** option should be enabled (Choose pages -> List page settings).

```
pageObj.reload({a:'reload'});
```

See also:

- JavaScript API: RunnerPage object
- Events: JavaScript OnLoad
- Choose pages screen
- List page settings
- JavaScript API

How to show dropdown list of US states

You can show the dropdown list of US states if US was selected in the *Country* list; hide it otherwise. Use the following code in JavaScript OnLoad event on **Add**/**Edit** pages in popup and inline.

> **Note**: Change `red` values to match your project.

```
var ctrlCountry = Runner.getControl(pageid, 'country');
var ctrlState = Runner.getControl(pageid, 'state');
```

```
ctrlCountry.on('change', function(e){
   if (this.getValue() == 'US'){
      ctrlState.show();
   }else{
      ctrlState.hide();
   }
});
```

You may want to hide the field label as well. Use the following code to hide or show the whole table row with the *State* edit control based on the *Country* field selection. This code works in popup and inline **Add**/**Edit** modes.

```
var ctrlCountry = Runner.getControl(pageid, 'country');

ctrlCountry.on('change', function(e) {
    if (this.getValue() == 'US') {
        pageObj.showField("state");
    } else {
        pageObj.hideField("state");
    }
});
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > getValue()
- JavaScript API: Control object > on()
- JavaScript API: Control object > show()
- JavaScript API: Control object > hide()
- JavaScript API: RunnerPage object > showField()
- JavaScript API: RunnerPage object > hideField()
- JavaScript API: RunnerPage object
- JavaScript API

How to control multi-step pages

Use the following code in JavaScript events like the JavaScript OnLoad for **Add** or **Edit** pages:

```
// 0 - first tab, 1 - second tab, etc.
pageObj.setCurrentStep(1);
```

This code works in the **JavaScript OnLoad** event or ClientBefore/ClientAfter events of custom buttons.

See also:

- JavaScript API: RunnerPage object
- Events: JavaScript OnLoad
- Tri-part events
- Page Designer: Insert custom button
- JavaScript API

How to return other field control in the inline mode

You can use the *ctrl.getPeer(field)* function to return the other field control in the inline mode.

Syntax

```
ctrl.getPeer(field);
```

Arguments

`field`

the name of the other field in the same row when in an inline mode.

Return value

Returns the other field control from the same page in the same row when in an inline mode. Works similar to the Runner.getControl function, but should be used in Field events.

Example

Get the 'price' control in an inline mode and then set its value to 1000.

```
var ctlPrice = ctrl.getPeer('price');
ctlPrice.setValue( 1000 );
```

See also:

- JavaScript API: Control object > getControl()
- JavaScript API: Control object > setValue()
- Field events
- Tri-part events
- JavaScript API

3.2.7 Tabs/Sections Javascript API

3.2.7.1 About Tabs/Sections API

The **Tabs/Sections API** allows you to manage tabs and sections added to **Add**, **Edit**, and **View** pages.

All tab and tab group indexes are zero-based. 0 - the first tab, 1 -the second tab, etc.

These methods work in the JavaScript OnLoad event on the **Add**/**Edit**/**View** pages.

Methods

Method	Description
getTabs([n])	Returns the **RunnerTabs object**, which represents a single tab group.
Tabs methods	
count()	Returns the number of tabs in the **Tab control** object.
activate(n)	Activates/shows the n-th tab.
activeIdx()	Gets an index of the currently selected tab.

hide(n)	Hides the n-th tab (zero-based).
show(n)	Shows the n-th tab (zero-based).
disable(n)	Makes the n-th tab disabled.
enable(n)	Makes the n-th tab enabled.
headerElement(n)	Returns the **jQuery object** of the tab header.
bodyElement(n)	Returns the **jQuery object** of the tab content.
addTab(headerHtml, paneHtml)	Creates a new tab and adds it to the end of a tab group.
moveTo(n, m)	Changes the order of the tabs. The n-th tab is moved to m-th position. The m-th tab is moved to the n-th position.
Sections methods	
getSectionCount()	Returns the number of sections on the page.
getSection(n)	Returns the **RunnerSection object** representing the n-th section on the page (zero-based).
headerElement()	Returns the **jQuery object** of the section header.
bodyElement()	Returns the **jQuery object** of the section content.
expand()	Expands the section.
collapse()	Collapses the section.

See also:

- Page designer. Working with table pages
- About Grid Row JavaScript API
- Choose pages elements
- About RunnerPage class

3.2.7.2 Tabs methods

getTabs

Returns the **RunnerTabs object**, which represents a single tab group.

Syntax

```
getTabs( [N] );
```

Arguments

`N`

a zero-based index of a tab group. If no argument specified, returns first tab group of the page.

Return value

Returns the **RunnerTabs object**.

If `N` is more than the number of tab groups, the method returns *null*.

Example

Activate the third tab in the first tab group.

```
var tabs = pageObj.getTabs(0);
tabs.activate(2);
```

See also:

- Tabs API: activate
- Sections API: getSection
- Grid Row JavaScript API: row.getKeys()
- RunnerPage object
- Tabs/Sections API
- JavaScript API: RunnerPage object
- JavaScript API

count

Returns the number of tabs in the **Tab control** object.

Syntax

```
count()
```

Arguments

No arguments.

Return value

Returns the number of tabs in the **Tab control** object.

Example

Count the number of tabs in a tab group.

```
var tabs = pageObj.getTabs();
var count = tabs.count();
```

See also:

- RunnerPage object: getTabs
- Tabs API: activate
- Sections API: getSectionCount
- About Tabs/Sections API

activate

Activates/shows the n-th tab. The tab index is zero-based.

Syntax

```
activate( n )
```

Arguments

n

a zero-based index of the tab to be activated.

Return value

No return value.

Example

Activate the third tab in the first tab group.

```
var tabs = pageObj.getTabs(0);
tabs.activate(2);
```

See also:

- RunnerPage object: getTabs
- Tabs API: ActiveIdx
- RunnerPage object: showField()
- RunnerPage class: showField()
- About Tabs/Sections API

activeIdx

Gets an index of the currently selected tab.

Syntax

```
activeIdx()
```

Arguments

No arguments.

Return value

A zero-based index of the active tab.

Example

Get an index of the currently selected tab.

```
var tabs = pageObj.getTabs();
var idx = tabs.activeIdx();
```

See also:

- RunnerPage object: getTabs
- Tabs API: Activate
- Grid Row JavaScript API: row.getKeys()
- Grid Row JavaScript API: row.recordId()
- About Tabs/Sections API

hide

Hides the n-th tab (zero-based).

Syntax

```
hide( n )
```

Arguments

n

a zero-based index of the tab.

Return value

No return value.

Example

Hides the third tab.

```
var tabs = pageObj.getTabs();
tabs.hide(2);
```

See also:

- RunnerPage object: getTabs
- Tabs API: show
- JavaScript API: RunnerPage object > hideField()
- RunnerPage class: hideField()
- About Tabs/Sections API

show

Shows the n-th tab (zero-based).

Syntax

```
show( n )
```

Arguments

n

a zero-based index of the tab.

Return value

No return value.

Example

Show the third tab.

```
var tabs = pageObj.getTabs();
tabs.show(2);
```

See also

- RunnerPage object: getTabs
- Tabs API: hide()
- JavaScript API: RunnerPage object > showField()
- RunnerPage class: showField()
- About Tabs/Sections API

disable

Makes the n-th tab disabled. The tab remains visible.

Syntax

```
disable( n )
```

Arguments

n

a zero-based index of the tab.

Return value

No return value.

Example

Make the fourth tab disabled.

```
var tabs = pageObj.getTabs();
tabs.disable(3);
```

See also:

- RunnerPage object: getTabs
- Tabs API: enable
- Control object: isReadonly()
- Control object: setDisabled()
- About Tabs/Sections API

enable

Makes the n-th tab enabled.

Syntax

```
enable( n )
```

Arguments

n

a zero-based index of the tab.

Return value

No return value.

Example

Make the fourth tab enabled.

```
var tabs = pageObj.getTabs();
tabs.enable(3);
```

See also:

- RunnerPage object: getTabs
- Tabs API: disable
- Control object: setEnabled()
- About Tabs/Sections API

headerElement

Returns the **jQuery object** of the tab header.

Syntax

```
headerElement( n )
```

Arguments

n

a zero-based index of the tab.

Return value

Returns the jQuery object of the tab header.

Example

Change the title of a tab to "**Active tab**" in bold.

```
var tabs = pageObj.getTabs();
tabs.headerElement(1).html('<b>Active tab</b>');
```

See also:

- RunnerPage object: getTabs
- Sections API: headerElement

- Editing the <head> section
- About Tabs/Sections API

bodyElement

Returns the **jQuery object** of the tab content.

Syntax

```
bodyElement( n )
```

Arguments

n

a zero-based index of the tab.

Return value

Returns the **jQuery object** of the tab content.

Example

Change the content of the tab to read "*This is the content of the tab*".

```
var tabs = pageObj.getTabs();
tabs.bodyElement(0).html( '<h1>This is the content of the tab</h1>' );
```

See also:

- RunnerPage object: getTabs
- Sections API: bodyElement
- Grid Row JavaScript API: row.setFieldText()
- About Tabs/Sections API

addTab

Creates a new tab and adds it to the end of a tab group.

Syntax

```
addTab( headerHtml, paneHtml )
```

Arguments

`headerHtml`

`paneHtml`

Return value

Returns the index of a new tab.

Example

Create a new tab "*New Tab*" that contains "*New tab content*" and add it to the end of a tab group.

```
var tabs = pageObj.getTabs();
tabs.addTab('New tab', 'New tab content');
```

See also:

- RunnerPage object: getTabs
- Menu builder
- Page Designer: Tabs and sections
- About Tabs/Sections API

moveTo

Changes the order of the tabs. The *n-th* tab is moved to the *m-th* position. The *m-th* tab is moved to the *n-th* position.

If *m* is greater than the number of tabs, the *n-th* tab is moved to the end.

Syntax

```
moveTo( n, m )
```

Arguments

n

a zero-based index of the tab.

m

a zero-based index of the tab.

Return value

Returns a new index of the *n-th* tab.

Example

Third tab is moved to the fourth position. The fourth tab is moved to the third position.

```
var tabs = pageObj.getTabs();
tabs.moveTo(2,3);
```

See also:

- RunnerPage object: getTabs
- Tabs API: addTab
- Page Designer: Tabs and sections
- About Tabs/Sections API

3.2.7.3 Sections methods

getSectionCount

Returns the number of sections on the page.

Syntax

```
getSectionCount()
```

Arguments

No arguments.

Return value

Returns the number of sections on the page.

Example

Count the number of sections.

```
var sections = pageObj.getSectionCount();
```

See also:

- Sections API: getSection
- Tabs API: count
- About Tabs/Sections API

getSection

Syntax

```
getSection( n )
```

Arguments

n

a zero-based index of the section.

Return value

Returns the **RunnerSection object** representing the `n`-th section on the page (zero-based).

Example

Return the first section.

```
var sec = pageObj.getSection(0);
```

See also:

- RunnerPage object: getTabs
- Sections API: getSectionCount
- Grid Row JavaScript API: row.getKeys()
- About Tabs/Sections API

headerElement

Returns the **jQuery object** of the section header.

Syntax

```
headerElement()
```

Arguments

No arguments.

Return value

Returns the **jQuery object** of the section header.

Example

Change the first section title to "**Active Section**" written in bold.

```
var section = pageObj.getSection(0);
section.headerElement().find('a').html('<b>Active Section</b>');
```

See also:

- Sections API: getSection
- Tabs API: headerElement
- Grid Row JavaScript API: row.fieldCell()
- Editing the <head> section
- About Tabs/Sections API

bodyElement

Returns the **jQuery object** of the section content.

Syntax

```
bodyElement()
```

Arguments

No arguments.

Return value

Returns the **jQuery object** of the section content.

Example

Change the first section content to read "This is the section content".

```
var section = pageObj.getSection(0);
section.bodyElement().html( '<h1>This is the section content</h1>' );
```

See also:

- Sections API: getSection
- Grid Row JavaScript API: row.fieldCell()
- Tabs API: bodyElement
- About Tabs/Sections API

expand

Expands a section.

Syntax

```
expand()
```

Arguments

No arguments.

Return value

No return value.

Example

Expand the first section.

```
var sec = pageObj.getSection(0);
sec.expand();
```

See also:

- Sections API: getSection
- Sections API: collapse

- About Tabs/Sections API

collapse

Collapses a section.

Syntax

```
collapse()
```

Arguments

No arguments.

Return value

No return value.

Example

Collapse the first section.

```
var sec = pageObj.getSection(0);
sec.collapse();
```

See also:

- Sections API: getSection
- Sections API: expand
- About Tabs/Sections API

3.2.8 Labels/Titles API

3.2.8.1 About Labels/Titles API

The **Labels**/**Titles API** allows you to work with the table/field labels and titles. **Table**/**field** names are case-insensitive.

API functions work in events like AfterAppInit, AfterTableInit, BeforeProcess of the appropriate page.

If the $language parameter is not specified, the current language is used.

Functions

Function	Description
getFieldLabel setFieldLabel	Gets or sets the field label.
getTableCaption setTableCaption	Gets or sets the table caption.
getProjectLogo setProjectLogo	Gets or sets the project logo.
getPlaceholder setPlaceholder	Gets or sets the field placeholder.
getFieldTooltip setFieldTooltip	Gets or sets the field tooltip.
getPageTitleTempl setPageTitleTempl	Gets or sets the page title.
getBreadcrumbsLabelTempl setBreadcrumbsLabelTempl	Gets or sets the breadcrumbs label template.

See also:

- Miscellaneous settings: Label Editor
- Menu builder
- Page Designer: Working with page elements

3.2.8.2 Methods

getFieldLabel

Gets the field label.

Syntax

```
Labels::getFieldLabel($table, $field, $language)
```

Arguments

$table

the table name.

$field

the field name.

$language

the language of the field. If the *$language* parameter is not specified, the current language is used.

Return value

Returns the field label. Labels can be set in the Label Editor.

Example

Gets the "carscars" table "descr" field label for the current language.

```
Labels::getFieldLabel("carscars", "descr");
```

See also:

- Labels/Titles API: setFieldLabel()
- Miscellaneous settings: Label Editor
- About Labels/Titles API

setFieldLabel

Sets the field label.

Syntax

```
Labels::setFieldLabel($table, $field, $label, $language)
```

Arguments

$table

the table name.

$field

the field name.

$label

a new field label. Labels can be set in the Label Editor.

$language

the language of the field. If the *$language* parameter is not specified, the current language is used.

Return value

No return value.

Example

Labels can be set in the Label Editor. This example shows how to change the "descr" field label of the "carscars" table to "New Description" for the current language.

> **Note**: You can use this code in the AfterAppInit, AfterTableInit, BeforeProcess events.

```
Labels::setFieldLabel("carscars", "descr", "New Description");
```

See also:

- Labels/Titles API: getFieldLabel()
- Miscellaneous settings: Label Editor
- About Labels/Titles API

getTableCaption

Gets the table caption.

Syntax

```
Labels::getTableCaption($table, $language)
```

Arguments

$table

the table name.

$language

the language of the caption. If the *$language* parameter is not specified, the current language is used.

Return value

Returns the table caption. Captions can be set in the Label Editor.

Example

Get the "carscars" table caption.

```
Labels::getTableCaption("carscars");
```

See also:

- Labels/Titles API: setTableCaption()
- Miscellaneous settings: Table labels
- About Labels/Titles API

setTableCaption

Sets the table caption.

Syntax

```
Labels::setTableCaption($table, $caption, $language);
```

Arguments

$table

the table name.

$caption

a new table caption. Captions can be set in the Label Editor.

$language

the language of the caption. If the *$language* parameter is not specified, the current language is used.

Return value

No return value.

Example

Captions can be set in the Label Editor. This example shows how to change the "carscars" table caption to "New Cars".

Note: You can use this code in the AfterAppInit, AfterTableInit, BeforeProcess events.

```
Labels::setTableCaption("carscars", "New Cars");
```

See also:

- Labels/Titles API: getTableCaption()
- Miscellaneous settings: Table labels
- About Labels/Titles API

getProjectLogo

Gets the project logo.

Syntax

```
Labels::getProjectLogo($language)
```

Arguments

$language

the language of the logo. If the *$language* parameter is not specified, the current language is used.

Return value

Returns the project logo. The project logo can be set in the Label Editor.

Example

Get the Project logo.

```
Labels::getProjectLogo();
```

See also:

- Labels/Titles API: setProjectLogo()
- Miscellaneous settings: Project logo
- Page Designer: Working with page elements
- About Labels/Titles API

setProjectLogo

Sets the project logo.

Syntax

```
Labels::setProjectLogo($label, $language)
```

Arguments

$label

the project logo. The *$label* parameter may contain any text or HTML code.

$language

the language of the logo. If the *$language* parameter is not specified, the current language is used.

Return value

No return value.

Example

Set the project logo.

Note: You can use this code in the AfterAppInit, AfterTableInit, BeforeProcess events.

```
Labels::setProjectLogo("<img src='http://mywebsite.com/images/logo.png'>");
```

See also:

- Labels/Titles API: getProjectLogo()
- Miscellaneous settings: Project logo
- Page Designer: Working with page elements
- Page Designer: Insert Text and Image
- About Labels/Titles API

getFieldTooltip

Gets the field tooltip.

Syntax

```
Labels::getFieldTooltip($table, $field, $language)
```

Arguments

`$table`

the table name.

`$field`

the field name.

`$language`

the language of the tooltip. If the *$language* parameter is not specified, the current language is used.

Return value

Returns the field tooltip. Tooltips can be set in the Label Editor.

Example

Get the *"descr"* field tooltip of the *"carscars"* table.

```
Labels::getFieldTooltip("carscars", "descr");
```

See also:

- Labels/Titles API: setFieldTooltip()
- Miscellaneous settings: Edit form tooltips
- Page Designer: Working with page elements
- About Labels/Titles API

setFieldTooltip

Sets the field tooltip.

Syntax

```
Labels::setFieldTooltip($table, $field, $tooltip, $language)
```

Arguments

$table

the table name.

$field

the field name.

$tooltip

the field tooltip.

$language

the language of the tooltip. If the *$language* parameter is not specified, the current language is used.

Return value

No return value.

Example

This example shows how to change the *"descr"* field tooltip of the *"carscars"* table to *"My new tooltip"*.

> **Note**: You can use this code in the AfterAppInit, AfterTableInit, BeforeProcess events.

```
Labels::setFieldTooltip("carscars", "descr", "My new tooltip");
```

See also:

- Labels/Titles API: getFieldTooltip()

- Miscellaneous settings: Edit form tooltips
- Page Designer: Working with page elements
- About Labels/Titles API

getPlaceholder

Gets the field placeholder.

Syntax

```
Labels::getPlaceholder($table, $field, $language)
```

Arguments

$table

the table name.

$field

the field name.

$language

the language of the placeholder. If the *$language* parameter is not specified, the current language is used.

Return value

Returns the field placeholder. Placeholder can be set in the Edit As settings or in the Label Editor on the **Miscellaneous settings** screen.

Example

Gets the "*descr*" field placeholder of the "*carscars*" table.

```
Labels::getPlaceholder("carscars", "descr");
```

See also:

- Labels/Titles API: setPlaceholder()
- Miscellaneous settings: Label Editor
- "Edit As" settings: Text Field
- Page Designer: Working with page elements
- About Labels/Titles API

setPlaceholder

Sets the field placeholder.

Syntax

```
Labels::setPlaceholder($table, $field, $placeholder, $language)
```

Arguments

$table

the table name.

$field

the field name.

$placeholder

the field placeholder.

$language

the language of the placeholder. If the *$language* parameter is not specified, the current language is used.

Return value

No return value.

Example

This example shows how to change the "*descr*" field placeholder of the "*carscars*" table to "*Cars description*".

> **Note**: You can use this code in the AfterAppInit, AfterTableInit, BeforeProcess events.

```
Labels::setPlaceholder("carscars", "descr", "Cars description");
```

See also:

- Labels/Titles API: getPlaceholder()
- Miscellaneous settings: Label Editor
- "Edit As" settings: Text Field
- Page Designer: Working with page elements
- About Labels/Titles API

getPageTitleTempl

This function works with page title templates and not with the titles directly. Examples of page title templates can be found in the Label Editor.

Syntax

```
Labels::getPageTitleTempl($table, $page, $language)
```

Arguments

$table

the table name.

$field

the field name.

$language

the language of the template. If the *$language* parameter is not specified, the current language is used.

$page

the **page** parameter accepts one of the following values:

PAGE_LIST

PAGE_PRINT

PAGE_ADD

PAGE_EDIT

PAGE_VIEW

PAGE_SEARCH

PAGE_EXPORT

PAGE_IMPORT

PAGE_REPORT

PAGE_RPRINT

PAGE_CHART

PAGE_MASTER_INFO_LIST

PAGE_MASTER_INFO_PRINT

PAGE_MASTER_INFO_REPORT

PAGE_MASTER_INFO_RPRINT

PAGE_MENU

PAGE_LOGIN

PAGE_REGISTER

PAGE_REMIND

PAGE_CHANGEPASS

Return value

Returns the page title. Page titles can be set in the Label Editor.

Example

This example shows how to get the **Edit** page title template.

```
Labels::getPageTitleTempl("carscars", PAGE_EDIT);
```

See also:

- Labels/Titles API: setPageTitleTempl()
- Labels/Titles API: getFieldLabel()
- Miscellaneous settings: Label Editor
- About Labels/Titles API

setPageTitleTempl

This function works with page title templates and not with the titles directly. Examples of page title templates can be found in the Label Editor.

Syntax

```
Labels::setPageTitleTempl($table, $page, $title, $language)
```

Arguments

$table

the table name.

$field

the field name.

$title

the page title.

$language

the language of the template. If the *$language* parameter is not specified, the current language is used.

$page

the **page** parameter accepts one of the following values:

PAGE_LIST

PAGE_PRINT

PAGE_ADD

PAGE_EDIT

PAGE_VIEW

PAGE_SEARCH

PAGE_EXPORT

PAGE_IMPORT

PAGE_REPORT

PAGE_RPRINT

PAGE_CHART

PAGE_MASTER_INFO_LIST

PAGE_MASTER_INFO_PRINT

PAGE_MASTER_INFO_REPORT

PAGE_MASTER_INFO_RPRINT

PAGE_MENU

PAGE_LOGIN

PAGE_REGISTER

PAGE_REMIND

PAGE_CHANGEPASS

Return value

No return value.

Example

This example shows how to change the **Edit** page title template.

Note: You can use this code in the AfterAppInit, AfterTableInit, BeforeProcess events.

```
Labels::setPageTitleTempl("carscars", PAGE_EDIT, "Edit car [{%id}]");
```

See also:

- Labels/Titles API: getPageTitleTempl()
- Labels/Titles API: setFieldLabel
- Miscellaneous settings: Label Editor
- About Labels/Titles API

getBreadcrumbsLabelTempl

This function gets the current breadcrumbs template.

Syntax

```
Labels::getBreadcrumbsLabelTempl($table, $masterTable, $page, $language)
```

Arguments

$table

the table name.

$masterTable

the master table name.

$language

the language of the breadcrumbs template. If the *$language* parameter is not specified, the current language is used.

$page

the page name. The **page** parameter accepts one of the following values:

PAGE_LIST

PAGE_PRINT

PAGE_ADD

PAGE_EDIT

PAGE_VIEW

PAGE_SEARCH

PAGE_EXPORT

PAGE_IMPORT

PAGE_REPORT

PAGE_RPRINT

PAGE_CHART

PAGE_MASTER_INFO_LIST

PAGE_MASTER_INFO_PRINT

PAGE_MASTER_INFO_REPORT

PAGE_MASTER_INFO_RPRINT

PAGE_MENU

PAGE_LOGIN

PAGE_REGISTER

PAGE_REMIND

PAGE_CHANGEPASS

Return value

Returns the current breadcrumbs template.

See also:

- Labels/Titles API: setBreadcrumbsLabelTempl()
- Page Designer: Working with page elements
- Master-details relationship between tables

- Menu builder
- About Labels/Titles API

setBreadcrumbsLabelTempl

This function sets the current breadcrumbs template.

Syntax

```
Labels::setBreadcrumbsLabelTempl($table, $label, $masterTable, $page, $language)
```

Arguments

$table

the table name.

$label

the breadcrumb template.

$masterTable

the master table name (if we are on details table page at the moment).

$language

the language of the breadcrumbs template. If the *$language* parameter is not specified, the current language is used.

$page

the page name. The page parameter accepts one of the following values:

PAGE_LIST

PAGE_PRINT

PAGE_ADD

PAGE_EDIT

PAGE_VIEW

PAGE_SEARCH

PAGE_EXPORT

PAGE_IMPORT

PAGE_REPORT

PAGE_RPRINT

PAGE_CHART

PAGE_MASTER_INFO_LIST

PAGE_MASTER_INFO_PRINT

PAGE_MASTER_INFO_REPORT

PAGE_MASTER_INFO_RPRINT

PAGE_MENU

PAGE_LOGIN

PAGE_REGISTER

PAGE_REMIND

PAGE_CHANGEPASS

Return value

No return value.

Example

By default the *Order Details* table uses the following **breadcrumbs menu**:

Home / Orders / Order Details [5187]

We want to change it, removing the "*Order details*" part:

Home / Orders / 5187

You can use any field from the details table here (**OrderID**) or from master table (**master**.**OrderID**)

> **Note**: You can use this code in the AfterAppInit, AfterTableInit, BeforeProcess events.

We can use either:

```
Labels::setBreadcrumbsLabelTempl("order details", "{%OrderID}", "orders");
```

or

```
Labels::setBreadcrumbsLabelTempl("order details", "{%master.OrderID}", "orders");
```

See also:

- Labels/Titles API: getBreadcrumbsLabelTempl()
- Page Designer: Working with page elements
- Menu builder
- Master-details relationship between tables
- About Labels/Titles API

3.2.9 Additional WHERE tabs API

3.2.9.1 About Additional WHERE tabs API

The **Additional WHERE tabs API** allows adding, deleting and modifying the Additional WHERE tabs from the server event code. All the methods of this API should be used in the After table initialized event only.

Any tab modifications made with this API apply to **List**, **Print** and **Export** pages together.

Methods

Method	Description

addTab	Adds a tab to the additional WHERE tabs on the **List**, **Print**, **Export**, or **Chart** page.
deleteTab	Deletes the additional WHERE tab.
setTabTitle	Sets the additional WHERE tab title.
setTabWhere	Sets tab WHERE clause for the additional WHERE tab title.

See also:

- SQL query screen: Additional WHERE tabs
- Event: After table initialized
- About RunnerPage class

3.2.9.2 Methods

addTab

The **addTab** method adds a tab to the Additional WHERE tabs on the **List**, **Print**, **Export**, or **Chart** page. This method should be used in the After table initialized event only.

Syntax

```
WhereTabs::addTab($table, $where, $title, $id);
```

Arguments

$table

the name of the table name in which to modify the tabs. Use the *$table* variable provided in the **After table initialized** event.

$where

a WHERE clause for the new tab.

$title

the title of the tab.

$id

the ID of the new tab.

Return value

`true`

if the tab was added successfully.

`false`

if the tab was not added successfully.

Example

Add the *'ANTON orders'* tab:

```
WhereTabs::addTab($table,"CustomerID='ANTON'", "ANTON orders", "anton");
```

See also:

- Additional WHERE tabs API: deleteTab
- Additional WHERE tabs API: setTabTitle
- Additional WHERE tabs API: setTabWhere
- SQL query screen: Additional WHERE tabs
- Event: After table initialized
- About Additional WHERE tabs API

deleteTab

The **deleteTab** method deletes one of the Additional WHERE tabs on the **List**, **Print**, **Export**, or **Chart** page. This method should be used in the After table initialized event only.

Syntax

```
WhereTabs::deleteTab($table, $id);
```

Arguments

`$table`

the name of the table name in which to modify the tabs. Use the *$table* variable provided in the **After table initialized** event.

`$id`

the id of the tab to be deleted.

Return value

No return value.

Example

Delete the "*anton*" tab:

```
WhereTabs::deleteTab($table, "anton");
```

See also:

- Additional WHERE tabs API: addTab
- Additional WHERE tabs API: setTabTitle
- Additional WHERE tabs API: setTabWhere
- SQL query screen: Additional WHERE tabs
- Event: After table initialized
- About Additional WHERE tabs API

setTabTitle

The **setTabTitle** method sets the title of one of the Additional WHERE tabs on the **List**, **Print**, **Export**, or **Chart** page. This method should be used in the After table initialized event only.

Syntax

```
WhereTabs::setTabTitle($table, $id, $title);
```

Argument

`$table`

the name of the table name in which to modify the tabs. Use the *$table* variable provided in the **After table initialized** event.

`$id`

the ID of the tab.

`$title`

the title of the tab.

Return value

`true`

if the title was set.

`false`

in all other cases.

Example

Set a new title to the tab:

```
WhereTabs::setTabTitle($table, "anton", "ANTON orders");
```

See also:

- Additional WHERE tabs API: addTab
- Additional WHERE tabs API: deleteTab
- Additional WHERE tabs API: setTabWhere
- SQL query screen: Additional WHERE tabs
- Event: After table initialized
- About Additional WHERE tabs API

setTabWhere

The **setTabWhere** method sets the WHERE clause of one of the Additional WHERE tabs on the **List**, **Print**, **Export**, or **Chart** page. This method should be used in the After table initialized event only.

Syntax

```
WhereTabs::setTabWhere($table, $id, $where);
```

Arguments

$table

the name of the table name in which to modify the tabs. Use the *$table* variable provided in the **After table initialized** event.

$id

the ID of the tab.

$where

a WHERE clause for the tab.

Return value

true

if the WHERE clause was set successfully.

false

in all other cases.

Example

Set a WHERE clause to the tab:

```
WhereTabs::setTabWhere($table, "anton", "CustomerID='ANTON'");
```

See also:

- Additional WHERE tabs API: addTab
- Additional WHERE tabs API: deleteTab
- Additional WHERE tabs API: setTabTitle
- SQL query screen: Additional WHERE tabs
- Event: After table initialized
- About Additional WHERE tabs API

3.2.10 Page class

3.2.10.1 About RunnerPage class

An object of the **RunnerPage class** represents the current page and is passed to the event as a *$pageObject* parameter.

Methods

Partially deprecated

The *Tab* methods are deprecated, we recommend using Additional WHERE tabs API instead.

Method	Description	Available on
getCurrentRecord()	Gets the current record.	**Edit**/**View** pages.
getMasterRecord()	Gets the master record.	All pages.
hideField()	Hides the field and field label.	BeforeDisplay event of the **Add**/**Edit**/**View**/**Register** pages.
setProxyValue()	Sets the variable to the given value. You may use this method to pass values from PHP to JavaScript.	All pages.
showField()	Shows the previously hidden field and field label.	BeforeDisplay event of the **Add**/**Edit**/**View**/**Register** pages.
addTab() (deprecated)	Adds a tab to the additional WHERE tabs on the **List** or **Chart** page.	BeforeProcess event of the **List** or **Chart** page.
deleteTab() (deprecated)	Deletes the additional WHERE tab.	BeforeProcess event of the **List** or **Chart** page.
setTabTitle() (deprecated)	Sets the additional WHERE tab title.	BeforeProcess event of the **List** or **Chart** page.
setTabWhere() (deprecated)	Sets tab WHERE clause for the additional WHERE tab title.	BeforeProcess event of the **List** or **Chart** page.
hideItem()	Hides the selected item.	After record processed event on the **List** and **Print** pages, Before

Method	Description	Available on
		display event.
showItem()	Shows the previously hidden item.	After record processed event on the **List** and **Print** pages, BeforeDisplay event.

See also:

- JavaScript API: RunnerPage object
- Additional WHERE tabs API
- Choose pages screen
- SQLQuery Screen: Additional WHERE tabs
- Global events

3.2.10.2 Methods

getCurrentRecord

Gets the current record. Available on **Edit**/**View** pages.

Syntax

```
getCurrentRecord()
```

Arguments

No arguments.

Return value

Returns the array.

Example

Get the current record and display the values of the *Make* and *Model* fields:

```
$data = $pageObject->getCurrentRecord();
echo $data["Make"] ." ".$data["Model"];
```

See also:

- RunnerPage class: getMasterRecord()
- JavaScript API: RunnerPage object > getSelectedRecordKeys()
- About RunnerPage class

getMasterRecord

Gets the master record. Available on all pages.

Syntax

```
getMasterRecord()
```

Arguments

No arguments.

Return value

Returns the array.

Example

Get the master record and display the values of the *Make* and *Model* fields:

```
$data = $pageObject->getMasterRecord();
echo $data["Make"] ." ".$data["Model"];
```

See also:

- RunnerPage class: getCurrentRecord()
- JavaScript API:RunnerPage object > getDetailsPage()

- JavaScript API: RunnerPage object > getDetailsPages()
- JavaScript API: RunnerPage object > getMasterPage()
- Example: Show data from a master table on the details view/edit/add page
- Master-details relationship between tables
- About RunnerPage class

hideField

The **hideField()** method allows you to hide a field and its label. This method is available in the BeforeDisplay event of the **List**/**Add**/**Edit**/**View**/**Register** pages.

Syntax

```
hideField($field)
```

Arguments

$field

a name of the field. Example: "*Make*".

Return value

No return value.

Example

Hide the *Make* field:

```
$pageObject->hideField("Make");
```

Note: fields hidden by the **hideField()** method can be displayed using JavaScript API: RunnerPage object > showField() method.

See also:

- RunnerPage class: showField()
- JavaScript API: RunnerPage object > hideField()
- JavaScript API: RunnerPage object > showField()
- Example: Hide empty fields on the View page
- About RunnerPage class

setProxyValue

Sets the variable to the given value. Use the **setProxyValue()** method to pass values from PHP to JavaScript: you assign value to a variable in PHP and then use it in JavaScript.

This method is available on all pages.

Syntax

```
setProxyValue($name, $value)
```

Arguments

$name

any variable name. Example: "*master*".

$value

the value assigned to the variable. *$value* may be a simple value (e.g., string, number) or an array.

Return value

No return value.

Example

Set the variables *name* and *master* to some values:

```
$pageObject->setProxyValue("name", $value);
$pageObject->setProxyValue("master",$pageObject->getMasterRecord());
```

Use the *name* and *master* variables in JavaScript OnLoad event:

```
alert(proxy['name']);
alert(proxy.master['Make']);
```

See also:

- RunnerPage class: getMasterRecord()
- Event: AfterAdd
- Event: AfterEdit
- Event: JavaScript OnLoad
- Example: How to display any page in a popup window
- About RunnerPage class

showField

Shows the previously hidden field and field label. The **showField()** method is available in the BeforeDisplay event of the **Add**/**Edit**/**View**/**Register** pages.

Syntax

```
showField($field)
```

Arguments

`$field`

a name of the field. Example: "*Make*".

Return value

No return value.

Example

Show the *Make* field:

```
$pageObject->showField("Make");
```

See also:

- RunnerPage class: hideField()
- JavaScript API: RunnerPage object > hideField()
- JavaScript API: RunnerPage object > showField()
- About RunnerPage class

addTab (deprecated)

> **Deprecated**
>
> The **addTab** method is deprecated, we recommend using Additional WHERE tabs API: addTab instead.

This method adds a tab to the Additional WHERE tabs on the **List** page. The **addTab** method is available in the BeforePocess event of **List** or **Chart** pages.

Syntax

```
$pageObject->addTab($where, $title, $id);
```

Arguments

$where

a WHERE clause for the new tab.

$title

the title of the tab.

`$id`

the ID of the new tab.

Return value

`true`

if the tab was added successfully.

`false`

if the tab was not added successfully.

Example

An example of adding a tab:

```
$pageObject->addTab("CustomerID='ANTON'", "ANTON orders", "anton");
```

See also:

- Additional WHERE tabs API: addTab
- Additional WHERE tabs API: deleteTab
- Additional WHERE tabs API: setTabTitle
- Additional WHERE tabs API: setTabWhere
- About Additional Where tabs API
- SQLQuery Screen: Additional WHERE tabs
- About RunnerPage class

deleteTab (deprecated)

Deprecated

The **deleteTab** method is deprecated, we recommend using Additional WHERE tabs API: deleteTab instead.

This method deletes one of the Additional WHERE tabs on the **List** page. The **deleteTab** method is available in the BeforePocess event of **List** or **Chart** pages.

Syntax

```
$pageObject->deleteTab($id);
```

Arguments

$id

the id of the tab to be deleted.

Return value

No return value.

Example

Delete the "*anton*" tab:

```
$pageObject->deleteTab("anton");
```

See also:

- Additional WHERE tabs API: addTab
- Additional WHERE tabs API: deleteTab
- Additional WHERE tabs API: setTabTitle
- Additional WHERE tabs API: setTabWhere
- About Additional Where tabs API
- SQLQuery Screen: Additional WHERE tabs
- About RunnerPage class

setTabTitle (deprecated)

Deprecated

The **setTabTitle** method is deprecated, we recommend using Additional WHERE tabs API: setTabTitle instead.

This method sets the title of one of the Additional WHERE tabs on the **List** page. The **setTabTitle** method is available in the BeforePocess event of the **List** or **Chart** pages.

Syntax

```
$pageObject->setTabTitle($id, $title);
```

Argument

$id

the ID of the tab.

$title

the title of the tab.

Return value

true

if the title was set.

false

in all other cases.

Example

Set a new title to the tab:

```
$pageObject->setTabTitle("anton", "ANTON orders");
```

See also:

- Additional WHERE tabs API: addTab
- Additional WHERE tabs API: deleteTab

- Additional WHERE tabs API: setTabTitle
- Additional WHERE tabs API: setTabWhere
- About Additional Where tabs API
- SQLQuery Screen: Additional WHERE tabs
- About RunnerPage class

setTabWhere (deprecated)

> **Deprecated**
>
> The **setTabWhere** method is deprecated, we recommend using Additional WHERE tabs API: setTabWhere instead.

This method sets the WHERE clause of one of the Additional WHERE tabs on the **List** page. The **setTabWhere** method is available in the BeforePocess event of the **List** or **Chart** pages.

Syntax

```
$pageObject->setTabWhere($id, $where);
```

Arguments

$id

the ID of the tab.

$where

a WHERE clause for the tab.

Return value

true

if the WHERE clause was set successfully.

false

in all other cases.

Example

Set a WHERE clause to the tab:

```
$pageObject->setTabWhere("anton", "CustomerID='ANTON'");
```

See also:

- Additional WHERE tabs API: addTab
- Additional WHERE tabs API: deleteTab
- Additional WHERE tabs API: setTabTitle
- Additional WHERE tabs API: setTabWhere
- About Additional Where tabs API
- SQLQuery Screen: Additional WHERE tabs
- About RunnerPage class

hideItem

This method is used to hide any item on the page that has an ID. It works both on server side and on the client side.

hideItem() is best used in After record processed or Before display events.

Syntax

```
$pageObject->hideItem("itemId");
```

Example

For elements in the data grid on the **List** and **Print** pages, use the *$recordId* parameter in the After record processed event:

```
$pageObject->hideItem("loginform_login", $recordId);
```

> **Note**: you can also use the JavaScript function *toggleItem()* in ClientBefore/ClientAfter events or in the Field events to hide/show the elements. See Insert custom button to learn more.

See also:

- RunnerPage class: showItem()
- JavaScript API: RunnerPage object > toggleItem()
- Example: How to hide the Edit link
- Tri-part events
- About RunnerPage class

showItem

This method is used to show the previously hidden item. It works both on the server side and on the client side.

We recommend using **showItem()** in After record processed or Before display events.

Syntax

```
$pageObject->showItem("itemId");
```

Example

For elements in the data grid on the **List** and **Print** pages, use the *$recordId* parameter in the After record processed event:

```
$pageObject->showItem("loginform_login", $recordId);
```

> **Note**: you can also use the JavaScript function *toggleItem()* in ClientBefore/ClientAfter events or in the Field events to hide/show the elements. See Insert custom button to learn more.

See also:

- RunnerPage class: hideItem()
- JavaScript API: RunnerPage object > toggleItem()
- Example: How to hide the Edit link
- Tri-part events
- About RunnerPage class

3.2.11 Search API

3.2.11.1 About Search API

This API allows you to control search panel, advanced search and 'All fields search' behavior.

Where to use the Search API?

The best events where you can use the **Search API** are AfterTableInitialized and **List** page: BeforeProcess events. You can read and write search conditions here. Changes are applied to SQL Query and are also shown on the search panel.

You can also use the **Search API** with the **List** page: BeforeDisplay event. You can also read and write search conditions here. The changes are nor applied to SQL Query but are shown on the search panel.

Functions

Function	Description
getSearchObject()	Gets the *SearchClause* object
$srchObj->getFieldValue()	Returns the search control value.
$srchObj->setFieldValue()	Replaces an existing value of a search field if the field is already added to the search panel. Adds the field to the search panel if didn't exist there previously.

$srchObj->getSecondFieldValue()	Returns the second search control value.
$srchObj->setSecondFieldValue()	Sets the second search control value if the BETWEEN option is selected.
$srchObj->getSearchOption()	Returns the search option.
$srchObj->setSearchOption()	Sets the search option for the field.
$srchObj->setSearchSQL()	Sets an SQL expression for the search. Replaces the current search expression for this field.
$srchObj->getAllFieldsSearchValue()	Gets the 'All fields search' value.
$srchObj->setAllFieldsSearchValue()	Sets the 'All fields search' value.

See also:

- Search Master and Details tables together
- JavaScript API: SearchField object
- JavaScript API: SearchController object (deprecated)

3.2.11.2 Methods

getSearchObject

Gets the *SearchClause* object.

If the *$table* parameter is not specified, the current table is used.

Syntax

```
SearchClause::getSearchObject($table)
```

Arguments

$table

the name of the table.

Return value

Returns the *SearchClause* object.

See also:

- Example: Print search parameters on the List page
- About Search API

getFieldValue

Returns the search control value.

Syntax

```
$srchObj->getFieldValue($field);
```

Arguments

`$field`

the name of the field.

Return value

Returns the search control value.

Example

Get the search value of the *Make* field.

```
$srchObj = SearchClause::getSearchObject("Cars");
$srchObj->getFieldValue("Make");
```

See also:

- Search API: getSearchObject

- Search API: setFieldValue()
- Example: Search Master and Details tables together
- Example: Print search parameters on the List page
- About Search API

setFieldValue

Replaces an existing value of a search field if the field is already added to the search panel.

Adds the field to the search panel if didn't exist there previously.

Syntax

```
$srchObj->setFieldValue($field, $value);
```

Arguments

$field

the name of the field.

$value

the value of the search field.

Return value

No return value.

Example

Replace an existing value of the *Department* search field with *10*. Add the following code to the AfterTableInit event:

```
$srchObj = SearchClause::getSearchObject("Employees");

$value = $srchObj->getFieldValue("Department");
if( $value == null ) {
```

```
$srchObj->setFieldValue("Department", 10 );
}
```

See also:

- Search API: getSearchObject
- Search API: getFieldValue()
- Event: AfterTableInit
- About Search API

getSecondFieldValue

Returns the second search control value, when BETWEEN is selected as a search option.

Syntax

```
$srchObj->getSecondFieldValue($field);
```

Arguments

$field

the name of the field.

Return value

Returns the second search control value, when BETWEEN is selected as a search option.

See also:

- Search API: setSecondFieldValue()
- Example: Print search parameters on the List page
- About Search API

setSecondFieldValue

Sets the second search control value if the BETWEEN option is selected.

Syntax

```
$srchObj->setSecondFieldValue($field, $value);
```

Arguments

$field

the name of the field.

$value

the value of the search field

Return value

No return value.

See also:

- Search API: getSecondFieldValue()
- About Search API

getSearchOption

Returns the search option.

Syntax

```
$srchObj->getSearchOption($field);
```

Arguments

$field

the name of the field.

Return value

Returns the search option, which is one of the following constants:

- CONTAINS
- EQUALS
- STARTS_WITH
- MORE_THAN
- LESS_THAN
- BETWEEN
- EMPTY_SEARCH
- NOT_CONTAINS
- NOT_EQUALS
- NOT_STARTS_WITH
- NOT_MORE_THAN
- NOT_LESS_THAN
- NOT_BETWEEN
- NOT_EMPTY

See also:

- Search API: setSearchOption()
- Choose fields screen: Search and Filter settings
- About Search API

setSearchOption

Sets the search option for the field. The list of search options is available in the **Arguments** section.

Syntax

```
$srchObj->setSearchOption($field, $option);
```

Arguments

$field

the name of the field.

$option

an option to be set. Accepts one of the following as the search option:

- CONTAINS
- EQUALS
- STARTS_WITH
- MORE_THAN
- LESS_THAN
- BETWEEN
- EMPTY_SEARCH
- NOT_CONTAINS
- NOT_EQUALS
- NOT_STARTS_WITH
- NOT_MORE_THAN
- NOT_LESS_THAN
- NOT_BETWEEN
- NOT_EMPTY

Example

How to make sure that the BETWEEN option is always selected for the *Year* field .

```
$srchObj = SearchClause::getSearchObject("Employees");
$value = $srchObj->getSearchOption("Year");
if ($value != BETWEEN) {
$srchObj->setSearchOption("Year", BETWEEN);
}
```

See also:

- Search API: getSearchObject

- Search API: getSearchOption()
- Choose fields screen: Search and Filter settings
- About Search API

setSearchSQL

Sets an SQL expression for the search field. Replaces the current search expression for the selected field.

Syntax

```
$srchObj->setSearchSQL($field, $sql);
```

Arguments

$field

the name of the field.

$sql

an SQL query to be used for the field.

Return value

No return value

Example 1

Change a search condition for the *Make* field.

```
$srchObj = SearchClause::getSearchObject("carsmodels");

$value = $srchObj->getFieldValue("make");

if( $value != null ) {
$srchObj->setSearchSQL("make", "make='$value' or id>12 and id<15");
}
```

Example 2

Here is how you can search master and details tables together. Let's say that you have *Orders* and *OrderDetails* tables and need to find orders that contain a certain product.

1. Modify the *Orders* SQL Query to add a dummy field named *'product'*. Make sure this field is searchable.

```
SELECT
        OrderID,
        CustomerID,
        EmployeeID,
        OrderDate,
        ShipAddress,
        ShipCity,
        ShipRegion,
        ShipPostalCode,
        ShipCountry,
        '' as product
FROM orders
```

2. *Orders* table, AfterTableInit event:

```
$srchObj = SearchClause::getSearchObject("orders");

$value = $srchObj->getFieldValue("product");

if( $value != null ) {
$srchObj->setSearchSQL("product", "OrderID in (select OrderID from OrderDetails where
ProductName like '%$value%')");
}
```

Note: in this event, we do a subquery to find all orders that contain the product in question.

See also:

- Search API: getSearchObject
- Search API: getFieldValue()
- SQL query screen: SQL tab

- About Search API

getAllFieldsSearchValue

Gets the *All fields search* value

Syntax

```
$srchObj->getAllFieldsSearchValue()
```

Arguments

No arguments.

Return value

Returns the *'All fields search'* value.

See also:

- Search API: setAllFieldsSearchValue()
- About Search API

setAllFieldsSearchValue

Sets the *All fields search* value.

Syntax

```
$srchObj->setAllFieldsSearchValue($value)
```

Arguments

$value

the value to be set.

Return value

No return value.

See also:

- Search API: getAllFieldsSearchValue()
- About Search API

3.2.12 Security API

3.2.12.1 About Security API

Security API allows you to work with permissions in your application.

Permissions need to be set only once per user session. That's why the **Security API** is best used in the After Successful Login event.

Functions

Function	Description
getUserGroup()	Returns the current user group name. Makes more sense to use with static permissions where each user belongs to a single user group.
getUserGroups()	Returns an array with all user groups.
getUserName()	Returns the current Username.
getDisplayName()	Returns the current Display Name.
setDisplayName()	Set the current Display Name.
isGuest()	Returns *true* if the user logged in as *Guest*, returns *false* otherwise.
isAdmin()	Returns *true* if the user is an *Admin*, returns *false* otherwise.
isLoggedIn()	Returns *true* if the user is logged in, returns *false* otherwise.
loginAs()	Logs the user in, no redirects.

logout()	Logs the user out, no redirects.
getOwnerId() setOwnerId()	These functions get or set the *OwnerID* for a specific table when Advanced Security options like **Users can see and edit their own data** are in use.
checkUsernamePassword()	This function checks the username and password and returns *true* if the username/password are correct, returns *false* otherwise.
getUserData()	Returns an array with user data from the *Login* table. Returns *false* if the user was not found.
currentUserData()	Returns an array with the current user data from the *Login* table.
getPermissions() setPermissions()	Gets/sets permissions for a certain table.
setAllowedPages()	Specifies which pages the current user can access.

See also:

- Security screen
- Registration and passwords
- User group permissions
- Dynamic permissions
- Advanced security settings

3.2.12.2 Methods

getUserGroup

Returns the current user group name. Makes more sense to use with static permissions where each user belongs to a single user group.

Syntax

```
Security::getUserGroup()
```

Arguments

No arguments.

Return value

Returns the current user group name.

Example

This example shows how to output the current user data.

```
$userData = Security::getUserGroup();
echo $userData;
```

Sample output

```
admins
```

See also:

- Security API: getUserGroups
- Security API: getUserName
- Security: User group permissions
- About Security API

getUserGroups

Returns an array with all user groups.

Syntax

```
Security::getUserGroups()
```

Arguments

No arguments.

Return value

Returns an array with all user groups.

Example

This example shows how to print the groups if they include a group named *accounts*.

```
$groups = Security::getUserGroups();
if( $groups["accounts"] ) {
print_r($groups);
}
```

Sample output

```
Array
(
    [accounts] => 1
    [<Admin>] => 1
)
```

See also:

- Security API: getUserGroup
- Security API: getUserName
- Security: User group permissions
- About Security API

getUserName

Returns the current Username.

Syntax

```
Security::getUserName()
```

Arguments

No arguments.

Return value

Returns the current Username.

Example

This example shows how to output the current user data:

```
$userData = Security::getUserName();
echo $userData;
```

Sample output

```
admin_user
```

See also:

- Security API: getUserGroup
- Security API: getUserGroups
- Security API: setAllowedPages
- Example: Hide controls on Add/Edit pages, based on the username
- About Security API

getDisplayName

Returns the current Display Name.

Syntax

```
Security::getDisplayName()
```

Arguments

No arguments.

Return value

Returns the current Display Name.

Example

This example shows how to output the current user data:

```
$userData = Security::getDisplayName();
echo $userData;
```

Sample output

```
admin_user
```

See also:

- Security API: setDisplayName
- About Security API

setDisplayName

Sets the current Display Name.

Syntax

```
Security::setDisplayName($str)
```

Arguments

`$str`

the Display name.

Return value

No return value.

Example

Change the current display name:

```
$str = "user_name" ;
Security::setDisplayName($str);
```

See also:

- Security API: getDisplayName
- About Security API

isGuest

Returns *true* if the user is logged in as Guest, returns *false* otherwise.

Syntax

```
Security::isGuest()
```

Arguments

No arguments.

Return value

Returns *true* if the user logged in as Guest, returns *false* otherwise.

Example

This example shows how to output the current user data.

```
$userData = Security::isGuest();
var_dump($userData);
```

Sample output

```
bool(false)
or
bool(true)
```

See also:

- Advanced security settings: Login as Guest
- Security API: isAdmin
- Security API: isLoggedIn
- About Security API

isAdmin

Returns *true* if user is an admin, returns *false* otherwise.

Syntax

```
Security::isAdmin()
```

Arguments

No arguments.

Return value

Returns *true* if user is an admin, returns *false* otherwise.

Example

This example shows how to output the current user data.

```
$userData = Security::isAdmin();
var_dump($userData);
```

Sample output

```
bool(false)
or
bool(true)
```

See also:

- Security API: isGuest
- Security API: isLoggedIn
- User group permissions
- Example: Hide controls on Add/Edit pages, based on the username
- About Security API

isLoggedIn

Returns *true* if user is logged in and not a guest, returns *false* otherwise.

Syntax

```
Security::isLoggedIn()
```

Arguments

No arguments.

Return value

Returns *true* if user is logged in and not a guest, returns *false* otherwise.

Example

This example shows how to output the current user data.

```
$userData = Security::isLoggedIn();
var_dump($userData);
```

Sample output

```
bool(false)
or
bool(true)
```

See also:

- Security API: isGuest
- Security API: isAdmin
- User group permissions
- About Security API

loginAs

Logs a user in, no redirects.

AftersuccessfulLogin event is called if the second parameter is set to *true*.

This function does not work with Active Directory or Facebook login.

Syntax

```
Security::loginAs($username, $callEvent = true)
```

Arguments

`$username`

name of the user to be logged in.

`$callEvent = true`

AftersuccessfulLogin event is called if the second parameter is set to true.

Return value

No return value.

Example

Authorize the current user:

```
$username = "User";
Security::loginAs($username, true);
```

See also:

- Security API: logout
- Active Directory
- Security: Facebook connect
- About Security API

logout

Logs the current user out without redirecting them.

Syntax

```
Security::logout()
```

Arguments

No arguments.

Return value

No return value.

Example

Log the current user out and redirect to another page.

