SQL Manager.net™
EMS® Software Development

Advanced Localizer for RAD Studio VCL User's Manual

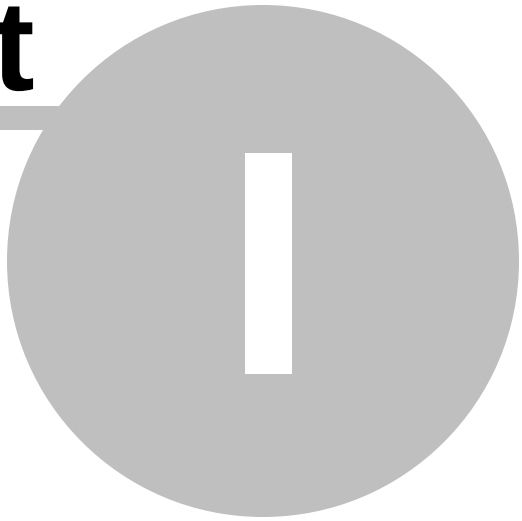
Advanced Localizer for RAD Studio VCL
User's Manual

This manual documents EMS Advanced Localizer for RAD Studio VCL, version 1.7.

Document generated on: 15.01.2025

Table of Contents

Part



1 Welcome to Advanced Localizer for RAD Studio VCL!

1.1 Overview

EMS Advanced Localizer for RAD Studio VCL is a component for Delphi and C++ Builder intended for adding the ability of multilingual support to your applications. Using powerful component editors of this suite you can easily and quickly localize the properties of your project components within each form, generate the template of language file containing current values of component properties, manage the localization files, specify the components and properties to be localized and choose other localization options. Applications that use **Advanced Localizer for RAD Studio VCL** can be localized in one touch at run-time and do not require to be reloaded. Moreover, there is an ability to write the component descendants that can work with user-defined formats of language files.

Key features

- Quick localization of the language-specific component properties within each form
- Easy management of localization files
- Support for 64-bit Windows target platform
- Switch languages during run-time with a single command, and without reloading
- Ability to write component descendants that can work with user-defined language files
- High productivity even on slow computers
- Detailed help system and a demo application for quicker mastering the product
- Powerful component and property editors that allow you to localize your project without writing a single line of code
- Delphi 2010, XE-XE8, 10 Seattle, 10.1 Berlin, 10.2 Tokyo, 10.3 Rio, 10.4 Sydney, 11 Alexandria, 12 Athens and C++ Builder 2010, XE-XE8, 10 Seattle, 10.1 Berlin, 10.2 Tokyo, 10.3 Rio, 10.4 Sydney, 11 Alexandria, 12 Athens

Product information

Homepage	https://www.sqlmanager.net/products/tools/advancedlocalizer
Support Ticket System	https://www.sqlmanager.net/support
Register on-line at	https://www.sqlmanager.net/products/tools/advancedlocalizer/buy

1.2 What's new

Version	Release date
Advanced Localizer for RAD Studio VCL 2.0.4	December 6, 2023

What's new in Advanced Localizer for RAD Studio VCL?