```
Security::logout();
//**********  Redirect to another page  ***********
header("Location: login.php");
exit();
```

See also:

- Security API: loginAs
- Events: Redirect to another page
- About Security API

getOwnerId

This function gets the *OwnerID* for a specific table when Advanced Security options like **Users can see and edit their own data** are in use.

Syntax

```
Security::getOwnerId($table)
```

Arguments

$table

the name of the table.

Return value

Returns the *OwnerID* for the specified table.

Example

This example shows how to output the current user data:

```
$table = "table1";
echo (Security::getOwnerId($table));
```

Sample output

```
1
```

See also:

- Security API: setOwnerId()
- Advanced security settings
- About Security API

setOwnerId

This function sets the *OwnerID* for a specific table when Advanced Security options like **Users can see and edit their own data** are in use.

Syntax

```
Security::setOwnerId($table, $ownerid)
```

Arguments

`$table`

the name of the table.

`$ownerid`

the owner id.

Return value

No return value.

Example

This example shows how to output the current user data:

```
$table = "posts";
$ownerid = "1" ;
Security::setOwnerId($table, $ownerid);
```

See also:

- Security API: getOwnerId
- Advanced security settings
- About Security API

checkUsernamePassword

This function checks username and password and returns *true* if the username/password are correct, returns *false* otherwise.

If username/password are incorrect and *$fireEvents* is set to *true*, PHPRunner runs the AfterUnsuccessfulLogin event.

This function doesn't perform the authorization, just validates the username and password.

Syntax

```
Security::checkUsernamePassword($username, $password, $fireEvents = false)
```

Arguments

$username

the username.

$password

the user's password.

`$fireEvents = false`

If username/password are incorrect and *$fireEvents* is set to true, PHPRunner runs the AfterUnsuccessfulLogin event.

Return value

`true`

if username/password are correct.

`false`

if username/password are not correct.

Example

This example shows how to output the current user data:

```
var_dump (Security::checkUsernamePassword($username, $password, $fireEvents =
false));
```

Sample output

```
bool(false)
or
bool(true)
```

See also:

- Event: After Unsuccessful Login
- Event: BeforeLogin
- Security screen: Security settings
- About Security API

getUserData

Returns an array with user data from the *Login* table.

Returns *false* if the user was not found.

Syntax

```
Security::getUserData($username)
```

Arguments

$username

the username.**Return value**

Array with user data from the *Login* table.

Returns *false* if the user was not found.

Example

This example shows how to output the current user data:

```
print_r (Security::getUserData($username));
```

Sample output

```
Array
(
    [ID] => 2
    [username] => admin_user
    [password] => admin
    [email] => no
    [fullname] => John Smith
    [active] => 1
)
```

See also:

- Security API: currentUserData
- Security screen: Security settings
- About Security API

currentUserData

Returns an array with the current user data from the *Login* table.

Syntax

```
Security::currentUserData()
```

Arguments

No arguments.

Return value

Returns an array with the current user data from the *Login* table.

Example

This example shows how to output the current user data:

```
$userData = Security::currentUserData();
print_r($userData );
```

Sample output

```
Array
(
    [ID] => 1
    [username] => admin
    [password] => admin
    [email] => no
    [fullname] => John

)
```

See also:

- Security API: getUserData

- Security screen: Security settings
- About Security API

getPermissions

Gets permissions for a certain table.

Syntax

```
Security::getPermissions($table)
```

Arguments

$table

the name of the table.

Return value

Returns an array with the table permissions.

Example

This example shows how to output the current user data:

```
$table = "table1";
print_r (Security::getPermissions($table));
```

Sample output

```
Array
(
    [A] => 1
    [E] => 1
    [D] => 1
    [S] => 1
    [P] => 1
    [I] => 1
)
```

See also:

- Security API: setPermissions
- Event: GetTablePermissions
- User group permissions
- About Security API

setPermissions

Sets permissions for a certain table for the current user. Permissions need to be set only once per user session, i.e., in the After Successful Login event.

Permissions should be passed in the form of an array where the keys are specific permission letters:

- **A** - add,
- **D** - delete,
- **E** - edit,
- **S** - search/list,
- **P** - print/export,
- **I** - import,
- **M** - admin permission. When advanced permissions are in effect (e.g., **Users can see/edit their own records only**), this permissions grants access to all records.

Syntax

```
Security::setPermissions($table, $rights)
```

Arguments

$table

the name of the table.

$rights

an array where the keys are specific permission letters.

Return value

No return value.

Example 1

Enable the *Add* and disable *Delete* functionality for the "*Cars*" table for the current user:

```
$rights = Security::getPermissions("Cars");
$rights["A"] = true;
$rights["D"] = false;
Security::setPermissions("Cars", $rights);
```

Example 2

Check if the current user has *Add* permissions for the "*Cars*" table:

```
$rights = Security::getPermissions("Cars");
if($rights["A"])
echo "add permission available";
```

See also:

- Security API: getPermissions
- Event: GetTablePermissions
- User group permissions
- About Security API

setAllowedPages

Specify which pages the current user can access.

The **setAllowedPages** function should be called before the user opens the page it affects. Therefore, it is best suited for AfterTableInitialized/AfterAppInit and After successful Login events.

Syntax

```
Security::setAllowedPages( $table, $pageType, $pages);
```

Arguments

$table {string}

the table that the page belongs to.

> **Note**: table names are case-sensitive and should be written exactly as they appear in PHPRunner. E.g., *"dbo_Cars"* can not be substituted with neither *"Cars"* nor *"dbo_cars"*.

When calling this function from the **AfterTableInitialized** event, use the **$table** variable.

Use the **GLOBAL_PAGES** constant to manage common pages - the pages that don't belong to any table: menu, login, register, etc.

$pageType {string}

one of the page types. The available page types are: list, add, edit, view, print, report, rprint, chart, dashboard, login, menu, register.

> **Note**: you can find the page type on the Page Designer screen in the page properties section:

$pages {array or string}

a page name or an array of page names the user should have access to.

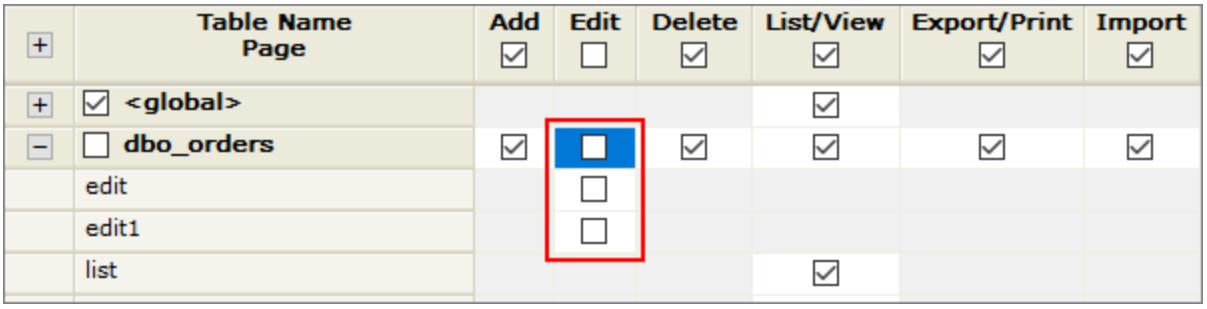
Return value

No return value.

Remarks

The **setAllowedPages** function does not affect the page types the user has no access to.

If, for example, access to **Edit** pages for the current user is prohibited on the Permissions page:

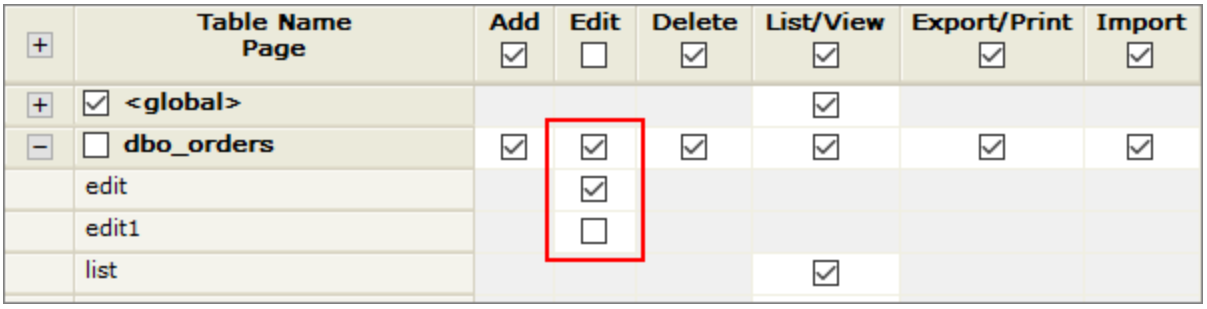


A call to **setAllowedPages** will not change anything. In this case, you can use this method together with the setPermissions function:

```
$rights = Security::getPermissions( "dbo_Orders" );
$rights["E"] = true;
Security::setPermissions( "dbo_Orders", $rights);
Security::setAllowedPages( "dbo_Orders", "edit", "edit1" );
```

On the other hand, **setAllowedPages** has priority over access rights to individual pages set on the Permissions page.

If, for example, the current user only has access to the **edit** page:



This code changes the page displayed to the user from **edit** to **edit1**:

```
Security::setAllowedPages( "dbo_Orders", "edit", "edit1" );
```

Example 1

Here are some basic examples of the **setAllowedPages** function.

After Table Initialized:

```
//allow access to the edit 1 page for the current user
Security::setAllowedPages( $table, "edit", "edit1" );
```

After App Initialized:

```
//allow access to edit and edit 1 pages for the current user
Security::setAllowedPages( "dbo_Cars", "edit", array( "edit", "edit1") );

//allow access to the menu1 page for the current user
Security::setAllowedPages( GLOBAL_PAGES, "menu", "menu1" );
```

Example 2

Show an alternative *Welcome* page, **menu1**, to a user named *mike*.

After successful login event:

```
if( Security::getUserName() == "mike" ) {
Security::setAllowedPages( GLOBAL_PAGES, "menu", "menu1" );
}
```

Example 3

You can call this function in other events than AfterTableInitialized/AfterAppInit and After successful Login.

Let's say you want the current user to gain access to the **edit1** page of the *Cars* table, use the following code before the user tries to access the **Edit** page:

```
// This code should not be put into the Cars - Edit page event
// Although Cars - List page events are fine
Security::setAllowedPages( "Cars", "edit", "edit1" );
```

See also:

- Security API: getPermissions
- Security API: setPermissions
- Security API: getUserName
- Security screen
- User group permissions
- Page Designer: Common pages
- Table pages
- Page Designer
- About Security API

Password hashing

You can hash your password manually using events.

Let's say that you want to provide an admin with direct access to the *Login* table. To do so, add the following code to the BeforeAdd event of the *Login* table:

For BCRYPT:

```
$values["password"] = getPasswordHash($values["password"]);
```

For MD5:

```
$values["password"] = md5($values["password"]);
```

Editing is a bit trickier and there are several approaches available and here is one of them.

On the **Edit** page, the password field is empty by default. If you do not want to change the password - leave it empty. If you want to change it - enter a new password. This will require us to implement the following events:

Edit page: Process record values event

```
$values["password"]="";
```

Edit page: BeforeEdit event

MD5:

```
if ($values["password"]!="")
 $values["password"] = md5($values["password"]);
else
 unset($values["password"]);
```

BCRYPT:

```
if ($values["password"]!="")
 $values["password"] = getPasswordHash($values["password"]);
else
 unset($values["password"]);
```

See also:

- Event: ProcessValues<PageName>
- Events: Before record updated
- Security: Encryption
- About Security API

3.2.13 SQLQuery class

3.2.13.1 About SQLQuery class

SQLQuery class allows you to modify the current SQL query stored in the *$query* object. The *$query* object is available only in the After table initialized event.

All the methods described below should be used in the After table initialized event. This ensures that the changes made are applied to all pages.

Methods

Method	Description
addWhere()	Adds a WHERE clause to the current SQL query.
replaceWhere()	Replaces the WHERE clause of the current SQL query.
addField()	Adds a field name to the end of the SELECT clause of the current SQL query.
deleteField()	Removes a field name from the SELECT clause of the current SQL query.
replaceField()	Replaces a field name in the SELECT clause of the current SQL query with a new one.

See also:

- Database API:Query()
- Event: BeforeQuery<PageName>
- Event: ListQuery
- Dynamic SQL query
- About SQL query screen
- SQL tab

3.2.13.2 Methods

addField

Adds a field name to the end of the SELECT clause of the current SQL query.

> **Note**: SQLQuery class methods are used in the After table initialized event.

Syntax

```
addField($calculatedField, $alias)
```

Arguments

`$calculatedField`

the name of the field. Example: "*size*".

`$alias`

an alias of the field name. Example: "*new_size*".

Note:*$alias* field values can be used in the following events:

- Before record processed, After record processed for the **List** page like *$data[$alias]*;
- Process record values, Before display for the **Edit** page like *$values[$alias]*;
- Copy page: Onload, Before record added, After record updated, Process record values for the **Add** page like *$values[$ alias]*;
- Process record values, Before display for the **View** page like *$values[$alias]*.

Return value

No return value.

Example 1.

To add *now()* as '*current_date*' to the end of the SELECT clause:

```
$query->addField("now()", "current_date");
```

Example 2.

To add *'size'+'adjust'* as '*new_size*' to the end of the SELECT clause:

```
$query->addField("size+adjust", "new_size");
```

See also:

- SQLQuery class: deleteField
- SQLQuery class: replaceField
- Grid Row JavaScript API: row.getFieldValue()
- Event: AfterTableInit
- SQL tab
- Table events
- About SQLQuery class

addWhere

If the current SQL query does not include a WHERE clause, *addWhere()* adds it as *where($condition)*. Otherwise, the WHERE clause is added as a new condition: *and($condition)*.

> **Note**: SQLQuery class methods are used in the After table initialized event.

Syntax

```
addWhere($condition)
```

Arguments

`$condition`

any condition clause. Example: *"id_sizes < 3 or id_sizes > 6"*. *id_sizes* is the name of the field.

Return value

No return value.

Example

Add a WHERE clause:

```
$query->addWhere("id_sizes < 3 or id_sizes > 6");
$query->addWhere("notes='".$_SESSION["UserID"]."'");
$query->addWhere("test ='passed'");
```

See also:

- SQLQuery class: replaceWhere
- Security: Additional WHERE tabs
- Example: Dynamic SQL query
- Event: AfterTableInit
- About SQLQuery class

deleteField

Removes the name of the field from the SELECT clause of the current SQL query.

> **Note**: SQLQuery class methods are used in the After table initialized event.

Syntax

```
deleteField($field)
```

Arguments

`$field`

the name of the field. Example: "*size*".

Return value

No return value.

Example

Remove the name of the field from the SELECT clause (the data from the field '*id_sizes*' is no longer requested):

```
$query->deleteField("id_sizes");
```

Note: use the ***deleteField*** function to delete the alias field created by the addField function.

See also:

- SQLQuery class: addField
- Event: AfterTableInit
- SQL tab
- About SQLQuery class

replaceField

Replaces a field name in the SELECT clause of the current SQL query with a new one.

Note: SQLQuery class methods are used in the After table initialized event.

Syntax

```
replaceField($replaceableField, $calculatedField, $alias)
```

Arguments

`$replaceableField`

a field name to be replaced. Example: "*size*".

`$calculatedField`

a new field name. Example: "*new_size*".

`$alias`

an alias of the new field name.

> **Note**: the *$alias* parameter can be omitted. In this case, *$replaceableField* value is taken as alias.

Return value

No return value.

Example 1

Replace the *'name'* with the *'englishName'* as *'name'*:

```
$query->replaceField("name", "englishName");
```

Example 2

Replace the *'price'* with the *'price*1.2'* as *'price'*:

```
$query->replaceField("price", "price*1.2");
```

See also:

- SQLQuery class: addField
- SQLQuery class: deleteField
- SQL tab
- Event: AfterTableInit
- About SQLQuery class

replaceWhere

Replaces the WHERE clause of the original SQL query with a new one.

> **Note**: SQLQuery class methods are used in the After table initialized event.

Syntax

```
replaceWhere( $condition)
```

Arguments

$condition

any conditional clause. Example: "*id_sizes < 3 or id_sizes > 6*". *id_sizes* is the field name.

Return value

No return value.

Example 1

Display records from the table where the value of the *test* field is *'passed'*:

```
$query->replaceWhere( "test = 'passed'" );
```

Example 2

Display records from where the value of the *size* field is 3 and the value of the *adjust* field is less than 2:

```
$query->replaceWhere( "size = 3 and adjust < 2" );
```

Example 3

Display all table records:

```
$query->replaceWhere( "" );
```

See also:

- SQLQuery class: addWhere
- Event: AfterTableInit
- Security: Additional WHERE tabs
- About SQLQuery class

3.2.14 PDF API

3.2.14.1 About PDF API

PDF API is used for manipulating PDF files. You can use this API in any JavaScript event.

You can apply it in the following cases:

- Creating a PDF, asking for user's email, emailing the PDF file to that email address;
- Saving a PDF on the hard drive under the name requested from the user;
- Creating a PDF and saving it in the database;
- Emailing selected records as separate PDF files.

Methods

Method	Description
Runner.PDF.open	Creates and opens a PDF file.
Runner.PDF.download	Downloads a PDF file.
AJAX helper object: ajax.addPDF	Creates a PDF file and sends it to the server.

Examples:

- How to create a PDF and save it to the disk
- How to save the View page as PDF from the List page
- How to email the current page as PDF
- How to email selected records as separate PDF files
- How to create PDF from all selected records except the first one

See also:

- PDF Parameters
- PDF view settings

- Events: JavaScript OnLoad
- About JavaScript API

3.2.14.2 PDF Parameters

Description

PDF parameters (*params*) are objects that determine how a PDF file is created.

PDF parameters are used in:

- Runner.PDF.open
- Runner.PDF.download
- AJAX helper object: ajax.addPDF

All individual parameters are optional and are used only if specified. Otherwise, the default values are used.

You can specify any of the parameters by adding the following expression to your code.

Example

Create and download View page for the table "*cars*" with an Id of 5.

```
var params = {
  table: 'cars',
  page: 'view',
  keys: '5'
}
Runner.PDF.download( params );
```

Data source parameters

`table {string}`

Defines the table or report name to be converted to PDF.

Default value: current table/report.

Note: when creating a PDF file based on a different table than the one where the button is placed, you must specify the *pageType* parameter as well.

For example, when you want to create a PDF based on the "*Orders*" table from a button added to the "*Customers*" page:

```
var params = {
table: 'Orders',
page: 'print',
}
```

pageType {string}

Defines the page type to be converted to PDF. This parameter becomes mandatory when the *table* parameter is specified.

Possible values are **view** and **print**:

- **view** - View page;
- **print** - Print page.

Default values:

- **print** in reports;
- **view** if the code is run on the View page;
- **print** in all other cases.

page {string}

Defines the page name. Default value corresponds to the default page specified with the *pageType*.

scope {number}

Defines which data should be added to the PDF file. This setting applies to the Print page only.

Possible values:

- **0** - this page only. Creates a PDF file with the same data you see on the current page. On the List page, this option uses settings selected on the Print panel;
- **1** - all pages;
- **2** - selection only. When on the List page, only the selected records are added to the PDF file.

Default value: 0.

`split {number}`

Specifies how to split the PDF file into pages. This setting applies to the Print page only.

Possible values:

- **0** - auto, fit as many records as possible into the page;
- **1** - fixed number of records. Specify how many records on each page you want to see by using the *records* option;
- **2** - read number of records from the printer panel.

Default value: 0

`records {number}`

When the *split* option value is 1 (fixed), this option determines how many records to show on each page.

Default value: 10

`selection {array or arrays}`

Defines which records to add to the PDF file. The value should be an array of arrays even if there is only one key field in the table: [[1], [2], [10]].

No default value.

`keys {array}`

When creating a PDF based on the View page, specify key field values here. This parameter is not needed when the code is added to the same View page.

No default value.

Example:

```
var params = {
pageType: 'view',
keys: ['10']
}
```

Document appearance options

`filename {string}`

Sets the PDF file name. Default value: 'file.pdf'.

`orientation {string}`

Sets the PDF orientation.

Possible values:

- **portrait**
- **landscape**

Default value: portrait.

`scale: {number}`

Makes the PDF document bigger or smaller. The **50** scale makes it 2 times smaller than the default **100** scale. You can adjust this parameter as you see fit. Default value: 100.

`backgroundImage: {string}`

Sets an image as the background of the PDF file. Specify the filename here. The file is read from the *<application root>/images* directory.

For example, if you specify:

```
backgroundImage: 'pdf/logo.jpg'
```

Then *image/pdf/logo.jpg* is used.

Only PNG and JPEG images can be added to PDF document. The image is resized to fit to the entire document page.

No default value.

`backgroundOpacity: {number}`

Defines the opacity level, or how transparent the background image should be.

Possible values range from **100** (fully opaque) to **0** (fully transparent). Default value: 100.

See also:

- Example: Creating PDF from all selected records except the first one
- Example: How to email selected records as separate PDF files
- PDF view settings
- About PDF API

3.2.14.3 Runner.PDF.open

Runner.PDF.open creates and opens PDF files. Files can be opened in the same browser tab or in a new one.

> **Note**: Microsoft Edge and Internet Explorer do not support the file preview feature. Files are downloaded instead.

Syntax

```
Runner.PDF.open( {}, pageObj, window );
```

Arguments

params

PDF parameters. Can be empty. Use {} for default settings.

pageObj

RunnerPage object. Available as a variable in all JavaScript events.

window

an optional **JavaScript Window object** used for opening a PDF file in the current browser tab. If omitted, the file is opened in a new browser tab. To open the file in the current browser tab, pass *window*.

Return value

No return value

Example

```
// when a new browser tab is created
Runner.PDF.open( {}, pageObj );

//when the file is opened in the current browser tab
Runner.PDF.open( {}, pageObj, window );
```

See also:

- Example: Creating PDF from all selected records except the first one
- PDF API: Runner.PDF.download
- JavaScript API: RunnerPage object
- JavaScript onload events
- About PDF API

3.2.14.4 Runner.PDF.download

Runner.PDF.download generates a PDF file and and prompts the file to download. It also determines the name of the file.

This method can be used in all JavaScript events.

Syntax

```
Runner.PDF.download ( {}, 'file.pdf', pageObj );
```

Arguments

params

PDF parameters. Can be empty. Use {} for default settings.

filename

the name of the file, for example, *file.pdf*.

pageObj

RunnerPage object. Available as a variable in all JavaScript events.

Return value

No return value

Example

```
Runner.PDF.download( {}, 'file.pdf',  pageObj );
return false;
```

See also:

- Example: Creating and saving a PDF file to disk
- PDF API: Runner.PDF.open
- JavaScript API: RunnerPage object
- JavaScript onload events
- About PDF API

3.2.14.5 ajax.AddPDF

Description

The **ajax.addPDF** method creates a PDF file and sends it to the Server event. The file can be emailed, saved to the disk or into the database.

Syntax

```
ajax.addPDF( name, settings, callback )
```

Arguments

`name {string}`

the name of the variable in which the PDF file contents are stored in the *$params* server array, so that it is available to use in the Server event.

`settings {object}`

the PDF settings. See PDF Parameters for details. Use an empty object {} for default settings.

`callback`

defines which callback function is called after the PDF file is created. You should call either the submit() or dialog() function in the callback to start the **Server** part of the event.

> **Note**: the *submit()* function is applied to start the **Server** part of the event when the basic routine cannot be used. It is necessary, when the Client Before event runs an asynchronous task, e.g., creating a PDF.
>
> See submit() to learn more.

Return value

No return value.

Examples

Example 1

This example shows the use of the name argument.

Client before:

```
ajax.addPDF( 'pdf', {} );
return false;
```

Note: when using the **addPDF** fucntion in the **Client Before** event, you should always return *false*. Just add the following line to the end of your event code:

```
return false;
```

Server:

```
file_put_contents( "file.pdf", $params["pdf"] );
```

Note: see Tri-part events to learn more about **Client before** and **Server** events.

Example 2

This example shows the use of the callback argument in the **ClientBefore** event.

```
ajax.addPDF( "pdf", {}, function() {
   ajax.submit();
```

```
});
return false;
```

See also:

- Example: How to create a PDF and save it to the disk
- Example: How to save the View page as PDF from the List page
- Example: How to email the current page as PDF
- Example: How to email selected records as separate PDF files
- Tri-part events
- About Dialog API
- About AJAX helper object
- About PDF API

3.2.14.6 Examples

Creating and saving a PDF file to disk

This example shows how to create a PDF file and save it into the ouput folder.

Client before:

First, we create a PDF file. Then, we make a dialog appear on the web page where a user can specify the filename.

```
ajax.addPDF( 'pdf', {}, function() {

   return ctrl.dialog( {
      title: 'Save results as PDF',
      fields: [{
         name: 'filename',
         value: 'file.pdf'
      }]
   });
});
return false;
```

Server:

After the user entered the filename, we transfer it to the **Server** event.

The server code checks the entered filename so that the file is saved in the correct format. In essence, the file is saved with a correct extension, whether the *.pdf* extension was specified by the user or not.

After the name check, the file is saved to the output folder.

```
$filename = str_replace( array( '/', '\\'), '', $params["filename"]);

$dot = strrpos( $filename, "." );
if( strtolower(substr( $filename, $dot )) !== '.pdf' ) {
   $filename .= '.pdf';
}
file_put_contents( $filename, $params["pdf"] );
```

Client after:

After the file is saved to the disk, it is opened in a new browser window.

```
window.open( params['filename'] );
```

Note: the **Client after** is optional. Add it if you want to open the downloaded file in a new window.

See also:

- AJAX helper object: ajax.addPDF
- PDF Parameters
- About Dialog API
- Tri-part events
- About PDF API

Creating PDF from all selected records except the first one

In this example, we create a PDF file from all selected records except the first one.

First, we need to specify the *selection* parameter.

Then, we use the **RunnerPDF: open** method.

Client before:

```
var params = {
   selection: pageObj.getSelectedRecordKeys().slice(1)
};
Runner.PDF.open( params, pageObj );
return false;
```

See also:

- PDF Parameters
- RunnerPDF: open
- Tri-part events
- About PDF API

Creating PDF of the current record's View page

Insert a button into the data grid on the **List** page. This button will create and open a PDF file of the current record's View page.

Add the following code to the Client before event of the button:

```
var params = {
   pageType: 'view',
   keys: row.getKeys()
};
Runner.PDF.open( params, pageObj );
return false;
```

Leave the **Server** and **Client after** events empty.

See also:

- PDF API: Runner.PDF.open
- Insert custom button
- PDF Parameters
- Tri-part events
- About PDF API

Emailing the current page as PDF

This example shows how to email the current page as PDF.

Change the red values to adjust the code to your project.

Client before:

First, we create a PDF file. Then, we make a dialog appear on the web page. With this dialog, the user can specify the *filename,* and the email address to which the PDF file should be sent.

```
ajax.addPDF( 'pdf', {}, function() {
   return ctrl.dialog( {
      title: 'Email this page',
      fields: [{
         name: 'email',
         value: 'email@address.com'
      },
      {
         name: 'filename',
         value: 'results.pdf'
      }]
   });
});
return false;
```

Server:

In the **Server** event, we create a temporary PDF file.

Next, we create a *$mail* array. The email parameters are placed into this array.

You can modify the email subject - *$mail["subject"]*, and the email body (message) - *$mail["body"]*.

After the email is sent, the result is returned to the "**Client after**":

- *$result "success" = true* - everything is fine, the message is sent.
- *$result "success" = false* - the message is not sent. In this case, an error message defined in the *$result* variable appears.

```
$path = $button->saveTempFile( $params["pdf"] );

$mail = array();
$mail["to"] = $params["email"];
$mail["subject"] = "PHPRunner PDF and Email demo";
$mail["body"] = "Check the results";
$attachment = array(
   "path" => $path,
  "name" => $params["filename"]
);
$mail["attachments"] = array( $attachment );

$ret = runner_mail( $mail );

if( $ret["mailed"] )
{
   $result["success"] = true;
}
else
{
   $result["message"] = $ret["message"];
  $result["success"] = false;
}
```

Client after:

The user gets a notification with the result.

```
if( result.success ) {
   ctrl.setMessage("sent ok");
} else {
```

```
    ctrl.setMessage("error sending " + result.message );
}
```

See also:

- AJAX helper object: ajax.addPDF
- PDF Parameters
- About Dialog API
- Tri-part events
- About PDF API

Emailing selected records as separate PDF files

This example shows how to email selected records as separate PDF files.

Change the red values to adjust the code to your project.

Client before:

First, we need to get information about the user-selected records, since the PDF files are created from these records. If no entries are selected, the code execution ends.

Then, we make a dialog appear on the web page. Using this dialog, the user can specify an email recipient's address, email subject and body.

We also set the parameters of the the PDF pageType to '**View**'.

```
var pdfParams = {},
    selectedRecords = pageObj.getSelectedRecords();

if( selectedRecords.length == 0 )
    return false;

params.recordCount = selectedRecords.length;
params.filenames = [];
```

```
var createOnePdf = function( idx ) {
   ajax.addPDF( 'pdf'+idx, pdfParams[idx], function() {
      delete pdfParams[idx];
      if( Object.keys( pdfParams ).length == 0 ) {
         showDialog();
      }
   });
}

var showDialog = function() {
  ctrl.dialog( {
   title: 'Email ' + params.recordCount + ' PDF files',
   fields: [
      {
         name: 'email',
         value: 'email@address.com'
      },
      {
         name: 'subject',
         value: 'Check out this data'
      },
      {
         name: 'body',
         value: 'This email is generated by Runner-created application',
         type: 'textarea'
      },
      ]
   });
}

selectedRecords.forEach( function( ajaxRow, idx ) {
   pdfParams[ idx ] = {
      pageType: 'view',
      records: ajaxRow.getKeys()
   };
   params.filenames.push( '_viewpage_id'+ajaxRow.getKeys()+'.pdf')
   createOnePdf( idx );

});

return false;
```

Server:

In the **Server** event, we create temporary PDF files for each record. After that, we send the email. The result is returned to the **Client after** event:

- *$result "success" = true* - everything is fine, the message is sent.
- *$result "success" = false* - the message is not sent. In this case, an error message defined in the *$result* variable appears.

```
// save generated files to disk and create an array of attachments

$attachments = array();
for( $i=0; $i< $params["recordCount"]; ++$i ) {
   $attachments[] = array(
      "path" => $button->saveTempFile( $params["pdf" . $i] ),
      "name" => $params["filenames"][$i]
   );
}
$mail = array();
$mail["to"] = $params["email"];
$mail["subject"] = $params["subject"];
$mail["body"] = $params["body"];
$mail["attachments"] = $attachments;

$ret = runner_mail( $mail );

if( $ret["mailed"] )
{
   $result["success"] = true;
}
else
{
   $result["message"] = $ret["message"];
$result["success"] = false;
}
```

Client after:

The user gets a notification with the result.

```
if( result.success ) {
   ctrl.setMessage("sent ok");
} else {
   ctrl.setMessage("error sending " + result.message );
}
```

See also:

- AJAX helper object: ajax.addPDF
- PDF Parameters
- About Dialog API
- Tri-part events
- About PDF API

3.2.15 Troubleshooting tips

3.2.15.1 Troubleshooting charts

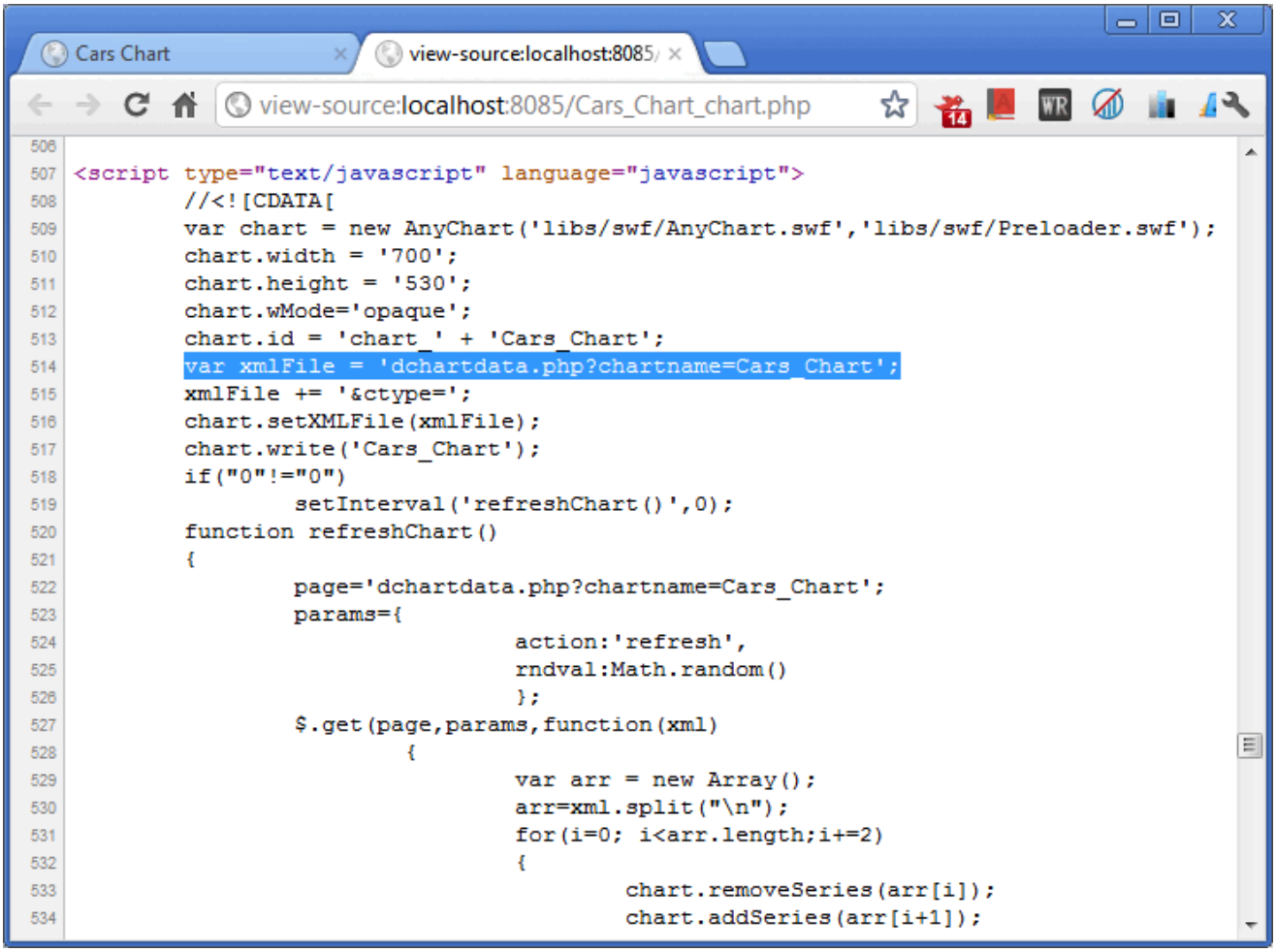
In this tutorial, we will find out how to troubleshoot chart errors in our applications. PHPRunner applications use a Flash-based charting component that receives data in an XML format.

If any error occurs while generating XML input you will see the following message instead of chart:

XML Parser failure:
The element type must be terminated by the matching end-tag.

Lets go a bit deeper. Right click anywhere on the page (except on chart itself) and choose **View source**. A new tab opens with the source code for the page.

On the *source* page, search for the first occurrence of *dchartdata.php?* string. You should see something like this:



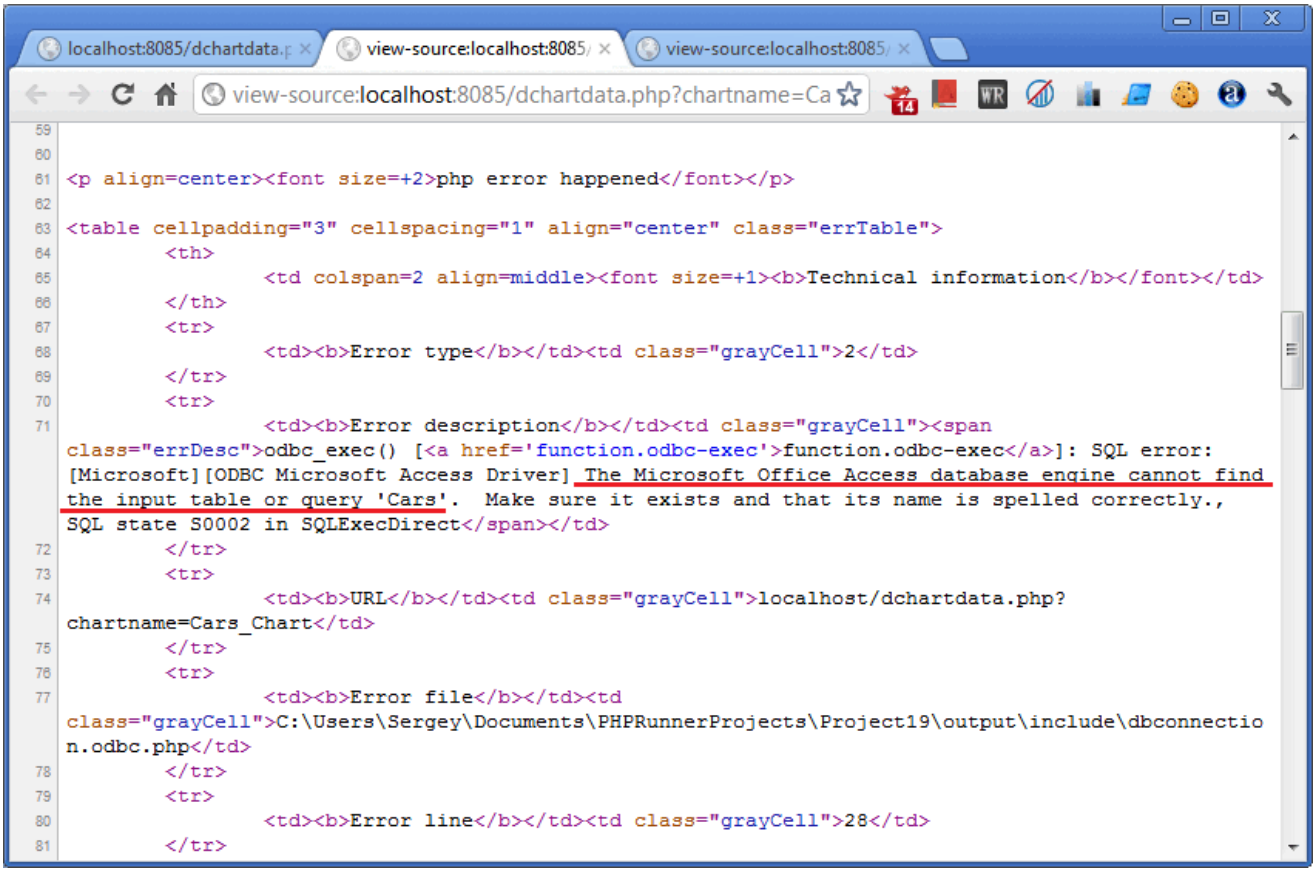
Highlight *dchartdata.php?chartname=Cars_Chart* part, right click on it and choose **Copy**.

Now paste it to the browser address bar replacing *Cars_Chart_chart.php*. URL is supposed to look like this:

http://yourwebsite.com/dchartdata.php?chartname=Cars_Chart. Press **Enter**.

dchartdata file generates XML. As we can see, something went wrong and this file produces an error.

View the *source* of this page again. Search for *php error happened* string.



Now we can see the actual error message. In this specific case, the *'Cars'* table is missing from the database.

There can be several reasons for such an error: the table might be removed or renamed after the project was built. To fix this, you either need to change the chart definition in your project or rename this table back.

See also:

- Creating charts
- Datasource tables: Renamed/deleted tables
- Cars template
- Error reporting
- Troubleshooting tips

3.2.15.2 Troubleshooting custom buttons

Let's say you have a cars database. You have added a button to the **List** page that should update the selected cars statuses as *'Sold'*.

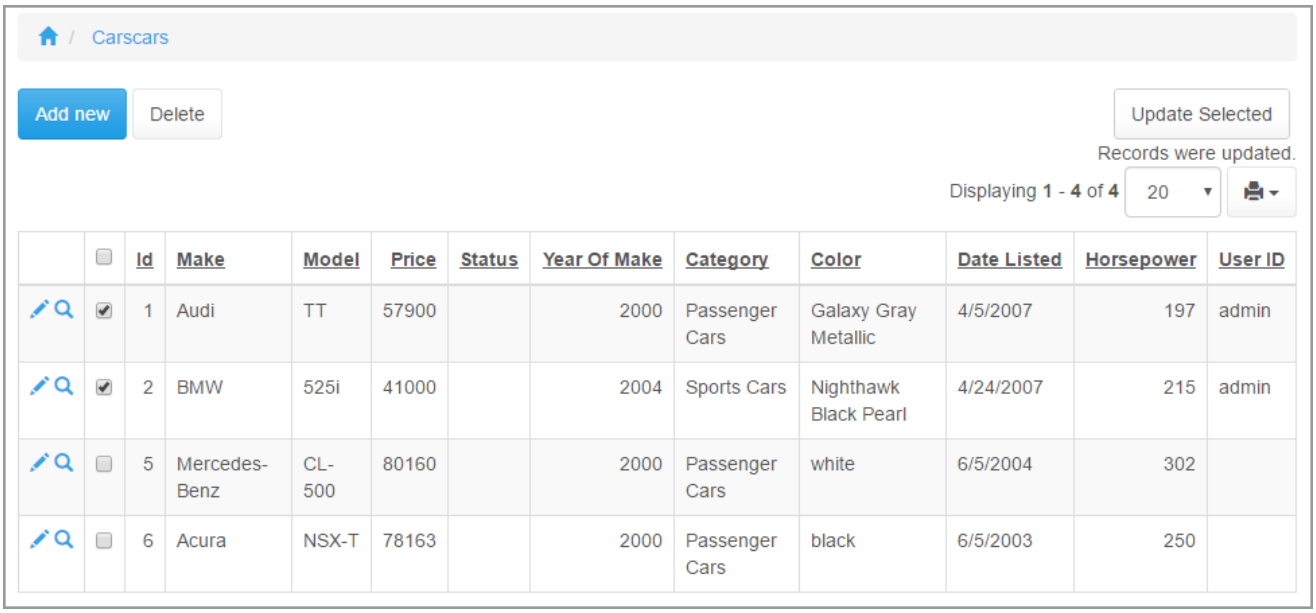
Your code looks good and passes the syntax check but still doesn't work when you run your application. What's worse - it doesn't produce any visible errors.

In this article, you can learn how to catch errors like the one above.

> **Note**: this article uses Google Chrome. Other browsers have similar options.

Using developer tools to troubleshoot custom buttons

Open your **List** page with the added button. You should see something like this:

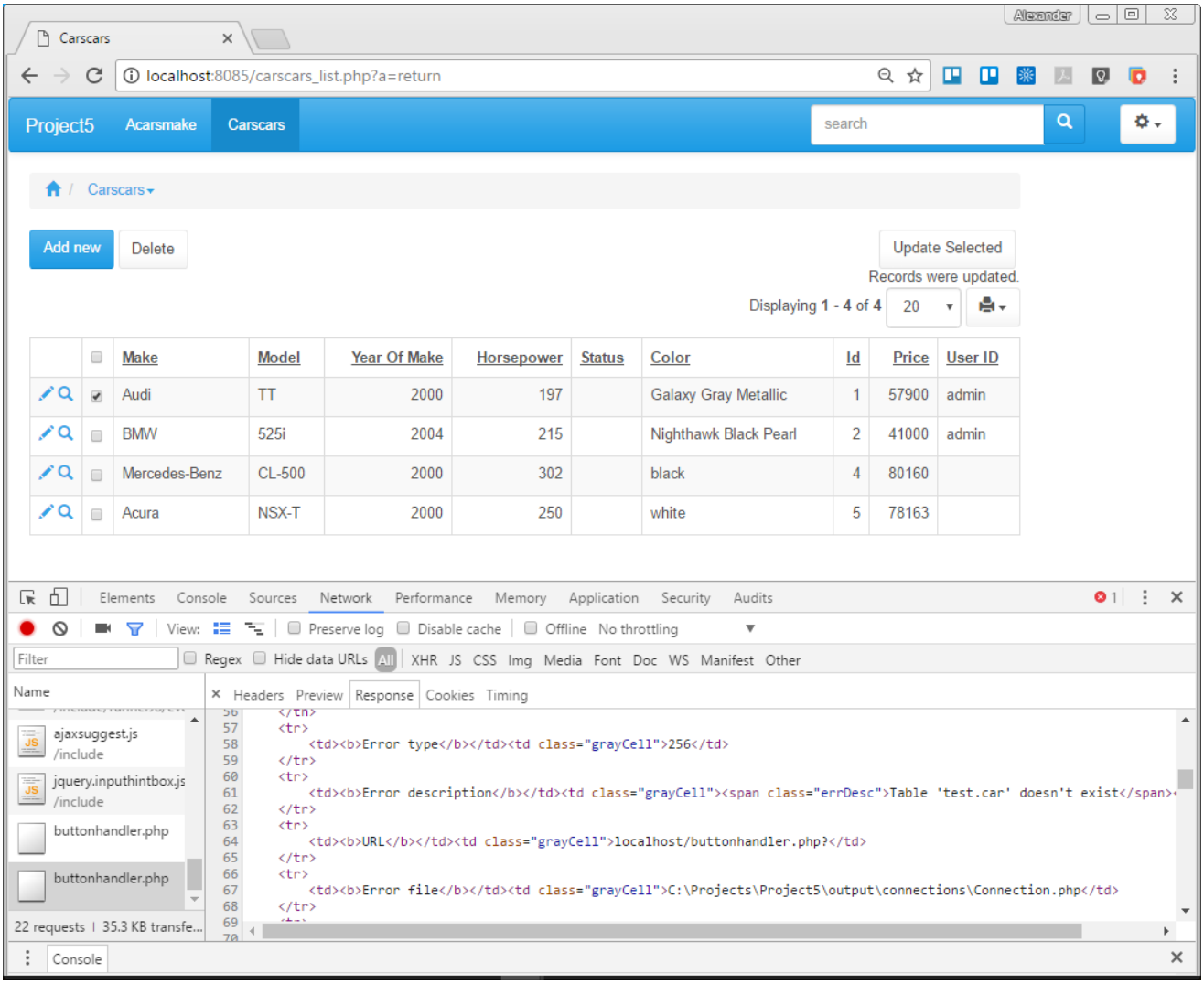


	Id	Make	Model	Price	Status	Year Of Make	Category	Color	Date Listed	Horsepower	User ID
	1	Audi	TT	57900		2000	Passenger Cars	Galaxy Gray Metallic	4/5/2007	197	admin
	2	BMW	525i	41000		2004	Sports Cars	Nighthawk Black Pearl	4/24/2007	215	admin
	5	Mercedes-Benz	CL-500	80160		2000	Passenger Cars	white	6/5/2004	302	
	6	Acura	NSX-T	78163		2000	Passenger Cars	black	6/5/2003	250	

Modern browsers provide developer tools. Hit *F12* to display the developers tools panel and proceed to the *Network* tab.

With the developer tools panel open on the **List** page, select a few records and click **Update Selected**. A new entry appears under the *Network* tab: browser executes *buttonhandler.php* file where the server side code is stored.

If you expand the *Response* tab, you can see the error description if there is any:



In this case, it says the following under the *Error description*:

"Table 'test.car' doesn't exist."

It looks like we have misspelled the table name in our code. Replacing *'car'* with *'carscars'* fixes the issue.

This can definitely be helpful, though some events can be hundreds lines of code long and a single error message can't always help.

To find the exact line of code that produces the error, scroll down the content of the *Response* tab to find the entry that points to *buttonhandler.php* file.

Here it is:

```
<td nowrap="nowrap">#4. </td>
<td nowrap="nowrap">buttonhandler.php:49</td>
<td nowrap="nowrap">buttonHandler_Update_Selected</td>
```

Now you can open the *buttonhandler.php* file in any text editor (Notepad++ recommended) and find the line 49:

In this case, line 49 contains:

```
CustomQuery($sql);
```

It points to the fact that something is not right with the SQL query.

Additional troubleshooting tips

You may want save a few troubleshooting steps by printing your SQL Queries on the web page instead of executing them. This way, you can see what exactly is happening on the server and catch any syntax errors faster.

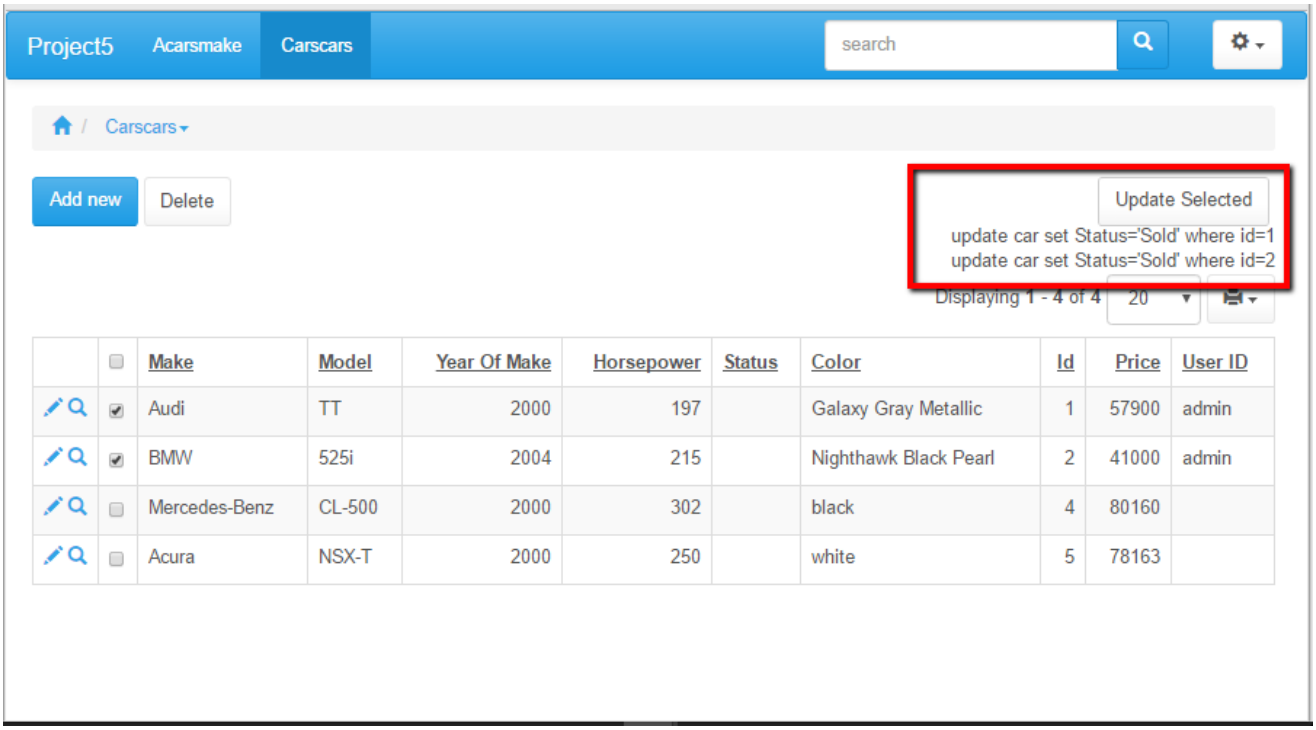
To do so, assign SQL queries to the *result["txt"]* variable and print it on the page using the following code in ClientAfter event.

```
ctrl.setMessage(result["txt"]);
```

Sample Server PHPRunner code:

```
$result["txt"]="";
foreach($keys as $idx=>$val)
{
$sql = "update car set Status='Sold' where id=".$val["ID"];
$result["txt"].=$sql."<br>";
//CustomQuery($sql);
}
```

And here is how this looks like in the app:



See also:

- Page Designer: Insert custom button
- Tri-part events
- About SQL query screen
- Error reporting
- Cars template

3.2.15.3 Troubleshooting Javascript errors

In this article, we are focusing on debugging and troubleshooting JavaScript errors. We will look at a few examples to illustrate the methods used.

We will be using developers tools that come with modern browsers.

> **Note**: this article uses Google Chrome. Other browsers have similar options.

Example 1

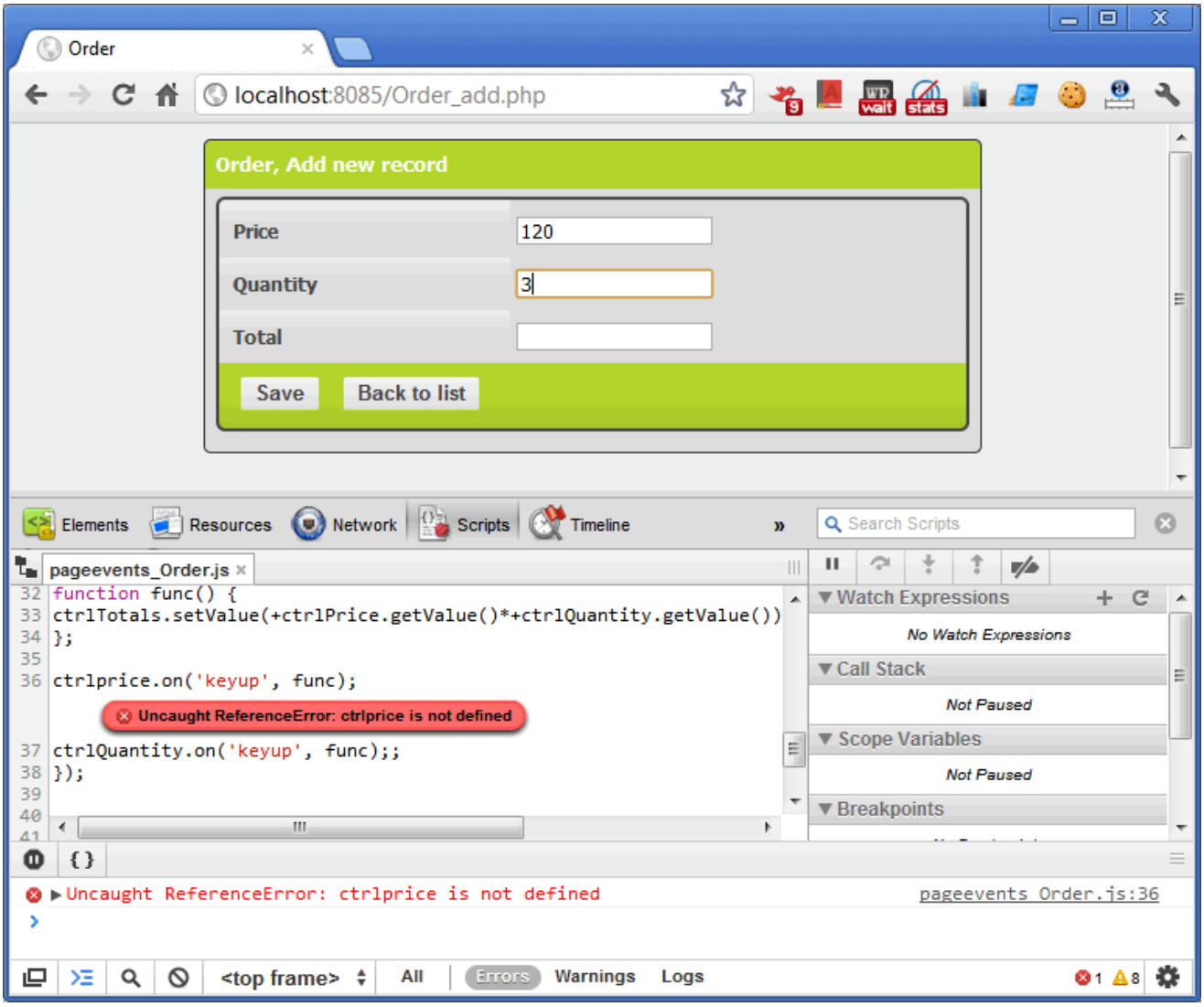
```
var ctrlPrice = Runner.getControl(pageid, 'Price');
var ctrlQuantity = Runner.getControl(pageid, 'Quantity');
var ctrlTotals = Runner.getControl(pageid, 'Total');

function func() {
ctrlTotals.setValue(+ctrlPrice.getValue()*+ctrlQuantity.getValue());
};

ctrlprice.on('keyup', func); //misspelled variable
ctrlQuantity.on('keyup', func);
```

Let's take a look at our first example where we intentionally misspelled the name of the variable in the JavaScript OnLoad event (*ctrlPrice* vs *ctrlprice*, variables are case sensitive). Since this is not a syntax error, the *Syntax Check* won't pick up on it but the browser will.

One thing to note here is when you are troubleshooting and you build your project, you might want to uncheck the **Compress javascript files** option on the Output screen. This will organize the code in a way that is much easier to follow.

So, when the page is loaded, press *F12* to display the developers tools panel.

We can see the error message there and by clicking it, we can get right to the line of code that causes the trouble. The error message in a popup says: '*ctrlprice is not defined*'. Replacing *ctrlprice* with the correct *ctrlPrice* fixes the issue.



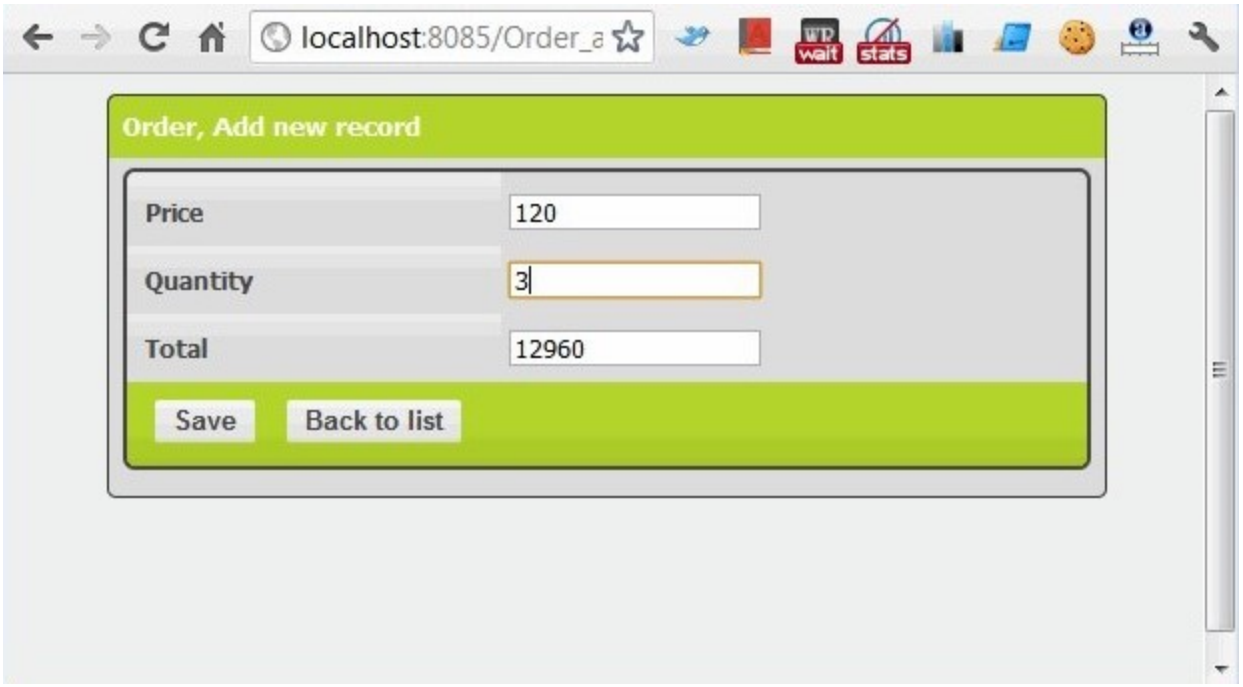
Example 2

Let's take a look at another example where we are calculating the amount on the fly and are not getting the anticipated result. We are calculating the order total by multiplying the number of units in the order by the price of the unit and adding a tax to the equation.

```
var ctrlPrice = Runner.getControl(pageid, 'Price');
var ctrlQuantity = Runner.getControl(pageid, 'Quantity');
var ctrlTotals = Runner.getControl(pageid, 'Total');

function func() {
var total=ctrlPrice.getValue()*ctrlQuantity.getValue();
```

```
ctrlTotals.setValue(total * total*0.1 );
};

ctrlPrice.on('keyup', func);
ctrlQuantity.on('keyup', func);
```

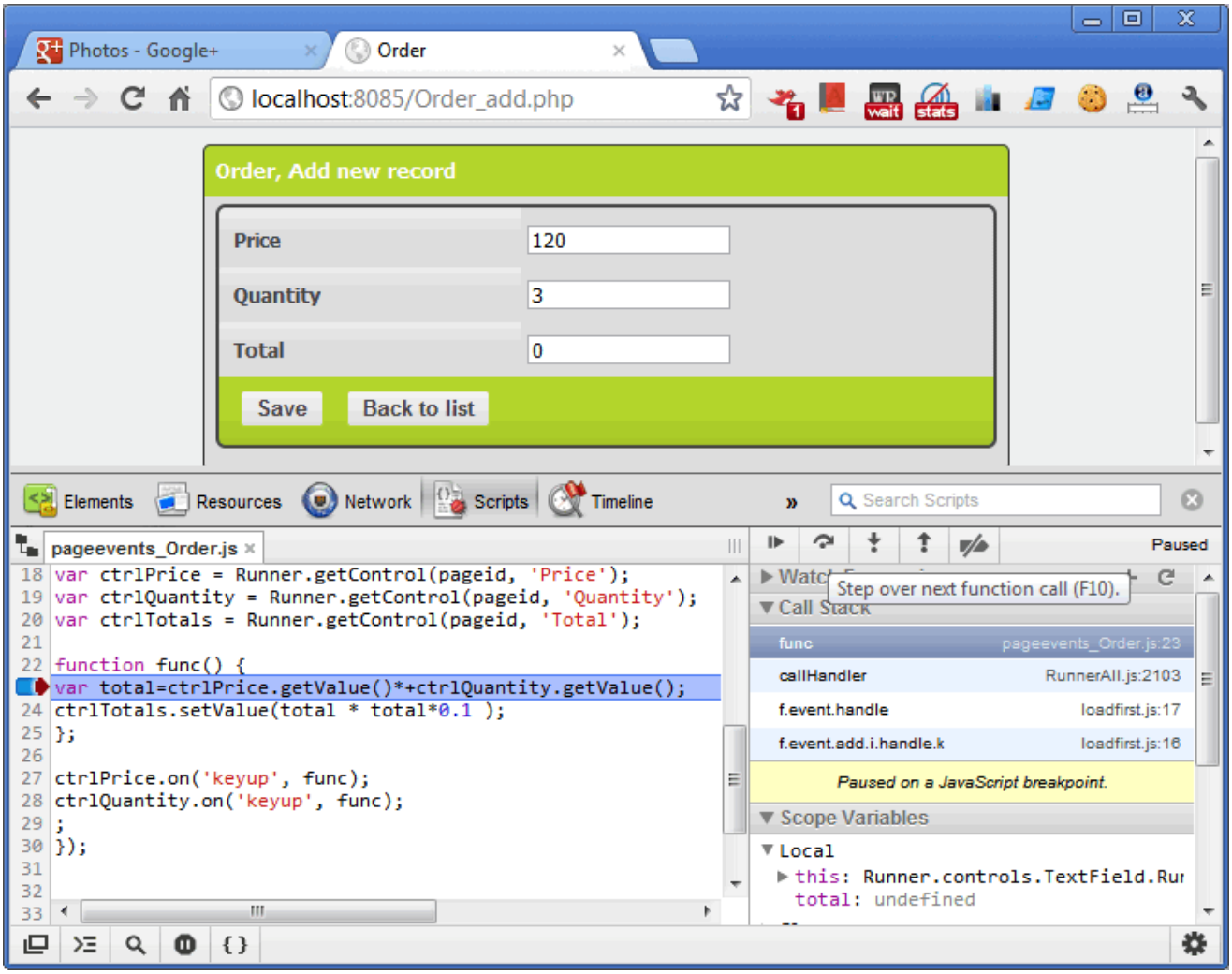
And here is the result it produces:



So, in the order where we have 3 units at $120 each with 10% sales tax, our total should be $396 dollars. However, the total we are getting is $12960.

To troubleshoot the issue we will insert a breakpoint in the JavaScript code and watch the values of each part of the equation separately to identify the error.

To add a breakpoint, we need to proceed to the *Scripts tab* in the developers tools panel, find our file (*pageevents_Order.js*), and click the line number we we want to set our breakpoint.



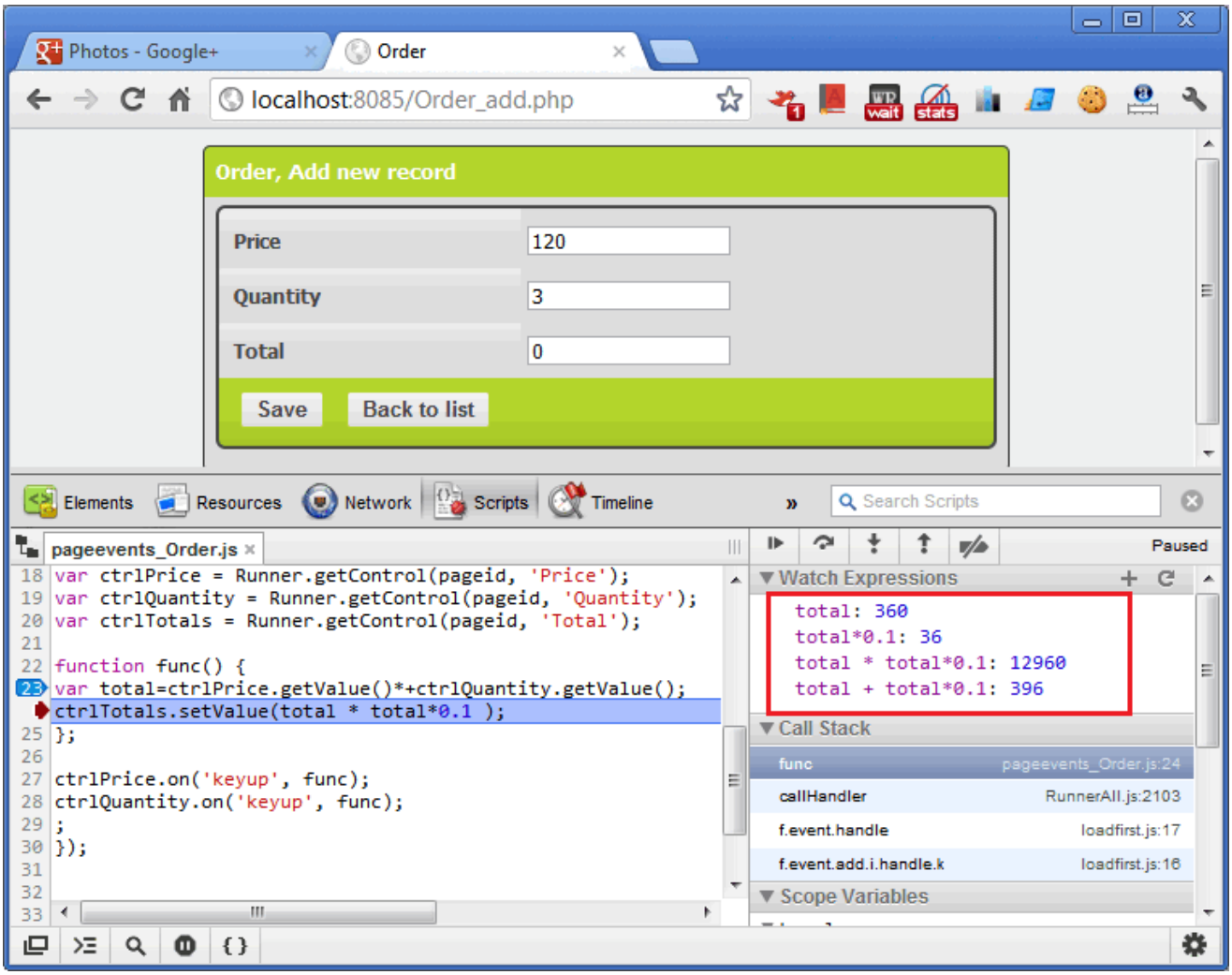
Note: in Google Chrome 78, the same tab is called *Sources*.

Once breakpoint is inserted, place the cursor on the *Quantity* field and press any arrow key on the keyboard. Program execution stops, and we can see what's going on. We can use *F10* key to move to the next line of code (*Step over*).

Now lets add a few watch expressions on the right side.

We can see that total and tax are calculated properly however we multiplying them instead of adding up which causes the error.

A simple correction in the equation logic solves the issue.



Real life examples are more complicated, though basic troubleshooting techniques are the same.

As a first step, make sure there are no runtime JavaScript errors then set a breakpoint and step through the code to find logic flaws.

You can also watch a video version of this tutorial.

See also:

- JavaScript API: Control object > getControl()

- JavaScript API: Control object > getValue()
- JavaScript API: Control object > setValue()
- JavaScript API: Control object > on()
- Output directory settings
- About JavaScript API
- Events: JavaScript OnLoad

3.2.15.4 Troubleshooting techniques

When your projects do not work as expected, especially when you use lots of events and custom code, you can use the following debugging techniques to resolve the issue.

Print all executed SQL statements on the web page

For this purpose, set the *$dDebug* variable in *include/appsettings.php* file to true.

It can be especially useful when you need to resolve master-details or advanced security issues.

```
$dDebug = true;
$dSQL = "";

$bUseMobileStyleOnly = false;
```

Another way is to add the following code to the AfterAppInit event:

```
$dDebug = true;
```

You can copy the SQL query and run it against your database manually to see if it returns correct results.

To run an SQL query manually, use the tools that come with the database (Query designer in MS Access, Enterprise Manager in SQL Server, phpMyAdmin in MySQL).

Print out variables using the echo statement

The *echo* command is used to write output to the browser. The following example sends the text "Hello World" to the browser:

```
echo "Hello World";
```

Here are several examples of how you can use this command:

Print the value of the variable

```
echo "Variable name: " . $variable;
```

With BeforeAdd/BeforeEdit events, you can print form field value

```
echo "field_name: " . $values["field_name"];
```

Note: visit the corresponding articles to learn more about BeforeAdd/BeforeEdit events.

Displaying a session variable that stores the authorized user's ID

```
echo "UserID: " . $_SESSION["UserID"];
```

Displaying the IP address of the authorized user

```
echo "Your IP address: " . $_SERVER['REMOTE_ADDR'];
```

Print all values in the array

```
echo "all entered values: ";
print_r($values);
```

See also:

- Troubleshooting charts

- Troubleshooting custom buttons
- Troubleshooting JavaScript errors
- Events: After application initialized
- Event: BeforeAdd
- Event: BeforeEdit
- Save user data in session variables
- About SQL query screen

3.2.16 Data formatting

The data is stored in a raw format in the database. For example, the *Date* type data is stored as *yyyy-mm-dd*. For pages like **List**, **View**, **Print**, you can customize data appearance by defining the View as settings in Page Designer.

Using the formatted data in events. ViewControl::Format function

If there is a need to use formatted data in events, for example, when sending emails to users, you can use the *ViewControl::Format* function.

Syntax

```
ViewControl::Format($data, $FieldName, $TableName);
```

Arguments

`$data`

raw data to be formatted.

`$FieldName`

the field that contains the data to be formatted.

`$TableName`

the table that contains the data to be formatted.

Example 1

Using the *ViewControl::Format* function in the AfterAdd event:

```
echo $values['Date Listed'] ."<br>";
echo ViewControl::Format($values, 'Date Listed', 'Cars' );
```

The first line of the code sample displays raw data (e.g., 2014-10-15), the second one - the formatted data (US date format, e.g., 10/15/2014).

Example 2

Data formatting for a button on a **View** page or a button inserted into the grid:

```
$data = $button->getCurrentRecord();
$var = ViewControl::Format($data, 'Date Listed', 'Cars');
```

This function can also be used for a Lookup wizard field. In this case, raw data is stored in the *Link* field, and the *ViewControl::Format* function displays the values of the *Display* field.

See also:

- Button object: getCurrentRecord()
- "View as" settings
- "Edit as" settings: Lookup wizard
- Events: AfterAdd

3.2.17 How to access fields in the field events

Access fields from ClientBefore, ClientAfter events

Variables available in the field events

`ctrl`

the current field, a Control object.

`pageObj`

a RunnerPage object.

Example:

Set the value of the current field to 100.

```
ctrl.setValue(100);
```

ctrl.getPeer(field)

Returns another field control from the same page of the same row in an **inline** mode:

```
var ctlPrice = ctrl.getPeer('price');
ctlPrice.setValue( 1000 );
```

pageObj.getControl(fieldName)

Returns the field control. Works in any events where the *pageObj* variable is available.

```
var ctlPrice = pageObj.getControl('price');
ctlPrice.setValue( 1000 );
```

Access fields from the Server events

`$result`

an array of values to return from the server.

`getCurrentRecord()`

returns an associative array with field values (field name => value).

```
$data = $ajax->getCurrentRecord();
$result["record"] = $data;
$result["email"] = $data["email"];
```

See also:

- JavaScript API: Date Control API > setValue()
- JavaScript API: RunnerPage object
- JavaScript API: Control object
- Field Events
- Tri-part events

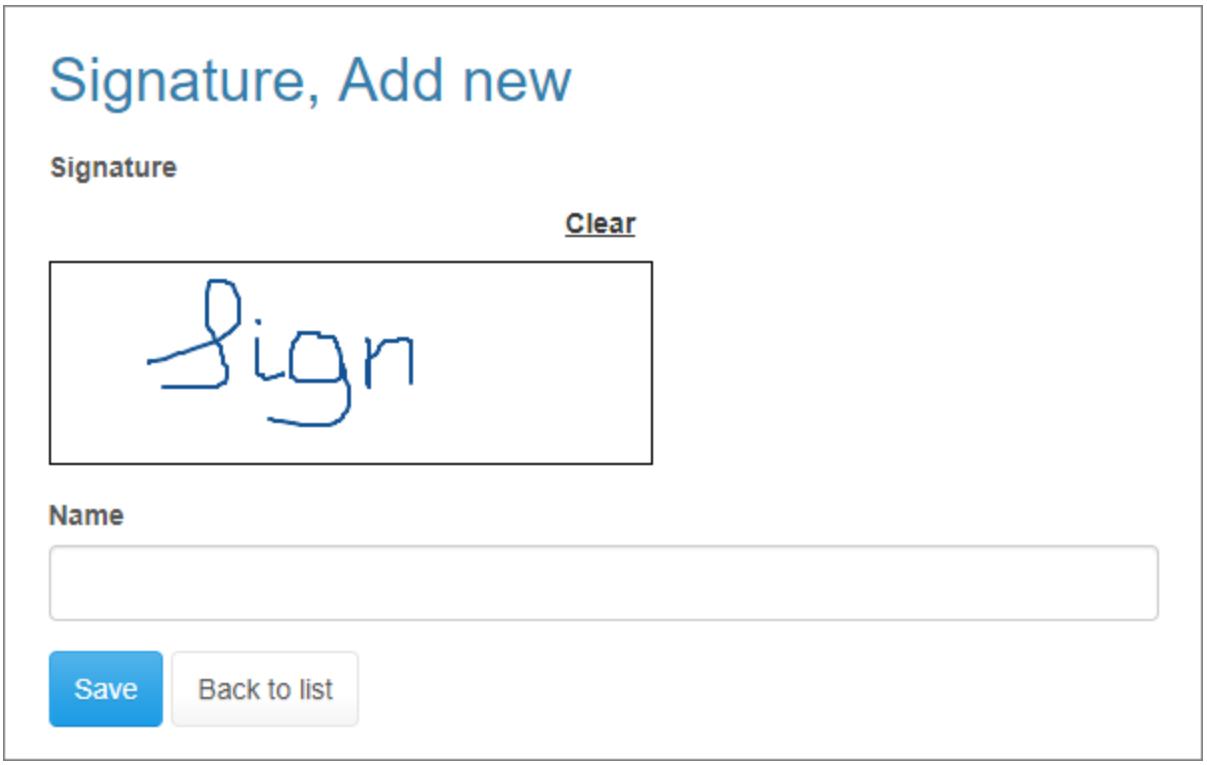
3.2.18 How to create a custom Edit control plugin

Quick jump

PHPRunner comes with an ability to create custom edit control plugins. In this article, we are going to show you how to create *ColorPicker* and *SignaturePad* plugins.

Before we proceed, check the following live demo that showcases plugins for both edit controls. The *SignaturePad* control works on mobile devices as well.

The *ColorPicker* and *SignaturePad* edit controls as seen in the browser:



ColorPicker control

Let's say we want to add a color picker control that allows users to select colors similarly to Adobe Photoshop. Since our software comes bundled with jQuery, let's search the web for "jQuery colorpicker".

This search returns several results, and the one we are going to use is miniColors. It looks nice and is easy to integrate. Sources can be found at Github.

- create a new folder under *<Documents>\PHPRunnerPlugins\edit* named ColorPicker;
- copy files *EditMyField.js* and *EditMyField.php* from *MyField* folder to the ColorPicker folder;
- rename those files to *EditColorPicker.js* and *EditColorPicker.php*, respectively;
- if your plugin needs extra files, copy them to the *ColorPicker* folder while keeping the folder structure;
- open *EditColorPicker.js* in any text editor and replace all occurrences of *EditMyField* with *EditColorPicker*. Do the same with the *EditColorPicker.php* file.
- use the *addJSFiles()* function in *EditColorPicker.php* to add references to external JavaScript files:

```
$this->pageObject->AddJSFile("jquery.miniColors.min.js");
```

Note: you may need to specify the loading order of JavaScript files. In this example file, *second.js* is loaded after *first.js*:

```
$this->pageObject->AddJSFile("second.js", "first.js");
```

- use addCSSFiles() function to add CSS files:

```
$this->pageObject->AddCSSFile("jquery.miniColors.css");
```

In the *EditColorPicker.php* file find the *buildUserControl()* function. This is were you build HTML code for your control.

If you leave the predefined code as is, it will display a simple text edit box. You can change this edit box, for example, by using a black theme:

```
class="black"
```

Lets see how to turn this edit box into a *colorpicker* control. According to colorpicker instructions we need to call for *miniColors* JavaScript referencing the edit box. We can do so adding the following code to constructor function:

```
$("#"+this.valContId).miniColors({
letterCase: 'uppercase'
});
```

The *letterCase* option tells the control to convert the manually entered color values to the upper case. *this.valContId* is the id of the control.

This is it; the control is ready. You can now launch PHPRunner and select the *ColorPicker* as Edit as type for any text field.

Additional enhancements

1. It would be nice if instead of the hex color value, we could show the visual representation of the selected color on the **List**/**View** pages. We can do so by choosing the View as type as Custom and adding the following code there:

```
$value="<span style='padding: 5px; background: ".$value."'>".$value."</span>";
```

2. By default, PHPRunner sets focus on the first edit control when an **Add** or **Edit** page is loaded. This may not be a desired behavior for the ColorPicker control, as the popup window will open every time the page is loaded. To prevent this from happening, implement the setFocus function, and return "*false*" every time.

Note: If you need to change the control behavior, check the functions and events in the *source\include\common\runnerJS\Control.js* file.

SignaturePad

The *SignaturePad* plugin allows adding a signature pad to your forms. *SignaturePad* works with both mouse and touch devices. We use a SignaturePad jQuery plugin that comes with excellent documentation and examples.

The basic setup is the same: create new folder for the *SignaturePad* plugin, copy and rename files, add files that plugin needs to the plugin directory.

This plugin is a bit more complicated and takes a few extra steps to integrate.

1. Build the HTML code

Here is how to do this in the *buildUserControl()* function:

```
echo '
<div class="sigPad" style="width: '.($this->width+2).'px;">
<ul class="sigNav">
<li class="clearButton"><a href="#clear">Clear</a></li>
</ul>
<div class="sig sigWrapper">
<div class="typed"></div>

<canvas class="pad" width="'.$this->width.'" height="'.$this->height.'"></canvas>
<input id="'.$this->cfield.'" type="hidden" '
.'name="'.$this->cfield.'" class="output">
</div>
</div>

';
```

2. Convert signature data to image

The signature data is recorded in a JSON format as a sequence of lines and is passed to the web server this way. We need to convert the JSON data to a PNG file and save it in the files folder on the web server.

Luckily all the hard work was already done, and all we need to do is to add a few lines of code to the *readWebValue()* function:

```
if ($this->webValue) {
   // save signature to file
   require_once 'signature-to-image.php';
   $img = sigJsonToImage($this->webValue, array(
            'imageSize' => array($this->width, $this->height)
```

```
            ,'bgColour' => $this->bgcolor
            ));
    $filename= $this->folder."/".generatePassword(15).".png";
    imagepng($img, $filename);
    $filesize = filesize($filename);

    // prepare image info to be saved in db
    $result[] = array("name" => $filename,
    "usrName" => 'signature.png', "size" => $filesize, "type" => "image/png",
    "searchStr" => 'signature.png'.":sStrEnd");
    $this->webValue = my_json_encode($result);
    }
```

Note: You can read more info about the JSON to image conversion here.

Note the way the file info is stored in the database. Since this version offers multiple files to upload, we need to be able to store more info than just the file name.

Besides the file name itself, we also save the file size, file type, and the path to the file in the JSON format. Here is how a typical file upload field looks like in the database:

```
[{"name":"files\/h8hsoz5hd23b0ik.jpg", "usrName":"Chrysanthemum.jpg",
"size":879394,"type":"image\/jpeg", "searchStr":"Chrysanthemum.jpg:sStrEnd"},
{"name":"files\/2p85jz854o6fbv8.jpg", "usrName":"Desert.jpg","size":845941,
"type":"image\/jpeg", "searchStr":"Desert.jpg:sStrEnd"},
{"name":"files\/pm4fu8uv2u6xc1w.jpg", "usrName":"Hydrangeas.jpg",
"size":595284,"type":"image\/jpeg", "searchStr":"Hydrangeas.jpg:sStrEnd"}]
```

3. Customization

Now we can customize our plugin. Users may need to change the appearance and behavior of the signature pad, for example:

- change the width and height of the signature pad;
- change the background color;
- change the folder where the image files are stored;
- make the signature pad field required.

As the first step, we need to learn to pass the settings from the PHPRunner wizard to the plugin. Proceed to the Edit as dialog in PHPRunner and click the **Add initialization script** button. Here is a sample set of settings for the *SignaturePad* control:

```
// signature field height
$this->settings["height"] = 100;
// signature field width
$this->settings["width"] = 300;
// signature background color
$this->settings["bgcolor"] = array(0xff, 0xff, 0xff);
// set it to true to make signature field required
$this->settings["required"]=false;
// folder to store signature files
$this->settings["folder"]="files";
```

This code is self-descriptive; you can pass any number of settings there. If you create a custom edit control plugin, place the sample initialization script to the sample.*php* file that needs to be located in the plugin folder.

Now we can access those settings in the plugin *initUserControl()* function. We can also pass the settings to the JavaScript part of the plugin.

```
// setting default value
$this->required = false;

// saving settings to the local variable
if ($this->settings["required"])
   $this->required=$this->settings["required"];

// passing settings to JavaScript
$this->addJSSetting("required", $this->required);
```

Adding new functionality

Making the signature field required

In the JavaScript *init()* function, add the following code:

```
if (this.required)
this.addValidation("IsRequired");
```

The signature is not required by default. To make it required, add the following line of code to the initialization script under Edit as properties:

```
$this->settings["required"]=true;
```

Setting the width and height of the signature pad

Passing the width and height from the Edit as settings in PHPRunner:

```
$this->settings["height"] = 100;
$this->settings["width"] = 300;
```

Then we can use *$this->width* and *$this->height* in the *buildUserControl()* function to specify the width and height of our control:

```
<canvas class="pad" width="'.$this->width.'" height="'.$this->height.'"></canvas>
```

Changing the folder where the signature images are stored

Passing the folder name from the Edit as settings in PHPRunner:

```
$this->settings["folder"]="files";
```

Using this variable in the *readWebValue()* function:

```
$filename= $this->folder."/".generatePassword(15).".png";
```

Changing the signature pad background color

Passing the background color from the Edit as settings in PHPRunner:

```
$this->settings["bgcolor"] = array(0xff, 0xff, 0xff);
```

The *bgcolor* array contains the color value in an RGB format (Red, Green, Blue, each color ranges from 0 to 255), *0xff* is a hexadecimal representation of 255. *array(0xff, 0xff, 0xff)* means white color.

Now we can use *this.bgColor* in the JavaScript control constructor function to pass the background color to the *SignaturePad* control:

```
$('.sigPad').signaturePad({drawOnly:true, bgColour: this.bgColor});
```

We also need to pass the backgound color to the *sigJsonToImage()* function that converts the JSON signature data to image. We use *$this->bgcolor* variable here in *readWebValue() PHP* function.

```
$img = sigJsonToImage($this->webValue, array(
      'imageSize' => array($this->width, $this->height)
      ,'bgColour' => $this->bgcolor
      ));
```

This is it. As you can see, creating a custom edit control plugin is relatively easy. You can do it with a few lines of code.

If you develop a new edit control plugin and want to share it with other users, feel free to contact our support team.

We have a plugin marketplace where you can sell the plugins you have created.

See also:

- JavaScript API: Control object > addValidation()
- JavaScript API: Control object > add CSS Class
- "Edit as" settings: ColorPicker
- "Edit as" settings: SignaturePad
- "View as" settings: Custom
- About Control object

3.2.19 How to display messages or tooltips for the fields

You can display messages or tooltips for the fields in the generated applications.

Variable

`ctrl`

the current field, a Control object.

Functions

> **Note**: these functions don't work in an **Inline Add/Edit** mode.

`ctrl.getTooltip()`

Returns the current tooltip text including HTML formatting.

`ctrl.setTooltip( message )`

Sets the tooltip message. HTML formatting is supported.

`ctrl.addTooltip( message )`

Append the message to the tooltip. HTML formatting is supported.

Example

```
ctrl.setTooltip( '<b>' + ctrl.getValue() + '</b> is the value of the field');
```

See also:

- About Control object
- Labels/Titles API: setFieldTooltip
- Miscellaneous settings: Edit form tooltips
- Field events

3.2.20 How to mark fields invalid

You can mark fields as valid or invalid in the generated application.

Variable

`ctrl`

the current field, a Control object.

Functions

`ctrl.setInvalid( message )`

Marks the control invalid.

`ctrl.setValid()`

Marks the control valid.

Note: *setValid()* only removes the status set by *setInvalid()*, it has no effect on other validations.

Note: the page can not be saved when there are invalid fields on it.

Example

```
if( ctrl.getValue() > 100 )
  ctrl.setInvalid( 'Price can not exceed 100' );
else
  ctrl.setValid();
```

See also:

- JavaScript API: Control object > getValue()
- JavaScript API: RunnerPage object > hideField()
- JavaScript API: RunnerPage object > showField()
- Field events
- About Control object

3.2.21 How to display data returned by stored procedure

The following SQL Server stored procedure example returns data from the "*test*" table.

```
CREATE PROCEDURE [dbo].[test_proc]
AS
BEGIN
SET NOCOUNT ON;
SELECT * from test
END
```

1. Add the test table or any other table to the project.

2. Add the following code to the **List** page: CustomQuery event:

```
return DB::Query("EXEC test_proc");
```

3. Add the following code to the **List** page: FetchRecords event:

```
return $rs->fetchAssoc();
```

4. Add the following code to the ListGetRowCount event:

```
return DBLookup("select count(*) from test");
```

See also:

- Database API:Query()
- DAL method: DBLookup
- QueryResult object: fetchAssoc()
- Event: ListQuery
- Event: ListFetchArray
- Event: ListGetRowCount
- How to execute stored procedures

3.2.22 How to execute stored procedures from events

Executing an SQL Server stored procedure from events

Calling the procedure without parameters:

```
CustomQuery("EXEC StoredProcNameHere");
```

Passing one of the field values as a parameter:

```
CustomQuery("EXEC StoredProcNameHere '" . $values["FieldName"] . "'");
```

Executing MySQL stored procedure from the event

Calling the procedure without parameters:

```
CustomQuery("CALL StoredProcNameHere");
```

Passing one of the field values as a parameter:

```
CustomQuery("CALL StoredProcNameHere ('" . $values["FieldName"] . "')");
```

Executing ORACLE stored procedure from the event

Calling the procedure without parameters:

```
CustomQuery("BEGIN STOREDPROCNAME END;");
```

Passing one of the field values as a parameter:

```
CustomQuery("BEGIN STOREDPROCNAME('" . $values["FieldName"] . "'); END;");
```

See also:

- Connecting to the database
- Connecting to Oracle database
- Table events
- Global events

3.2.23 PHPRunner session variables

> **Replaced**
>
> Starting with PHPRunner version 9.7, **session variables** were replaced with the functionality available via Security API and Search API.
>
> The only exception is the *language* variable.

You can still use the session variable *$_SESSION["language"]* that stores the selected language, e.g., "English" or "Spanish".

Examples

Example 1

Get the current value of this variable, use the following function that returns the selected language.

```
mlang_getcurrentlang();
```

Example 2

If you need to assign the value to the variable, use the variable directly.

```
$_SESSION["language"]="English";
```

See also:

- Localizing PHPRunner applications
- Save user data in session variables
- About Security API
- About Search API

3.2.24 PHPRunner templates

Introduction

PHPRunner uses the built-in template language. PHPRunner templates cleanly separate your presentation layer (HTML, CSS, etc.) from the application code. The main idea is to simplify visual templates by moving all logic and JavaScript to PHP files.

Template language reference

- all template tags are enclosed within delimiters *{* and *}*. All content outside delimiters is displayed as static HTML.
- template variables start with the dollar *$* sign. They may contain numbers, letters, and underscores.

Example

```
{$variable_name}
```

The string *{BEGIN block_name} ... {END block_name}* is used to define the block section (loops, condition statements).

Functions

`assign ($name, $val)`

is used to assign the value *$val* to the variable *$name*.

Working with the templates

Here is a basic example of how templates work. Add the following code to the Before display event:

```
$xt->assign('name', 'george smith');
$xt->assign('address', '45th & Harris');
```

The template file contains the output with template tags that PHPRunner replaces with assigned content.

Template file

```
<html>
<head>
<title>User Info</title>
</head>
<body>
User Information:<p>
Name: {$name}<br>
Address: {$address}<br>
</body>
</html>
```

Output

```
<html>
<head>
<title>User Info</title>
</head>
<body>
User Information:<p>
Name: george smith<br>
Address: 45th & Harris<br>
</body>
</html>
```

Changing a button label dynamically

Here is how you can change a custom button label dynamically.

1. Insert a **Custom Button** into one of the pages using Page Designer. Use the following as button label:

```
{$button_label}
```

2. Use the following code in the BeforeDisplay event of the page in question:

```
$xt->assign("button_label", "Edit Customer");
```

{BEGIN} ... {END} blocks

The template file contains a set of code sections wrapped by *{BEGIN ...} and {END ...}*.

Template file

```
{BEGIN Model_fieldblock}
  <tr><td class=shade width=150>Model</td><td width=250>
  {$Model_value} 
  </td></tr>
{END Model_fieldblock}
```

Use the following code in the Before display event:

```
$xt->assign("Model_fieldblock",true);
```

When you use *true*, the HTML code between {BEGIN ...} and {END ...} appears in the output.

```
$xt->assign("Model_fieldblock",true);
```

When you use *false*, the HTML code between {BEGIN ...} and {END ...} is hidden.

See also:

- About templates
- Template files processing rules (Files.txt)
- Template language
- Page Designer: Working with page elements

3.2.25 Template files processing rules (Files.txt)

The Files.txt file

The *Files.txt* file (located in the source directory) contains the list of template processing rules.

```
<source file> <destination file> [table name]
```

The `[table name]` is an optional parameter that must be provided for the table specific templates, such as *list.php*, *edit.php*, etc. If the *[table name]* parameter is a specified file, it is processed for each table in the project. Otherwise the template file is processed once.

Files.txt supports a conditional compilation. See Template language reference for more info on the syntax.

Example

```
##if @t.bAdd || @t.bInlineAdd##
add.php ##@t.strShortTableName##_add.php ##@t.strDataSourceTable##
##endif##
```

This code snippet tells the wizard to create an **Add** page (*tablename_add.php* file) if the **Add** or **Inline Add** options have been selected for this table.

Each business template (Cars, Events, PayPal, etc.) also has *files.txt* which overrides the rules defined in the main file.

If you add your files to the template, it's necessary for you to add new lines to the *files.txt* file to define how the new files should be processed.

See also:

- PHPRunner templates
- Template language
- About business templates
- Choose pages screen

3.2.26 Template language

Quick jump

Template language is the framework that powers visual and code templates in PHPRunner. Most template language expressions are references to the project files. Template language elements are wrapped by ## characters:

```
##if @field.m_bAddPage##
```

Expressions

1. Strings

Example:

- "*string1*" - string1;
- "*this is a* \"*string*\"" - this is a "string";
- "*"\"\\\" is a backslash*" - "\" is a backslash.

2. Numbers

Examples:

- 2
- 3.3
- -2

3. Variables

Variables start with the @ character.

Examples:

- *@BUILDER* - the root element of the project file.
- *@TABLE* - a pseudo-variable that points to the currently selected table.
- *@a*, *@field* - regular variables.
- *@TABLE.arrFieldObj* - an array of fields that belong to the current table.
- *@field.EditFormatObj.m_strDefaultValue* - the default field value.

Variables belong to one of the following data types: *strings*, *numbers*, *objects* and *arrays*.

4. Boolean expressions

- 0,"" - false.
- Anything else - true.

Operators

1. Comparison operators

- == - equals.
- *!=* or <> - does not equal.
- < - less than.
- <= - less than or equal to.
- > - greater than.
- >= - greater than or equal to.

You can only compare numbers and strings. The comparison result can be either 0 or 1.

2. Boolean

- *or* or ||
- *and* or &&
- *not* or *!*

The result can be either 0 or 1.

3. Parenthesis

Example:

```
(@field.m_bAddPage or @field.m_strLabel=="ID") and not @Table.m_strCaption==""
```

4. Dot operator

To access structure members, the dot . operator is used.

Example:

```
@field.m_ListFormatObj.m_nColWidth
```

5. The Array length property is '.len'

Example:

```
@TABLE.m_arrFieldObj.len
```

6. Priority order

There is an established order with the priority of each operator. From greatest to lowest, the priority order is as follows:

1. *.*
2. *.len*
3. parenthesis
4. comparison operators
5. *not*
6. *and*
7. *or*

Statements

1. Display a value of a variable or an expression

Examples:

- *##3##* - displays 3.
- *##@field.m_bAddPage##* - displays 0 or 1.
- *##@field.m_strLabel##* - displays the field label, for example, *'Year of Make'*.

2. Conditional statement

```
if <Boolean expression>, elseif <Boolean expression>, else, endif
```

Examples:

```
##if @field.m_bAddPage##
 ...
##elseif @field.m_strLabel=="ID"##
 ...
##else##
...
##endif##
```

3. Loop statements

1)

```
Foreach <array> as <variable> , endfor
```

The variable is created when loop starts and destroyed with the *'endfor'*.

Example:

```
##foreach @TABLE.m_arrFieldObj as @field##
if strField="##@field.m_strName##" then
  Label = "##@field.m_strLabel##"
##endfor##
```

2)

```
Repeat <number> [variable], endrepeat
```

Repeat the loop body N times. Variable, if specified, ranges from 1 to *<number>*.

3)

```
Filter
```

Allows nodes filtering.

Example. Get a list of fields that require validation:

```
##foreach @TABLE.m_arrFieldObj as @field filter @field.m_strValidateAs order @field.
m_nEditPageOrder##
  ##if @first##
    include("include/validate.phpaspcs");
  ##endif##
##endfor##
```

4) Nested loops.

Repeat and *Foreach* loops can be nested.

Example:

```
##foreach @BUILDER.Tables as @t##
  ##foreach
@t.arrMasterTables[strMasterTable==@TABLE.strDataSourceTable].arrMasterKeys as @dk##
  $masterquery.="&masterkey##@index##=".rawurlencode($data["##@dk s##"]);
  ##endfor##
  $showDetailKeys["##@t.strShortTableName##"]=$masterquery;
##endfor##
```

5) Loop variable @index

The loop variable *@index* takes on the values *1, 2, ..., N* through each of the N iterations of the loop body.

6) Pseudo-variables @first and @last

The *@first* variable takes the value of:

- 1 - during the first loop pass;
- 0 - otherwise.

It is useful when you need to perform some action only once, i.e., skip a comma in front of table name:

```
$tables = Array("Table1","Table2","Table3");
##if !@first## , ##endif##
```

In loops, *@first* terminates the execution of the nearest enclosing *foreach* or `repeat` statement. Control then passes to the statement that follows the terminated statement, if any:

```
##foreach @TABLE.m_arrFieldObj as @field##
##if @field.m_EditFormatObj. m_strValidateAs && @first##
  include("include/validate.phpaspcs");
##endif##
##endfor##
```

7) Order - a sort order other than default.

Example. Get a list of fields ordered by *nEditPageOrder* (the field order on the edit page):

```
##foreach @TABLE.m_arrFieldObj as @field order @field. nEditPageOrder##
##if @field.m_EditFormatObj. m_strValidateAs && @first ##
  include("include/validate.phpaspcs");
##endif##
##endfor##
```

Output modifiers

Modifiers are required to encode quotes, slashes and other characters that can break template language elements. You can combine several modifiers. Modifier order is important.

Example:

```
##@field.m_strLabel hs##
```

List of modifiers abbreviations:

- *hs* - *shapehtml* is applied first, *shapescript* is applied after that.
- *s* - replaces " with \" *for PHP*.
- *q* - strings wrapped by single quotes.
- *h* - HTML-encodes the string.
- *j* - replaces ' with \'.
- *n* - replaces spaces with * *.
- *u* - URL-encodes the string.
- *w* - adds wrappers around the field name (*[field name]* or `*field name*`).
- *t* - adds wrappers around the table name *([dbo].[table name]* or `*table name*`).
- g - replaces all non alphanumeric characters with underscores.
- *p* - builds the parameter name for UPDATE, INSERT, DELETE (.NET specific).
- *c* - removes spaces.
- *8* - converts a UTF8 string to a national language charset.
- *o* - removes the owner/schema from table name.
- *d* - converts the name to CamelCase. Example: "*Order details*" becomes *OrderDetails*.
- *l* - replaces line feeds and carriage returns with spaces.
- *a* - builds a valid variable name from the table name. Used in Data Access Layer.
- *f* - builds a valid variable name from the field name. Used in Data Access Layer.
- *x* - builds a valid property name (.NET Data Access Layer specific).
- *y* - builds a valid class name (.NET Data Access Layer specific).

Macros and constants

Macros and constants are processed and replaced with the actual code. Macros and constants are defined in the *macros.txt* file.

Constant definition example:

```
##define <name> <value>##
##define FORMAT_DATABASE_IMAGE "Database image"##
##define EDIT_DATE_SIMPLE 0##
```

Macro definition example:

```
##define UseRTE(@field)
(@field.strEditFormat==EDIT_FORMAT_TEXT_AREA && @field.m_EditFormatObj.m_bUseRTE)
##
```

Macros and constants are processed in the same way. Therefore, we suggest to follow this naming convention: constant names are written in upper case (e.g., FORMAT_DATABASE_IMAGE), macro names use CamelCase convention (e.g. UseCalendar). Spaces are not allowed in macro or constant names.

Example:

```
##if @field.strViewFormat==FORMAT_DATABASE_IMAGE##
##if UseRTE(@field)##
##foreach Fields as @f##
##Master.strCaption##
```

Additional language elements

Specific array element

To access a specific array element, use <array>[<condition>].

Example:

```
##@TABLE.arrFieldObj[strName==@TABLE.strKeyField].strLabel##
```

This example shows how to access the Label property of a key column field or any other field.

See also:

- PHPRunner templates
- Template files processing rules (Files.txt)

3.2.27 runner_mail function

Description

The **runner_mail** function is a wrapper for the **mail()** function in PHPRunner.

To use it, you need to set up email parameters (**From**, **SMTP server**, **SMTP server port**, **SMTP server username**, **SMTP server password**) in the Email settings.

Syntax

```
runner_mail($params)
```

Arguments

$params

an array with input parameters. The following parameters are supported:

Email parameters

from

the sender's email address. If none is specified, an email address from the Email settings is used.

fromName

the sender's name.

to

the receiver's email address.

cc

email addresses of secondary recipients.

`bcc`

email addresses of recipients whose addresses are not to be revealed to other recipients of the message.

`replyTo`

the reply email address.

`priority`

the message priority (use '1' for urgent, '2' - high, '3' - normal).

`body`

the plain text message body.

`htmlbody`

the html message body (do not use the 'body' parameter in this case).

`charset`

the html message charset. If none is specified, the default website charset is used.

`attachments`

the description of the attachments. The `'path'` (a path to the attachment) is required. Other parameters are optional: `'name'` overrides the attachment name, `'encoding'` sets the file encoding, `'type'` sets the MIME type.

Attachments work only when the **PHP 'mail'** function is disabled.

Return value

An array with the following keys:

- **mailed**: (*true* or *false*) indicates whether the email was sent or not.
- **message**: an error message in case the email was not sent.

Examples

Send simple email

```
$email="test@test.com";
$message="Hello there\nBest regards";
$subject="Sample subject";
runner_mail(array('to' => $email, 'subject' => $subject,
  'body' => $message));
```

Send HTML email

```
$email="test@test.com";
$message="Hello there\n<b>Best regards</b>";
$subject="Sample subject";
runner_mail(array('to' => $email, 'subject' => $subject,
  'htmlbody' => $message, 'charset' => 'UTF-8'));
```

Example with error handling

```
$email="test@test.com";
$subject="Sample subject";
$body="test";
$arr = runner_mail(array('to' => $email, 'subject' => $subject,
  'body' => $body));
// if error happened print a message on the web page
if (!$arr["mailed"])
{
  echo "Error happened: <br>";
  echo $arr["message"];
}
```

Send email with BCC and CC fields

```
$email="test@test.com";
$message="Hello there\nBest regards";
$subject="Sample subject";
runner_mail(array('to'  =>  $email, 'cc' => 'test2@test.com',
'bcc' => 'test3@test.com', 'subject' => $subject, 'body' => $message));
```

Send email with From and FromName fields

```
$email="test@test.com";
$message="Hello there\nBest regards";
$subject="Sample subject";
$from="bgates@microsoft.com";
$fromName="Bill Gates";
runner_mail(array('to'  =>  $email, 'subject' => $subject,   'body' =>
$message, 'fromName'=> $fromName, 'from' => $from));
```

Send email to multiple recipients

```
$email="test@test.com,test2@test.com,test3@test.com";
$message="Hello there\nBest regards";
$subject="Sample subject";
runner_mail(array('to' => $email, 'subject' => $subject, 'body' => $message));
```

Send email with attachments

```
$from = "myemail@test.com";
$to = "myclient@test.com";
$msg = "Find some documents (Invoice.pdf, Photo.jpg, signature.jpg) attached.";
$subject="Documents";

$attachments = array();
// Attachments description. The 'path'(a path to the attachment) is required. Others
parameters are optional:
//'name' overrides the attachment name, 'encoding' sets a file encoding, 'type' sets
a MIME type
$attachments =  array(
        array('path' => getabspath('files/Invoice.pdf')),
        array('path' => getabspath('files/Photo12.jpg'),
                    'name' => 'Photo.jpg',
                    'encoding' => 'base64',
                    'type' => 'application/octet-stream'),
        array('path' => getabspath('files/signature.jpg'))
) ;

$ret = runner_mail(array('from' => $from, 'to' => $to, 'subject' => $subject, 'body'
=> $msg, 'attachments' => $attachments));

if(!$ret["mailed"]){
    echo $ret["message"];
}
```

Send email with new record data. Use this code in the After Add event.

```
$from = "myemail@test.com";
$to = "myclient@test.com";
$msg = "The new record was added: ";
$subject="New record";

$msg.= "Name: ".$values["name"]."\r\n";
$msg.= "Email: ".$values["email"]."\r\n";
$msg.= "Age: ".$values["age"]."\r\n";

$ret=runner_mail(array('to' => $email, 'subject' => $subject, 'body' => $msg,
'from'=>$from));
if(!$ret["mailed"]){
    echo $ret["message"];
}
```

See also:

- Email settings
- Events: AfterAdd

- runner_sms function
- Event editor
- About predefined actions
- Dialog API

3.2.28 runner_sms function

Description

Sends a text message to a specified phone number.

Syntax

```
runner_sms($number, $message)
```

Arguments

$number

a phone number.

$message

a text message.

Return value

An array with the following keys:

- **success**: (*true* or *false*) indicates whether the sms was sent or not.
- **error**: an error message in case the sms was not sent.

Example

Send an sms to a phone number. This function works only if you have set up the Two-factor authentication.