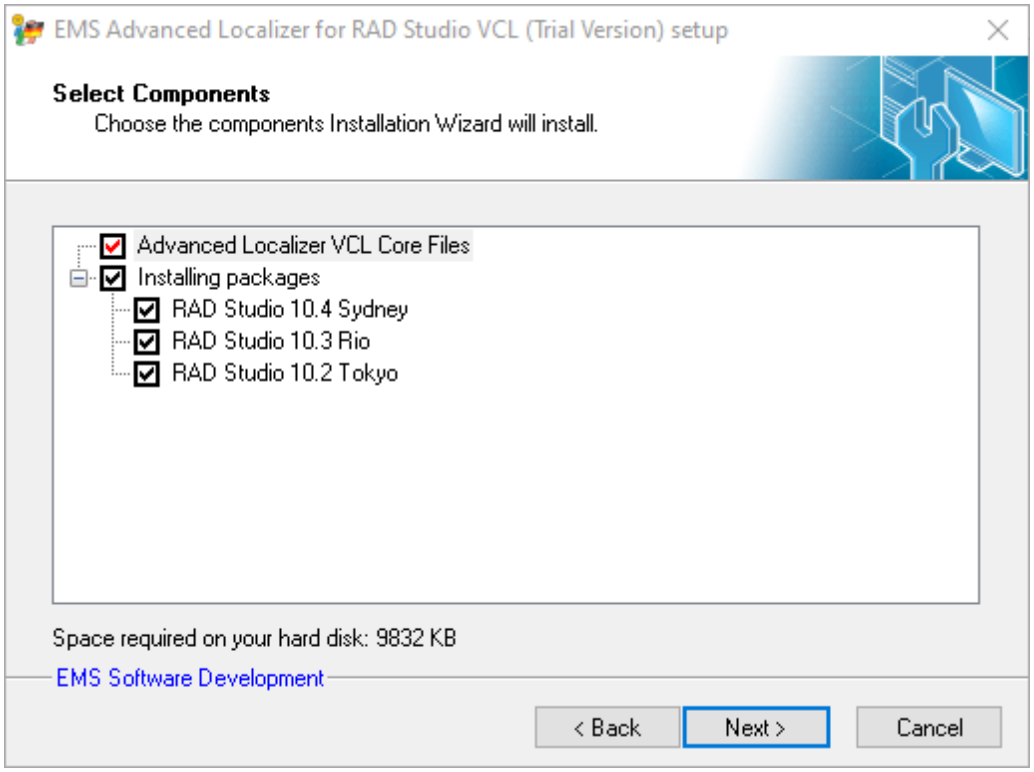
- Support for RAD Studio 12 Athens implemented.
- Fixed paths for 32-bit Clang compiler in RAD Studio options.

1.3 Installation

To install the **trial version** of **Advanced Localizer for RAD Studio VCL** onto your system:

- download the distribution package of **Advanced Localizer for RAD Studio VCL** from the download page available at our website;
- unzip the downloaded file to any local directory, e.g. *C:\unzipped*;
- close all currently opened Delphi and/or C++ Builder IDEs, if any;
- run the executable setup file from the local directory and follow the instructions of the installation wizard.

During the installation you will need to select the packages to install:



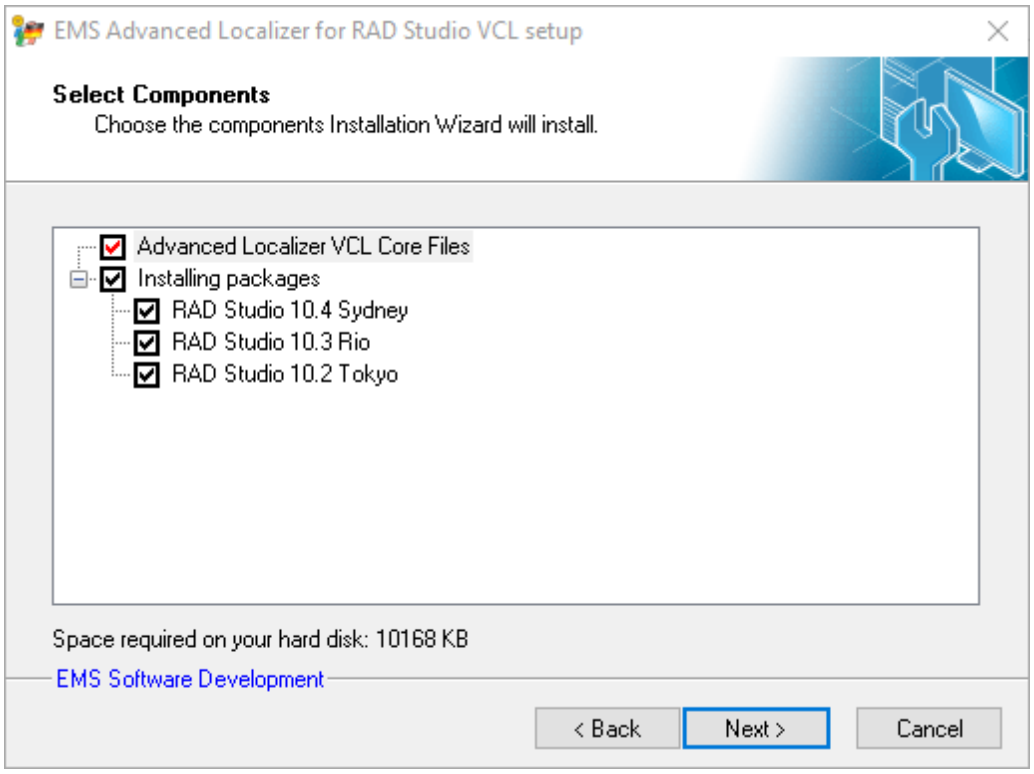
When you are done, you can finish installation of the **trial version** of **Advanced Localizer for RAD Studio VCL**.