```
$number="+12345678901";
$message="You verification code";

runner_sms($number, $message);
```

See also:

- Two-factor authentication
- SMS settings
- Multiple SMS providers
- runner_mail function

3.2.29 Useful links

Useful programming links

- PHP documentation: http://www.php.net/manual/en/
- PHP string functions: http://www.php.net/manual/en/ref.strings.php
- PHP date and time functions: http://us.php.net/manual/en/ref.datetime.php
- General SQL functions and tutorial: http://www.webcheatsheet.com/sql/interactive_sql_tutorial/
- MySQL documentation: http://dev.mysql.com/doc/refman/5.0/en/index.html
- MySQL date and time functions: http://dev.mysql.com/doc/refman/5.0/en/date-and-time-functions.html
- MySQL string functions: http://dev.mysql.com/doc/refman/5.0/en/string-functions.html
- MS Access functions: http://www.webcheatsheet.com/sql/access_functions/
- Oracle documentation: http://www.oracle.com/pls/db111/homepage
- SQL Server documentation: http://technet.microsoft.com/en-us/library/ms130214.aspx
- HTML reference: https://dev.w3.org/html5/html-author/
- CSS reference: http://htmlhelp.com/reference/css/

3.2.30 Using JOIN SQL queries

Lets say you need to pull data from two or more joined tables to show on the **List**/**View**/**Edit** page. The data should be searchable and editable (except for the joined fields that cannot be updated).

In this example we use two tables:

Cars

ID	Make	Model	UserID
1	Acura	NSX-T	1
2	Ford	Crown Victoria	2
3	Volkswagen	Passat	1
4	Toyota	Avalon	2
5	Audi	TT	3

Users

ID	Username
1	Bob
2	Admin
3	Bill
4	Tina

Here is the default SQL query PHPRunner builds for the *Cars* table:

```
select [ID],
[Make],
[Model],
[UserID],
From [Cars]
```

Modify it like so:

```
select Cars.[ID],
[Make],
[Model],
[UserID],
UserName
From [Cars]
inner join users on cars.userid=users.id
```

If you don't specify which table the *ID* field belongs to, an error message similar to this one appears:

```
Error message:

[Microsoft][ODBC Microsoft Access Driver] The specified field '[ID]' could refer to
more than one table listed in the FROM clause of your SQL statement.
```

See also:

- SQL Query screen
- Open Database Connectivity (ODBC)

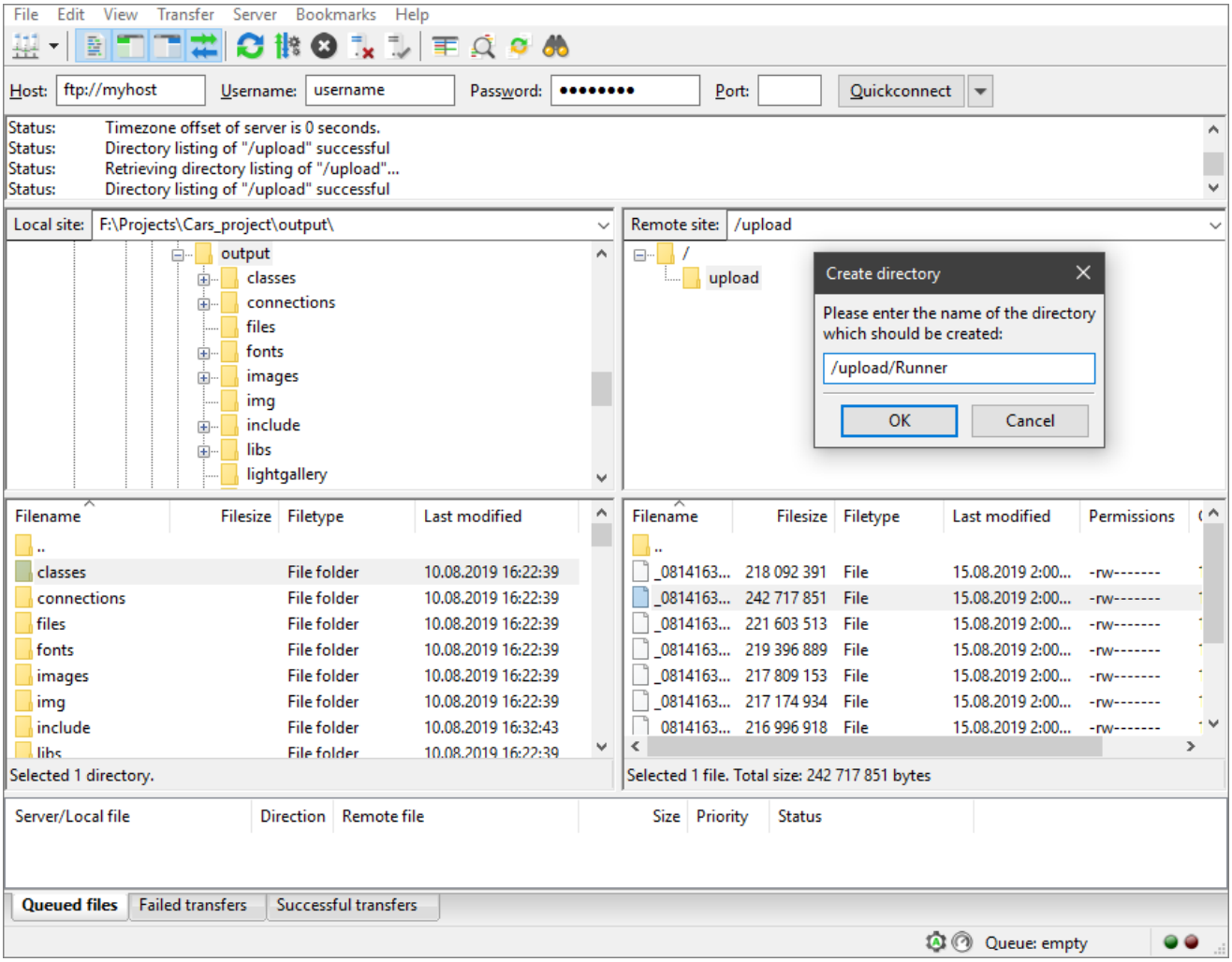
3.3 Publishing PHP application to the Web server

3.3.1 Upload web application using a third-party FTP client

After you have built an application with PHPRunner, you may use any FTP client to upload it to a remote server. Here is how you can do it using **FileZilla**.

First of all, you need to connect to the remote server. Fill in the *Host*, *Username*, *Password*, *Port* fields and click the **Quickconnect** button.

> **Note**: you can see the connection status in the panel below the **Quickconnect** button.



The left panel represents your local files/site. Open the **Output folder** of your project in the local files.

> **Note**: to learn more about the output settings, see Output directory settings.

The right panel represents the remote server file structure. Create a new folder for your project on the remote server - right-click anywhere in the bottom-right panel and select **Create new directory** and enter it.

Once you opened both folders (local and remote), select all local folders and files in the output directory, right-click on them and select **Upload**. The selected files are added to the **Queue** and uploaded to the remote server.

When the contents of the **Output folder** have been copied to the remote server, open the browser, type in the name of your website and add */%name of the folder you made%* after it. In our example, the URL looks like this: *http://www.xlinesoft.com/upload/Runner/.*

See also:

- After you are done
- Output directory settings
- FTP upload
- Desktop applications

3.4 Demo Account

3.4.1 About Demo Account

What is a Demo Account?

Demo Account is a free service provided by xlinesoft.com to PHPRunner customers. By using **Demo Account**, you agree to the following Terms and Conditions.

Demo Account allows quickly uploading the generated by PHPRunner application to our demo server for testing purposes.

You may find this feature useful if:

- you don't have a web server on your local machine to test the application;
- you don't have a web hosting account;
- you need to show someone the generated application (colleagues, friends, or our support staff).

Using Demo Account

To open and use **Demo Account**, proceed to the Finished screen in PHPRunner and click the **Demo Account** button. To create an account, enter your email address and choose any password you want.

> **Note**: you need these credentials later if you wish to manage your account, i.e., delete projects.

After the account is created, use the **Upload** button to transfer the application to the demo server.

> **Note**: your application automatically opens in the browser after the successful upload.

Demo Account transfers the generated pages and database to the demo server. Currently supported databases are MS Access, SQL Server, MySQL.

To view your account, proceed to http://demo.asprunner.net/Account/AccountView.aspx. The account page allows you to manage the uploaded projects.

Demo Account limitations

- Since this account is designed for demo purposes, only the first 1000 records in each SQL Server/MySQL database table are transferred to the server. MS Access databases are transferred to the server entirely.
- The maximum size of the archived project is 100Mb.

See also:

- Demo Account Terms and Conditions
- After you are done
- Quick start guide
- Connecting to the database

3.4.2 Terms and Conditions

Xlinesoft.com Demo Account Terms and Conditions

Agreement between User and xlinesoft.com.

A violation of any of the below Terms and Conditions will result in the termination of your account, with or without warning.

We provide a free Demo Account for our users. xlinesoft.com reserves the right to cancel any account for any reason or no reason at all. xlinesoft.com provides this service to any user that abides by our terms and conditions. Failure to comply with these rules will result in account termination. Furthermore, anyone who conducts illegal activities may be prosecuted, and have their personal information disclosed to the appropriate authority. xlinesoft.com reserves the right to change the terms and conditions at any time, and it is the member's responsibility to check for any updates of these terms. You are entirely liable for all activities conducted through your account.

Your use of the xlinesoft.com Demo Account is conditional upon your acceptance without modification of the terms, conditions, and notices contained herein. Your use of our Services constitutes your agreement to all such terms, conditions, and notices.

The following rules apply while using xlinesoft.com Demo Account:

1. Sign Up - Basic Terms

1.1 Upon after creating a Demo Account through xlinesoft.com software, you will become a user of xlinesoft.com Demo Account.

1.2 xlinesoft.com does not issue credits for outages incurred through service disablement resulting from Terms and Conditions violations.

2. Modification of Terms of Service

2.1 We reserve the right to change the terms and conditions of this agreement at any time without prior notice. Such changes will be posted on our web site at www.xlinesoft.com. You agree to review any changes to this agreement and, if such changes are not acceptable to you, immediately terminate your use of the xlinesoft.com Demo Account. If you continue to use the xlinesoft.com service after the effective date of such changes, such use will constitute acceptance of the changes.

3. Modifications to xlinesoft.com Demo Account

3.1 By accepting this Agreement, you hereby acknowledge and agree that, in our sole discretion, we may modify or discontinue, temporarily or permanently, any aspect of the xlinesoft.com Demo Account at any time with or without prior notice, including, without limitation, modification or discontinuance of advertising, content, and applications appearing as part of the xlinesoft.com Demo Account. You agree that we will not be liable to you or any third party for any modification, suspension or discontinuance of the xlinesoft.com Demo Account.

4. Privacy

4.1 xlinesoft.com adheres to a Privacy Statement and will not release any confidential information about you unless required by law or regulatory authority or while assisting an investigation concerning fraud.

5. No Unlawful or Prohibited Use, Spam and Termination

5.1 xlinesoft.com Demo Account may be used for lawful purposes only. As a condition of your use of the xlinesoft.com Demo Account, you agree that you will not use our Website for any purpose that is unlawful, illegal or prohibited by these Terms of Services, or our other terms, conditions or notices. You agree that you will not modify, copy, distribute, transmit, reproduce, publish, license, transfer, store, sell or create derivative works from, any data, software, Services or products obtained through our Website. This includes, but is not limited to: copyrighted material; trademarks; trade secrets or other intellectual property rights used without proper authorization; material that is obscene, defamatory, constitutes an illegal threat or violates export control laws.

Examples of unacceptable content or links include:

- pirated software;
- hacker programs or archives;
- Warez sites;
- MP3;
- IRC bots.

Such misuse can result in the cancellation of your entire account and the blacklisting of your domains without notice. You or xlinesoft.com may terminate your Demo Account at any time for any reason.

See also:

- About Demo Account

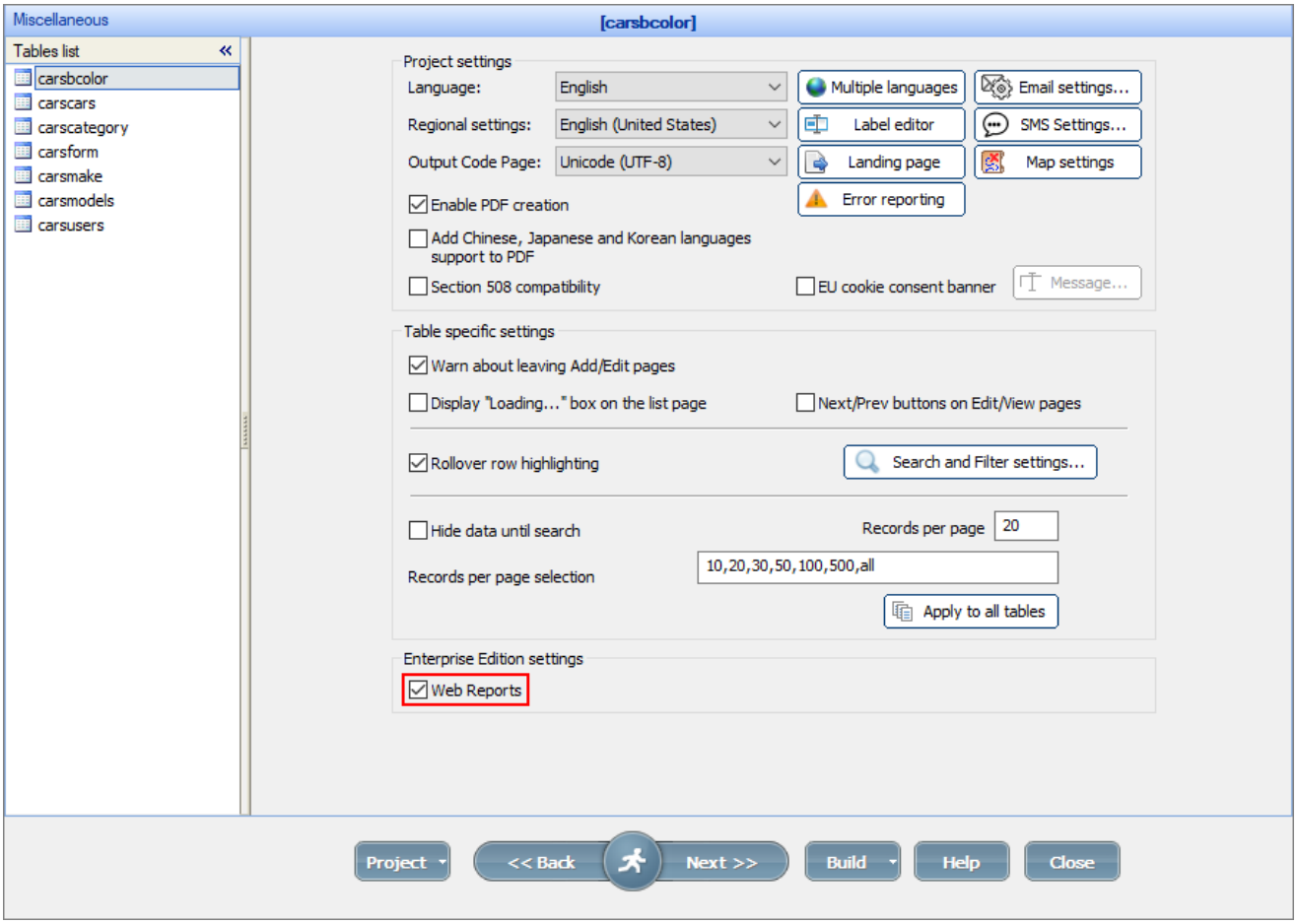
3.5 Web reports

3.5.1 Online report/chart builder

Quick jump

The Enterprise Edition of PHPRunner includes an online report/chart builder that allows you to view, create, edit, and delete charts and reports in the browser.

To enable the online report/chart builder, select the **Web Reports** checkbox on the Miscellaneous screen and build the project.

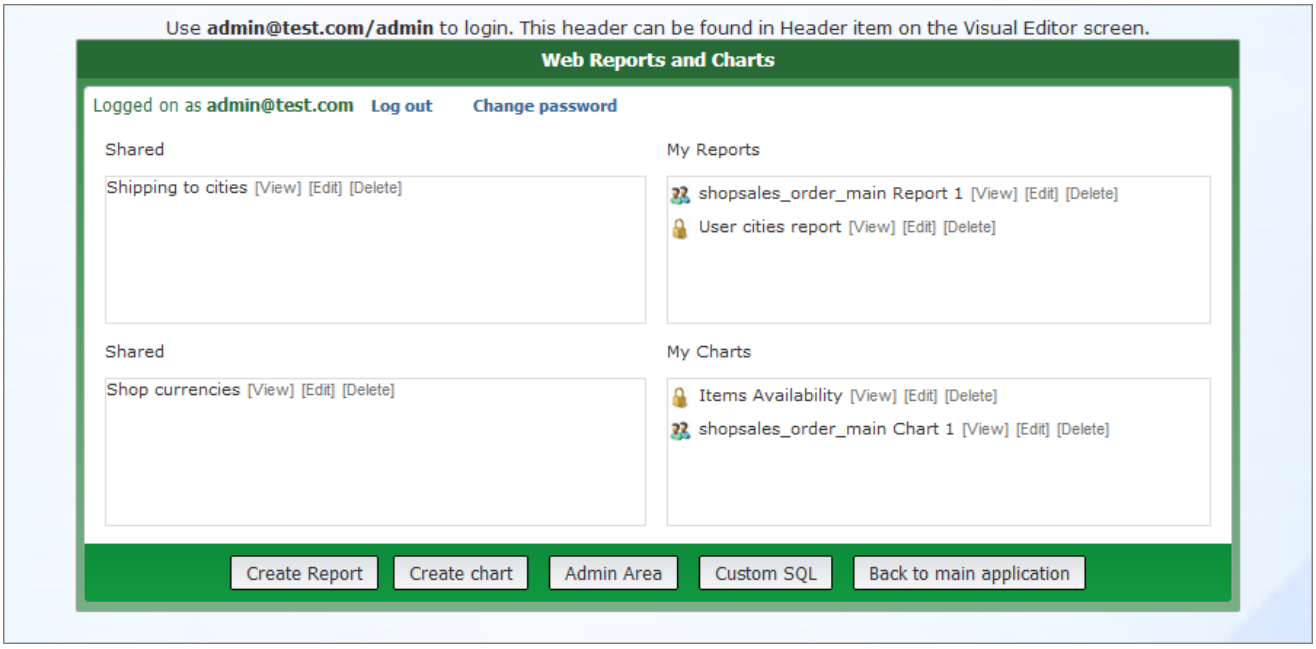


Then launch the online report/chart builder by running *webreport.php* in the browser (e.g., *http://www.yourwebsite.com/project1/webreport.php*).

> **Note**: the online report/chart builder uses the PHPRunner project security model. So if you use a **Login** page, you need to log into your application first.

Working with the online report/chart builder

The start page presents you with the list of charts and reports. If your project uses permissions, the charts and reports are divided into shared and private ones.



You can view, edit, or delete the charts and reports if the permissions allow you to do so. You can also create a new chart or a new report.

Use **Advanced search** to narrow down the results while viewing charts and reports. When viewing a report, you can also print the current page or the whole report, open a report as a Microsoft Word or Microsoft Excel document.

Here is an example of a web report:

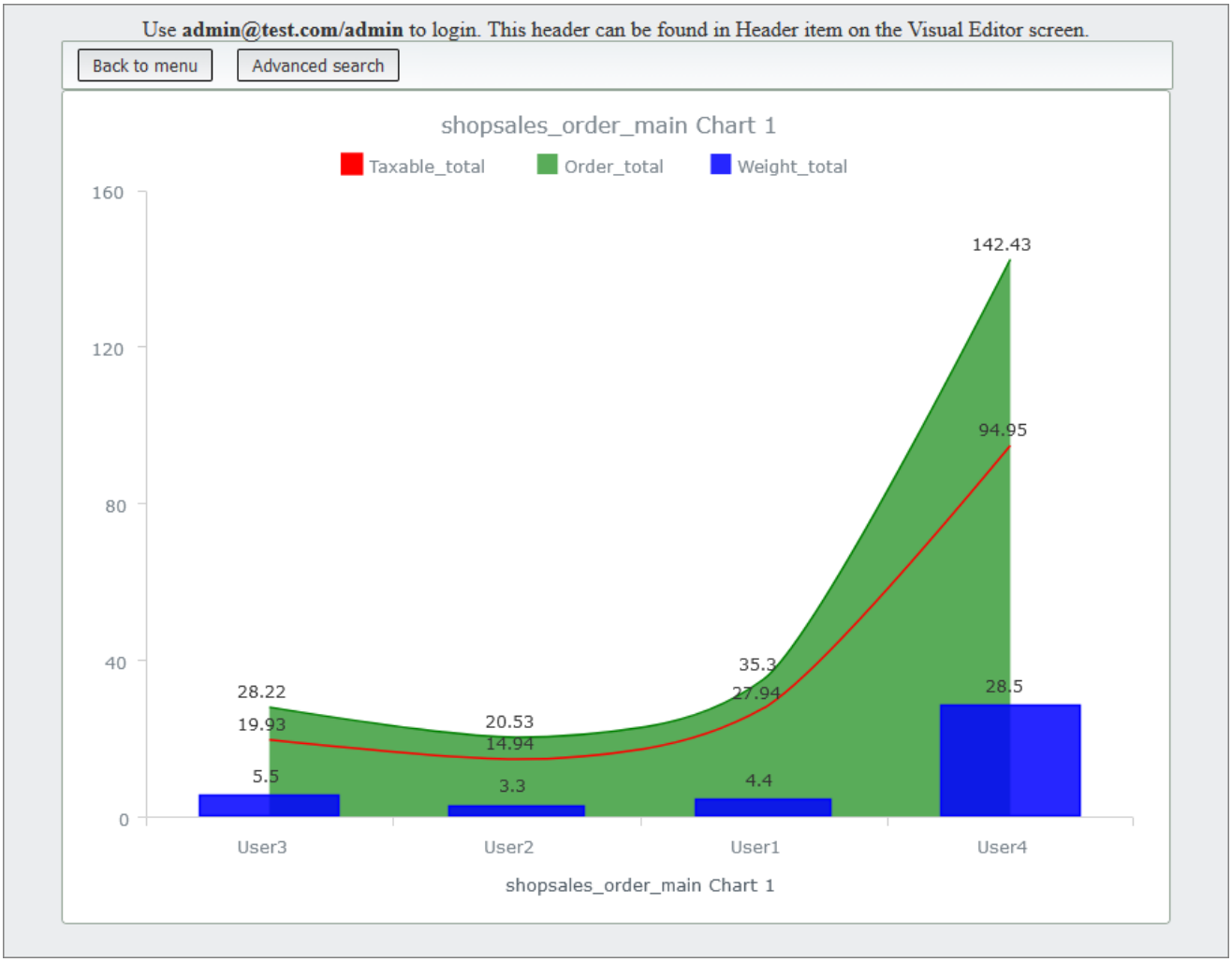
shopsales_order_main Report 1 | Back to menu | Advanced search | Print this page | Print whole report

Groups per page: 5

user_id	Order Number	Customer Email	Billing Name	Billing Address	Billing City	Weight Total	Order Cost	Taxable Total	Tax Amount	Order Total	Pay Type	Order Comments	Order Status	Order Number	Line_no	Product	Quantity ordered	Quantity shipped	sell_cost	Comments
15																				
	4																			
		user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4	7	ISBN 5-352-0005	1			
		user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4	11	CD14466	1			
		user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4	8	ISBN 15-23-444	1			
		user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4	12	CD12345	1			
		user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4	9	ISBN 1-23-456	1			
		user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4	10	CD43215	1			
	Summary for Order Number 4 - 6 records total																			
	Sum					76.2													0	
	Min					12.70														
	Max					12.70														
	6																			
		user4@user.com	User4	168-02 Jewel Ave	New York	28.50	64.50	94.95	5.48	142.43	CHK		PLACED	6	15	ISBN 1-23-456	5			
		user4@user.com	User4	168-02 Jewel Ave	New York	28.50	64.50	94.95	5.48	142.43	CHK		PLACED	6	16	ISBN 5-352-0005	5			
	Summary for Order Number 6 - 2 records total																			
	Sum					57													0	
	Min					28.50														
	Max					28.50														
Summary for user_id 15 - 8 records total																				
Sum						133.2													0	
Min						12.70														
Max						28.50														

user_id	Order Number	Customer Email	Billing Name	Billing Address	Billing City	Weight Total	Order Cost	Taxable Total	Tax Amount	Order Total	Pay Type	Order Comments	Order Status	Order Number	Line_no	Product	Quantity ordered	Quantity shipped	sell_cost	Comments
Page summary 8 - records total																				
Sum						133.2													0	
Min						12.70														
Max						28.50														

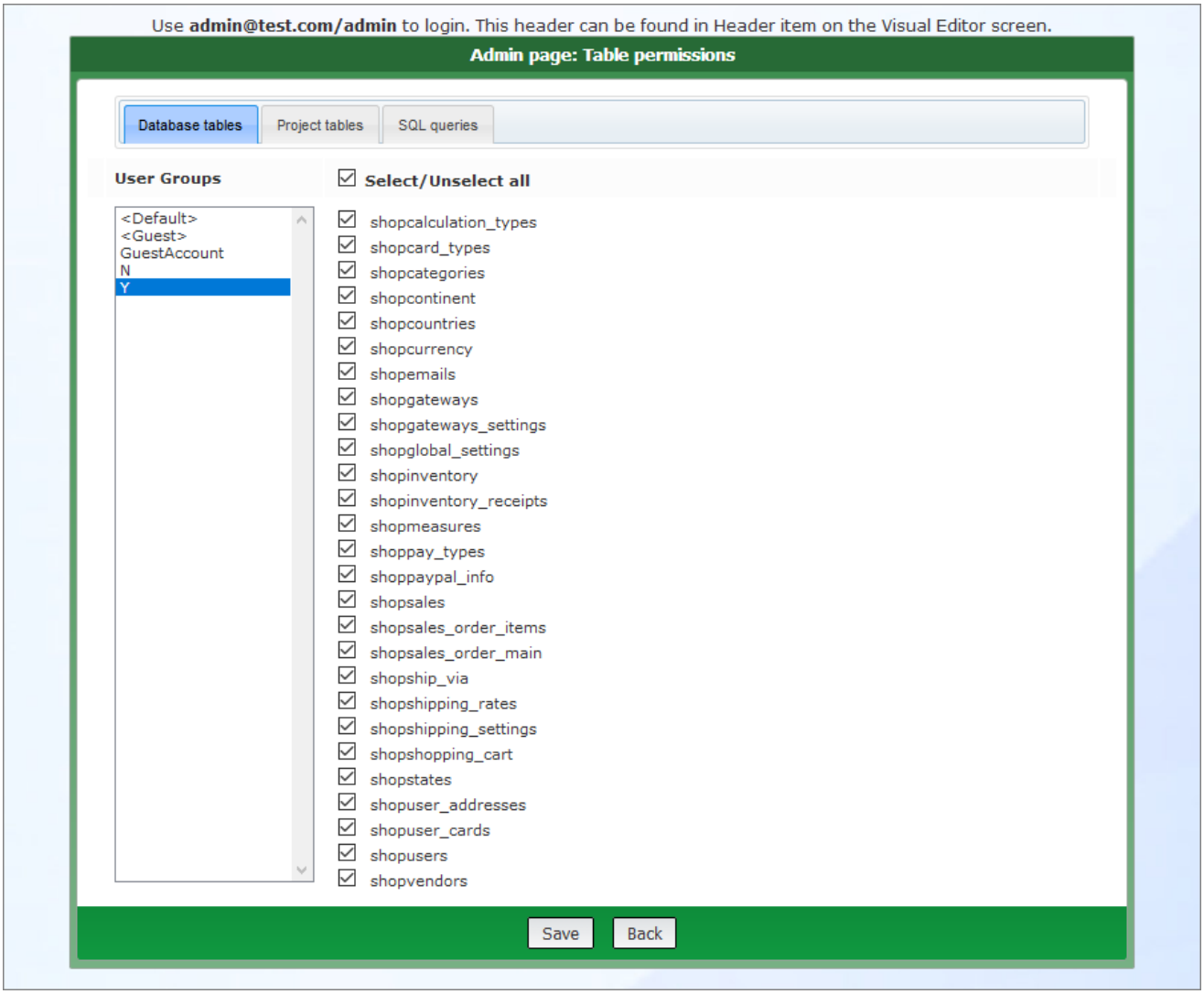
Here is an example of a web chart:



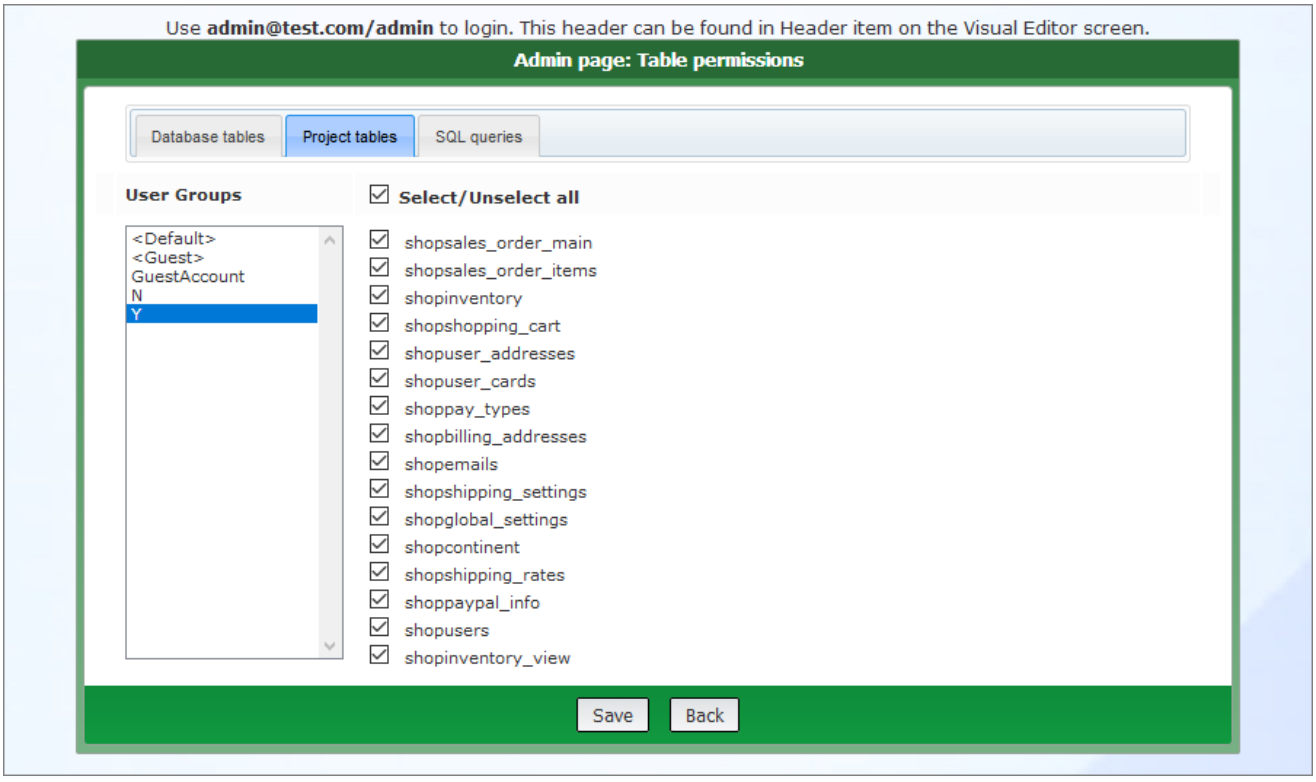
Admin area

The Admin area allows you to set up the group permissions for creating web reports and charts using the **Database tables**, **Project tables**, and **SQL queries** tabs. You can learn more about these tabs on Creating web reports or Creating web charts articles.

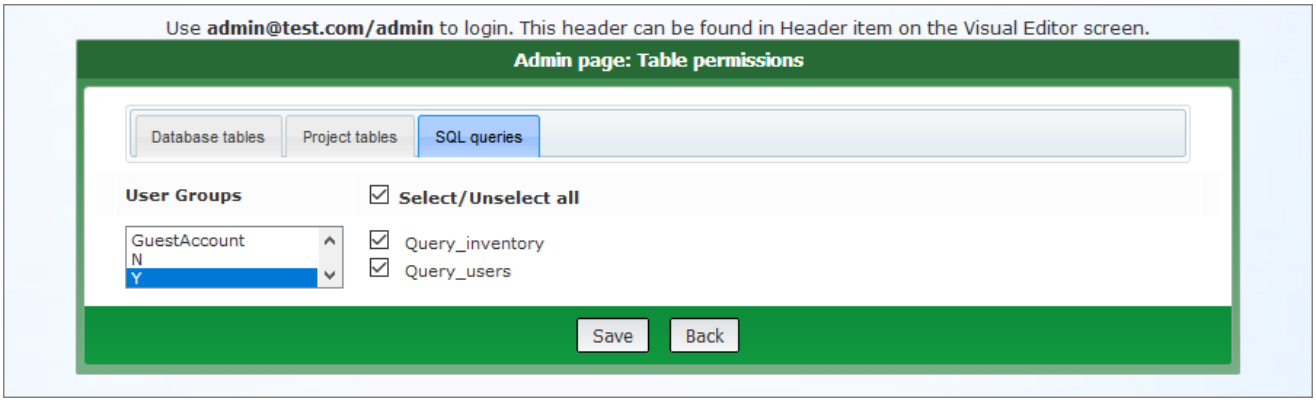
Here is the **Database tables** permissions tab:



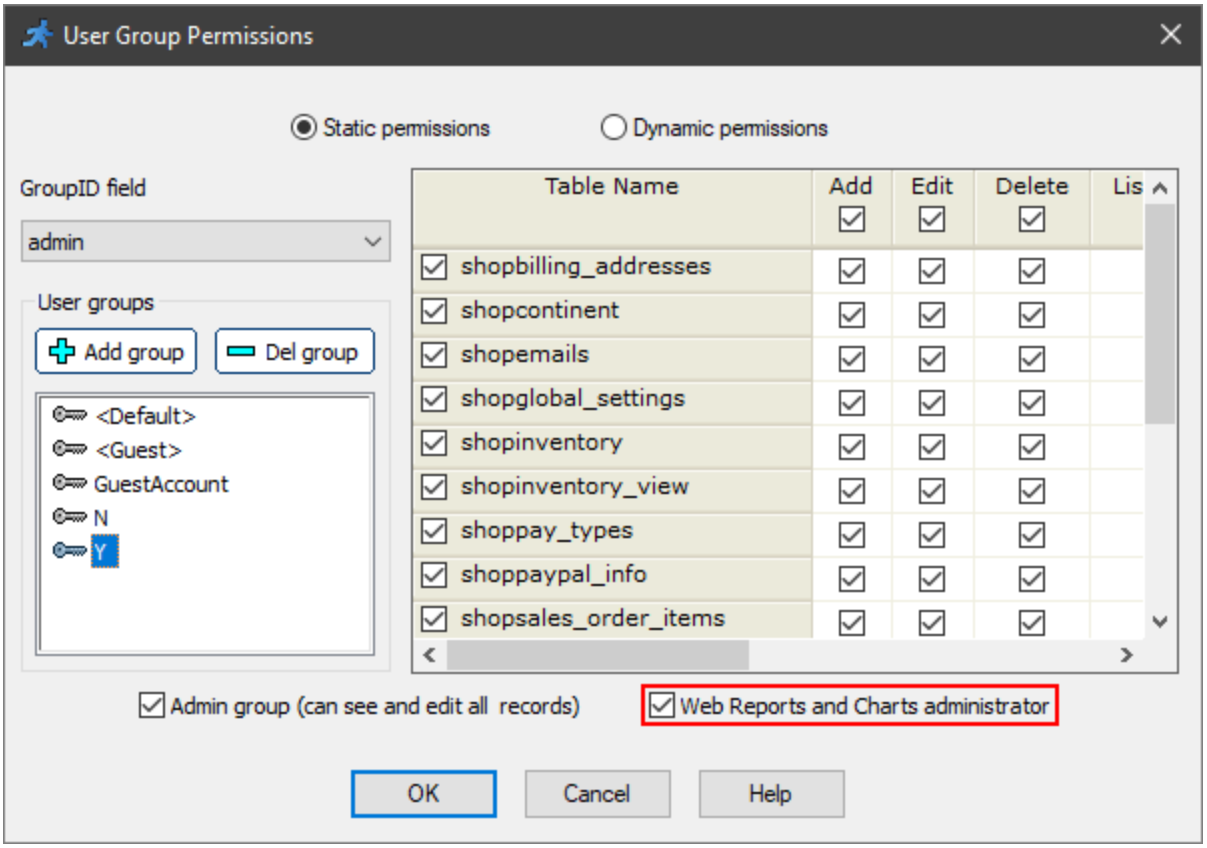
Here is the **Project tables** permissions tab:



Here is the **SQL queries** permissions tab:

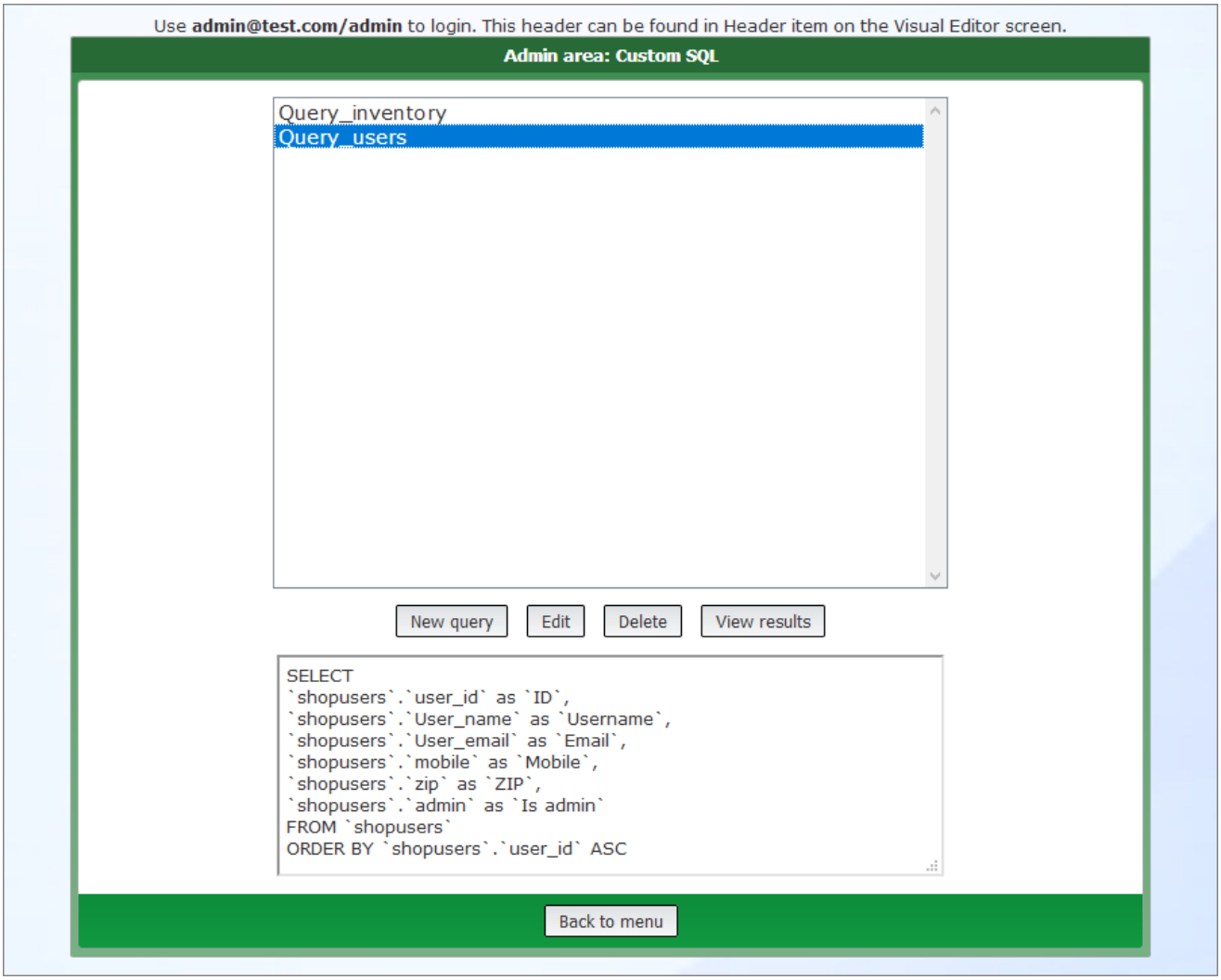


The **Admin** area is available only for the web reports and charts administrator who can be assigned in the Security -> Permissions popup.



Custom SQL

A user with admin permissions (admin) can create, edit, and delete custom SQL queries by clicking the **Custom SQL** button. These queries can be used by other users as data sources for reports and charts. For more information, see Custom SQL.



See also:

- Creating web reports
- Creating web charts
- Custom SQL
- Creating and configuring reports
- Creating charts

3.5.2 Creating web reports

Quick jump

To create a new report, click the **Create Report** button on the start page and follow the steps to configure the report.

The **Back** and **Next** buttons allow you to move to previous and next pages correspondingly. Use the **Jump to** button to jump to any page in the report creation. The **Save** button saves the report and redirects you to the start page. Use the **SQL Query** button to view the SQL query and its results. The **Preview** button allows you to preview the report.

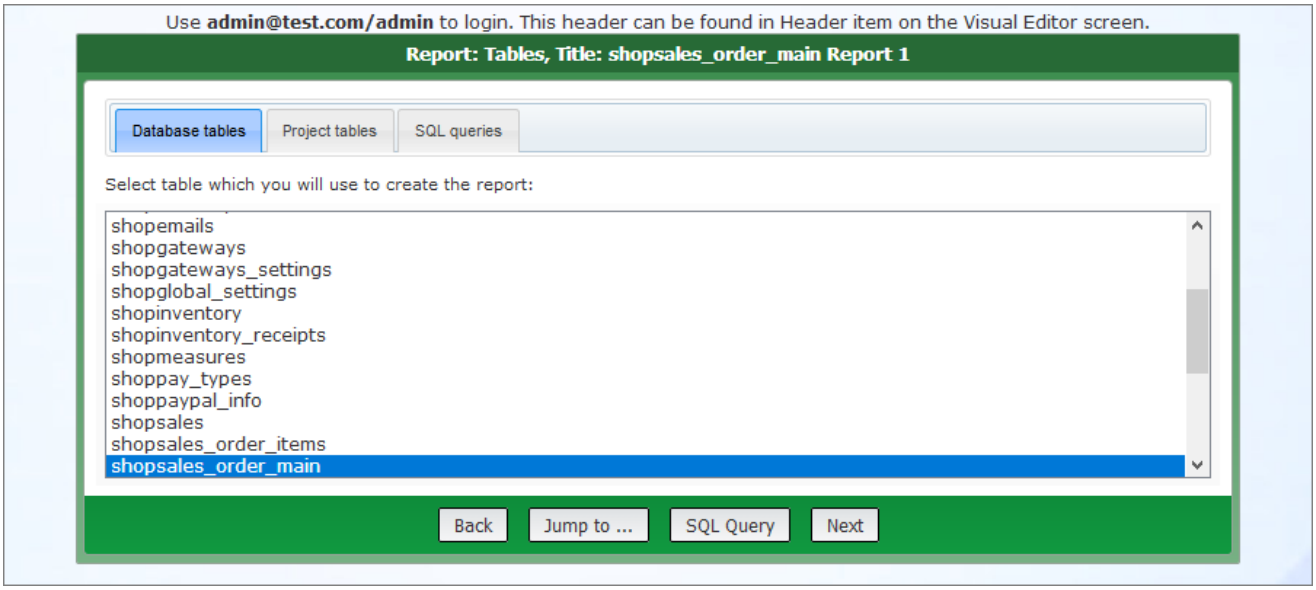
You can find the description of the report creation steps below.

Tables

This page allows you to choose a table or an SQL query as a data source for your report.

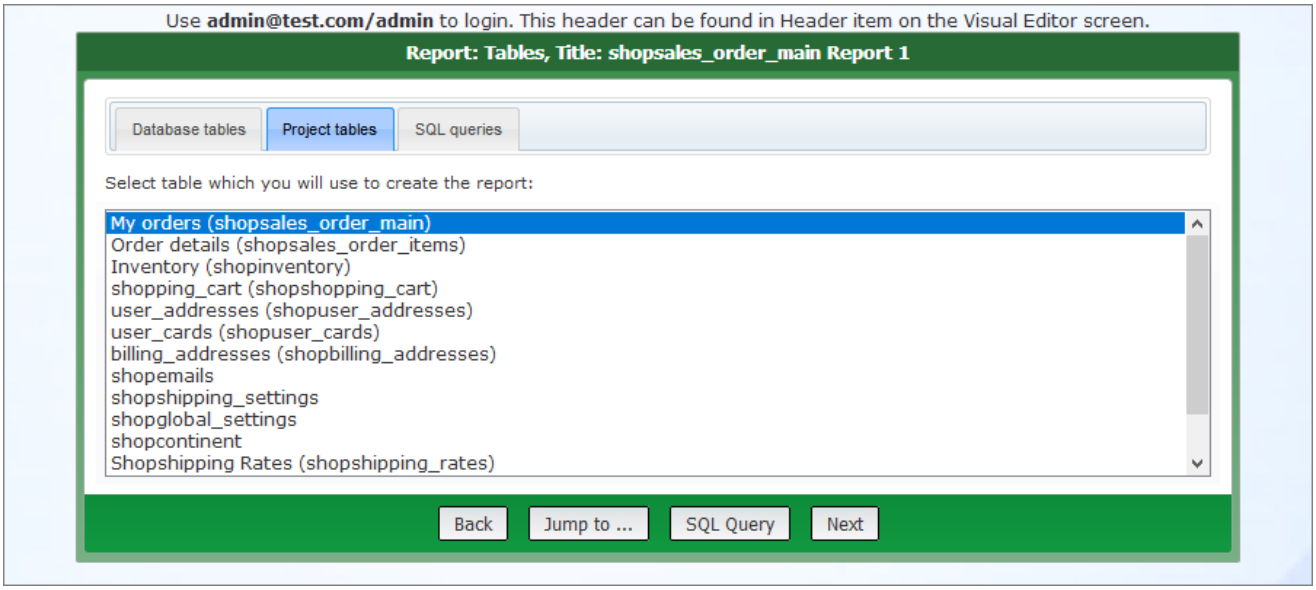
Database tables

- All tables in the database are available for selection. User tables (custom views) are not available.
- When viewing a report, user permissions for the tables (static and dynamic permissions, advanced security options) and "view/edit" field settings do not work. You need to edit the permissions for each report on the Dynamic permissions page.
- During the report configuration, you can create table relations (SQL joins) to query data from two or more tables and add additional search conditions using the WHERE clause.



Project tables

- All tables and custom views added to the project in the left panel of the Datasource tables screen are available for selection.
- When viewing a report, user permissions for the tables and "view/edit" field settings work as usual.
- During the report configuration, you are not able to create table relations (SQL joins). The SQL queries defined on the SQL query screen in PHPRunner are used to query the data for the report.
- The tables for selection are displayed as **Caption (table title)**, e.g., *My orders (shopsales_order_main)*.



SQL queries

This tab allows you to select the queries created on the Custom SQL page as a data source for your report.

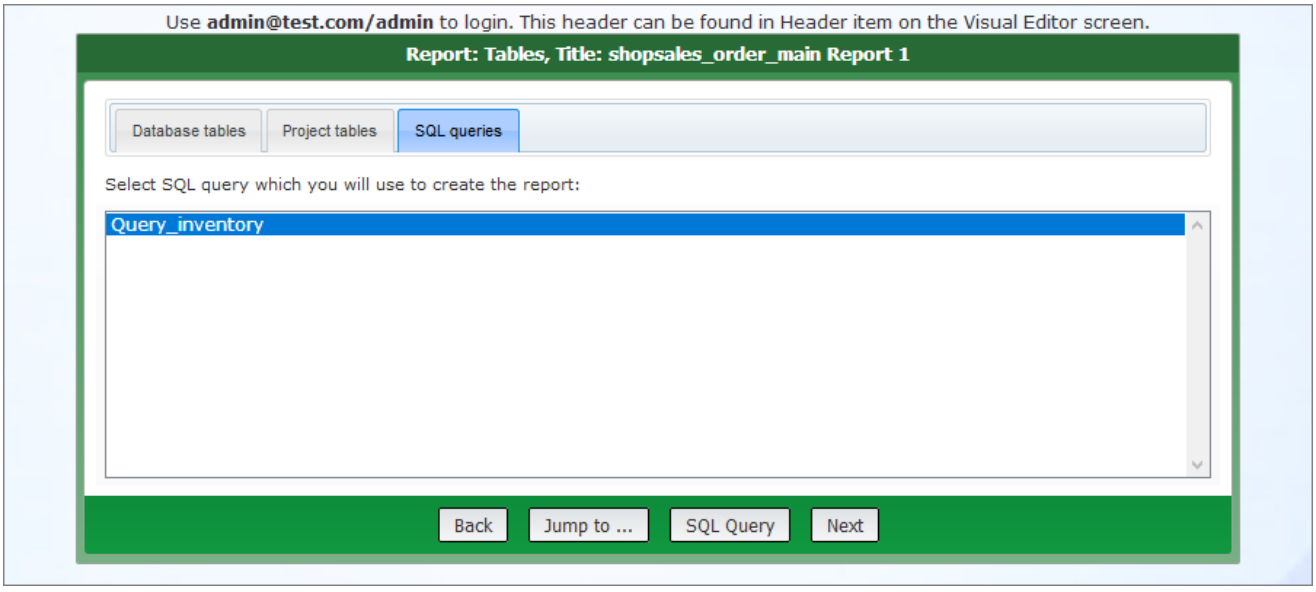
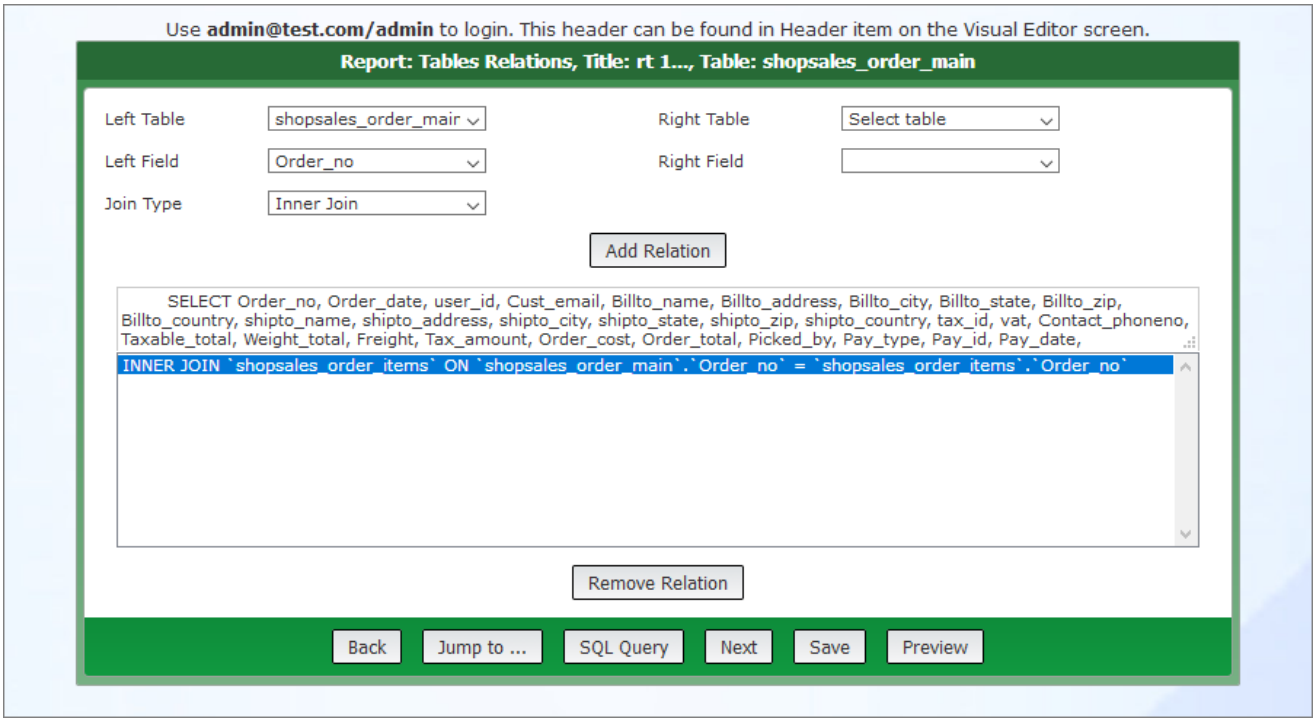


Table relations

Note: this page becomes available if you selected a table from the **Database tables** tab as a data source.

On the **Table relations** page, you can create table relations (SQL joins) to query data from two or more tables, based on the relationship between certain fields in these tables. You can add **Inner Join**, **Left Join**, **Right Join**, and **Full Outer Join**.

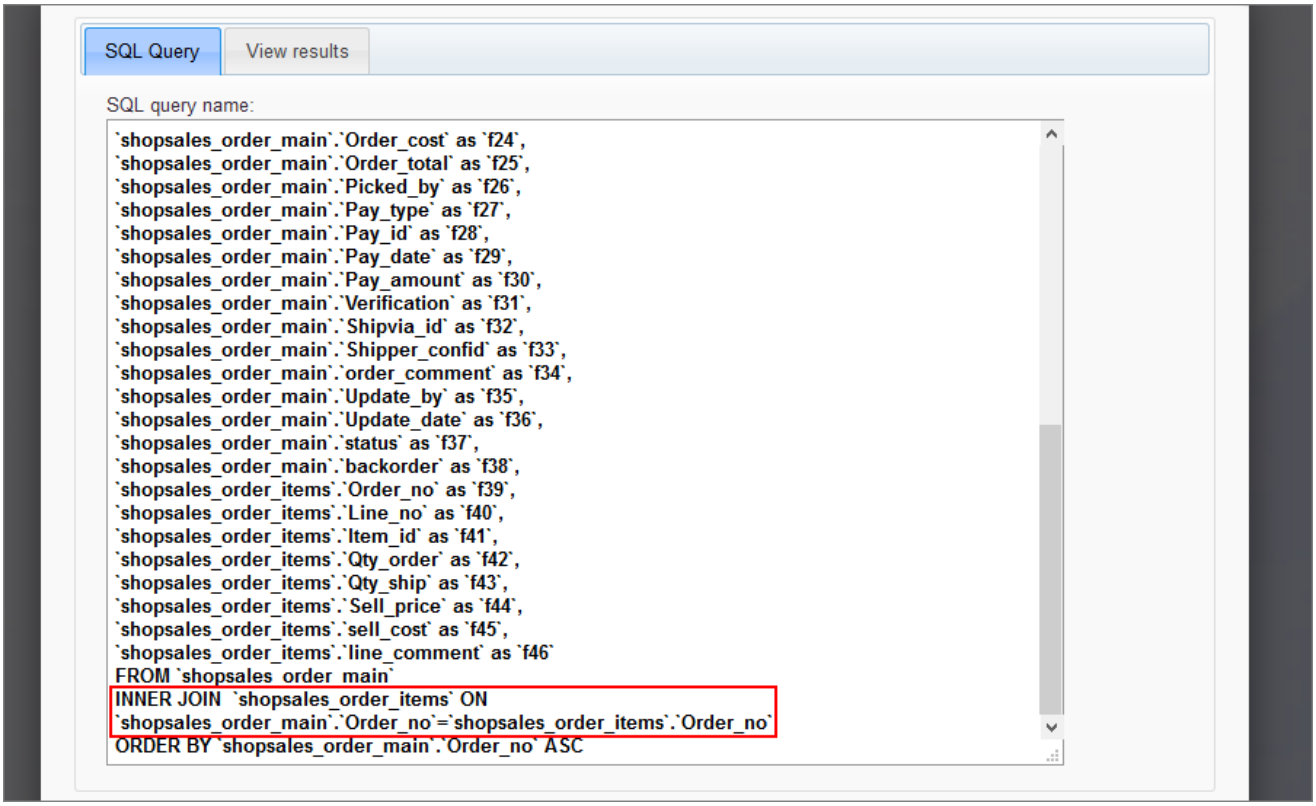
To add an SQL join, choose tables and fields to be joined, pick one of the join types, and click **Add Relation**. The JOIN clause is added below the SELECT clause.



Note: you can add several table relations.

Use the **SQL Query** button to view the complete SQL query and its results.

Here is how the query looks like for the example above:



Here is how the results look like for the example above:

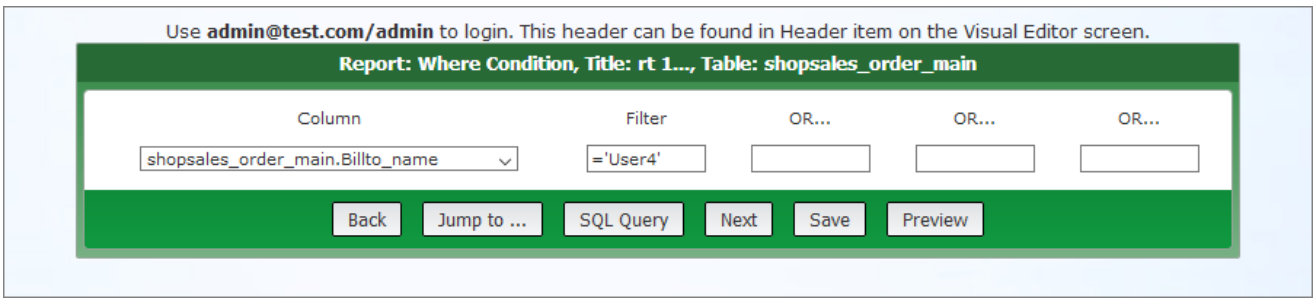
f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	f15	f16	f17	f18	
14	user3@user.com	User3	312 Alexander ave	New York	NY	10454	US	User3	312 Alexander ave	New York	NY	10454	US			1
14	user3@user.com	User3	312 Alexander ave	New York	NY	10454	US	User3	312 Alexander ave	New York	NY	10454	US			1
13	user2@user.com	User2	60 Madison st	New York	NY	10038	US	User2	60 Madison st	New York	NY	10038	US			1
13	user2@user.com	User2	60 Madison st	New York	NY	10038	US	User2	60 Madison st	New York	NY	10038	US			1
12	user1@user.com	User1	1432 Pacific St	New York	NY	11216	US	User1	1432 Pacific St	New York	NY	11216	US			1
12	user1@user.com	User1	1432 Pacific St	New York	NY	11216	US	User1	1432 Pacific St	New York	NY	11216	US			1
15	user4@user.com	User4	168-02 Jewel Ave	New York	NY	11365	US	User4	168-02 Jewel Ave	New York	NY	11365	US			1
15	user4@user.com	User4	168-02 Jewel Ave	New York	NY	11365	US	User4	168-02 Jewel Ave	New York	NY	11365	US			1

Use the **Remove Relation** button to delete the selected table relation.

WHERE condition

> Note: this page becomes available if you selected a table from the **Database tables** tab as a data source.

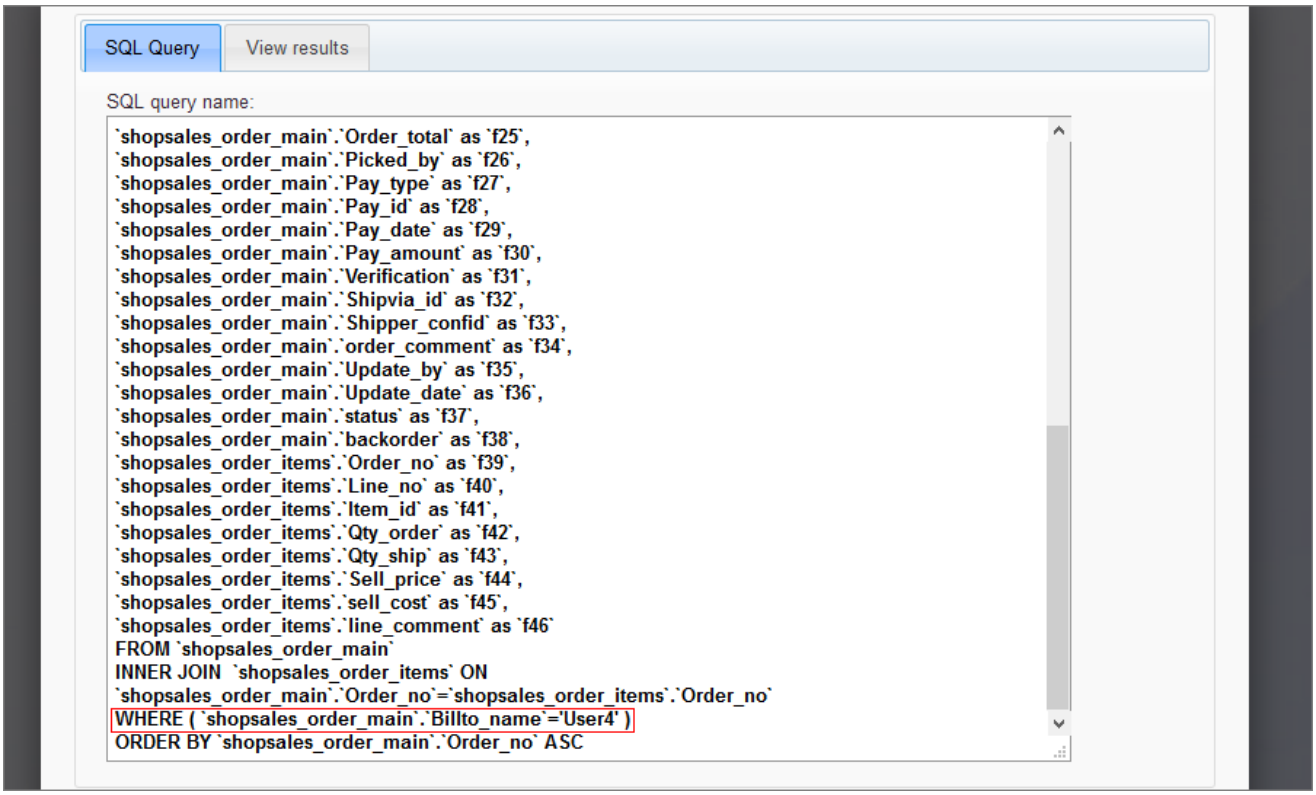
On the WHERE condition page, you can add additional filter conditions using the WHERE clause. Select the field and type in the filter criteria using the text boxes on the right. The filter should be added as <operator><value>. E.g., `='USA'; =2009; <>'red'; >10`.



Note: if you need several criteria, fill in the first one. Additional fields appear after you reload the page.

Use the **SQL Query** button to view the complete SQL query and its results.

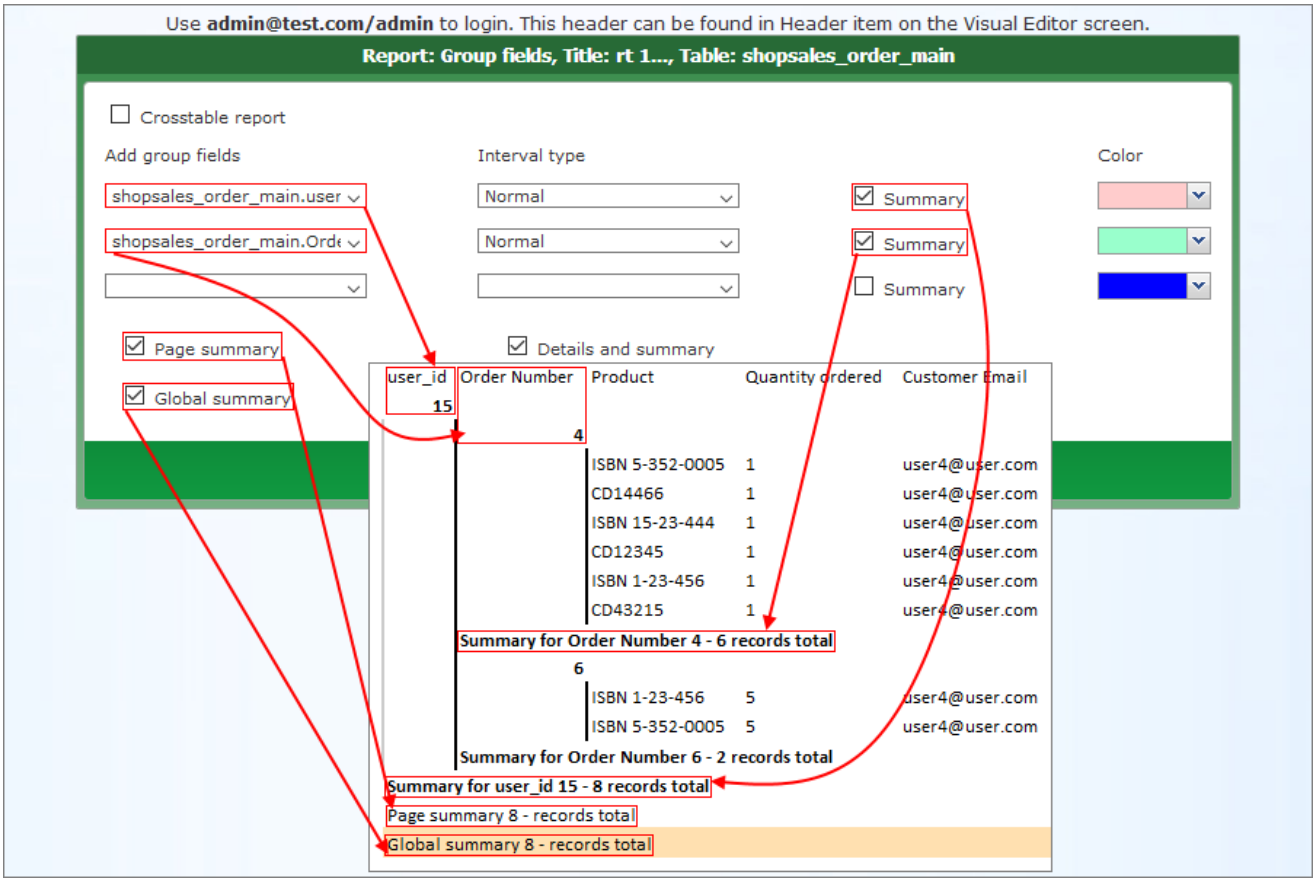
Here is how the query looks like for the example above:



Here is how the results look like for the example above:

Group fields

On the **Group fields** page, you can group the results by one or more columns. The following picture explains how this page works.



See also:

- Creating and configuring reports: Group fields

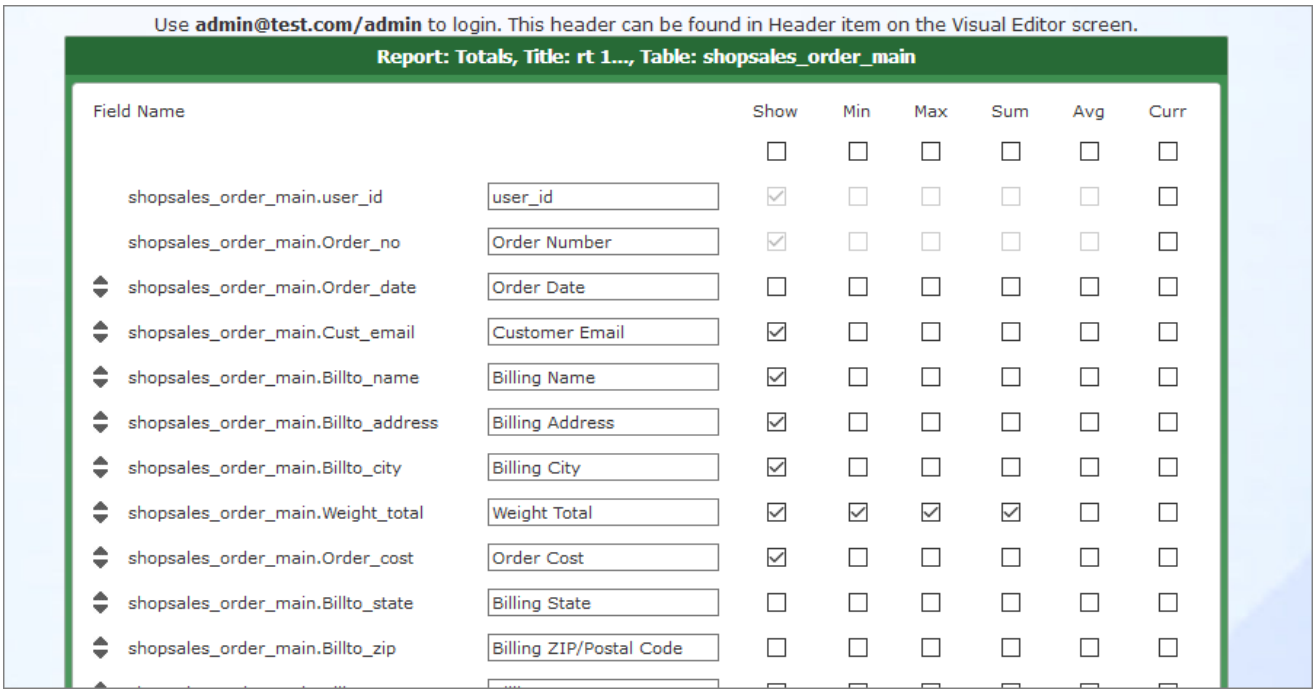
If you clear the **Details and summary** checkbox, only the summary is shown in the report.

You can use other interval types, besides **Normal** (a new group starts when the group field value changes). Available interval types are different for each data type.

Totals

On the **Totals** page, you can choose what fields to display in the report and specify their labels. You can also apply aggregate functions like MIN, MAX, SUM, and AVERAGE. The results of these calculations are displayed after each group and at the end of the page/report.

Use arrows on the left of the field names to change the order of the fields.



Note: you can't modify the settings for the group fields.

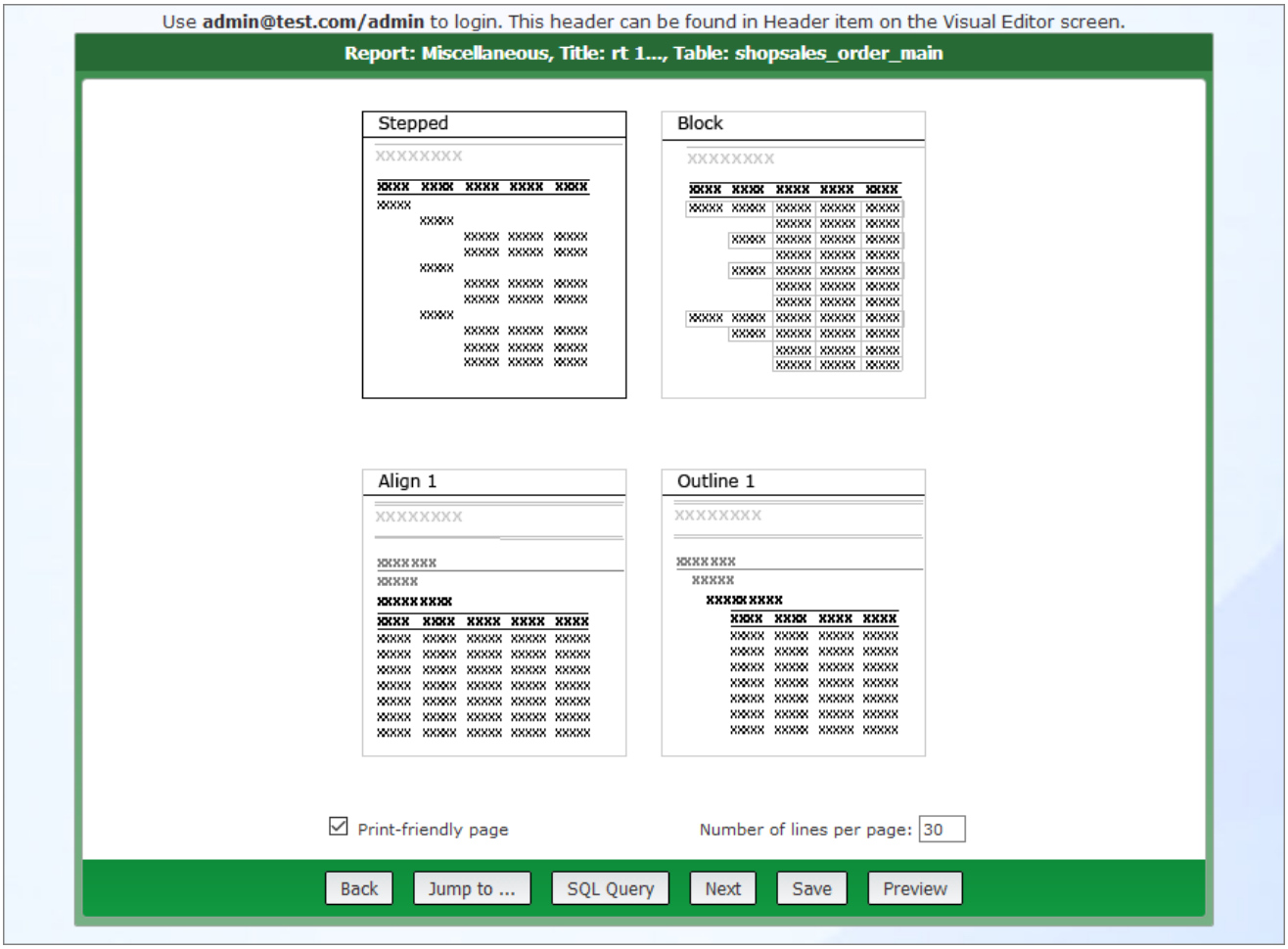
Miscellaneous

On the **Miscellaneous** page, you can choose the report layout. If you use grouping, you can choose between **Stepped**, **Block**, **Outline**, and **Align** layouts. If you don't use grouping, you can use only the **Tabular** layout, which is similar to the default **List** page.

The **Print-friendly page** checkbox enables/disables the following features:

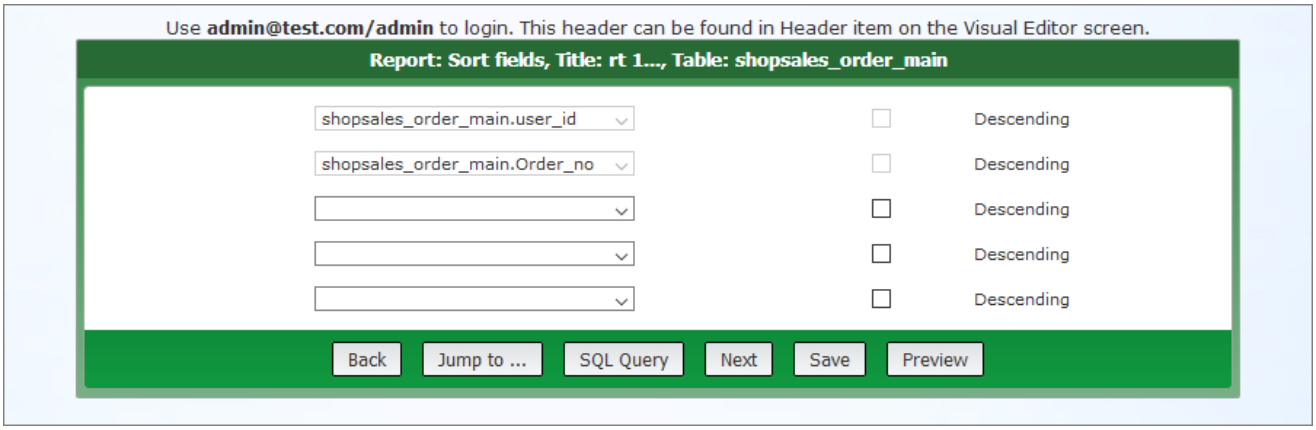
- printing the whole report or its page using a printer-friendly version;
- opening the report as a Microsoft Word or Microsoft Excel document.

Use the **Number of lines per page** option to determine where to insert the page break, when you print the entire report.



Sort fields

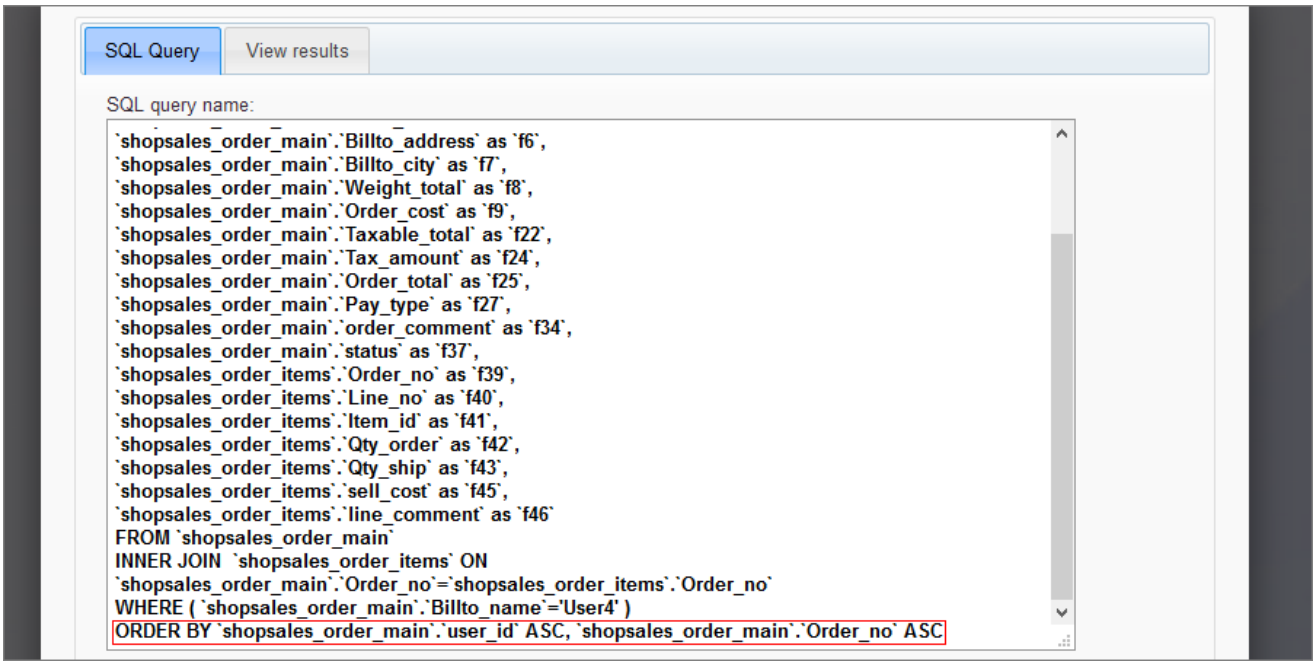
On the **Sort fields** page, you can define the sort order for the records in the report.



> **Note**: you can't modify the settings for the group fields.

Use the **SQL Query** button to view the complete SQL query and its results.

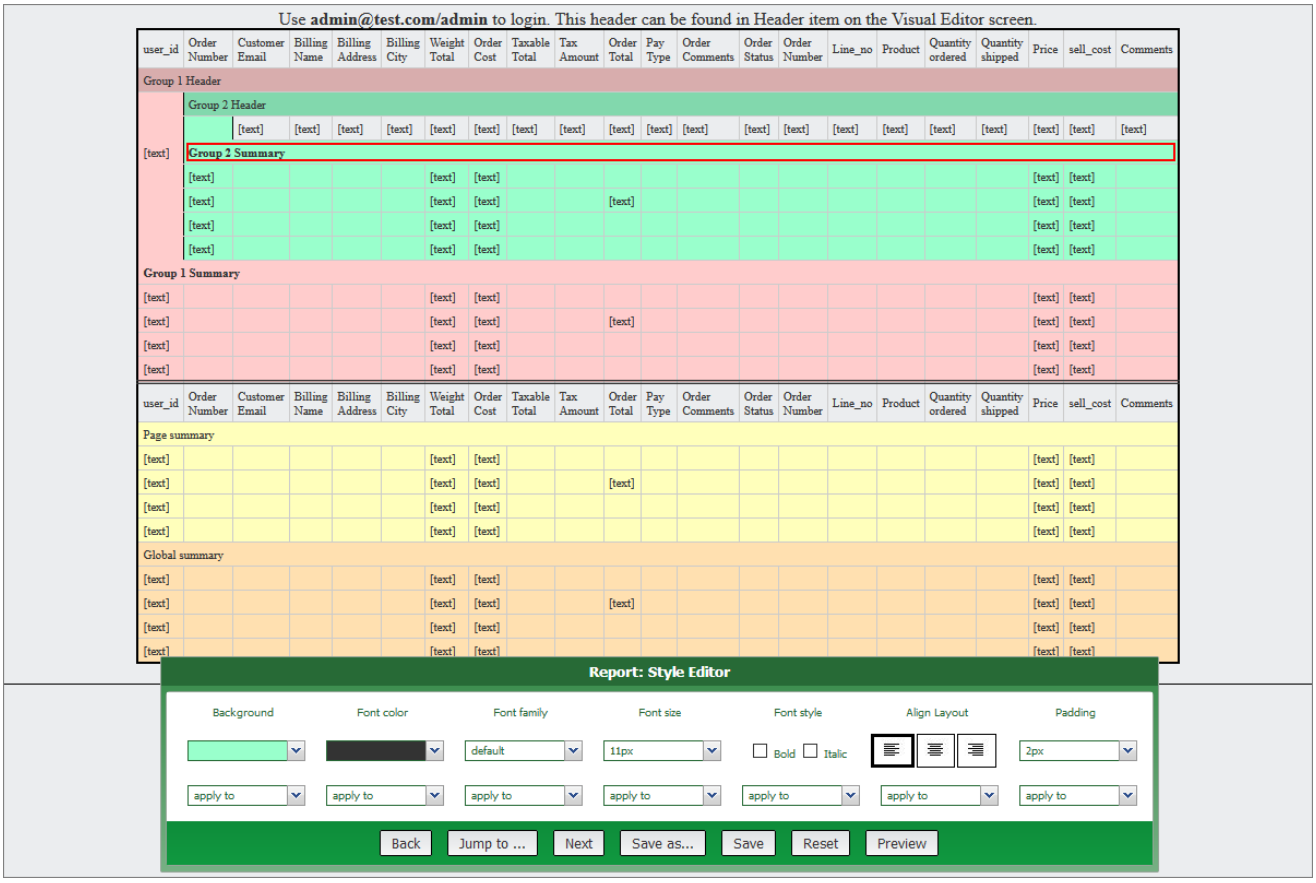
Here is how the query looks like for the example above:



Style Editor

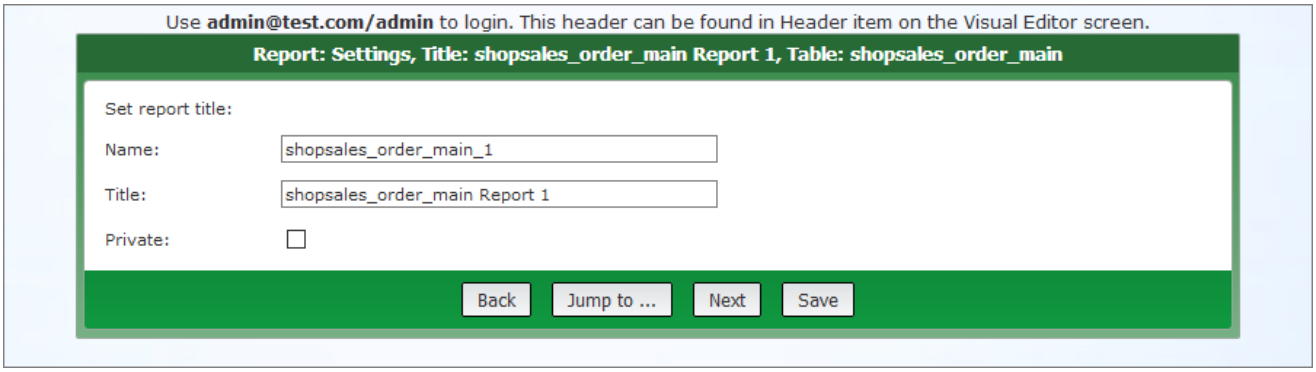
On the **Style Editor** page, you can define the font settings and background color for the cells in the report.

Select the cell and define the style settings for it. Using the **apply to** dropdown, you can apply the selected style to a group (row), field (column), or the entire report. Use the **Reset to default** button to return to default settings.



Settings

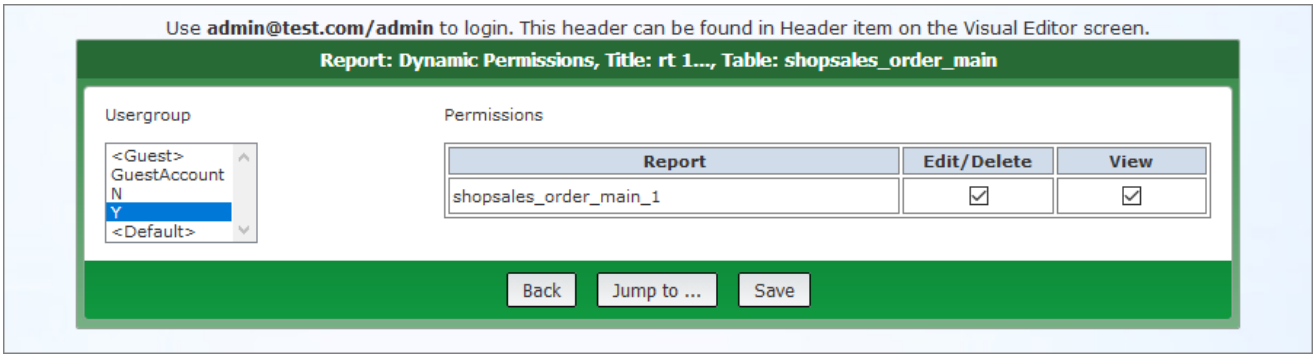
On the **Settings** page, you can define the report name and title. If your project uses Security settings, you also have an option to make the report private. Private reports are not accessible by anyone but the owner. Non-private (public) ones appear under the "shared" section on the start page.



Dynamic permissions

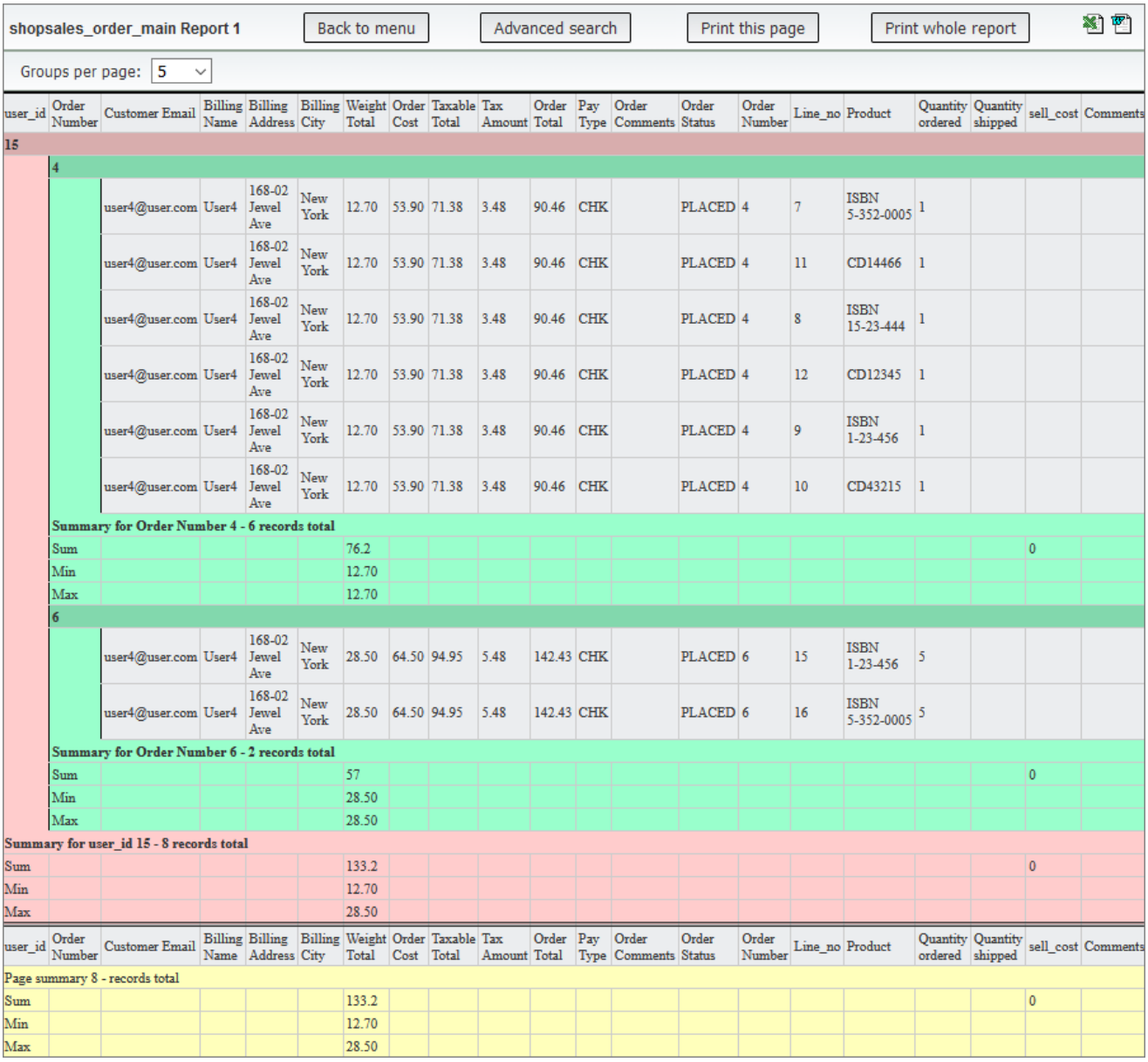
> Note: this page is available if you enabled dynamic permissions in PHPRunner and the report is not marked as private on the previous step.

On the **Dynamic permissions** page, you can assign user group permissions to view/edit/delete the report.



Result

Here is how the resulting report looks like:



shopsales_order_main Report 1 | Back to menu | Advanced search | Print this page | Print whole report

Groups per page: 5

user_id	Order Number	Customer Email	Billing Name	Billing Address	Billing City	Weight Total	Order Cost	Taxable Total	Tax Amount	Order Total	Pay Type	Order Comments	Order Status	Order Number	Line_no	Product	Quantity ordered	Quantity shipped	sell_cost	Comments
15																				
	4																			
		user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4	7	ISBN 5-352-0005	1			
		user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4	11	CD14466	1			
		user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4	8	ISBN 15-23-444	1			
		user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4	12	CD12345	1			
		user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4	9	ISBN 1-23-456	1			
		user4@user.com	User4	168-02 Jewel Ave	New York	12.70	53.90	71.38	3.48	90.46	CHK		PLACED	4	10	CD43215	1			
	Summary for Order Number 4 - 6 records total																			
	Sum					76.2													0	
	Min					12.70														
	Max					12.70														
	6																			
		user4@user.com	User4	168-02 Jewel Ave	New York	28.50	64.50	94.95	5.48	142.43	CHK		PLACED	6	15	ISBN 1-23-456	5			
		user4@user.com	User4	168-02 Jewel Ave	New York	28.50	64.50	94.95	5.48	142.43	CHK		PLACED	6	16	ISBN 5-352-0005	5			
	Summary for Order Number 6 - 2 records total																			
	Sum					57													0	
	Min					28.50														
	Max					28.50														
Summary for user_id 15 - 8 records total																				
Sum						133.2													0	
Min						12.70														
Max						28.50														

user_id	Order Number	Customer Email	Billing Name	Billing Address	Billing City	Weight Total	Order Cost	Taxable Total	Tax Amount	Order Total	Pay Type	Order Comments	Order Status	Order Number	Line_no	Product	Quantity ordered	Quantity shipped	sell_cost	Comments
Page summary 8 - records total																				
Sum						133.2													0	
Min						12.70														
Max						28.50														

See also:

- Online report/chart builder
- Creating web charts
- Custom SQL
- Creating and configuring reports
- User group permissions

- Dynamic permission
- Security screen

3.5.3 Creating web charts

Quick jump

To create a new chart, click the **Create Chart** button on the start page and follow the steps to configure the chart.

The **Back** and **Next** buttons allow you to move to previous and next pages correspondingly. Use the **Jump to** button to jump to any page in the chart creation. The **Save** button saves the chart and redirects you to the start page. Use the **SQL Query button** to view the SQL query and its results. The **Preview** button allows you to preview the chart.

You can find the description of the chart creation steps below.

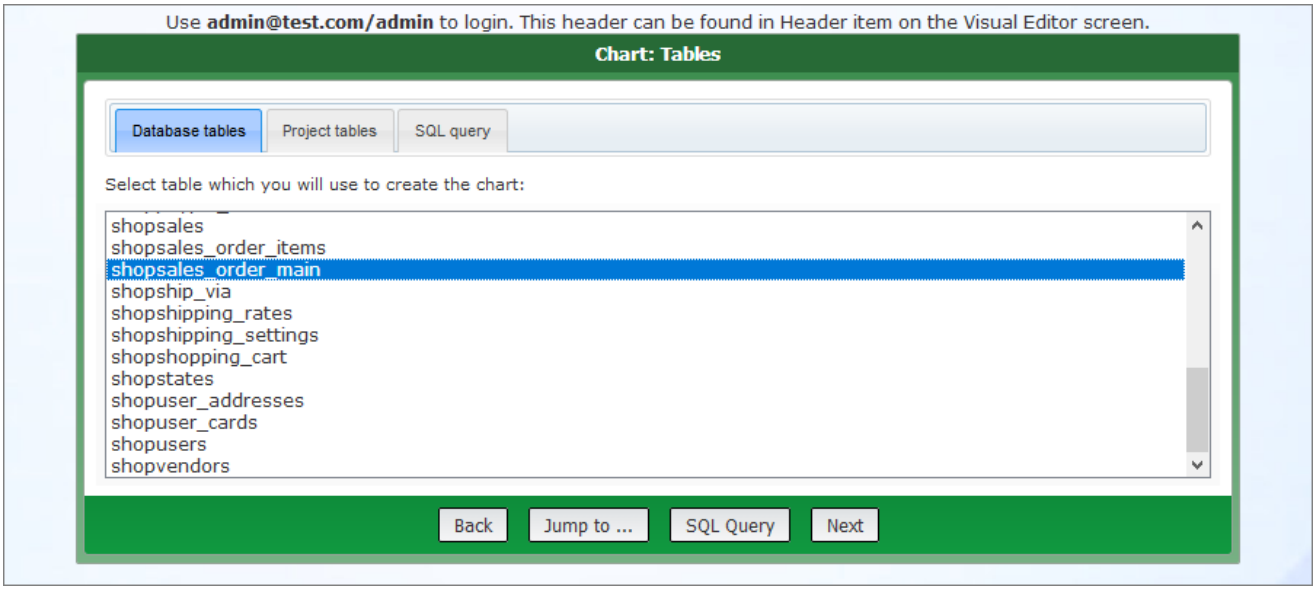
Tables

This page allows you to choose a table or an SQL query as a data source for your chart.

Database tables

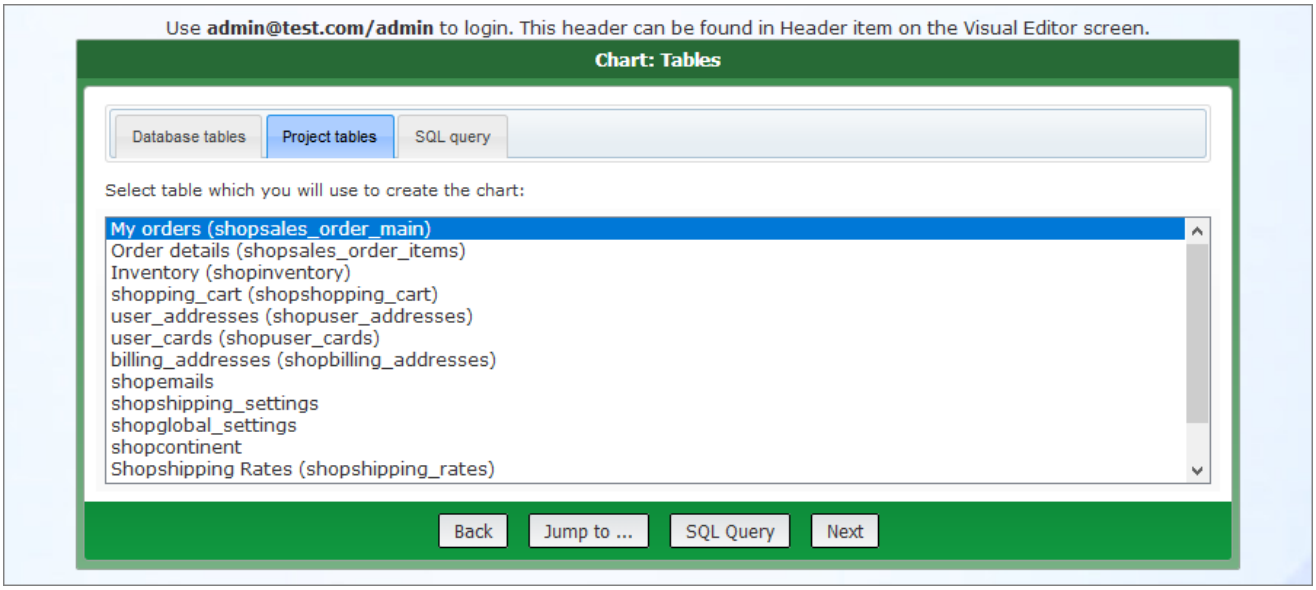
- All tables in the database are available for selection. User tables (custom views) are not available.
- When viewing a chart, user permissions for the tables (static and dynamic permissions, advanced security options) and "view/edit" field settings do not work. You need to edit the permissions for each chart on the Dynamic permissions page.

- During the chart configuration, you can create table relations (SQL joins) to query data from two or more tables and add additional filter conditions using the Group by page.



Project tables

- All tables and custom views added to the project in the left panel of the Datasource tables screen are available for selection.
- When viewing a chart, user permissions for the tables and "view/edit" field settings work as usual.
- During the chart configuration, you are not able to create table relations (SQL joins). The SQL queries defined on the SQL query screen in PHPRunner are used to query the data for the report.
- The tables for selection are displayed as **Caption (table title)**, e.g., *My orders (shopsales_order_main)*.



SQL queries

This tab allows you to select the queries created on the Custom SQL page as a data source for your chart.

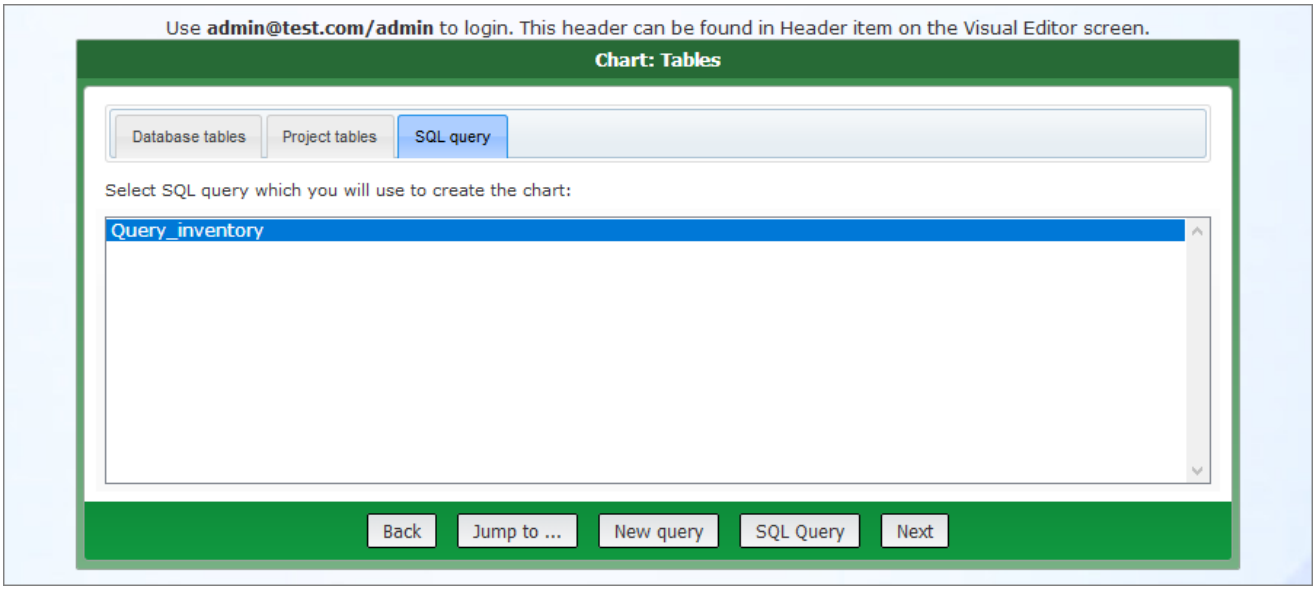
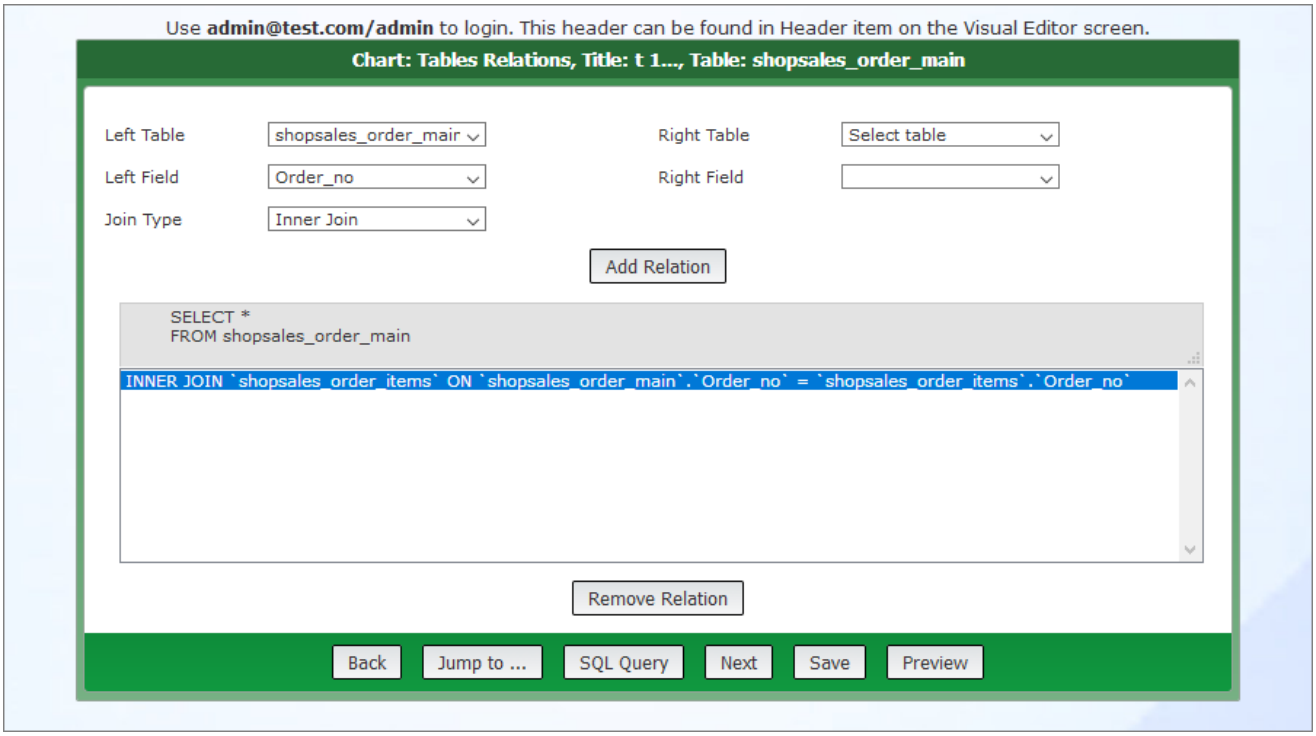


Table relations

> **Note**: this page becomes available if you selected a table from the **Database tables** tab as a data source.

On the **Table relations** page, you can create table relations (SQL joins) to query data from two or more tables, based on the relationship between certain fields in these tables. You can add **Inner Join**, **Left Join**, **Right Join**, and **Full Outer Join**.

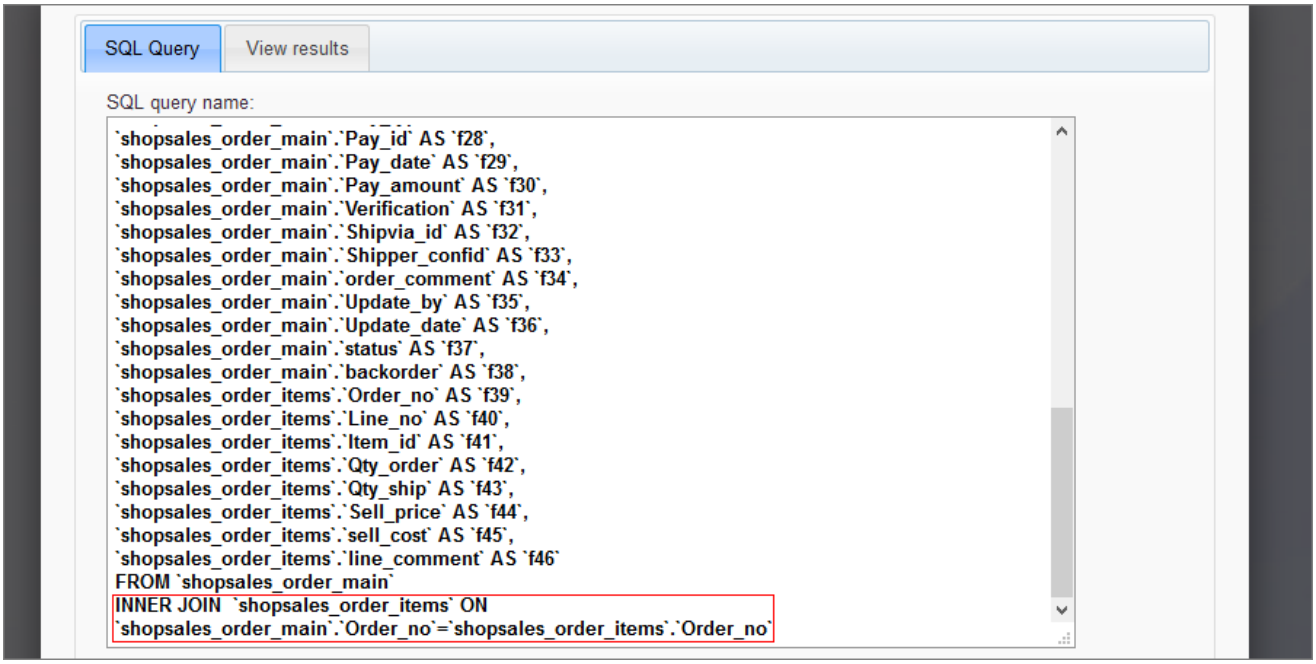
To add an SQL join, choose the tables and fields to be joined, pick one of the join types, and click **Add Relation**. The JOIN clause is added below the SELECT clause.



> **Note**: you can add several table relations.

Use the **SQL Query** button to view the complete SQL query and its results.

Here is how the query looks like for the example above:



Here is how the results look like for the example above:

Use the **Remove Relation** button to delete the selected table relation.

Group by

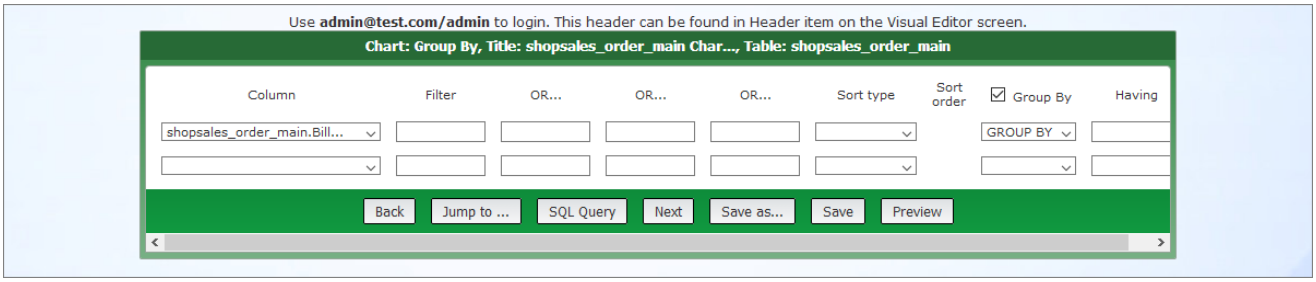
> **Note**: this page becomes available if you selected a table from the **Database tables** tab as a data source.

On the WHERE condition page, you can add additional filter conditions using the WHERE clause. Select the field and type in the filter criteria using the text boxes on the right. The filter should be added as <operator><value>. E.g., `='USA'; =2009; <>'red'; >10`.

You can also choose the **Ascending**/**Descending** sort type to sort the records in the chart. Select a field in the first column and choose the sort type and sort order to do so.

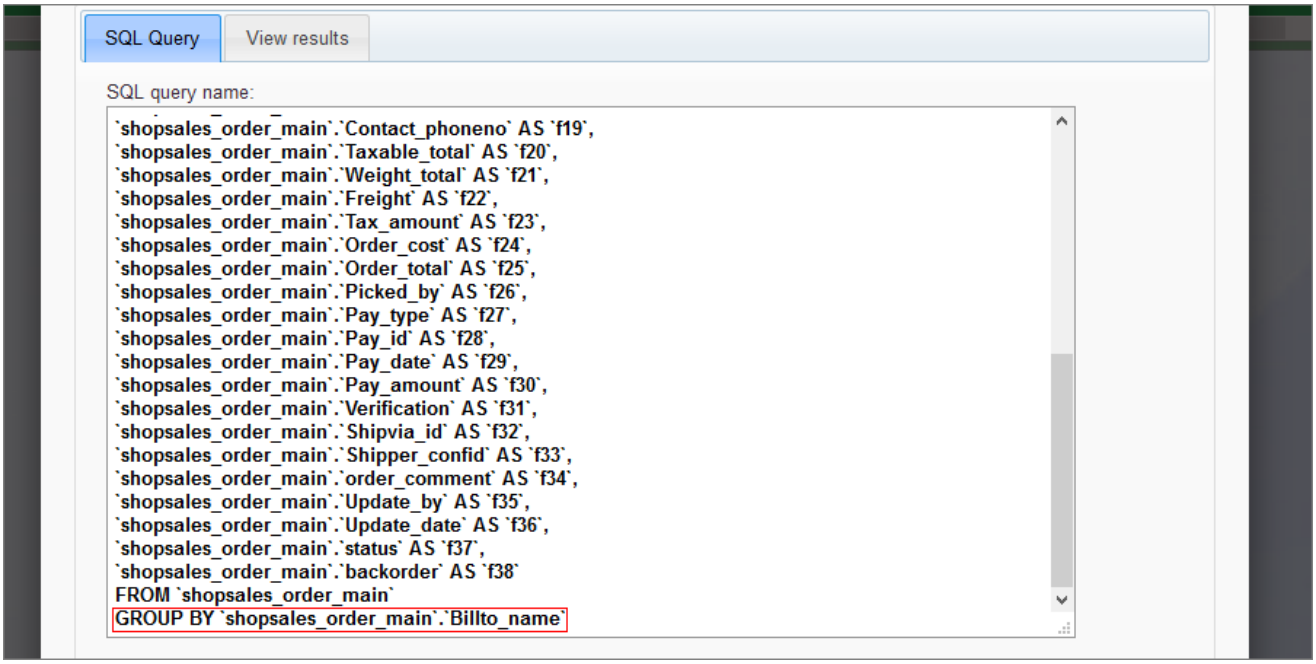
Additionally, you can group the results by one or more columns and apply aggregate functions like MIN, MAX, SUM, AVERAGE, and COUNT. Select a field in the first column, enable the **Group By** checkbox, and choose one of the values in the dropdown under this checkbox.

You can filter the records that the GROUP BY clause returns using the HAVING clause. Type in the condition as <operator><value> in the text box under Having to do so. E.g., `>10 or = 500`.



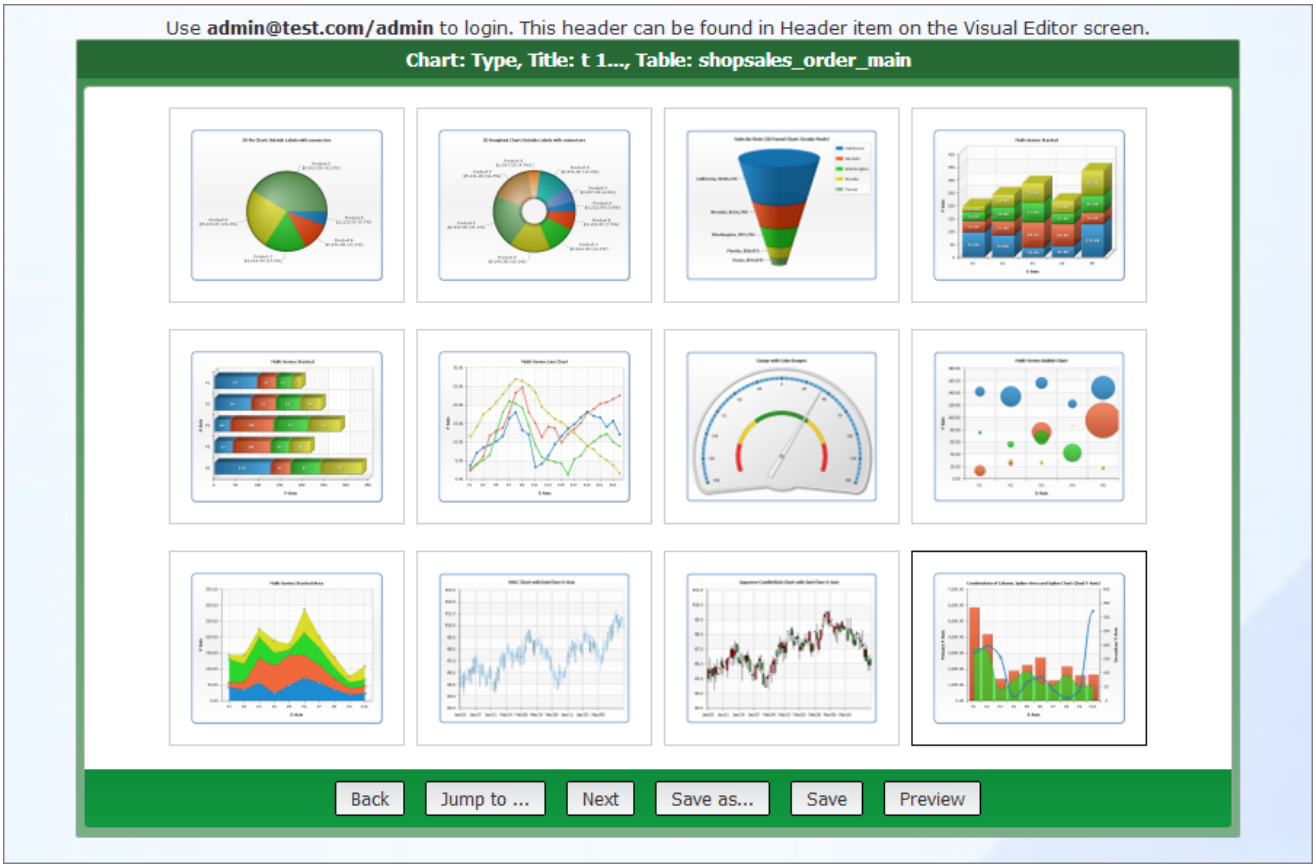
Use the **SQL Query** button to view the complete SQL query and its results.

Here is how the query looks like for the example above:



Type

On the **Type** page, you can select one of the chart types. For more information, see Chart types.

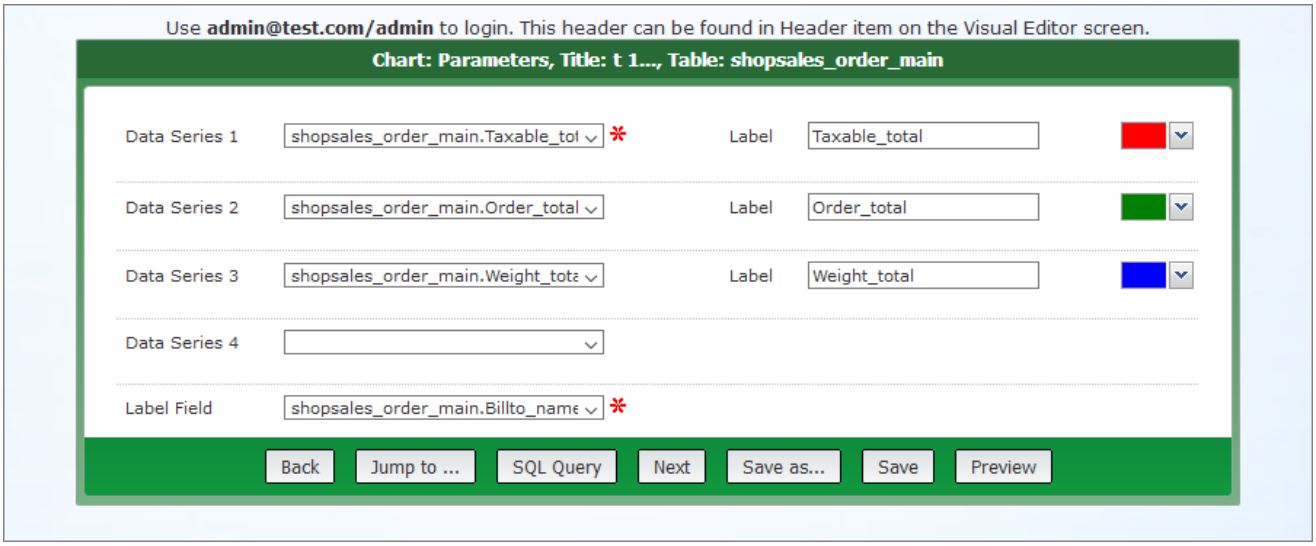


Parameters

On the **Parameters** page, you can choose the Data series fields (fields with data) and the **Label** field (field with data labels).

You can add any number of **Data series** fields. Additional **Data series** dropdown boxes appear automatically once you've used the available ones.

For more information about choosing the **Data series** fields for certain chart types, see Chart types.



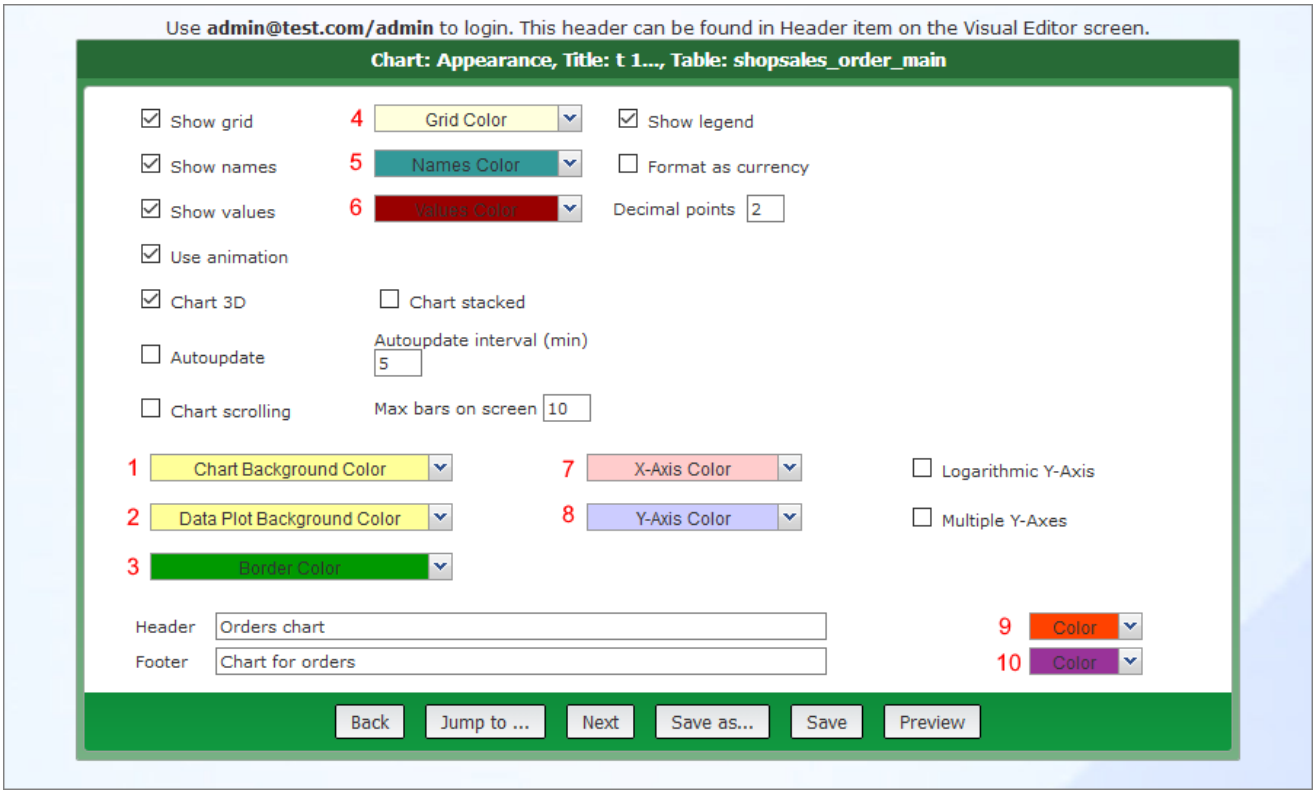
Note: only the numeric fields are available to be chosen as the **Data series**.

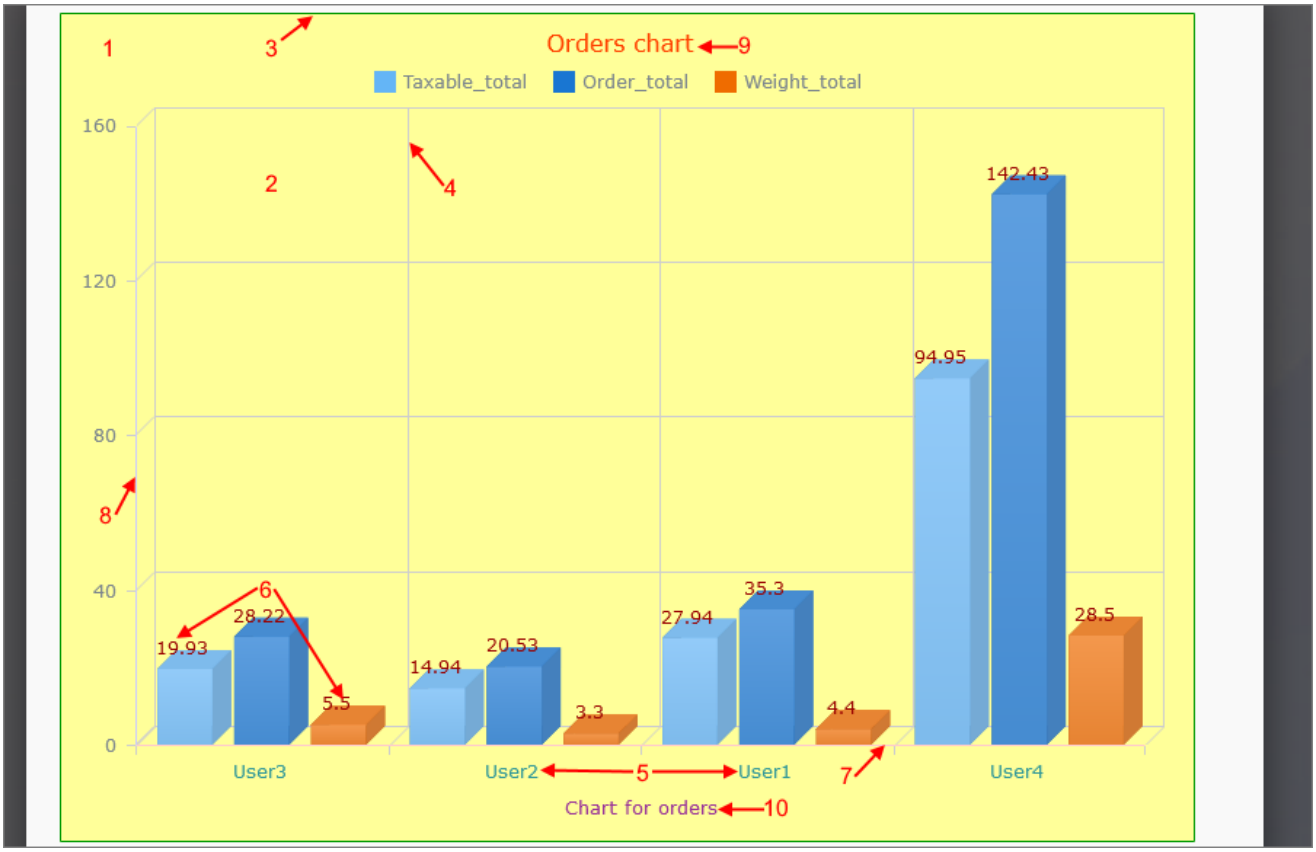
The **Label** text boxes allow you to change the labels of the **Data series** fields.

The **Color** dropdowns define the colors of the **Data series** fields.

Appearance

On the **Appearance** page, you can configure the chart appearance on the web page. The following two pictures show how changing the different colors affects the resulting Column chart.





- Select the **Autoupdate checkbox** to auto-refresh the chart every N seconds.

- The **Use animation checkbox** enables the chart animation upon opening a chart.

- Use the **Logarithmic Y-Axis** option to convert a linear value axis to a logarithmic value axis.

If you select one of the 2D charts (e.g., 2D Column chart), additional options become available:

- Use the **Chart 3D** option to build a 3D chart instead of a 2D one.

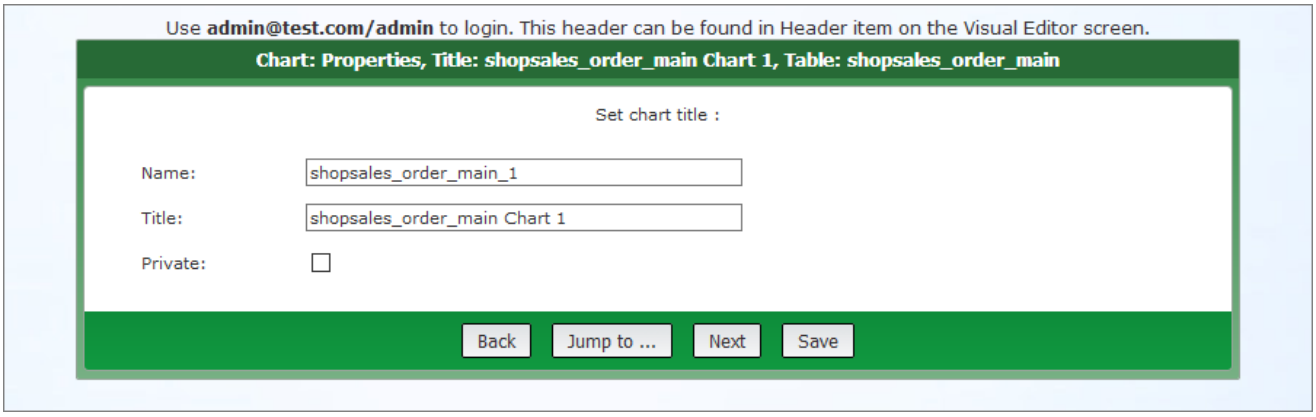
- Use the **Chart stacked** option to display a chart where the chart elements are stacked on top of each other.

The **Y-axis label** input box appears only if you have multiple Data series fields.

For more information about each chart type settings, see Chart types.

Properties

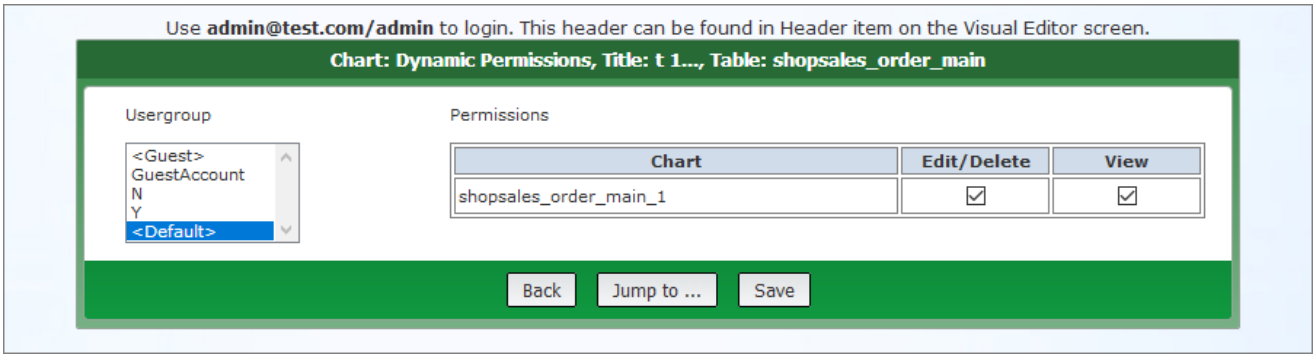
On the **Properties** page, you can define the chart *name* and *title*. If your project uses Security settings, you also have an option to make the chart private. Private charts are not accessible by anyone but the owner. Non-private (public) ones appear under the "shared" section on the start page.



Dynamic permissions

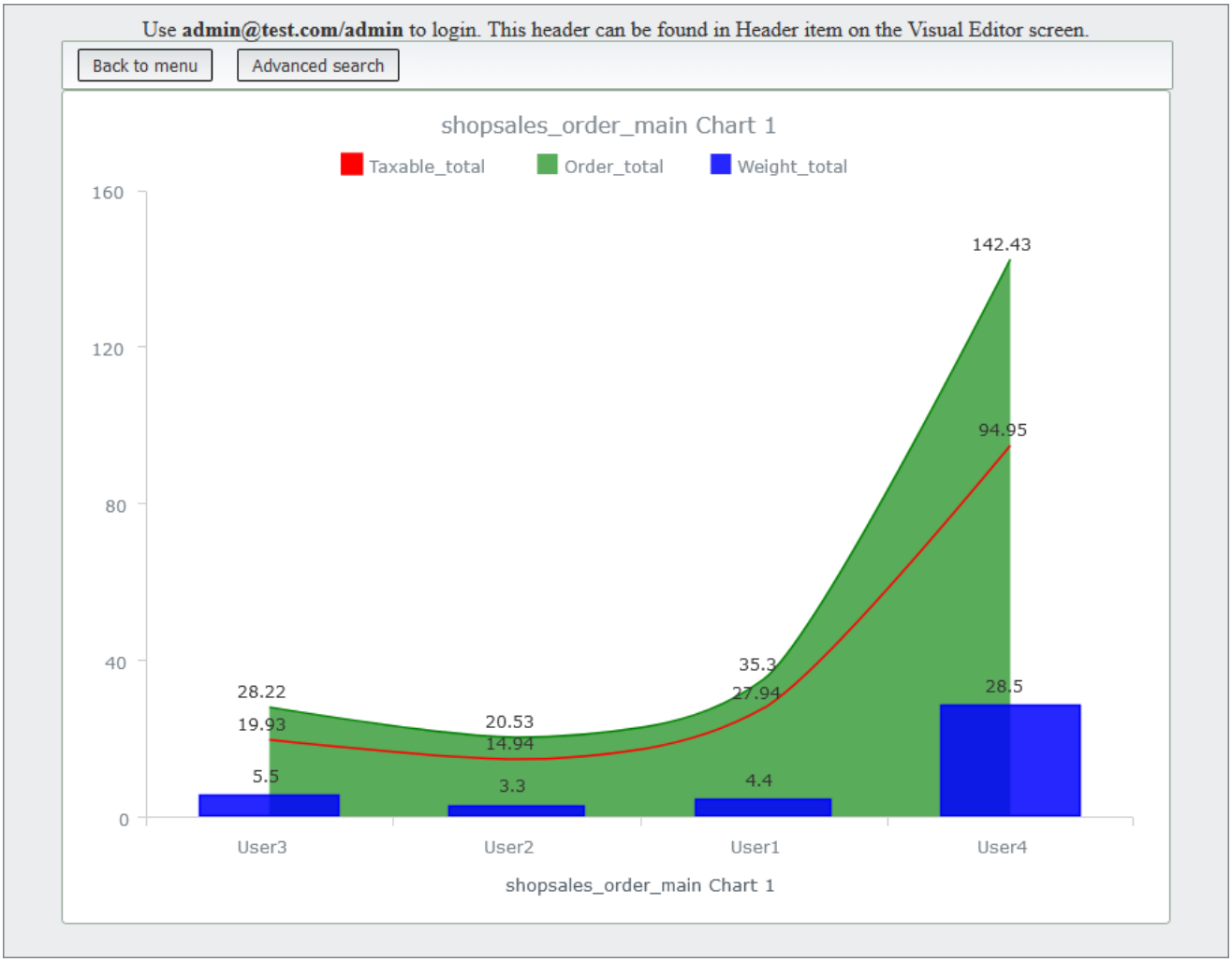
> **Note**: this page is available if you enabled dynamic permissions in PHPRunner and the chart is not marked as private on the previous step.

On the **Dynamic permissions** page, you can assign user group permissions to view/edit/delete the chart.



Result

Here is how the resulting Combined chart looks like with the default appearance settings.



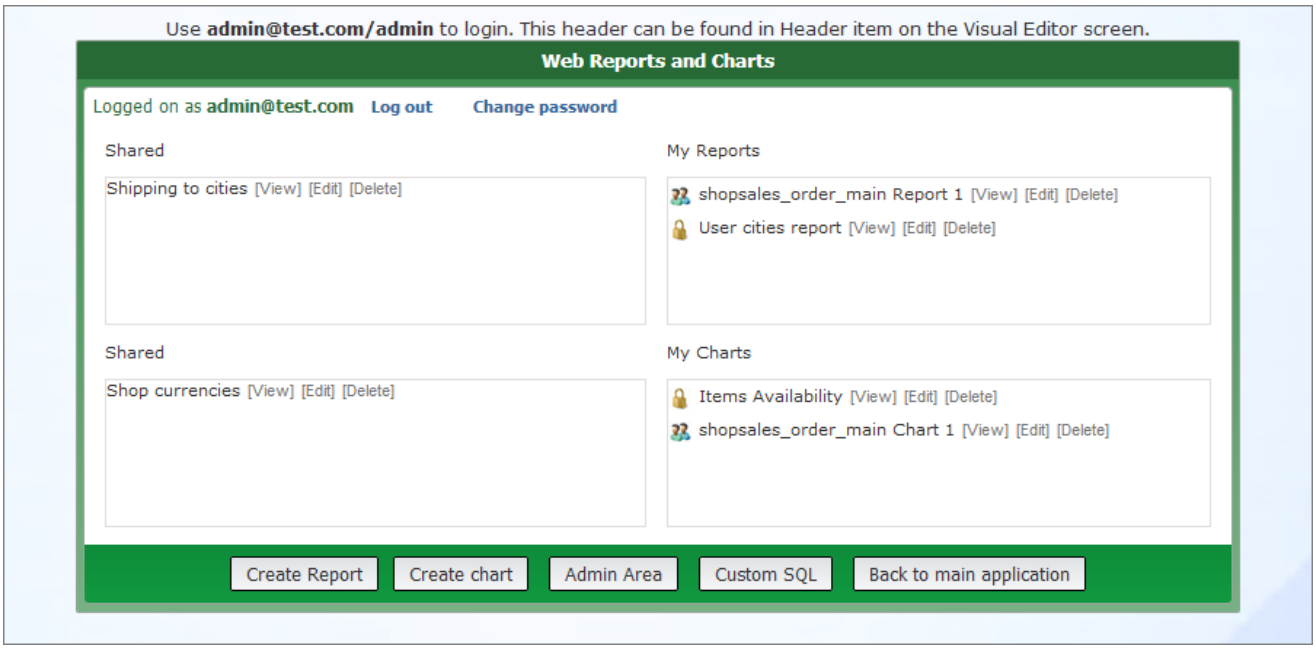
See also:

- Online report/chart builder
- Creating web reports
- Custom SQL
- Creating charts
- A list of chart types
- User group permissions
- Dynamic permission
- Security screen

3.5.4 Custom SQL

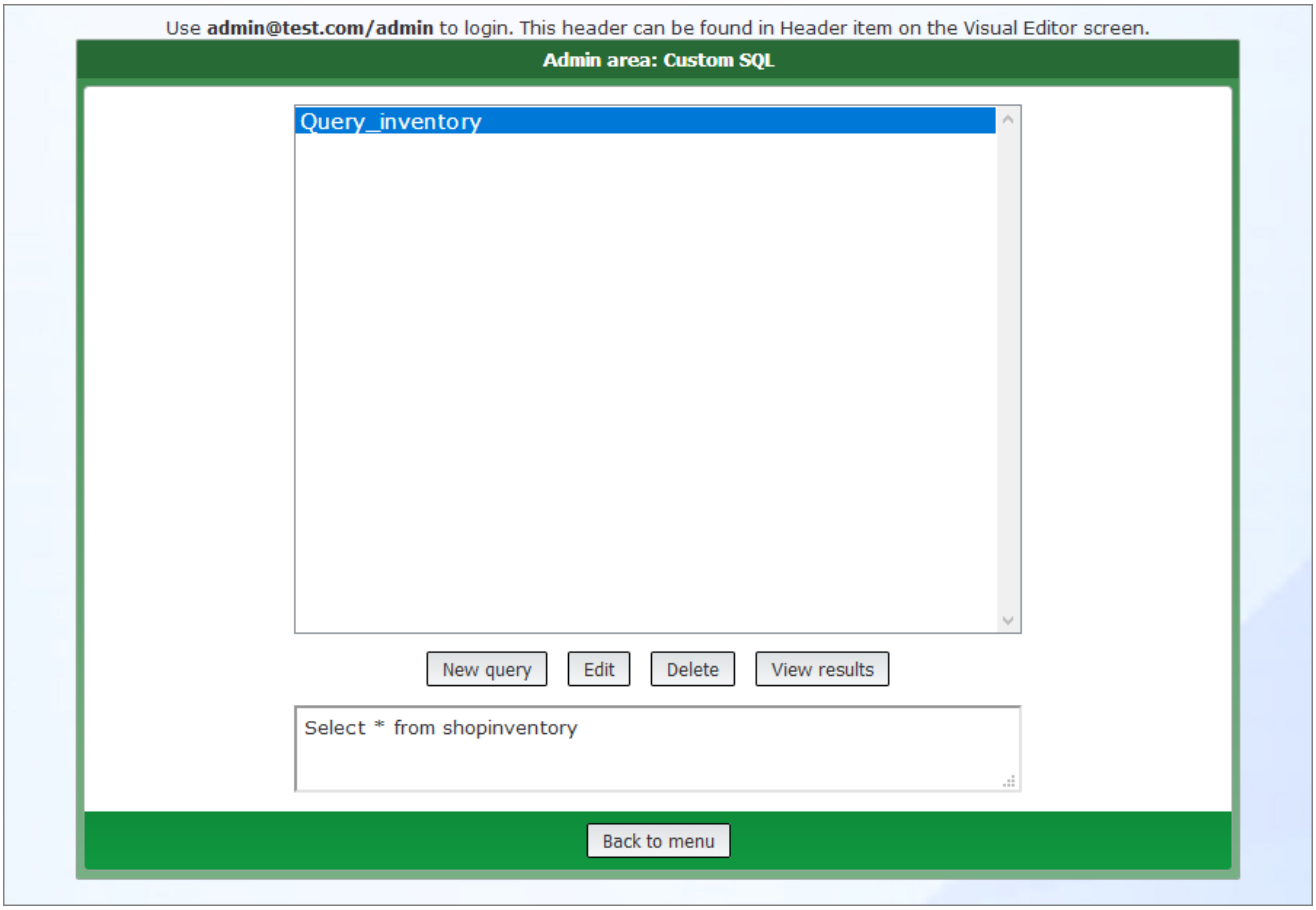
A user with admin permissions (admin) can create custom SQL queries that can be used by other users as a data source for reports and charts. Admin can edit and delete custom SQL queries as well.

To view/create/edit/delete custom SQL queries, click the **Custom SQL** button on the start page.



To create a new custom SQL query:

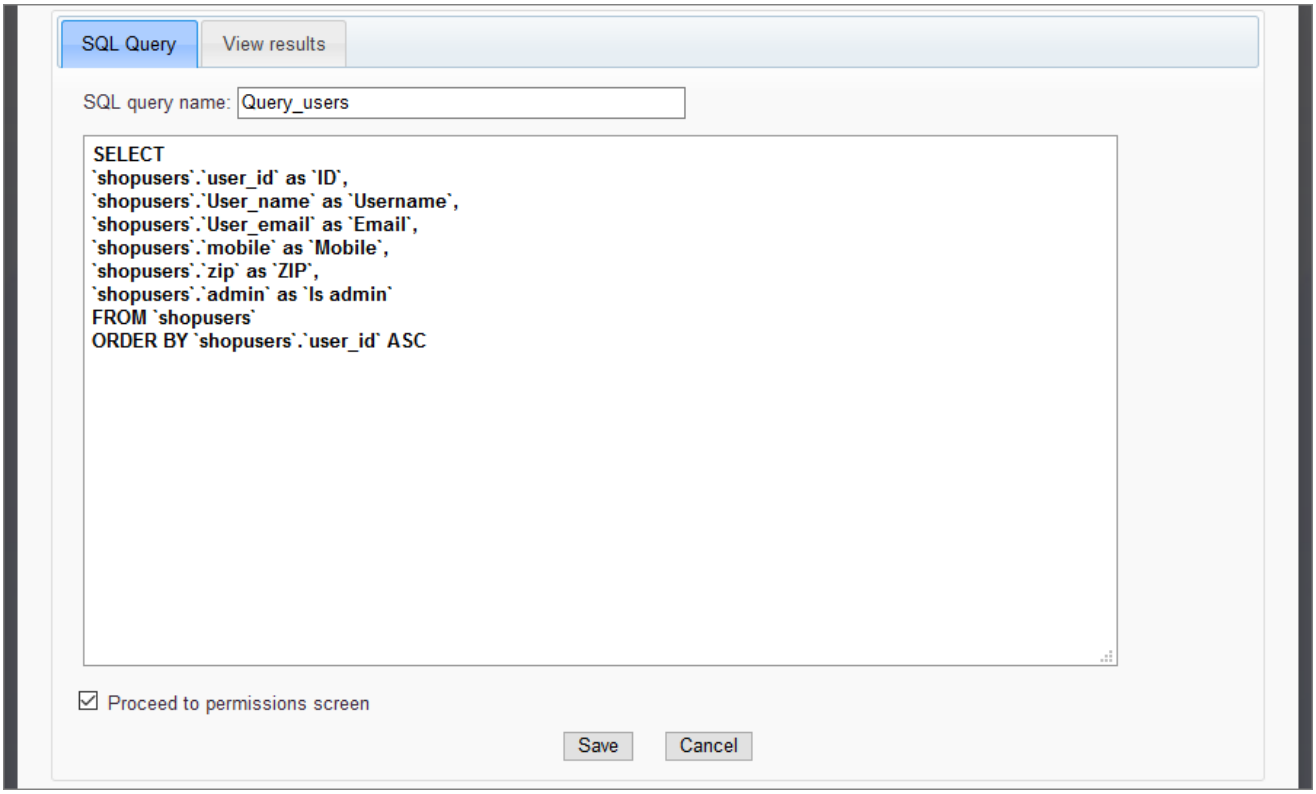
1. Click the **Custom SQL** button on the start page to open the **Custom SQL** page.



2. Click the **New query** button.

3. Enter the *SQL query name* and the *query code*. If you wish to set up permissions for the created query, select the **Proceed to permissions screen** checkbox.

> **Note**: you can set up the permissions later on the Admin page.



> **Note**: you can use stored procedures in the query code for the supported databases (MS SQL Server, MySQL, Oracle, etc.).

Here is an example of calling the stored procedure with parameters in MS SQL Server:

```
EXEC [Employee Sales by Country] '10/10/2007','10/10/2009'
```

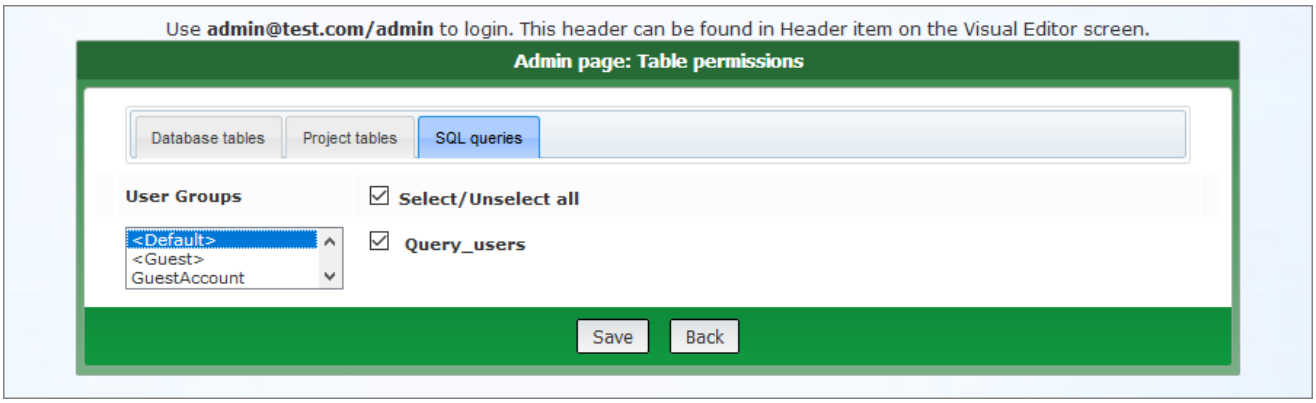
Here is an example of calling the stored procedure without the parameters in MS SQL Server:

```
EXEC [Ten Most Expensive Products]
```

4. Click **Save**.

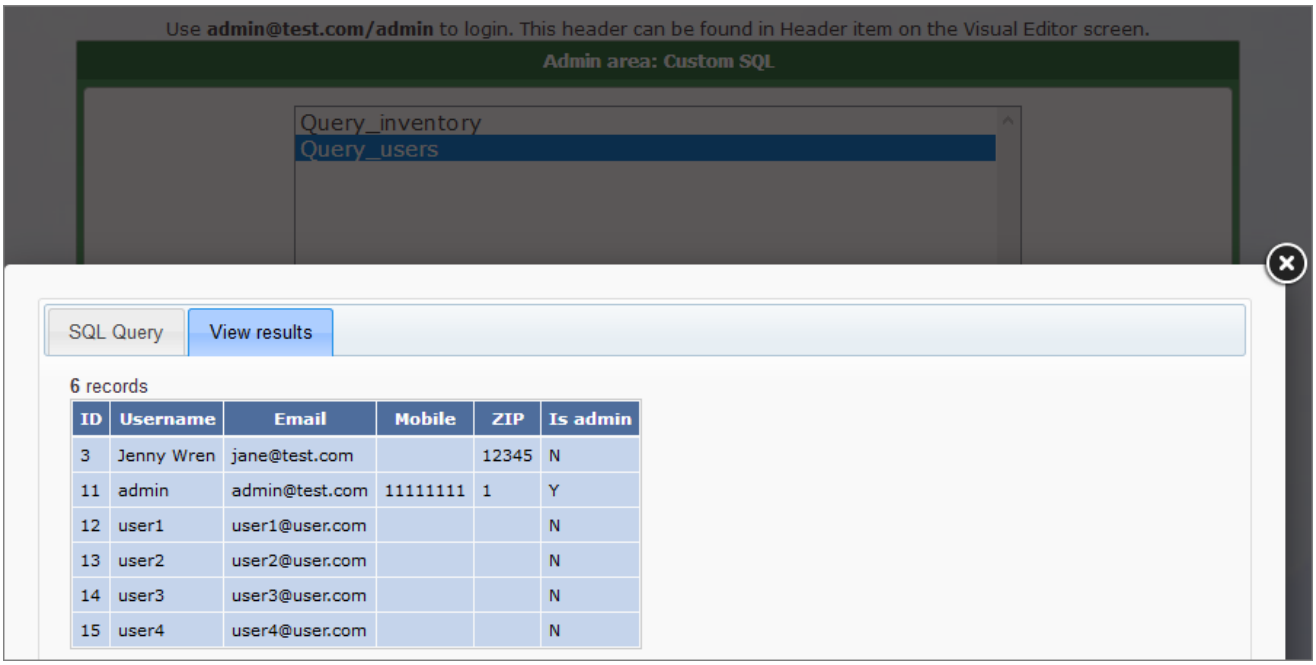
5. Configure the permissions.

If you selected the **Proceed to permissions screen** checkbox, you are redirected to the **Permissions** screen, where you can set up the group permissions for the created query.

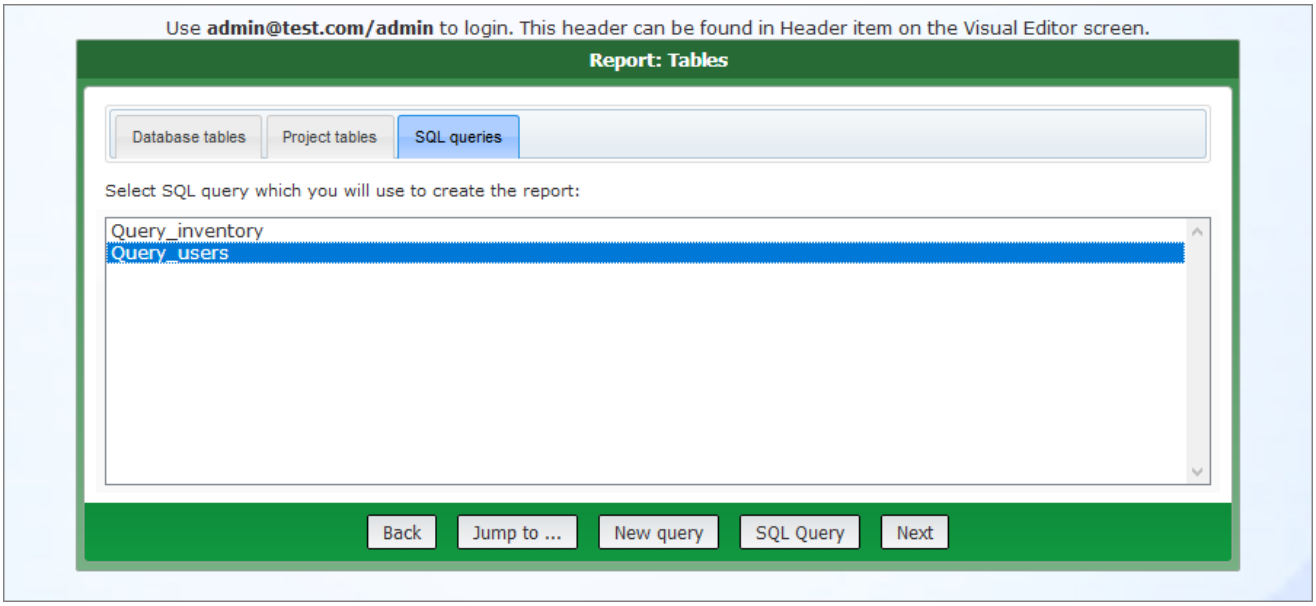


Click **Save** to save the permissions and go back to the **Custom SQL** page.

6. You may now select the created query and click the **View results** button to preview the resulting table.



When creating a report/chart, anyone who has permissions can choose the custom SQL query as a data source. Switch to the **SQL queries** tab on the **Tables** page to do so.



See also:

- Online report/chart builder
- Creating web reports
- Creating web charts
- User group permissions
- Dynamic permission
- Creating charts
- Creating reports
- About SQL query

3.6 How to install a local web server (XAMPP)

PHPRunner comes with a built-in web server. However, you may want to install your web server. This page explains how to install a local web server (XAMPP) and configure PHPRunner to use it.

XAMPP is a free package that includes an easy-to-setup web server (Apache), database server (MySQL), and a server-side scripting language (PHP). More about XAMPP.

This guide uses Windows 10 64-bit and XAMPP v7.3.5.

Download XAMPP

- Visit https://www.apachefriends.org/download.html.
- Download XAMPP Installer for Windows.

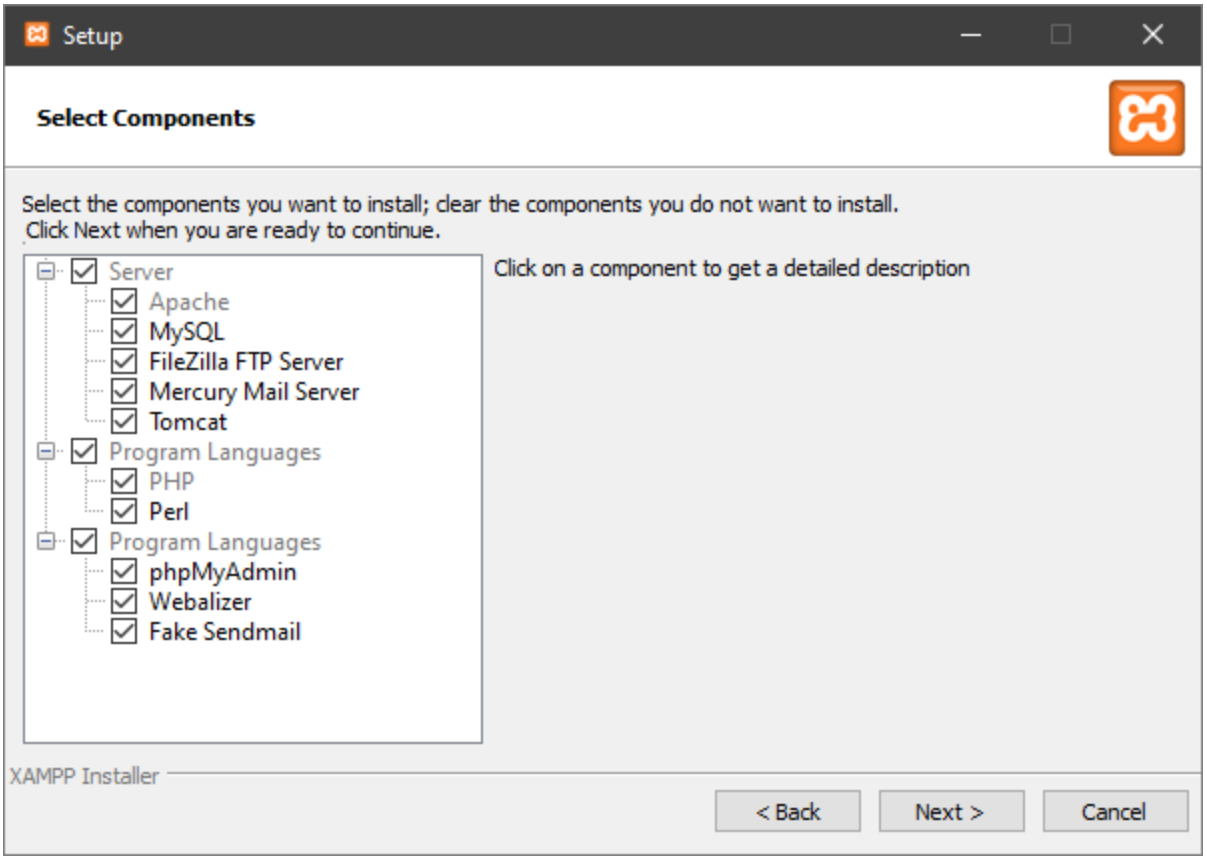
Install XAMPP

> **Note**: if you are running the Skype VOIP application, or if you are running IIS Server, you may need to exit them before proceeding.

1. Run the installer.

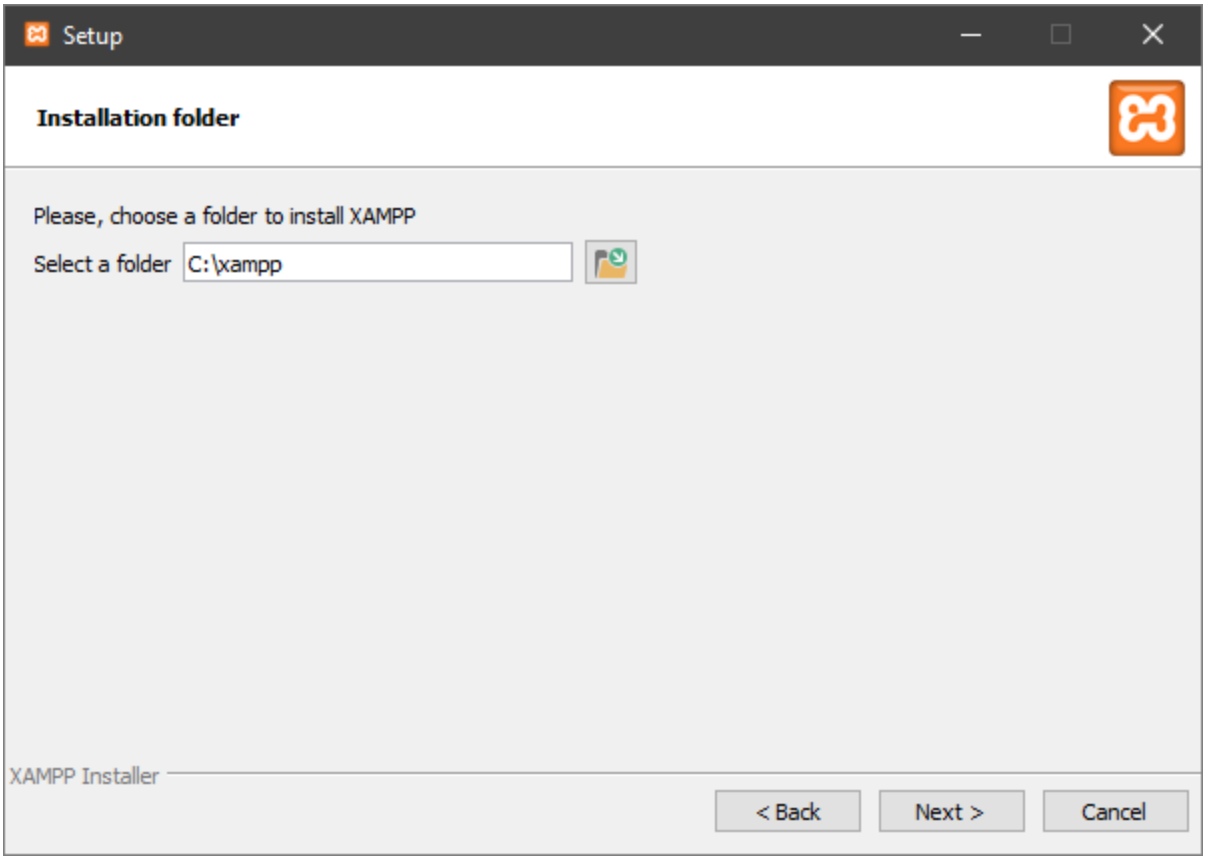
2. You can select to install **Apache** and **MySQL** as services, which makes them start automatically every time you start Windows. That allows other applications you may have, such as an IIS server, to run simultaneously.

If you don't choose this option, you need to use the **XAMPP Control Panel** application to start the servers individually each time you need them. This may be desirable if you don't intend to use your servers that often.



3. Choose a destination to install XAMPP.

Note: if the **User Account Control (UAC)** is turned on in the OS, Apache doesn't recommend installing XAMPP into the *C:\Program Files* folder.



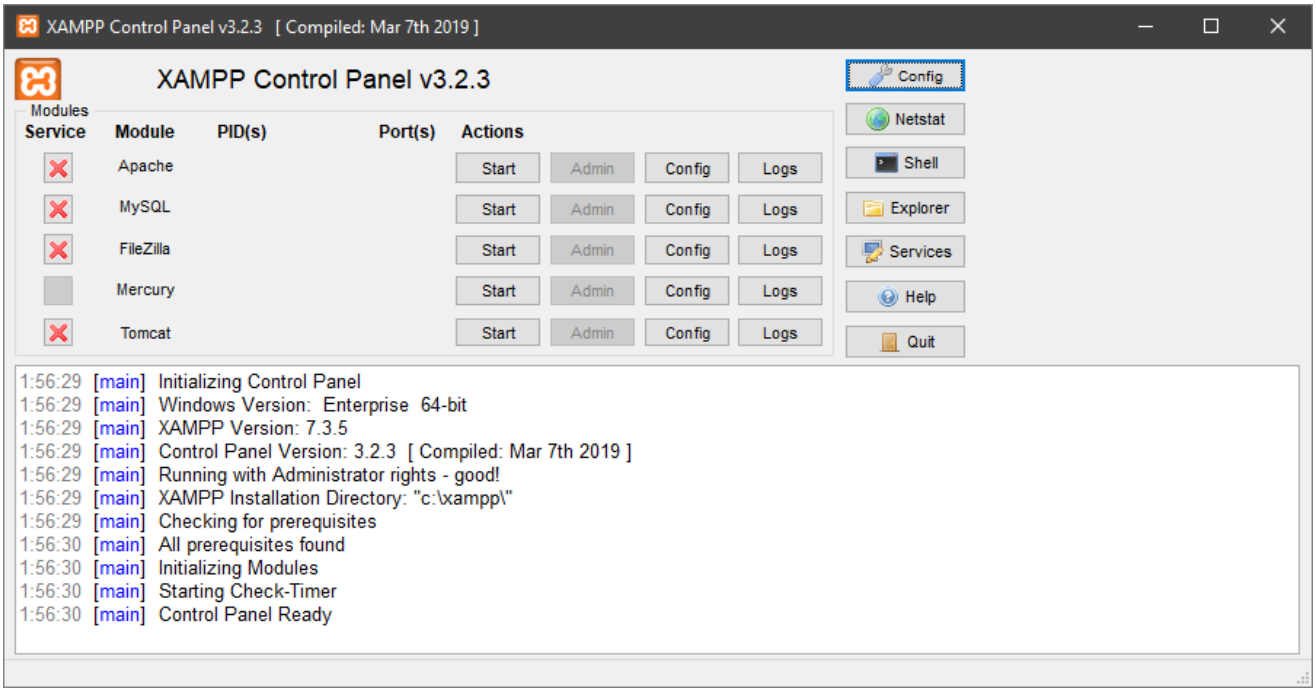
4. XAMPP installation starts. After it is complete, click **Finish**.

5. Next, you are prompted to start the **XAMPP Control Panel**, which can also be opened by selecting **Start** » **All Programs** » **XAMPP** » **XAMPP Control Panel**, or by running **C:\xampp\xampp-control.exe**. This tool lets you start and stop the various servers installed as part of XAMPP. If Apache and MySQL were installed as services, select **No**. Otherwise - run the **XAMPP Control Panel**.

Using XAMPP

The **XAMPP Control Panel** is used to control and monitor the status of services that XAMPP has installed. It is recommended to tun it as **Administrator**.

When the control panel is running, the following icon is visible in your system tray  . Double-clicking this icon brings up the **Control Panel**.



If you see a red **X** under **Service** - that means the module is not installed as a service. In that case, you need to start and stop them manually. Click **Start** to run the modules.

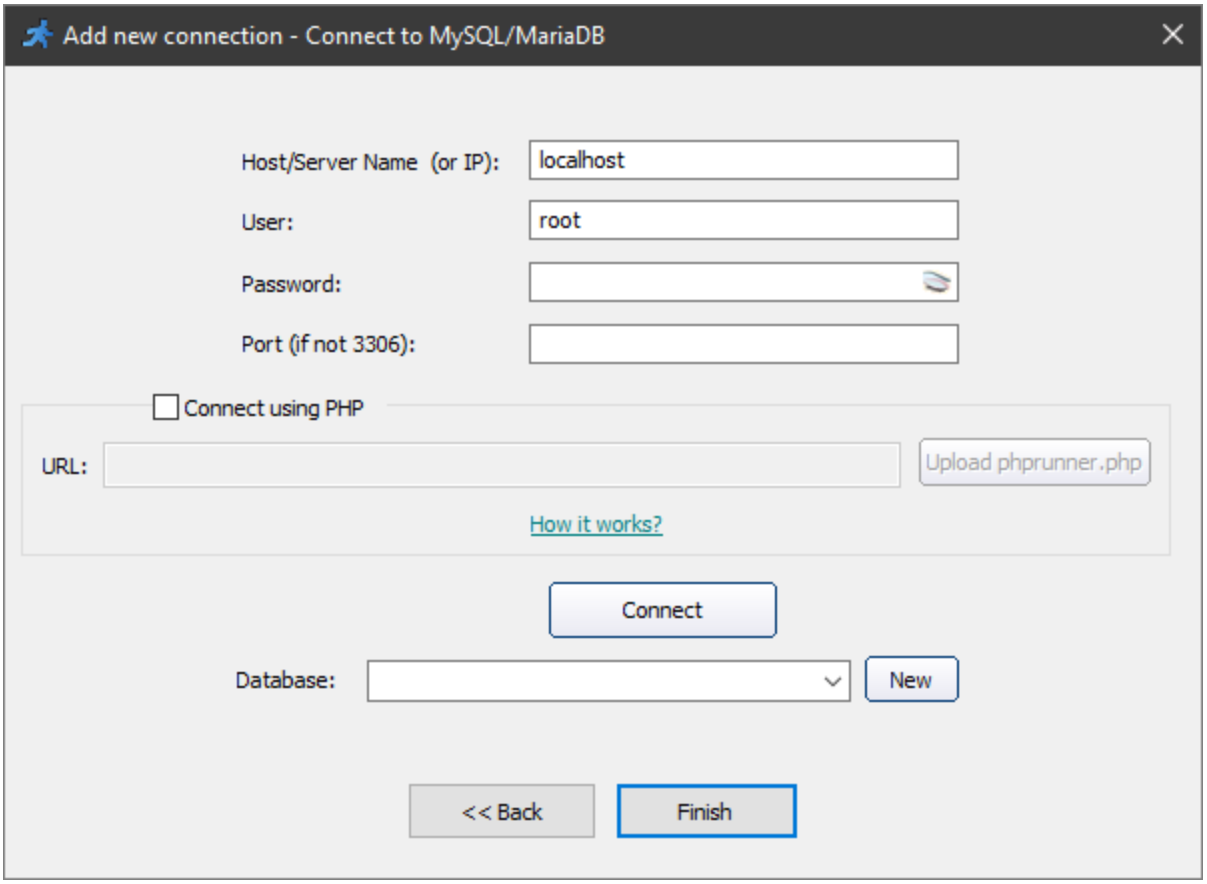
Go to *http://localhost/* in the browser. If you are directed to a page with the XAMPP logo, your installation was successful.

You can add or edit the files in *C:\xampp\htdocs* to change what you see at *http://localhost/*. Right after installation, this folder contains all the files for the XAMPP welcome web page. You can remove or back up these files so that they do not conflict in any way.

The XAMPP configuration interface is available at *http://localhost/xampp/index.php* and is located at *C:\xampp\htdocs\xampp*.

> **Note**: To send emails via PHP, you need to modify the *php.ini* file. It is stored at *C:\xampp\php*. Change the line `sendmail_from = me@example.com` to `sendmail_from = your_email_address`.

When connecting to MySQL, type *localhost* in the *Host/Server name (or IP)* field and *root* in the *User* field. Leave the *Password* field blank.



On the Output directory screen, switch to **I have my own web server** option and enter the URL manually (i.e., *http://localhost/project1*). Change the output directory as well to one of web server subdirectories (i.e., *C:\xampp\htdocs\project1*).

See also:

- Connecting to the database
- Output directory settings
- After you are done
- Quick start guide

3.7 How to add external css/PHP/js files

Follow the instructions below to add your files with JavaScript/CSS/PHP code to the project.

Copy your files to the project folder

Copy your files to the source folder in the project directory (e.g., *..\Documents\PHPRunnerProjects\Project1\source*). Organize the files into subfolders if needed.

For example, you want to add images *right.png* and *wrong.png* to the images folder, a new HTML template *quiz.htm* to the templates folder, and a simple CSS file *quiz.css*. To perform this:

- Create the folder called *images* inside the source folder and copy the images *right.png* and *wrong.png* there.
- Create the folder called *templates* inside the source folder and copy the file *quiz.htm* there.
- Copy the file *quiz.css* directly into the source folder.

When building the project, all files from the source folder are copied to the output folder while keeping the structure of the subfolders.

Link your files to pages

JavaScript

To link a JavaScript file to a page, select the page on the Editor screen and switch to the **HTML mode**. Then add the following code after the **<END body>** tag:

```
<script type="text/javascript" src="include/customfile.js"></script>
```

> **Note**: for **Add**/**Edit**/**View** pages shown in a popup, add the same code snippet to the **List** page.

To link a JavaScript file to all project pages, select the **Header** item on the **Editor** screen and switch to the **HTML mode**. Then add the code as stated above.

CSS

To link a CSS file to a page, switch to the **HTML mode** on the **Editor** screen and add your code just after the following line:

```
<link REL="stylesheet" href="style.css" type="text/css">
```

To link a CSS file to all project pages, select the **Header** item on the **Editor** screen and switch to the **HTML mode**. Then add the code as stated above.

PHP

It is recommended to use events to add your PHP code. Add the following code to the After table initialized event to link a PHP file to a page or After application initialized event to link a PHP file to all pages:

```
include("customfile.php");
```

Accessing PHPRunner session variables from external PHP files

If you need to access PHPRunner session variables from your custom file, add the following as the first line in your PHP file:

```
include("include/dbcommon.php");
```

This code snippet assumes that your external PHP file is located in the main PHPRunner folder.

See also:

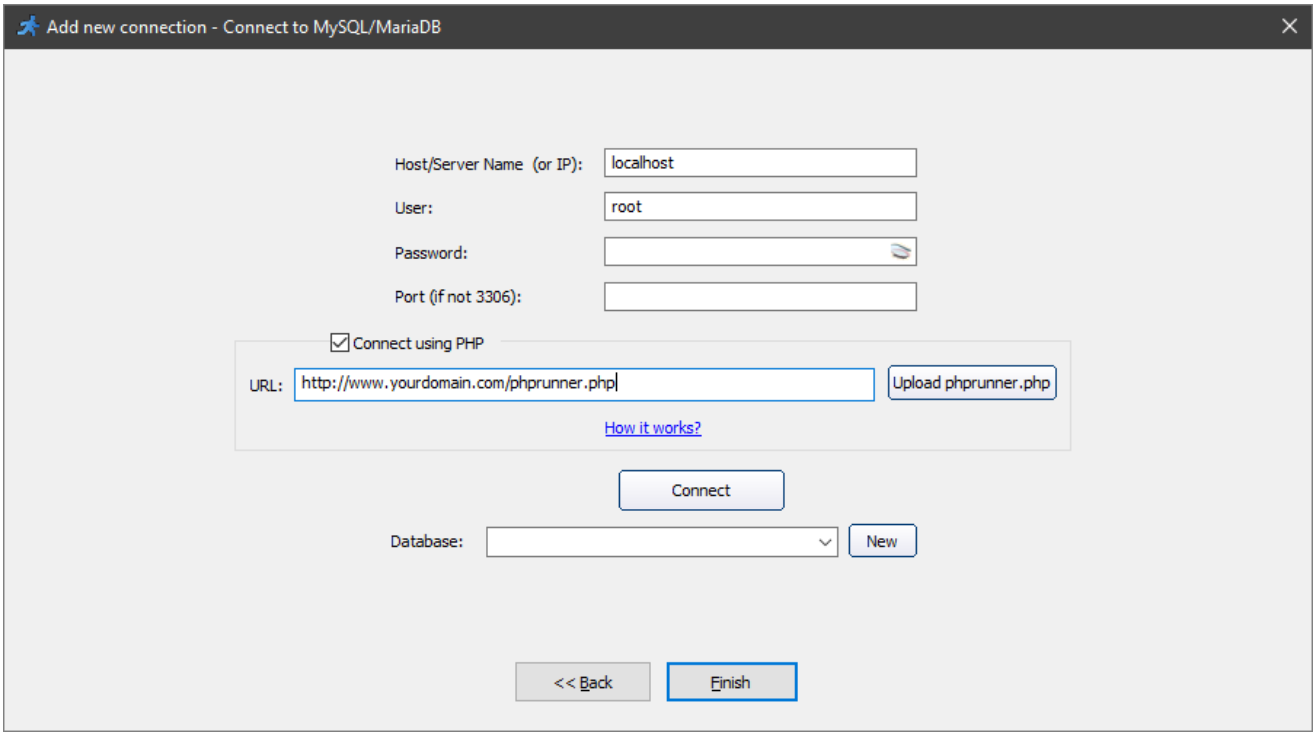
- Editor screen
- Editing the <head> section
- Event editor
- AfterTableInit
- AfterAppInit

3.8 Connecting to a remote MySQL database via PHP

If your MySQL server doesn't allow a direct connection, you can use PHP to connect to it.

Perform the following steps to proceed.

1. On the Connect to MySQL screen enable the **Connect using PHP** checkbox and upload the connection script *phprunner.php* to your site manually or by using the **Upload phprunner.php** button. The *phprunner.php* file can be found in the installation folder, usually *C:\Program Files\PHPRunner 10.3.*

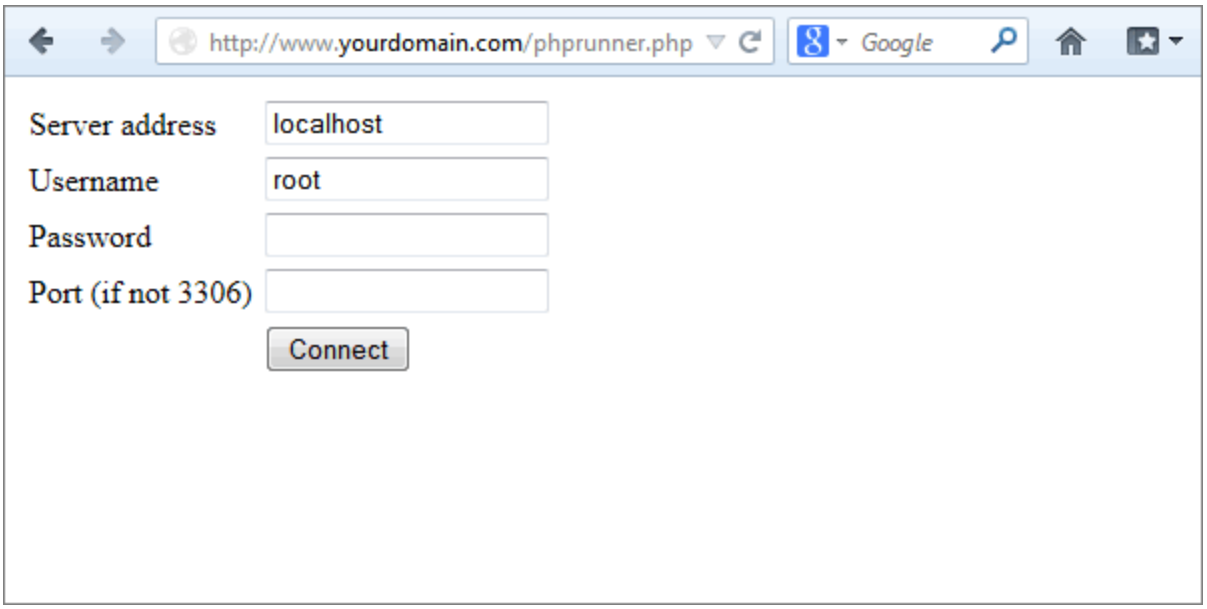


2. Test *phprunner.php* in browser:

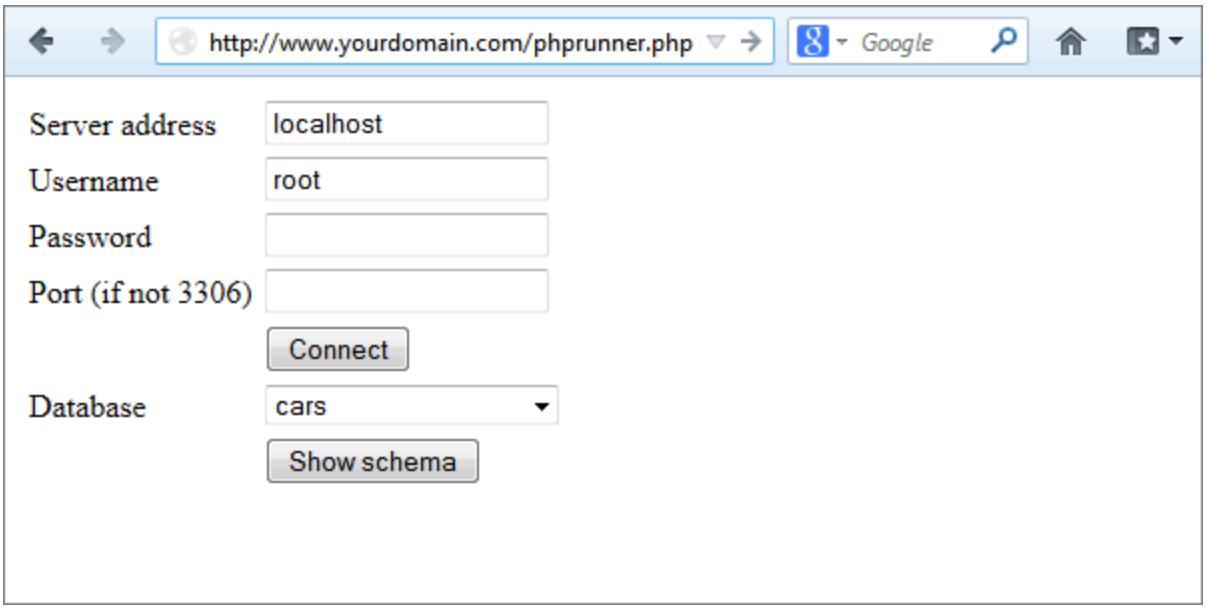
- Open the *phprunner.php* file in the browser. You can access it at *http://www.yourdomain.com/subdirectory/phprunner.php*. Use the full path to the uploaded file in the URL instead of *yourdomain.com/subdirectory/*.

- Type in the *Host/Server Name* (*localhost* if your Web and MySQL server are located at the same host), *user name*, *password*, and click **Connect**.

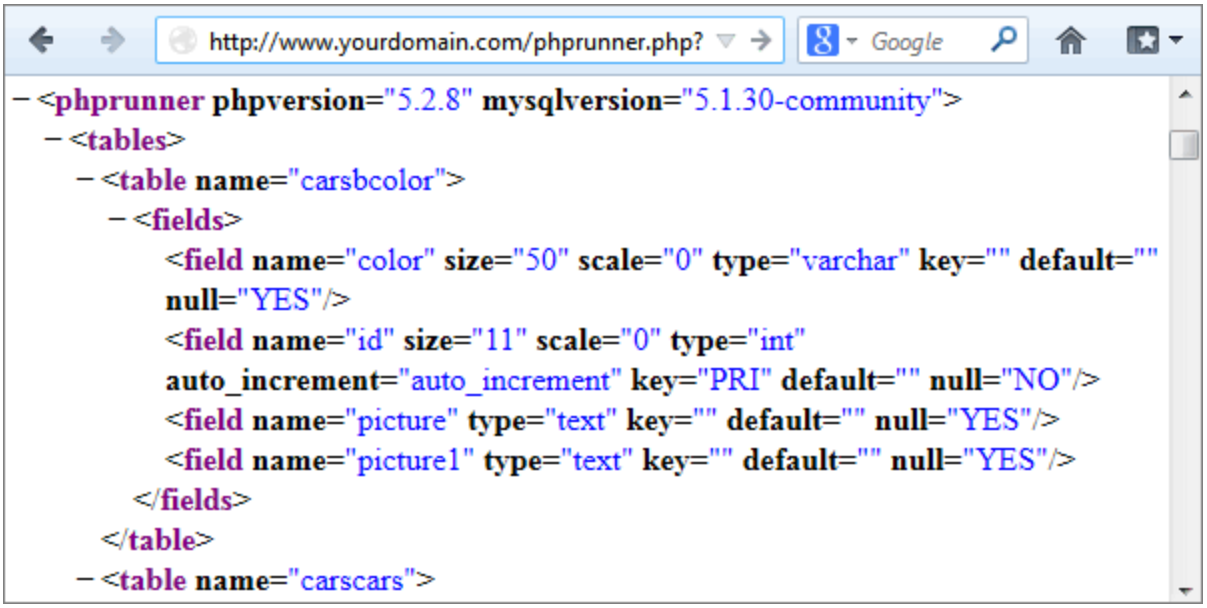
> **Note**: the *username* and *password* are encrypted before sending.



- Select the **Database** you wish to connect to.



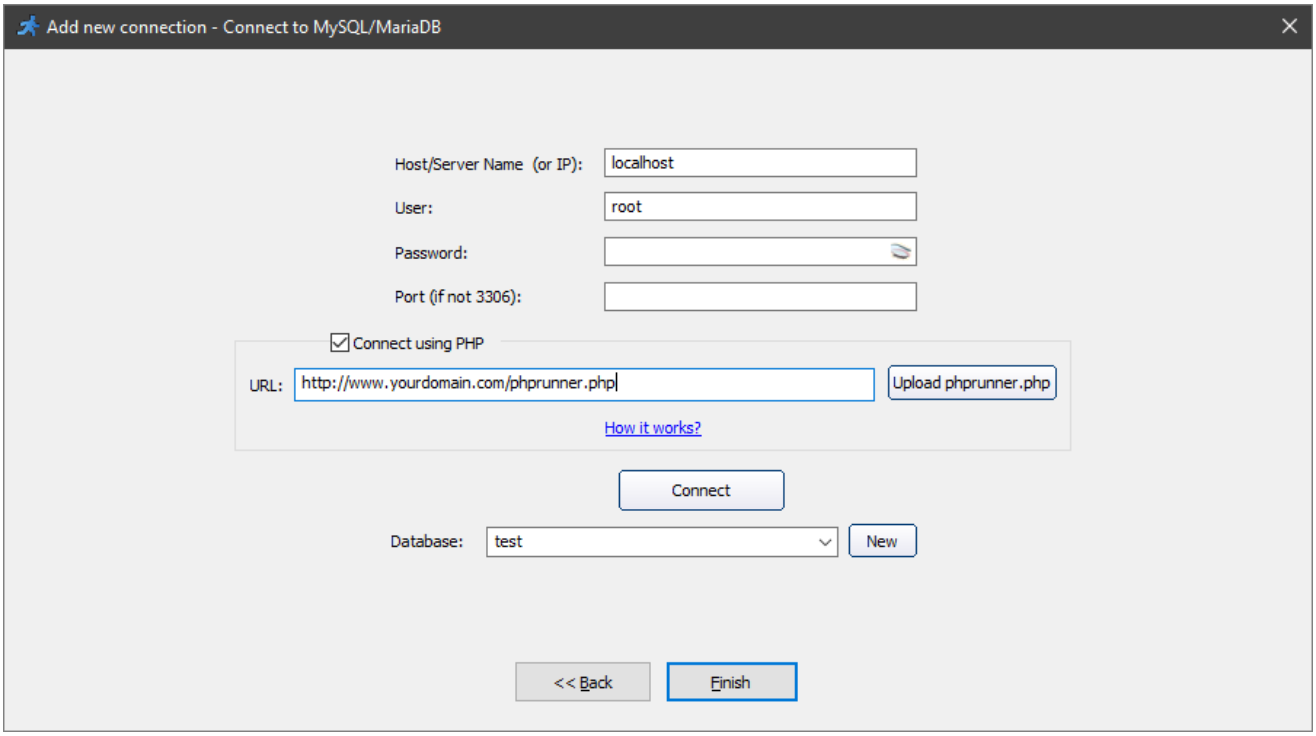
- Click **Show schema** to get the XML representation of the structure of your database.



3. On the **Connect to MySQL** screen, enter the URL of the connection script *phprunner.php*, type in the *Host/Server Name*, *username*, *password*, and click **Connect**.

> **Note**: the connection settings are the same as the ones you used in the previous step while testing *phprunner.php* in the browser.

Then select the database and click **Next >>** (**Finish** if you are adding a new connection).



> **Note**: some MySQL servers don't allow getting a list of databases. In this case, you need to type in the database name manually.

After successfully connecting to the database, you can continue configuring the application.

See also:

- Connecting to the database
- Multiple database connections
- Quick start guide

3.9 Working with MS Access databases

1. If you need to work with Microsoft Access databases, we suggest installing a 32-bit version of PHPRunner.

2. You also need to install the Microsoft Access Database Engine to be able to connect to your database. You need to install it on the local machine and the web server.

Download the 32-bit version (*AccessDatabaseEngine.exe* file) here:

Microsoft Access Database Engine

and install it.

If it doesn't install (i.e., you see a warning related to the 64-bit Office components conflict), try running it from the command prompt with the "/passive" parameter:

```
AccessDatabaseEngine.exe /passive
```

If you are trying to install Microsoft Access Database Engine 2016 or a newer version, use the "/quiet" parameter:

```
AccessDatabaseEngine.exe /quiet
```

3. On the web server, make sure you have provided full permissions to web server users for the folder where the database file resides.

4. Make sure that your php.ini file contains the following line:

```
extension=php_com_dotnet.dll
```

See also:

- Connecting to the database
- Open Database Connectivity (ODBC)

3.10 Working with Oracle databases

If you need to work with Oracle databases, you need to download and install the **Oracle Instant Client** from this page:

https://www.oracle.com/database/technologies/instant-client/downloads.html

You need to choose a 32 or 64-bit client version depending on PHPRunner version installed on your computer.

If you are using MS Internet Information Services and the PHPRunner wizard has successfully connected to the database, but when you run the generated app, it fails to connect to Oracle, the possible reason might be that an IIS user doesn't have access to the *tnsnames.ora* file.

The suggestion is to use the server:port/instance format when connecting to Oracle. Here is a sample *tnsnames.ora* file:

```
AADEV =
(DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = TestOracle.server.gov)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SID = AAINST)
      (SERVER = DEDICATED)
    )
  )
extension=php_com_dotnet.dll
```

Instead of using `AADEV` in PHPRunner, use `TestOracle.server.gov:1521/AAINST`

See also:

- Connecting to the database
- Quick start guide

3.11 Open Database Connectivity (ODBC)

ODBC

Open DataBase Connectivity (ODBC) is a system provided in Windows to allow exchanging data between different database applications.

Go to **Control Panel** -> **Administrative tools** to find ODBC Data Sources (32&64-bit) applications.

PHPRunner allows you to extract data from an ODBC Data Source Name (DSN). A DSN is a convenient way of storing all of the datasource connection information (file name, path, user I.D, connect password, database type, etc.) under a simple name.

If you want to use PHPRunner with databases like MySQL, Informix, or MS SQL server, you need to install and configure appropriate ODBC drivers for your database. You can find links to the appropriate ODBC drivers below.

Visit the Microsoft documentation site to learn more about ODBC:

https://docs.microsoft.com/en-us/sql/odbc/microsoft-open-database-connectivity-odbc

Latest ODBC Drivers

Microsoft databases

https://docs.microsoft.com/en-us/sql/connect/odbc/download-odbc-driver-for-sql-server.

Oracle

https://www.oracle.com/database/technologies/instant-client/downloads.html

MySQL

https://dev.mysql.com/downloads/connector/odbc/

PostgreSQL

https://odbc.postgresql.org/

Informix

https://www.ciscounitytools.com/Applications/CxN/InformixODBC/InformixODBC.html

DB2

https://www-01.ibm.com/support/docview.wss?uid=swg27016878

SQLite

https://www.devart.com/odbc/sqlite/download.html

See also:

- Connecting to the database
- Quick start guide

3.12 AJAX-based functionality

Quick jump

PHPRunner comes with a built-in AJAX-based functionality:

- You can easily search for information with a Google-like auto-suggest feature.
- If you want to see the details records, you can mouse over the link, so you don't need to proceed to the **Details** page.

- You can choose an option from a lookup-enabled field. The options are refreshed each time you type something into the text box. Web pages with AJAX-driven dependent dropdown boxes are loaded faster.

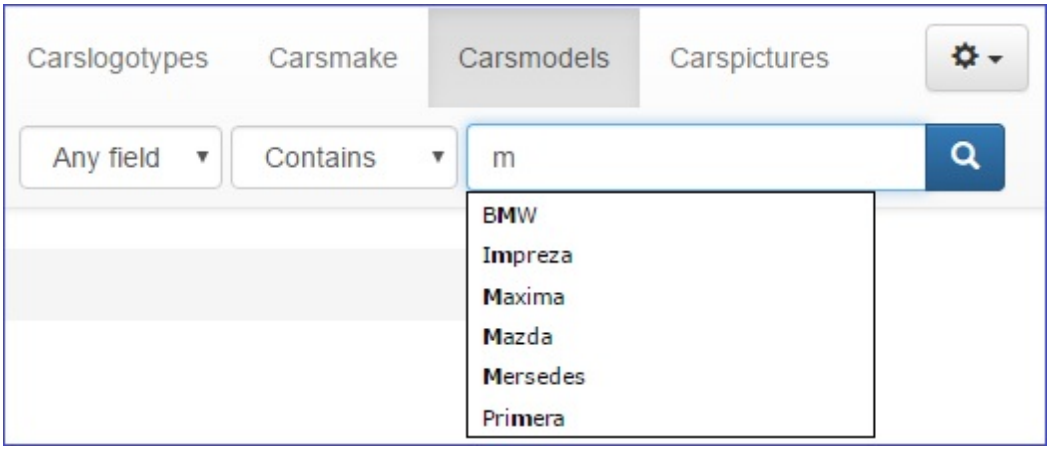
> **Note**: if you don't want to use the AJAX-based functionality, change the `$useAJAX` and `$suggestAllContent` variables value in the *include\appsettings.php* file to *false*.

```
$useAJAX = false;
$suggestAllContent = false;
```

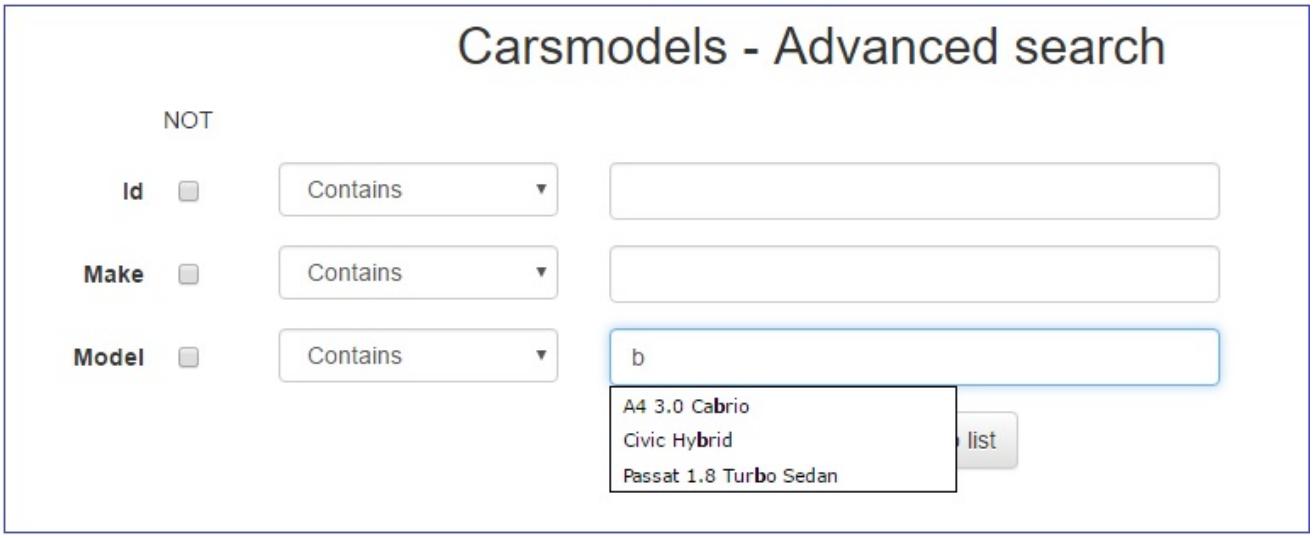
AJAX-based Auto-Suggest

This feature is similar to Google search suggestions; it makes your generated app more user-friendly.

To see how it looks like, begin typing in the search box. The picture below demonstrates the search suggest feature on the **Basic search** page:



On the **Advanced search** page, the search suggestion looks like this:



The **Auto-Suggest feature** comes in two versions. By default, the search suggestion results include all values which contain the search phrase. If you want the suggestion to show only the values beginning with the search phrase, you should add the following code to the AfterAppInit event:

```
$suggestAllContent = false;
```

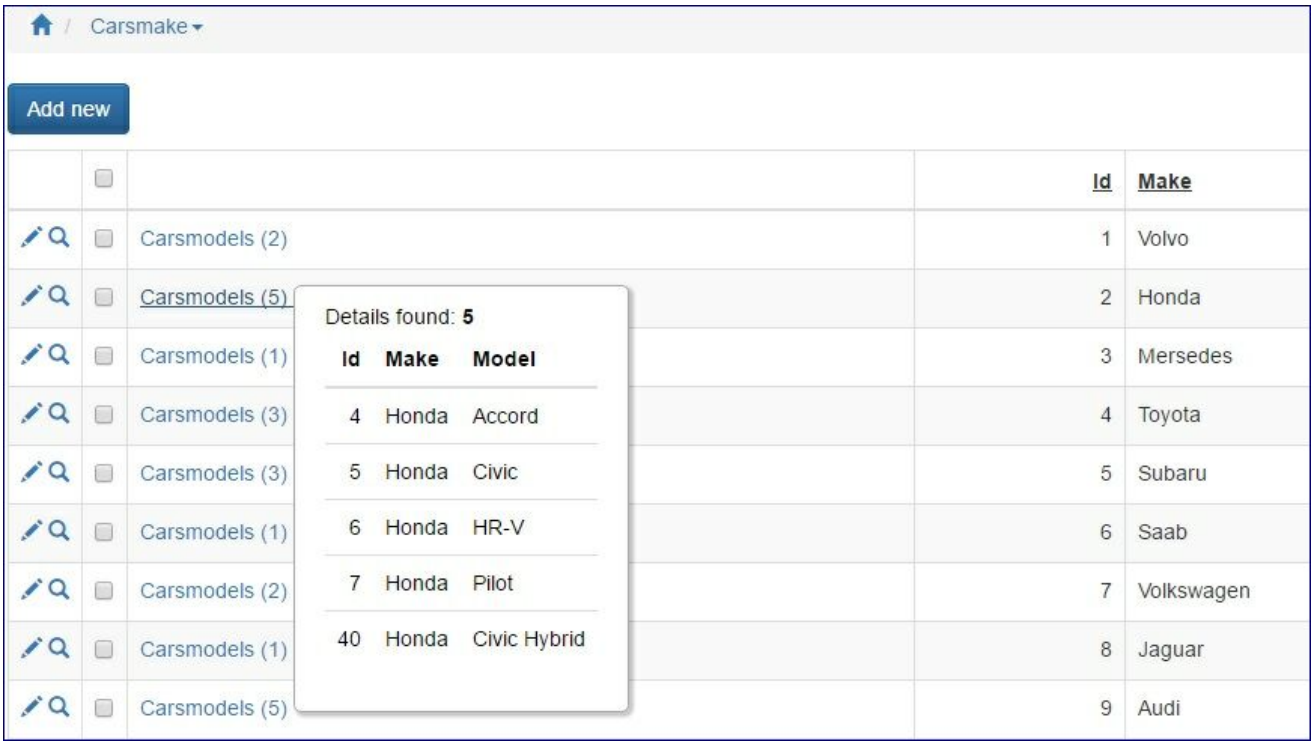
Note: if a field is hidden/not shown on some page (**List**/**View**/**Add**/**Edit**/**Export**), then the field values do not appear in the search suggestion results. This is done to secure confidential data like passwords and credit card numbers.

AJAX-based dependent dropdown boxes

In PHPRunner, dependent dropdown boxes are AJAX driven. The content of the dropdown boxes is loaded in real-time using AJAX technologies instead of loading all the content upon the initial opening of the web page. This means web pages are loaded faster.

Details Records Preview

In PHPRunner, you can see the details records preview directly on the **List** page. All you need to do is to mouse over the link. The following picture demonstrates how the details preview looks like:



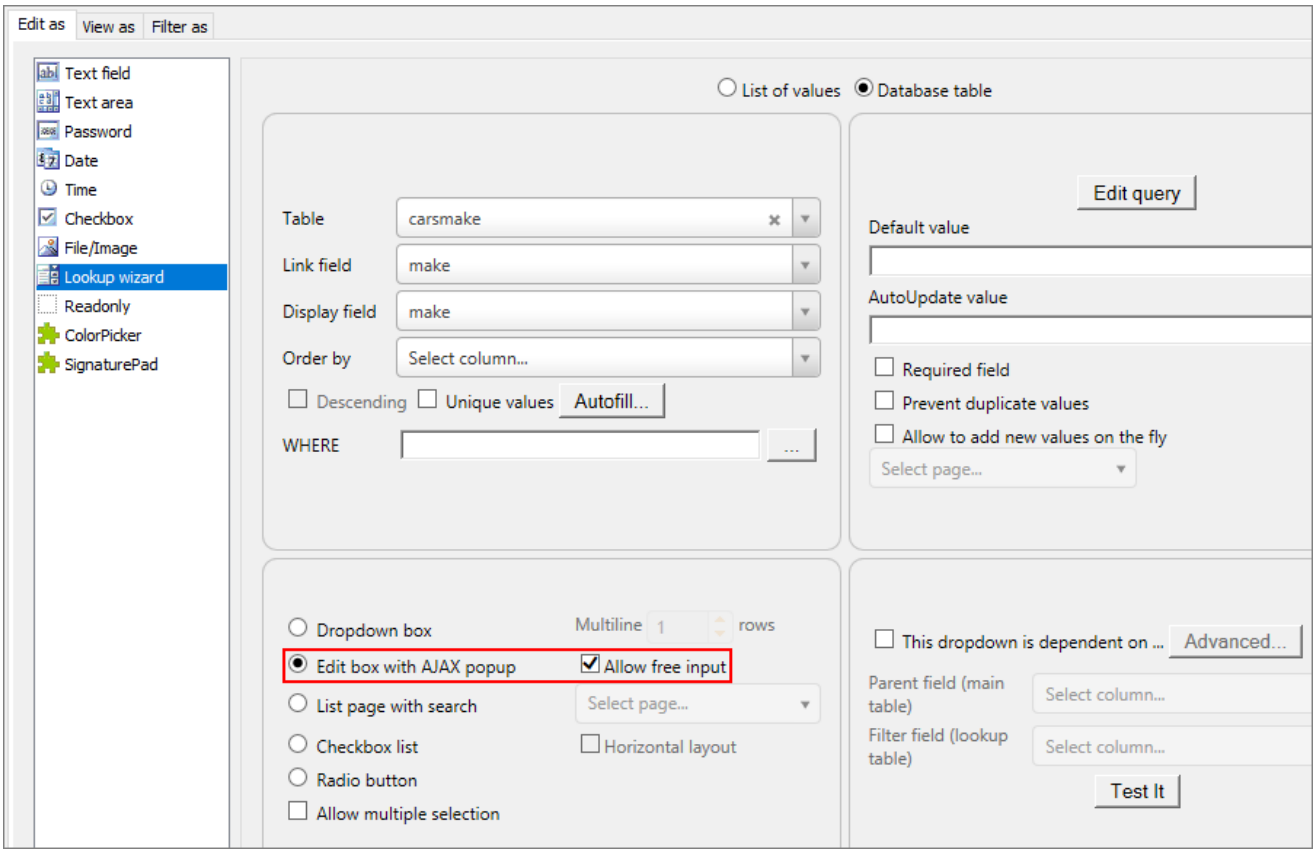
The number of records in the details preview is limited. Only the first ten details are displayed. If there is an image in the details, then only the first five details records are displayed.

For more information about how to enable the details records preview, see Master-details relationship.

Lookup wizard as an Edit box with AJAX popup

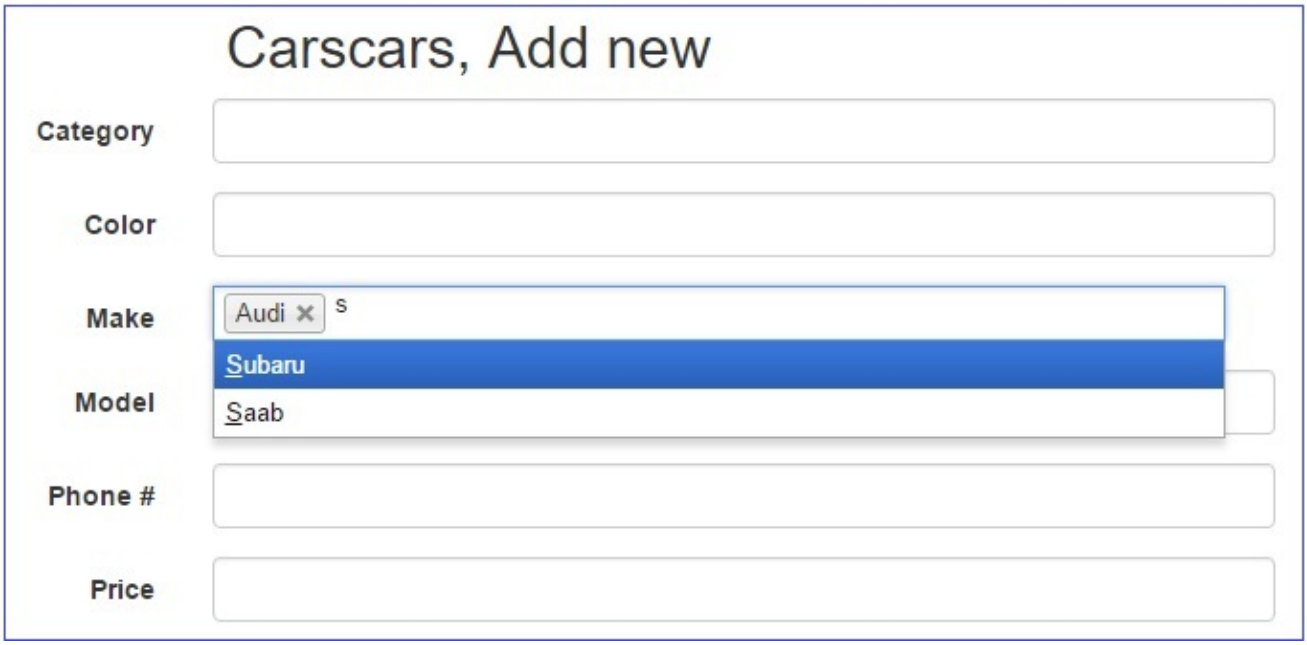
This feature is added to facilitate searching through a large amount of data. You can choose a value from the AJAX popup which is refreshed each time you type in the text box, instead of searching through all the values in the dropdown boxes.

To turn on this feature, you should select the **Edit box with AJAX popup** checkbox on the Edit as: Lookup wizard settings tab of the selected field.



After you build your project, you can see how it works on the **Edit**, **Add**, and **Search** pages. When you begin to type in the **Edit** box, an AJAX popup appears, and you can choose the value from the suggest list.

> **Note**: if you type in the value that doesn't exist in the database and click away, the border of the text box changes color to red. When you correct the value, the border changes back.



The application looks for the occurrence of the typed in string anywhere in the list. For example, when you enter *co*, it can show both *Corolla* and *Accord*. If you want to change this behavior and make it look for the values starting with the entered value, i.e., *Corolla* only, add the following code to the AfterAppInit event:

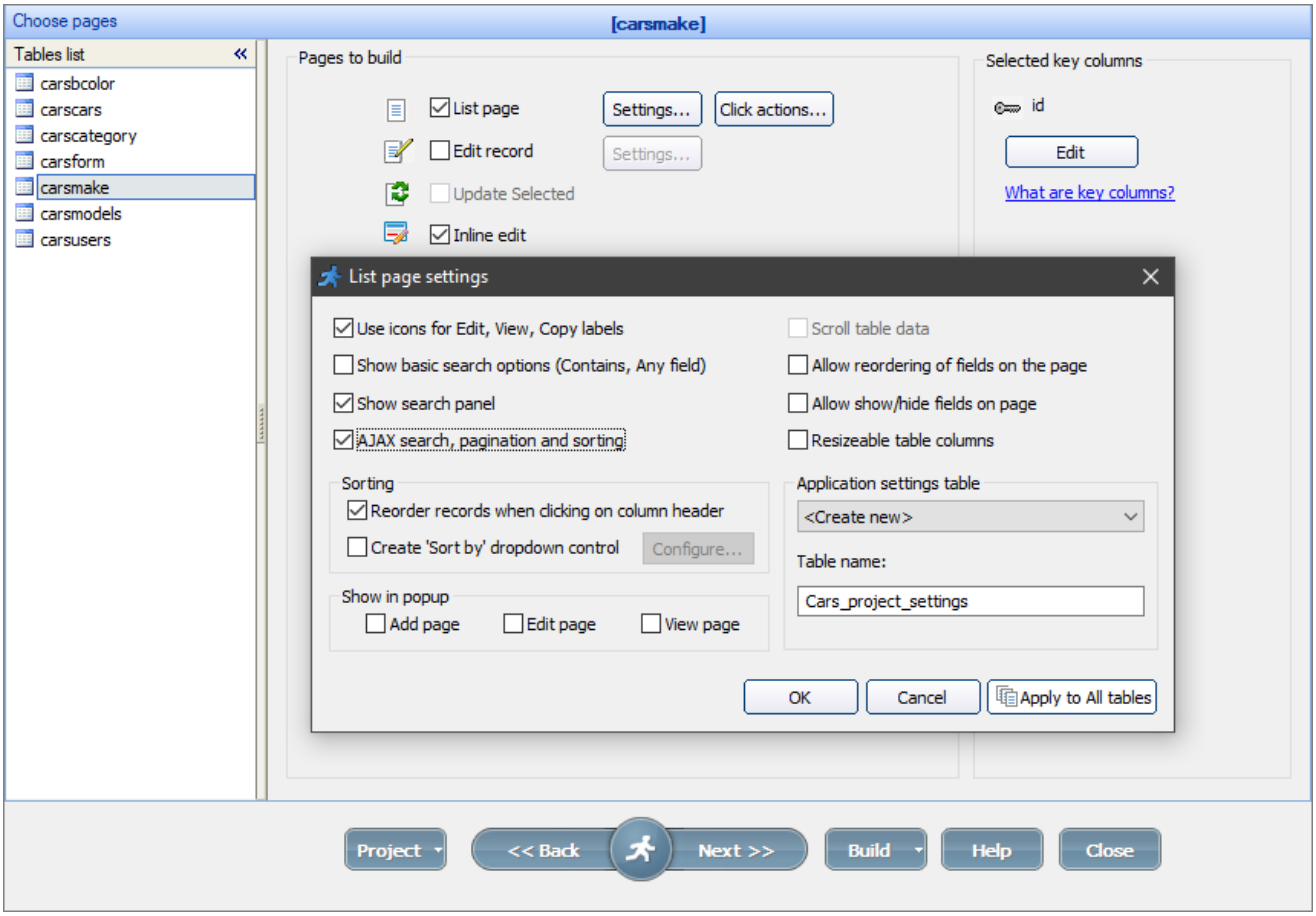
```
$ajaxSearchStartsWith = true;
```

If you try to submit the form with a wrong value entered in the text box, the form is instead submitted with the previous correct value entry.

AJAX-based pagination/sorting/search

This option enables AJAX search, pagination and sorting that allows updating data without reloading the entire page.

To turn this feature on, proceed to the Choose pages screen and click the **Settings** button next to the **List page** checkbox. Then select the **AJAX search, pagination and sorting** checkbox.



See also:

- Master-details relationship
- Lookup wizard
- "Edit as" settings
- AfterAppInit
- Event editor
- Choose pages screen
- List page settings

3.13 Localizing PHPRunner applications

Quick jump

This article covers the aspects of creating multilingual websites with the help of PHPRunner. This process includes the following steps:

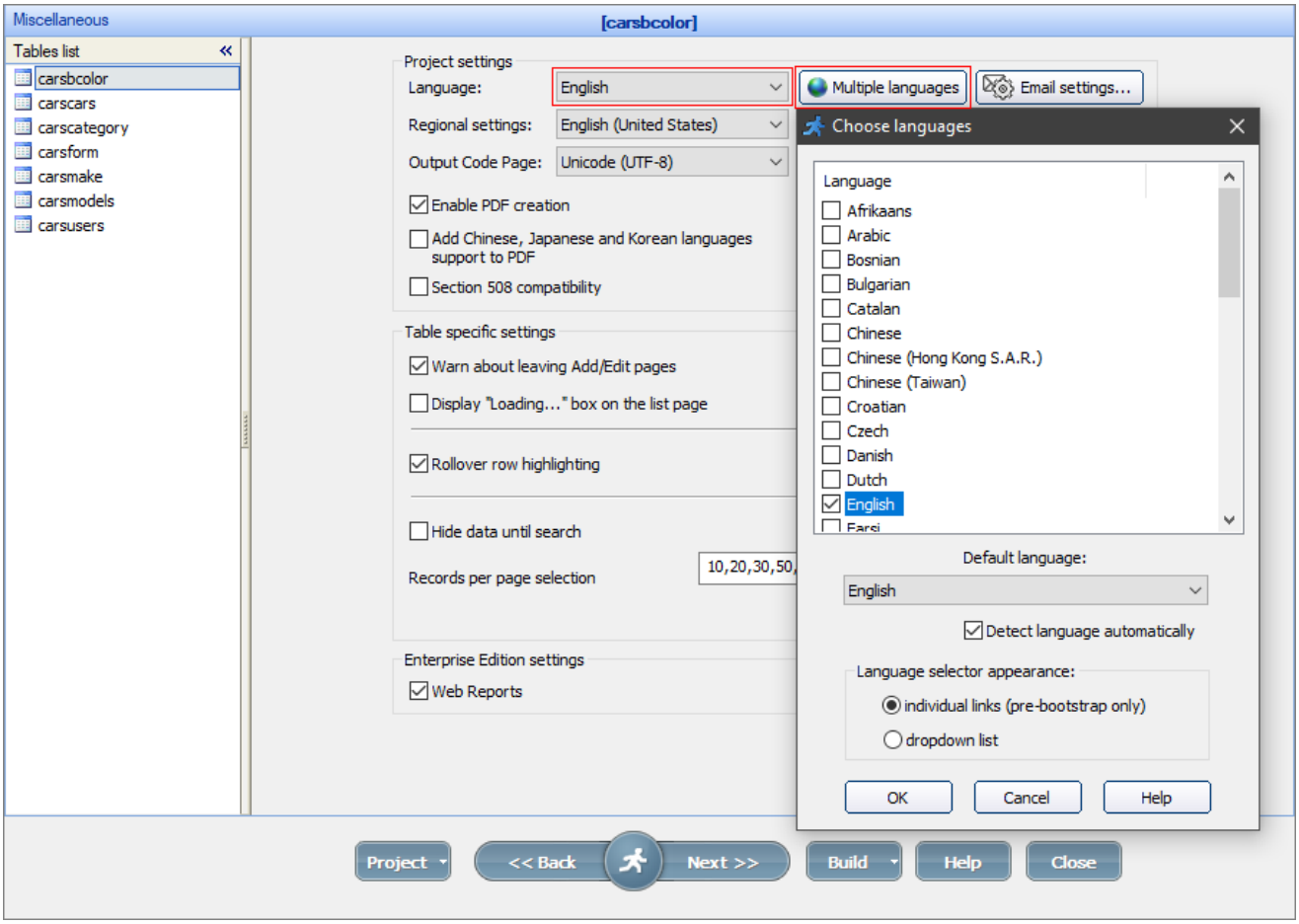
- Translation of system messages
- Translation of table/field names and custom labels
- Translation of data from the database

Translation of system messages

First of all, you need to define the language(s) of standard texts in the website interface - "system messages".

On the Miscellaneous screen, you can choose one or more supported languages for your website.

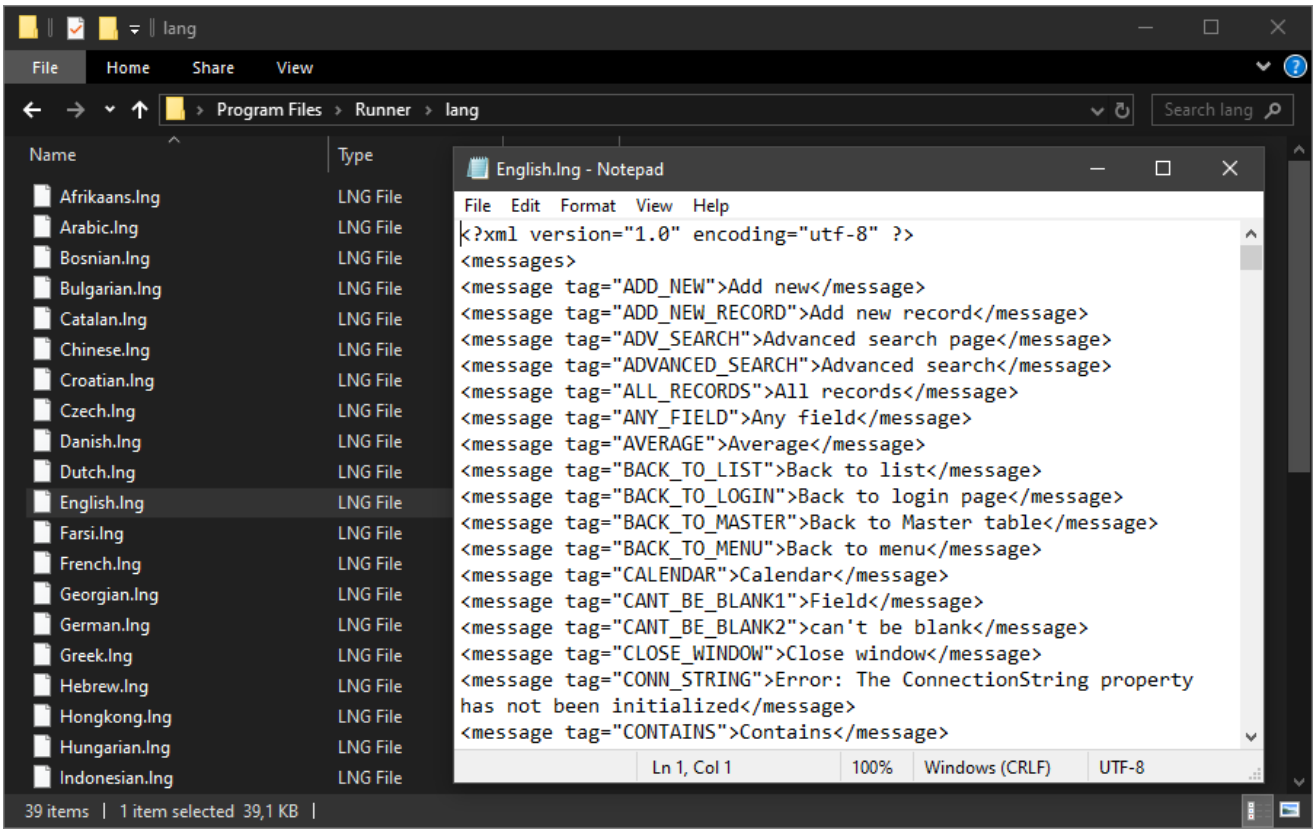
Use the **Language** dropdown box if you wish to choose one language. By clicking the **Multiple languages** button, you can select several languages and give the user the ability to choose the language while logging in.



PHPRunner includes translations of system messages into the following languages:

Afrikaans	Arabic	Bosnian	Bulgarian	Catalan	Chinese
Chinese (Hong Kong S.A.R.)	Chinese (Taiwan)	Croatian	Czech	Danish	Dutch
English	Farsi	French	Georgian	German	Greek
Hebrew	Hungarian	Indonesian	Italian	Japanese	Malay
Norwegian (Bokmal)	Polish	Portuguese (Brazil)	Portuguese (Standart)	Romanian	Russian
Slovak	Spanish	Swedish	Tagalog (Phillippines)	Thai	Turkish
Urdu	Welsh				

Translations of system messages are stored in the language files (*.lng) located in the lang directory (*C:\Program Files\PHPRunner 10.3\lang*). For example, the system messages for the English language are stored in the *English.lng* file:



To change the translation of system messages, modify the corresponding language file (*.lng).

To add a translation of system messages in a new language, create a copy of *English.lng* file and translate all phrases there. Then modify the *languages.cfg* file that is also located in the *lang* directory by adding this line (change red values to match your needs):

```
<language filename="YourLanguageFile.lng" name="YourLanguageName" lcid="YourLocaleID"
codepage="YourCodepage" charset="YourCharset" />
```

In this line:

`filename`

the name of the new .lng file.

`name`

the language name as it would appear in the PHPRunner wizard.

`lcid`

a locale ID. The list of Locale IDs can be found at http://xlinesoft.com/articles/asp_regional_settings.htm.

`codepage`

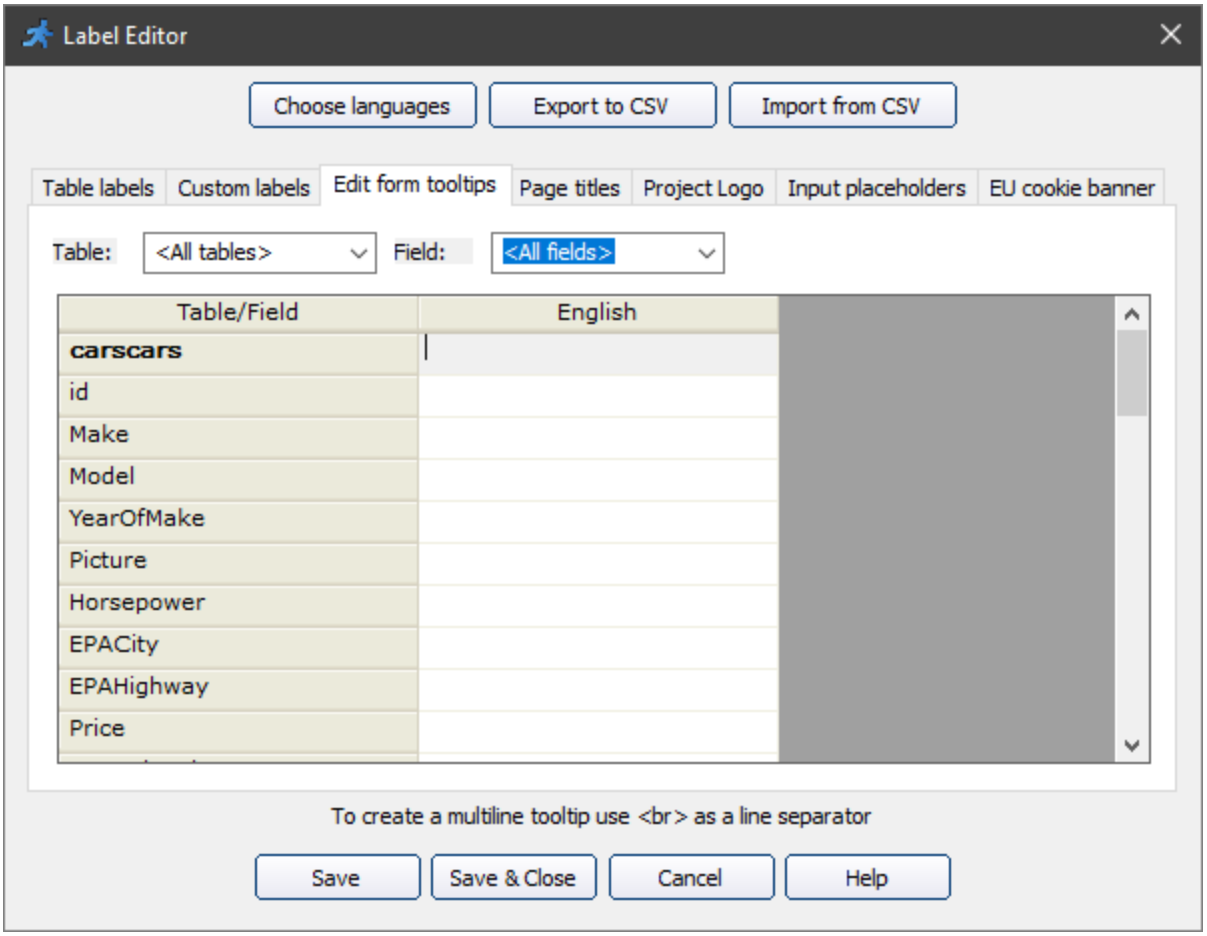
a codepage code. The list of codepage codes can be found at http://en.wikipedia.org/wiki/Code_page.

`charset`

a charset code. The list of charset codes can be found at http://www.webcheatsheet.com/html/character_sets_list.php.

Translation of table/field names and custom labels

Use the Label Editor on the Miscellaneous screen to translate table and field names/labels (**Table labels** tab). You can also add and translate custom labels there (**Custom labels** tab), and add tooltips for the **Edit** forms (**Edit form tooltips** tab).



Use **Custom labels** to translate menu items, tab/section names, error messages in regular expressions, and your custom validation plugins. When creating a menu item, new tab/section or regular expression, use the **Multilanguage** button to create and translate the custom label.

With **Label Editor**, you can also create custom labels to display messages to users.

Use the methods listed below to access labels with the PHP code:

Method	Description
GetTableCaption($table)	Returns the table caption.
GetFieldLabel($table, $field)	Returns the field label.

GetCustomLabel("LABEL_ID")	Returns the custom label. LABEL_ID - the custom label identifier.
GetFieldToolTip($table, $field)	Returns the field edit tooltip.
mlang_message($tag)	Returns the standard message. You can find the list of message tags in the *English.lng* file (*C:\Program Files\PHPRunner 10.3\lang\English.lng*).

Use the methods listed below to access the labels with the JavaScript code:

Method	Description
Runner.getCustomLabel("LABEL_ID")	Returns custom label. LABEL_ID - the custom label identifier.
Runner.lang.constants.CONSTANT_ID	Returns standard message. CONSTANT_ID - the message constant identifier. You can find the list of constants in the *RunnerLang.js* file (*C:\Program Files\PHPRunner10.3\source\include\common\runnerJS\constants\language\RunnerLang.js*).

Example 1

Use a custom label with a PHP code.

If you have a custom multilanguage label *Message1* and want to place this label on a page, add a PHP code snippet to the page and use the following code in it:

```
echo GetCustomLabel ("Message1");
```

Example 2

Use a custom label with a JavaScript code.

If you have a custom label *Error_Message_1* for your validation plugin, use the following code in your JavaScript function:

```
GetCustomLabel("Error_Message_1");
```

Note: the **GetCustomLabel** function is applicable only for editing fields on the **Add**/**Edit**/**Register**/**List** pages with an **Inline Add**/**Edit**.

Example 3

Use the field label instead of the field name.

In this PHP code snippet that sends an email with new data, the **GetFieldLabel** method is used to get the field label.