To install the **full version** of **Advanced Localizer for RAD Studio VCL** onto your system:

- download the distribution package of **Advanced Localizer for RAD Studio VCL** from the download page available at our website;
- unzip the downloaded file to any local directory, e.g. *C:\unzipped*;
- close all currently opened Delphi and/or C++ Builder IDEs, if any;
- run the executable setup file from the local directory and follow the instructions of the installation wizard.

Enter valid registration information in the appropriate boxes: **Registration name** and **Registration Key**. See details on getting this information.

During the installation you will need to select the packages to install:



When you are done, you can finish installation of the **full version** of **Advanced Localizer for RAD Studio VCL**.

Note: If the above given instructions have been insufficient for successful installation of the component suite, please refer to the *readme.1st* file distributed with the product.

1.4 Registration

All purchases are provided by **PayPro Global** registration service. The **PayPro Global** order process is protected via a secure connection and makes on-line ordering by credit/ debit card quick and safe.

PayPro Global is a global e-commerce provider for software and shareware sales via the Internet. It accepts payments in US Dollars, Euros, Pounds Sterling, Japanese Yen, Australian Dollars, Canadian Dollars or Swiss Franks by Credit Card (Visa, MasterCard/ EuroCard, American Express, Diners Club), Bank/Wire Transfer.

If you want to review your order information, or you have questions about ordering or payments please visit our PayPro Global Shopper Support, provided by **PayPro Global.**

Please note that all of our products are delivered via ESD (Electronic Software Delivery) only. After purchase you will be able to immediately download the registration keys. Also you will receive a copy of registration keys by email. Please make sure to enter a valid email address in your order. If you have not received the keys within 2 hours, please, contact us at sales@sqlmanager.net.

Product distribution	**PayPro Global**
Advanced Localizer for RAD Studio VCL Component Full version (with sources)*	Register Now!
Advanced Localizer for RAD Studio VCL Component Trial version	Download Now!

***EMS Maintenance Program** provides the following benefits:

- Free software bug fixes, enhancements, updates and upgrades during the maintenance period
- Free unlimited communications with technical staff for the purpose of reporting Software failures
- Free reasonable number of communications for the purpose of consultation on operational aspects of the software

After your maintenance expires you will not be able to update your software or get technical support. To protect your investments and have your software up-to-date, you need to renew your maintenance.

You can easily reinitiate/renew your maintenance with our on-line, speed-through Maintenance Reinstatement/Renewal Interface. After reinitiating/renewal you will receive a confirmation e-mail with all the necessary information.

1.5 How to register Advanced Localizer

To register your newly purchased copy of **EMS Advanced Localizer for RAD Studio VCL**, perform the following steps:

- receive the notification letter from **Digital River** with the registration info;
- enter the **Registration Name** and the **Registration Key** from this letter while installing the **full version** of the product.

See also:

Registration

1.6 Other EMS Products

Quick navigation


MySQL


Microsoft SQL Server


PostgreSQL


InterBase / FireBird


Oracle


IBM DB2


Tools & components

MySQL


SQL Management Studio for MySQL
EMS SQL Management Studio for MySQL is a complete solution for database administration and development. SQL Studio unites the must-have tools in one powerful and easy-to-use environment that will make you more productive than ever before!


SQL Manager for MySQL
Simplify and automate your database development process, design, explore and maintain existing databases, build compound SQL query statements, manage database user rights and manipulate data in different ways.


Data Export for MySQL
Export your data to any of 20 most popular data formats, including MS Access, MS Excel, MS Word, PDF, HTML and more.


Data Import for MySQL
Import your data from MS Access, MS Excel and other popular formats to database tables via user-friendly wizard interface.


Data Pump for MySQL
Migrate from most popular databases (MySQL, PostgreSQL, Oracle, DB2, InterBase/Firebird, etc.) to MySQL.


Data Generator for MySQL
Generate test data for database testing purposes in a simple and direct way. Wide range of data generation parameters.


DB Comparer for MySQL
Compare and synchronize the structure of your databases. Move changes on your development database to production with ease.


DB Extract for MySQL
Create database backups in the form of SQL scripts, save your database structure and table data as a whole or partially.


SQL Query for MySQL
Analyze and retrieve your data, build your queries visually, work with query plans, build charts based on retrieved data quickly and more.


Data Comparer for MySQL
Compare and synchronize the contents of your databases. Automate your data migrations from development to production database.

Scroll to top

Microsoft SQL Server

SQL Management Studio for SQL Server
EMS SQL Management Studio for SQL Server is a complete solution for database administration and development. SQL Studio unites the must-have tools in one powerful and easy-to-use environment that will make you more productive than ever before!

EMS SQL Backup for SQL Server
Perform backup and restore, log shipping and many other regular maintenance tasks on the whole set of SQL Servers in your company.

SQL Administrator for SQL Server
Perform administrative tasks in the fastest, easiest and most efficient way. Manage maintenance tasks, monitor their performance schedule, frequency and the last execution result.

SQL Manager for SQL Server
Simplify and automate your database development process, design, explore and maintain existing databases, build compound SQL query statements, manage database user rights and manipulate data in different ways.

Data Export for SQL Server
Export your data to any of 20 most popular data formats, including MS Access, MS Excel, MS Word, PDF, HTML and more

Data Import for SQL Server
Import your data from MS Access, MS Excel and other popular formats to database tables via user-friendly wizard interface.

Data Pump for SQL Server
Migrate from most popular databases (MySQL, PostgreSQL, Oracle, DB2, InterBase/Firebird, etc.) to Microsoft® SQL Server™.

Data Generator for SQL Server
Generate test data for database testing purposes in a simple and direct way. Wide range of data generation parameters.

DB Comparer for SQL Server
Compare and synchronize the structure of your databases. Move changes on your development database to production with ease.

DB Extract for SQL Server
Create database backups in the form of SQL scripts, save your database structure and table data as a whole or partially.

SQL Query for SQL Server
Analyze and retrieve your data, build your queries visually, work with query plans, build charts based on retrieved data quickly and more.

Data Comparer for SQL Server
Compare and synchronize the contents of your databases. Automate your data migrations from development to production database.

Scroll to top

PostgreSQL

SQL Management Studio for PostgreSQL
EMS SQL Management Studio for PostgreSQL is a complete solution for database administration and development. SQL Studio unites the must-have tools in one powerful and easy-to-use environment that will make you more productive than ever before!

EMS SQL Backup for PostgreSQL
Creates backups for multiple PostgreSQL servers from a single console. You can use automatic backup tasks with advanced schedules and store them in local or remote folders or cloud storages

SQL Manager for PostgreSQL
Simplify and automate your database development process, design, explore and maintain existing databases, build compound SQL query statements, manage database user rights and manipulate data in different ways.