```
//**********  Send email with new data  ************
// do not forget to setup email parameters like From, SMTP server, etc.
// using the 'Miscellaneous settings -> Email settings' dialog
$email="test@test.com";
$from="admin@test.com";
$msg="";
$subject="New data record";

foreach($values as $field=>$value)
{
  if(!IsBinaryType(GetFieldType($field)))
    $msg.= GetFieldLabel("table_name", $field) ." : ".$value."\r\n";
}

$ret=runner_mail(array('to' => $email, 'subject' => $subject, 'body' => $msg,
'from'=>$from));
if(!$ret["mailed"])
  echo $ret["message"];
```

Note: don't forget to replace *table_name* with the correct name of the table.

Translating data in the database

There are two approaches, in general, to translating the data in the database:

Store data for each language in a separate table and redirect users to the corresponding page, depending on the selected language

Example. To implement this approach, use the following code in the BeforeProcessList event of the *mydata_english* table:

```
if ($_SESSION["language"]=="Spanish")
  {
    header("Location: mydata_spanish_list.php");
    exit();
  }
  else if ($_SESSION["language"]=="French")
    {
      header("Location: mydata_french_list.php");
      exit();
    }
```

Store data for all languages in one table by adding a new field to indicate the language.

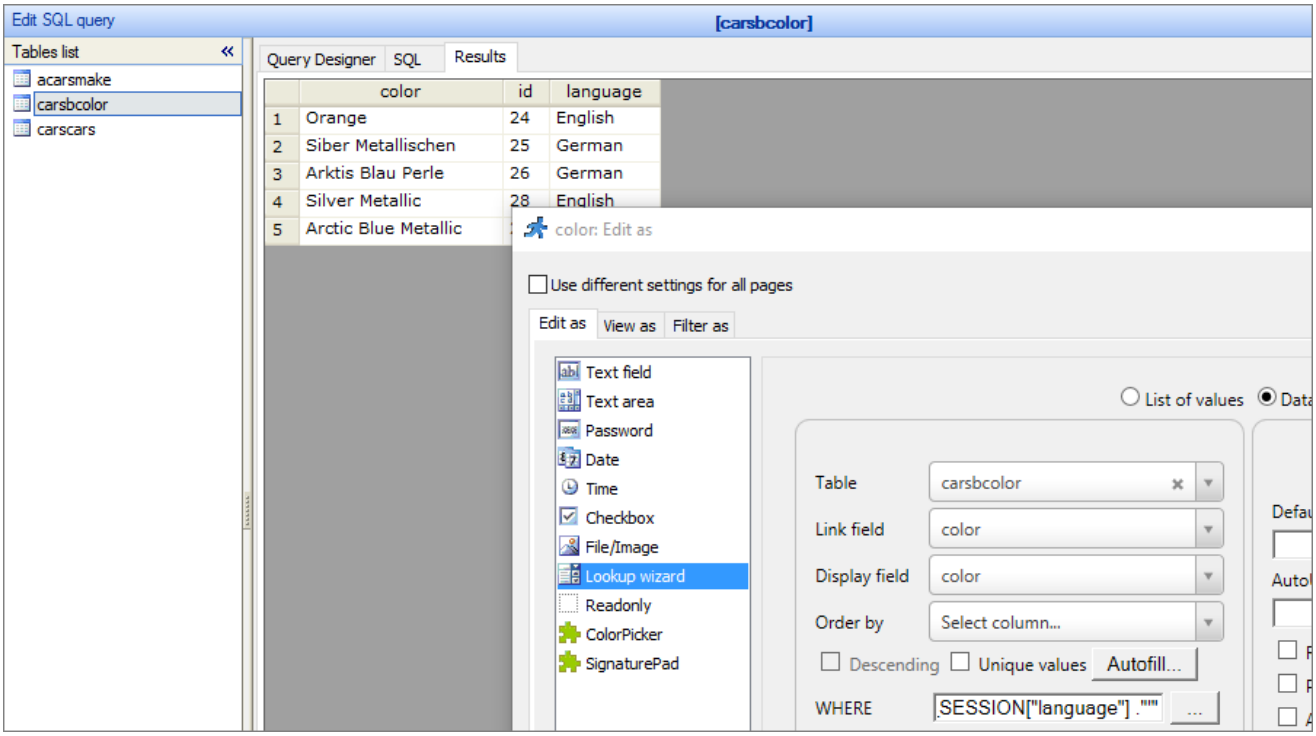
> **Note**: this approach is easier to configure.

To implement this approach, you can use the method described in the Dynamic SQL query topic and the following code:

```
$strWhereClause = whereAdd($strWhereClause, "language='".
$_SESSION["language"] ."'");
```

If you have a lookup table where the data is stored in multiple languages, use the following code in the WHERE clause in the Lookup wizard to display the data only for the currently selected language:

```
"language='". $_SESSION["language"] ."'"
```



See also:

- Miscellaneous settings
- How to create a custom Edit control plugin
- Insert code snippet
- Lookup wizard
- Dynamic SQL query
- Event editor

3.14 Rich Text Editor plugins

To choose a **Rich Text Editor** type, go to the Page Designer screen, set the Edit as type of any text field to the Text area, and select the **Use Rich Text Editor** checkbox.

PHPRunner supports the following third-party Rich Text Editors:

- **Basic Rich Text Editor**: https://kevinroth.com/rte/demo.htm
- **CKEditor**: https://ckeditor.com/
- **InnovaStudio Editor**: http://www.innovastudio.com/

For more info on each editor, documentation, and examples, visit the vendors' websites.

Configuration

Basic Rich Text Editor

This editor comes built-in into PHPRunner. You don't need to download or configure anything - select it from the list of available Rich Text Editors.

This editor is lightweight, and the footprint is minimal. Documentation, support forum, and examples are available at http://kevinroth.com/rte/

InnovaStudio Editor

To use this Rich Text Editor:

- Download **InnovaStudio Editor** from our website at http://www.asprunner.com/files/innovaeditor.zip.
- Create a folder named *innovaeditor* under *C:\Program files\PHPRunner 10.3\source\plugins*.
- Unzip **InnovaStudio Editor** files to the folder *C:\Program files\PHPRunner 10.3\source\plugins\innovaeditor*.

After that, **InnovaStudio Editor** becomes available for selection in the **Text area** dialog.

InnovaStudio Editor adds about 1000 files to the generated application; the footprint is about 3.5Mb. **InnovaStudio Editor** documentation and examples can be found in the documentation folder.

CKEditor

Proceed to http://ckeditor.com and download the latest version of **CKEditor**. Unzip it into the *C:\Program Files\PHPRunner 10.3\source\plugins* folder. After that, **CKEditor** becomes available for selection in the **Text area** dialog.

CKEditor adds 400+ files to the generated application; the footprint is about 2.5Mb. **CKEditor** documentation is available on the at https://ckeditor.com/docs/.

To enable the image upload feature, download and install **CKFinder**: https://ckeditor.com/ckfinder/download/

The documentation for **CKFinder** is available at: https://ckeditor.com/docs/ckfinder/latest/

See also:

- Page Designer screen
- "Edit as" settings
- Edit as: Text area
- How to create a custom Edit control plugin

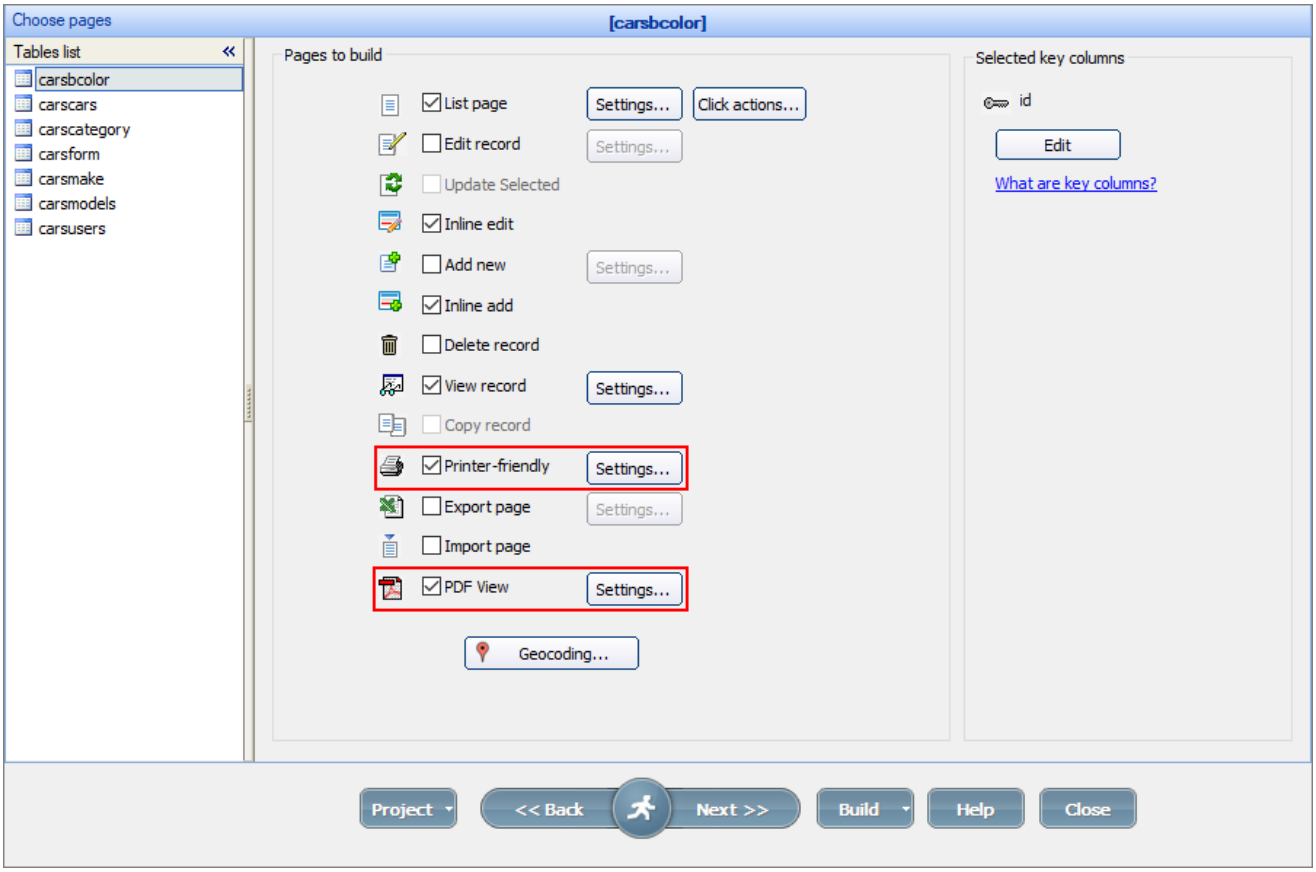
3.15 PDF view settings

PHPRunner comes with a built-in option to download pages as PDF documents.

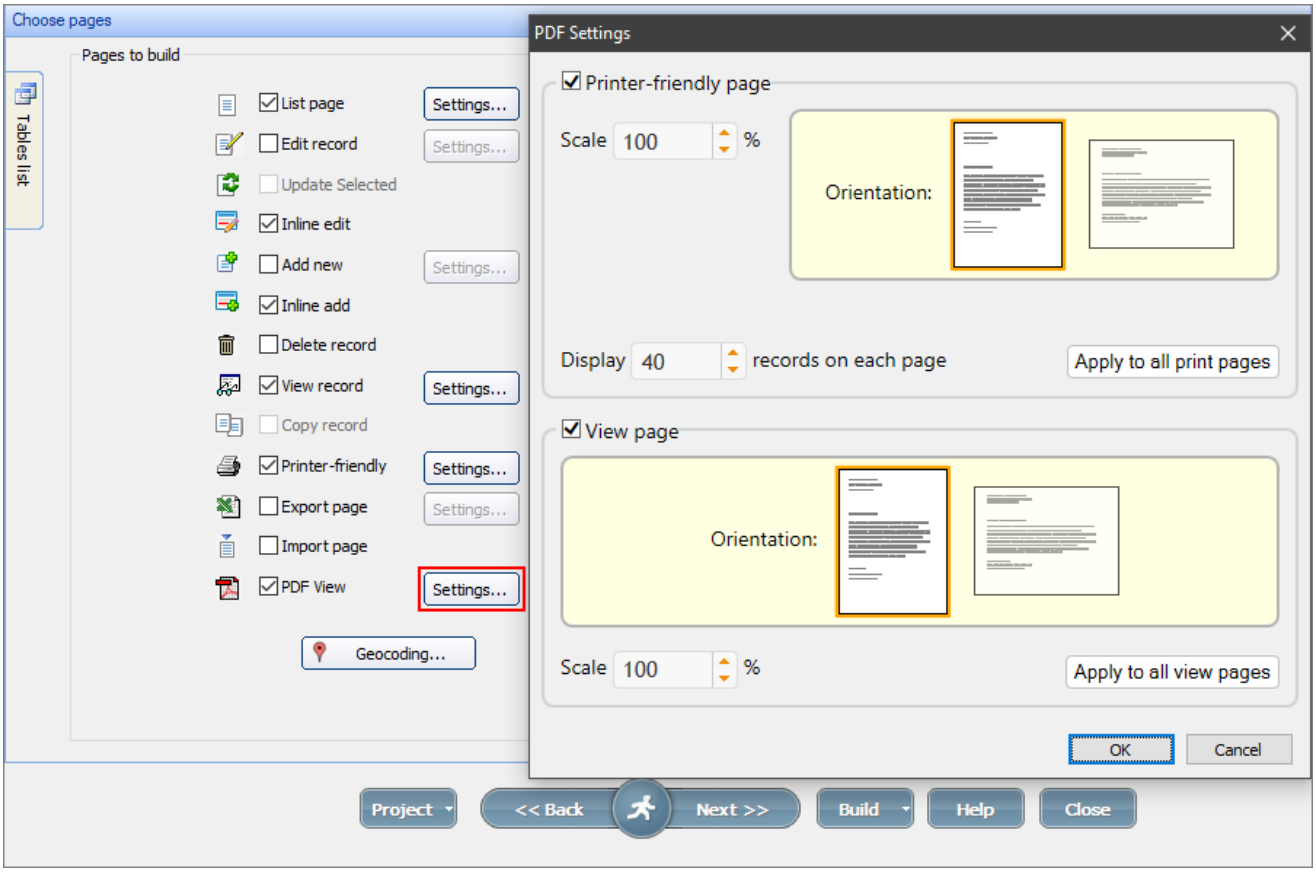
You can enable the PDF view option on the Choose pages screen or a **View**/**Print** page of the Page Designer screen.

Choose pages screen

1. Proceed to the **Choose pages** screen in PHPRunner and select the **PDF View** checkbox.



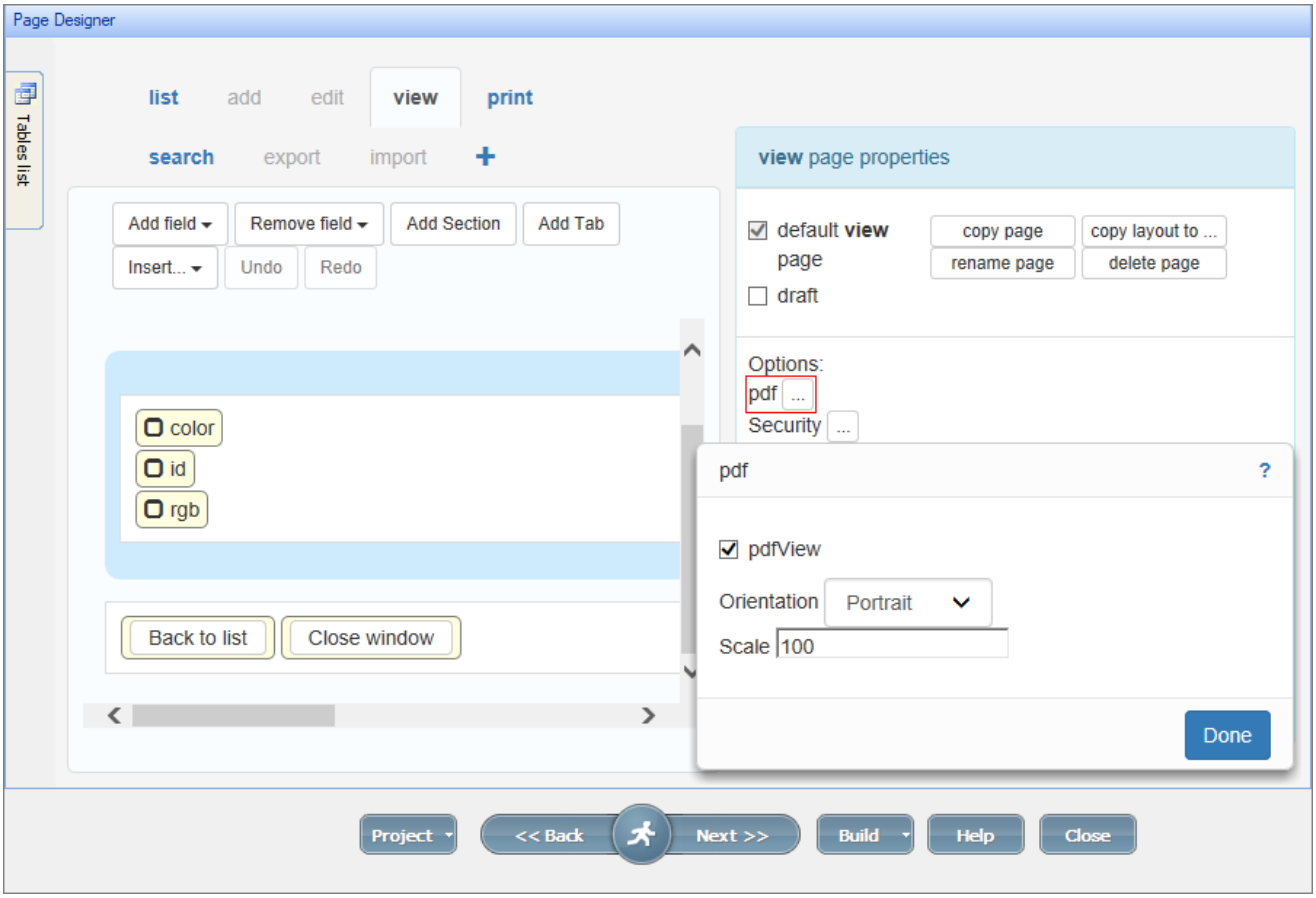
Click the **Settings** button to configure these options:



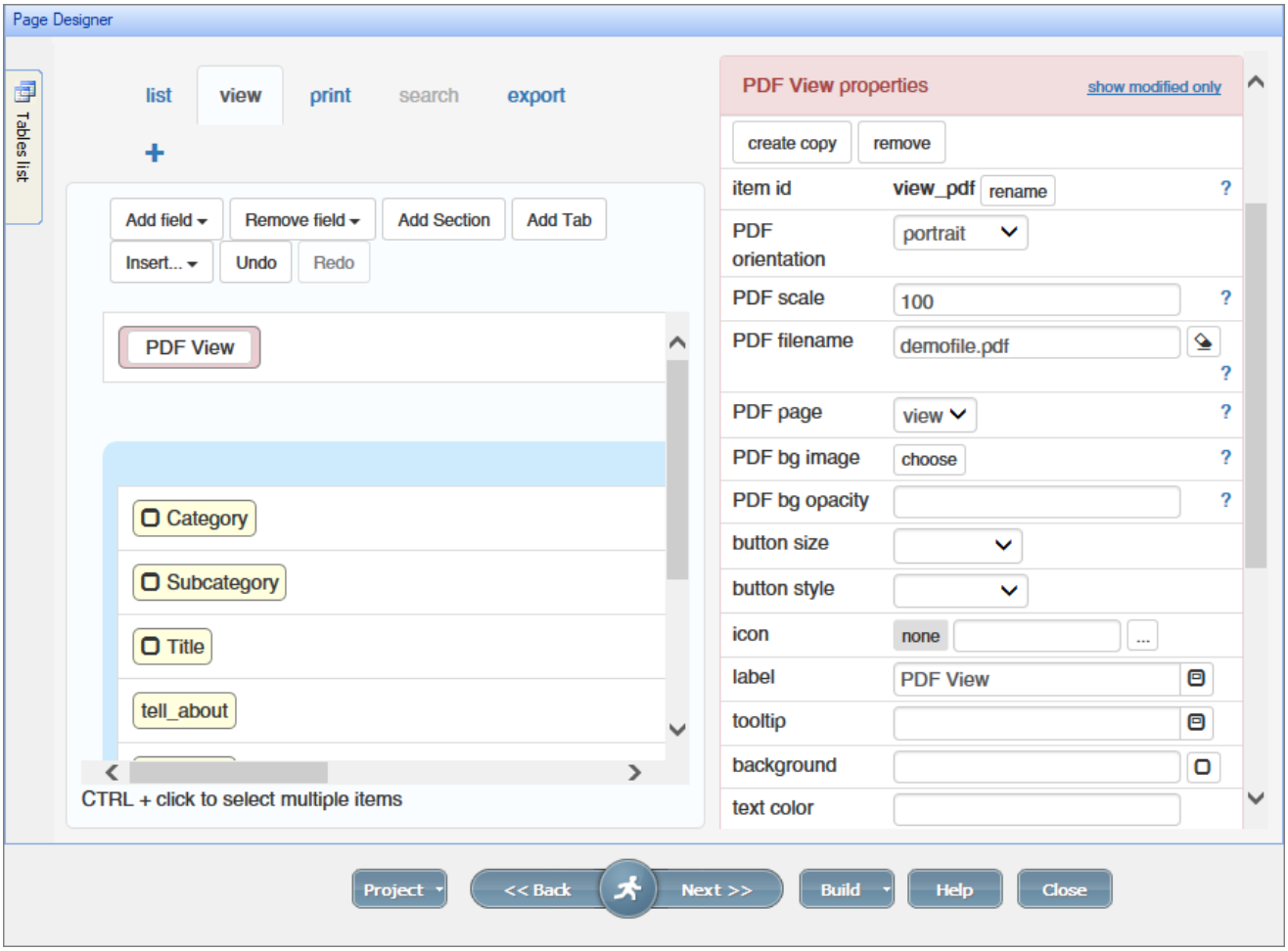
For more information, see Printer-friendly/PDF view settings.

Page Designer

2. Proceed to the **View**/**Print** pages on the **Page Designer** screen and check the page properties. The **View**/**Print** pages have the **pdf** option. Click the **...** button to enable or disable **PDF View**, set the **scale** and **orientation**.



With the **pdfView** option enabled, a **PDF View** button appears on the page. You can drag-n-drop it anywhere in the Page Designer screen.



The **PDF View** button has several unique properties:

`PDF orientation`

You can choose between the **portrait** and **landscape** orientation.

`PDF scale`

Set the **scale** of the resulting document.

`PDF filename`

Set the **name** of the PDF file.

`PDF page`

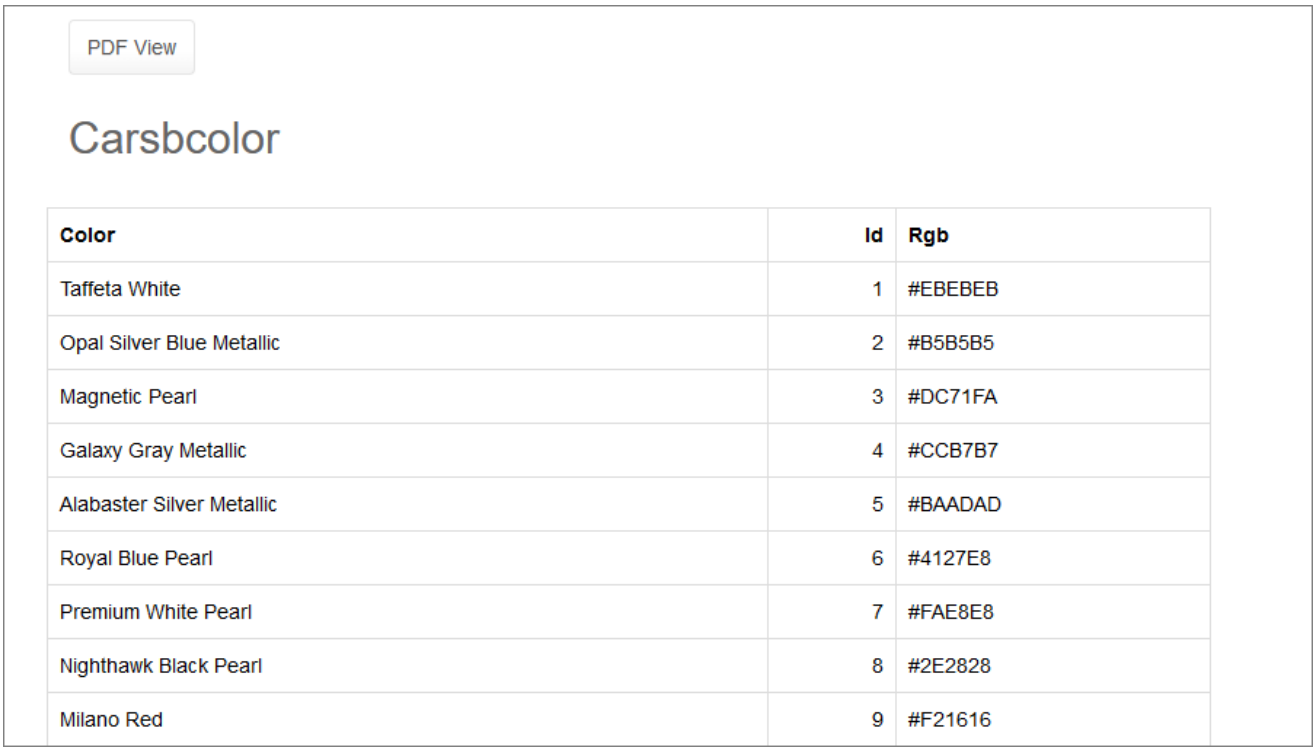
Choose between **View** or **Print** pages to export to PDF.

`PDF bg image, PDF bg opacity`

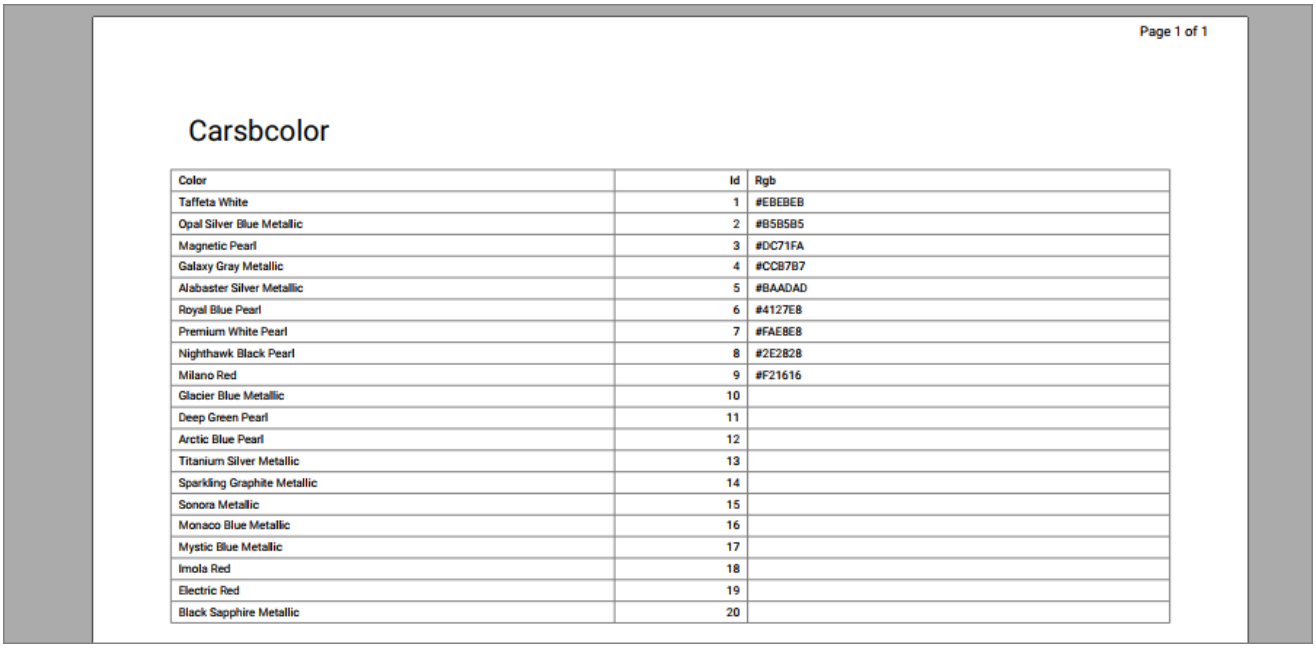
Starting with PHPRunner version 10.3, you can choose an image as the **background** of the resulting PDF file and set its **opacity**.

To learn more about properties of buttons, see Working with page elements.

Here is an example of a **Print** page with a **PDF View** button:



Here is the generated PDF document for the **Print** page above:

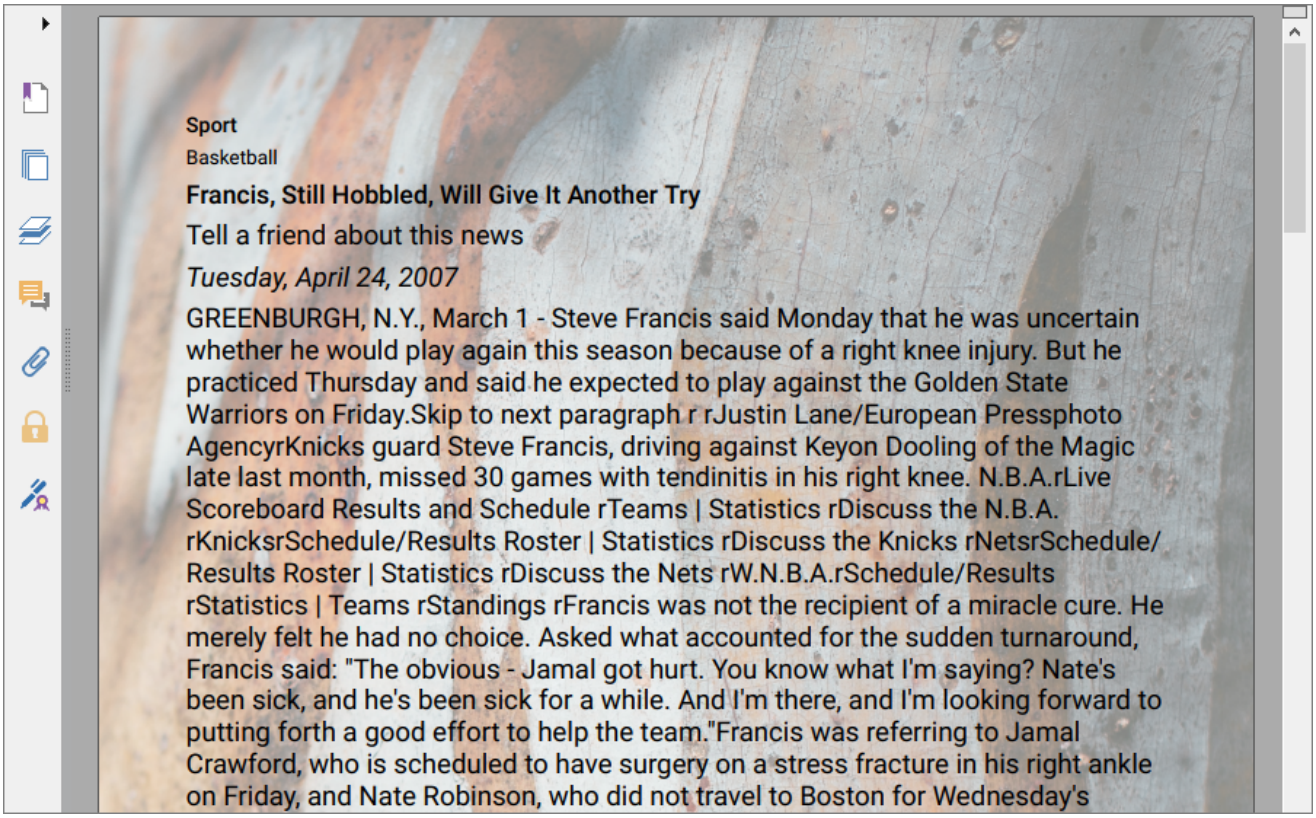


Page 1 of 1

Carsbcolor

Color	Id	Rgb
Taffeta White	1	#EBEBEB
Opal Silver Blue Metallic	2	#B5B5B5
Magnetic Pearl	3	#DC71FA
Galaxy Gray Metallic	4	#CCB7B7
Alabaster Silver Metallic	5	#BAADAD
Royal Blue Pearl	6	#4127E8
Premium White Pearl	7	#FAE8E8
Nighthawk Black Pearl	8	#2E2828
Milano Red	9	#F21616
Glacier Blue Metallic	10	
Deep Green Pearl	11	
Arctic Blue Pearl	12	
Titanium Silver Metallic	13	
Sparkling Graphite Metallic	14	
Sonora Metallic	15	
Monaco Blue Metallic	16	
Mystic Blue Metallic	17	
Imola Red	18	
Electric Red	19	
Black Sapphire Metallic	20	

Here is an example of a PDF document with a custom **background**:



Sport

Basketball

Francis, Still Hobbled, Will Give It Another Try

Tell a friend about this news

Tuesday, April 24, 2007

GREENBURGH, N.Y., March 1 - Steve Francis said Monday that he was uncertain whether he would play again this season because of a right knee injury. But he practiced Thursday and said he expected to play against the Golden State Warriors on Friday.Skip to next paragraph r rJustin Lane/European Presspho to AgencyrKnicks guard Steve Francis, driving against Keyon Dooling of the Magic late last month, missed 30 games with tendinitis in his right knee. N.B.A.rLive Scoreboard Results and Schedule rTeams | Statistics rDiscuss the N.B.A. rKnicksrSchedule/Results Roster | Statistics rDiscuss the Knicks rNetsrSchedule/ Results Roster | Statistics rDiscuss the Nets rW.N.B.A.rSchedule/Results rStatistics | Teams rStandings rFrancis was not the recipient of a miracle cure. He merely felt he had no choice. Asked what accounted for the sudden turnaround, Francis said: "The obvious - Jamal got hurt. You know what I'm saying? Nate's been sick, and he's been sick for a while. And I'm there, and I'm looking forward to putting forth a good effort to help the team."Francis was referring to Jamal Crawford, who is scheduled to have surgery on a stress fracture in his right ankle on Friday, and Nate Robinson, who did not travel to Boston for Wednesday's

See also:

- PDF view settings
- View as: PDF
- Choose pages screen
- Page Designer screen
- Working with table pages
- Working with page elements

3.16 Testing the mobile version of your web app

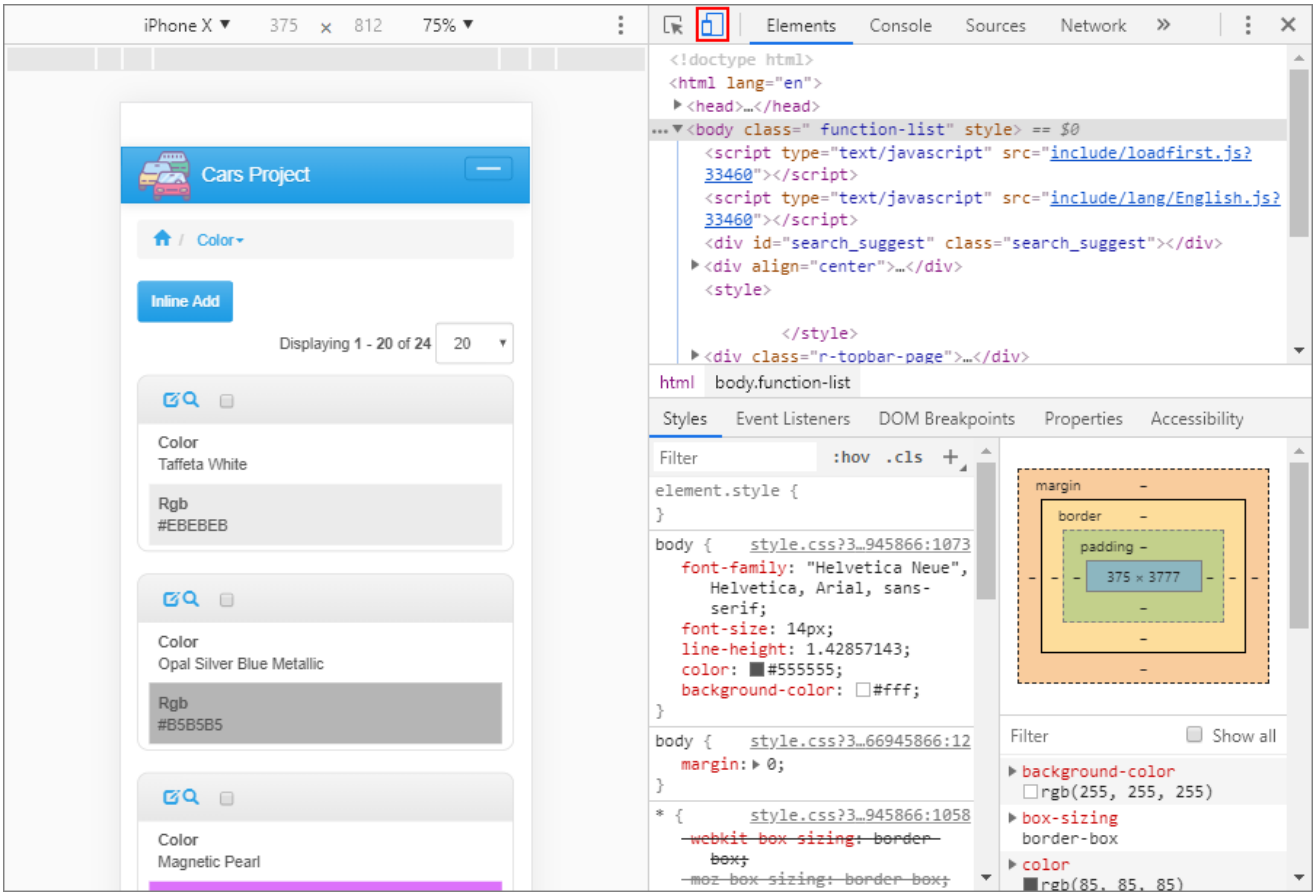
There are several different ways for you to test the mobile version of your application.

Using the developer tools in the browser

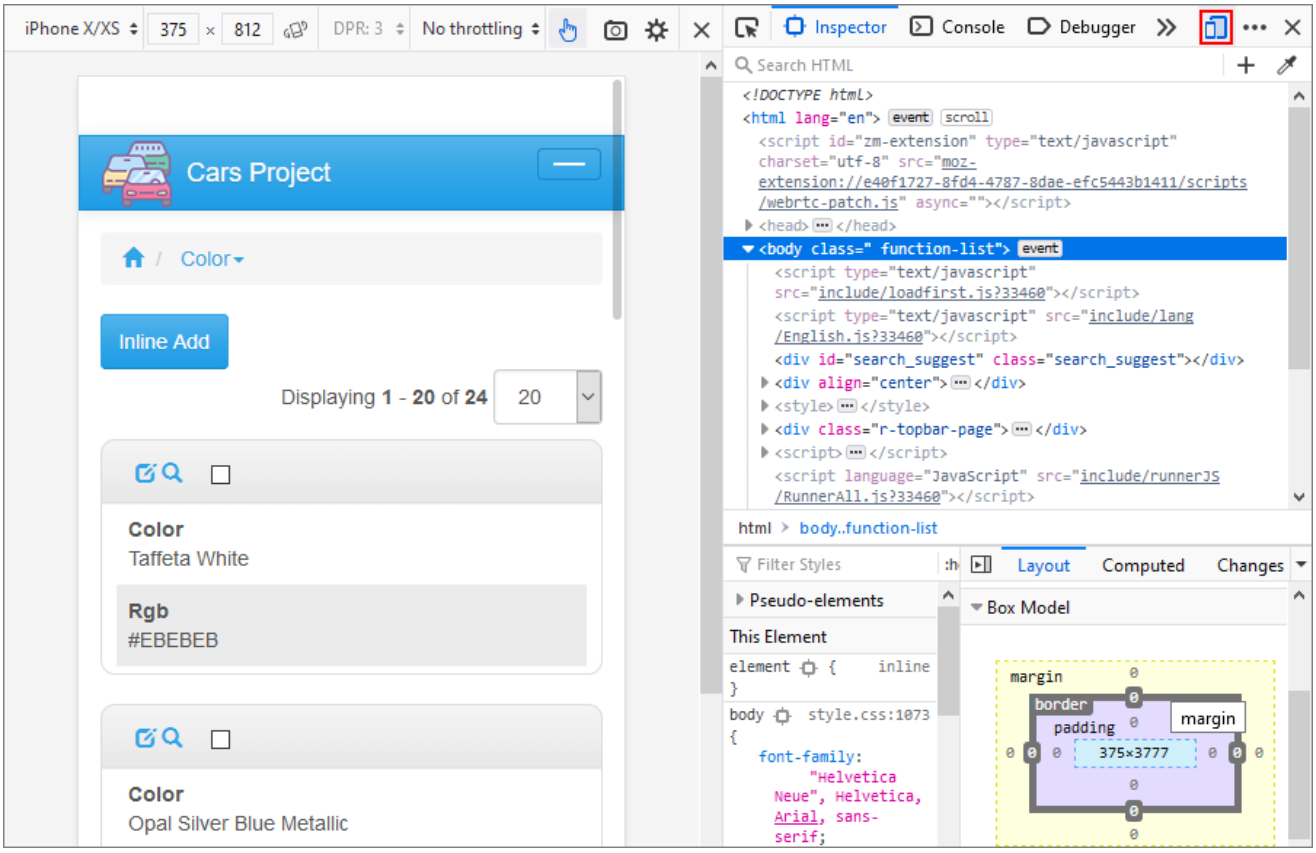
Modern browsers have an option to emulate the way mobile devices show any web page using the Developer tools.

Hit F12 to open the Chrome/Firefox Developers Tools. To emulate the mobile mode:

Click the **Toggle device toolbar** button in the Chrome Developer tools panel.



Click the **Responsive Design Mode** button in the Firefox Developer tools panel.



You can select different devices or set the resolution yourself and choose horizontal or vertical orientation.

Using your mobile device

If your mobile device and desktop computer are on the same network, you can test your app with your smartphone or tablet.

> **Note**: the built-in web server that comes with PHPRunner is only designed for local testing. You need to use a web server, for example, XAMPP.

Choose the output folder of the project under your web server root folder and build the app. Open the URL, e.g., *http://desktop_computer_name/project_name* in the browser on your mobile device.

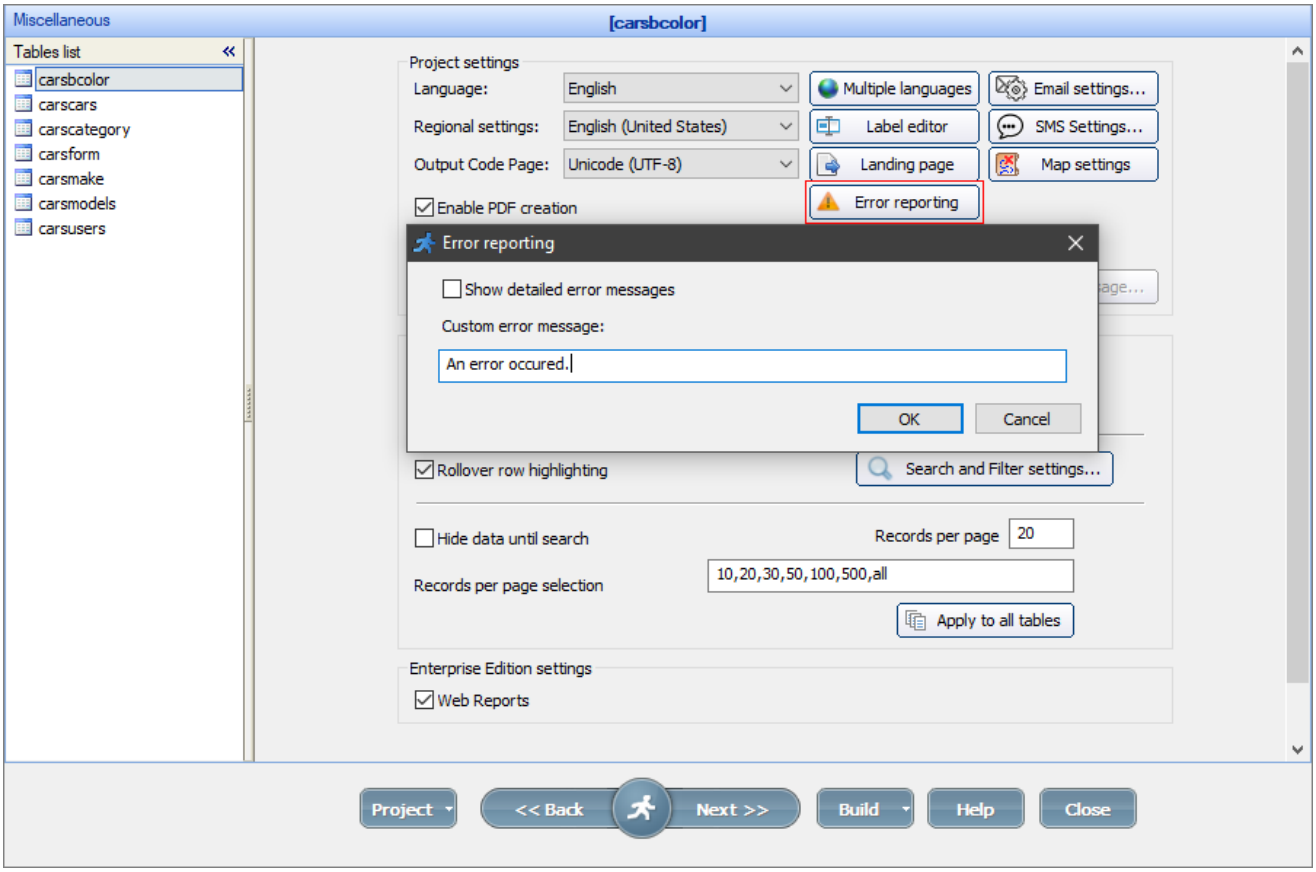
Alternatively, you can upload your project to the Demo Account, and open the link on your mobile device.

See also:

- After you are done
- Demo Account
- How to install a local web server (XAMPP)

3.17 Error reporting

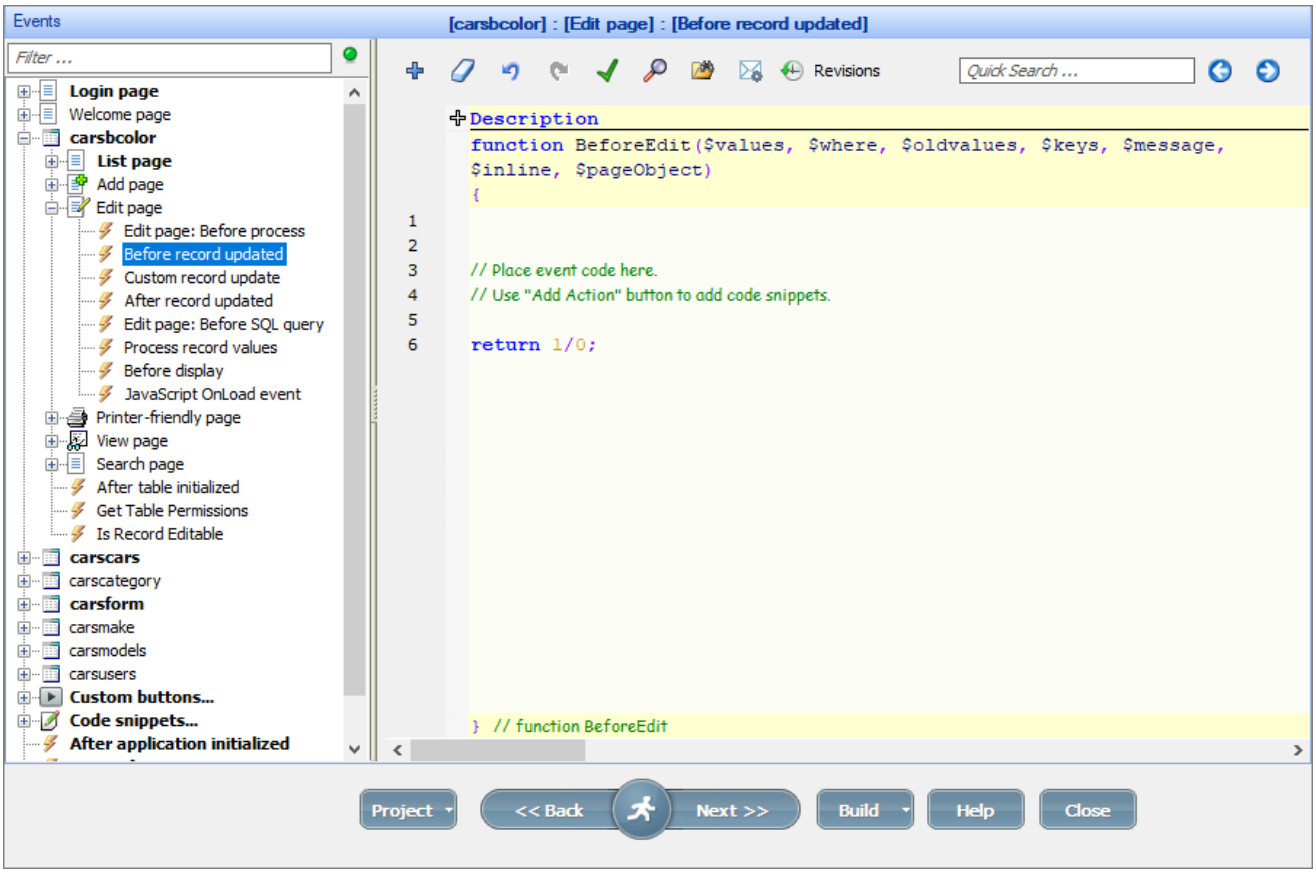
You can display a custom error message instead of detailed error messages. To enable this option, go to Miscellaneous screen -> Error reporting, clear the **Show detailed error messages** checkbox, and enter the custom message.



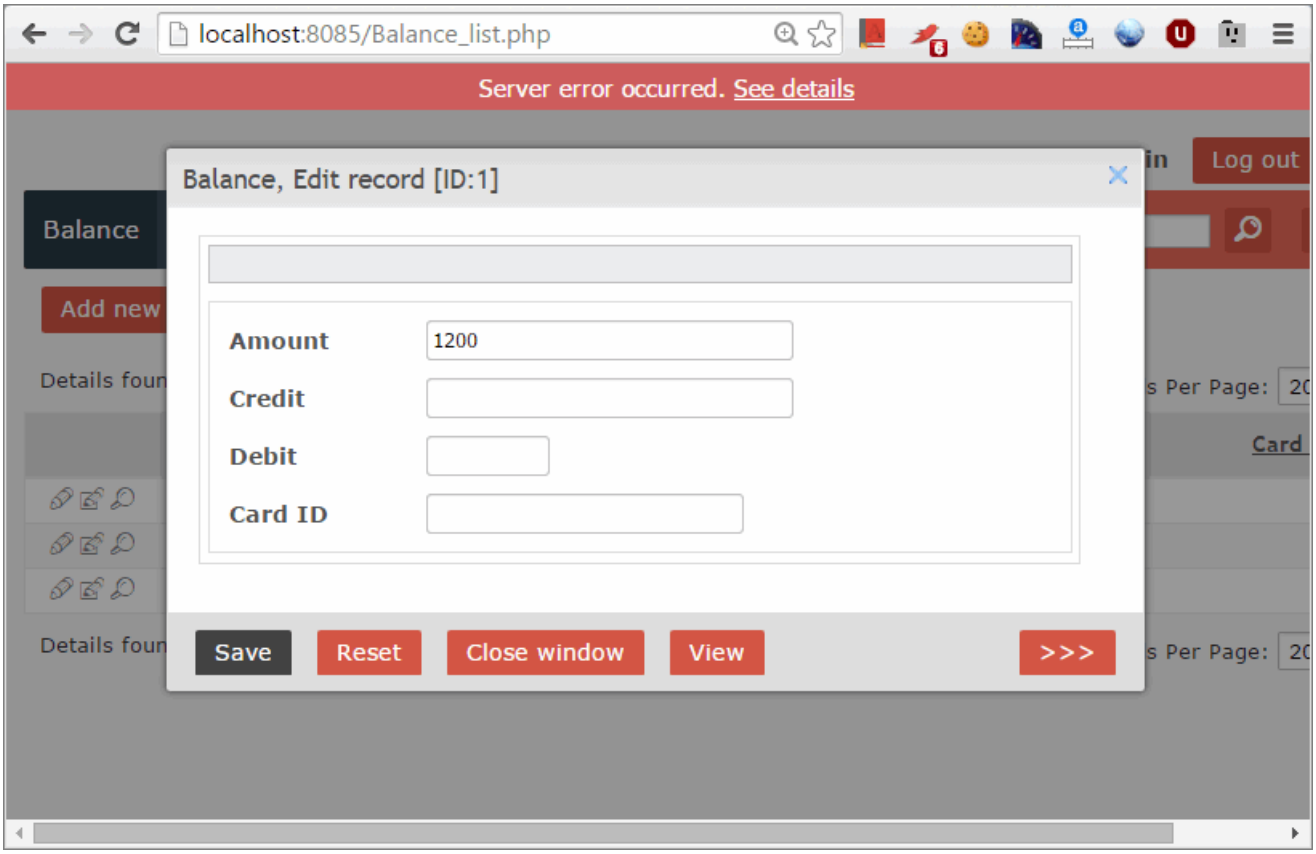
Displaying server errors in AJAX mode

You can view the details of server errors in real time while browsing the generated app. Here is an example.

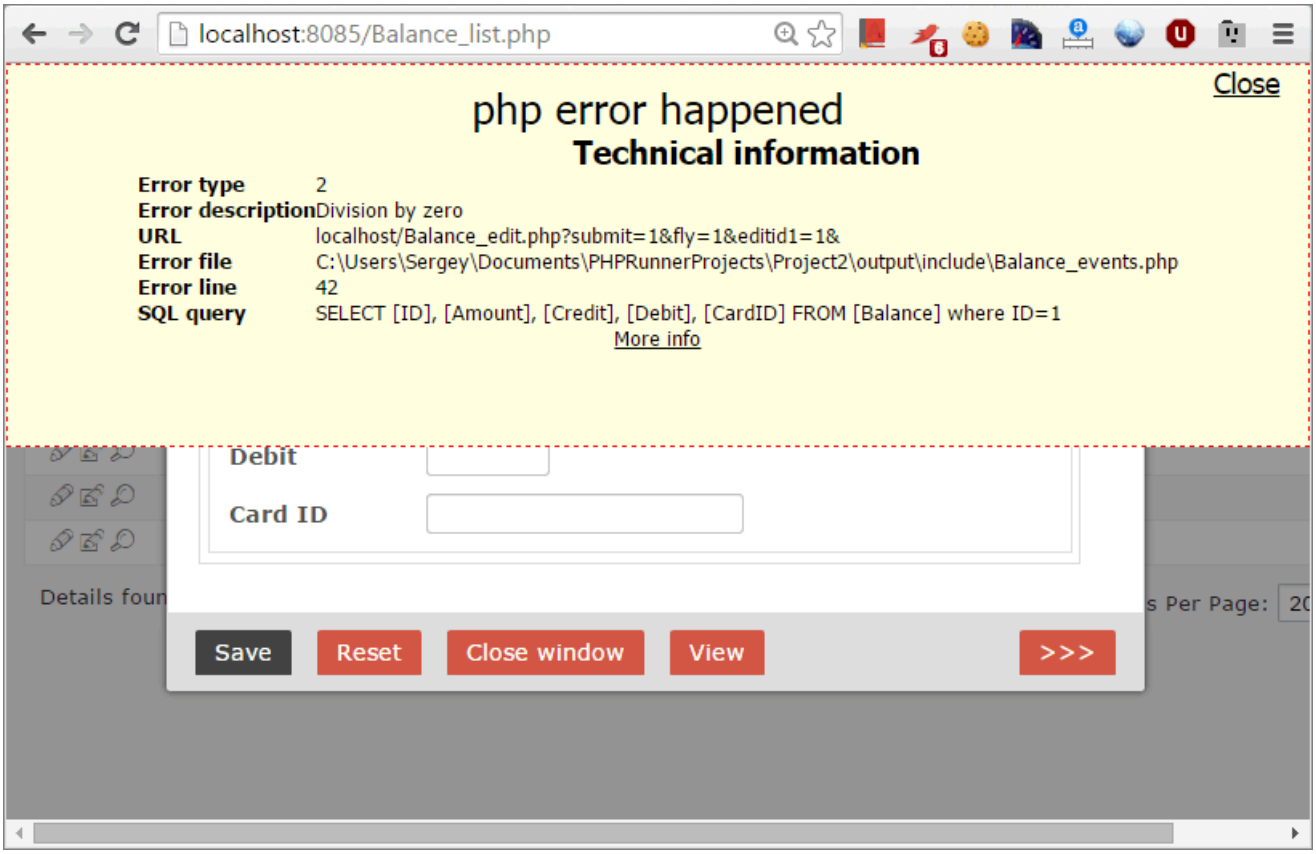
The **Edit** page is set to be displayed in a popup. The BeforeEdit event for this page contains an error: division by 0.



When trying to save the updated record, an *'error occurred'* message appears at the top of the page.



Click **See details** to view the extended error information.



See also:

- Miscellaneous settings
- Choose pages screen
- List page settings
- Event editor

4 Order PHPRunner online

PHPRunner may be ordered through our online store seven days a week, 24 hours a day.

Click here to order your copy now!

Visit PHPRunner online