Data Export for PostgreSQL
Export your data to any of 20 most popular data formats, including MS Access, MS Excel, MS Word, PDF, HTML and more

Data Import for PostgreSQL
Import your data from MS Access, MS Excel and other popular formats to database tables via user-friendly wizard interface.

Data Pump for PostgreSQL
Migrate from most popular databases (MySQL, SQL Server, Oracle, DB2, InterBase/Firebird, etc.) to PostgreSQL.

Data Generator for PostgreSQL
Generate test data for database testing purposes in a simple and direct way. Wide range of data generation parameters.

DB Comparer for PostgreSQL
Compare and synchronize the structure of your databases. Move changes on your development database to production with ease.

DB Extract for PostgreSQL
Create database backups in the form of SQL scripts, save your database structure and table data as a whole or partially.

SQL Query for PostgreSQL
Analyze and retrieve your data, build your queries visually, work with query plans, build charts based on retrieved data quickly and more.

Data Comparer for PostgreSQL
Compare and synchronize the contents of your databases. Automate your data migrations from development to production database.

Scroll to top

InterBase / Firebird

SQL Management Studio for InterBase/Firebird
EMS SQL Management Studio for InterBase and Firebird is a complete solution for database administration and development. SQL Studio unites the must-have tools in one powerful and easy-to-use environment that will make you more productive than ever before!

SQL Manager for InterBase/Firebird
Simplify and automate your database development process, design, explore and maintain existing databases, build compound SQL query statements, manage database user rights and manipulate data in different ways.

Data Export for InterBase/Firebird
Export your data to any of 20 most popular data formats, including MS Access, MS Excel, MS Word, PDF, HTML and more

Data Import for InterBase/Firebird
Import your data from MS Access, MS Excel and other popular formats to database tables via user-friendly wizard interface.

Data Pump for InterBase/Firebird
Migrate from most popular databases (MySQL, SQL Server, Oracle, DB2, PostgreSQL, etc.) to InterBase/Firebird.

Data Generator for InterBase/Firebird
Generate test data for database testing purposes in a simple and direct way. Wide range of data generation parameters.

DB Comparer for InterBase/Firebird
Compare and synchronize the structure of your databases. Move changes on your development database to production with ease.

DB Extract for InterBase/Firebird
Create database backups in the form of SQL scripts, save your database structure and table data as a whole or partially.

SQL Query for InterBase/Firebird
Analyze and retrieve your data, build your queries visually, work with query plans, build charts based on retrieved data quickly and more.

Data Comparer for InterBase/Firebird
Compare and synchronize the contents of your databases. Automate your data migrations from development to production database.

Scroll to top

Oracle

SQL Management Studio for Oracle
EMS SQL Management Studio for Oracle is a complete solution for database administration and development. SQL Studio unites the must-have tools in one powerful and easy-to-use environment that will make you more productive than ever before!

SQL Manager for Oracle
Simplify and automate your database development process, design, explore and maintain existing databases, build compound SQL query statements, manage database user rights and manipulate data in different ways.

Data Export for Oracle
Export your data to any of 20 most popular data formats, including MS Access, MS Excel, MS Word, PDF, HTML and more.

Data Import for Oracle
Import your data from MS Access, MS Excel and other popular formats to database tables via

user-friendly wizard interface.


Data Pump for Oracle
Migrate from most popular databases (MySQL, PostgreSQL, MySQL, DB2, InterBase/Firebird, etc.) to Oracle


Data Generator for Oracle
Generate test data for database testing purposes in a simple and direct way. Wide range of data generation parameters.


DB Comparer for Oracle
Compare and synchronize the structure of your databases. Move changes on your development database to production with ease.


DB Extract for Oracle
Create database backups in the form of SQL scripts, save your database structure and table data as a whole or partially.


SQL Query for Oracle
Analyze and retrieve your data, build your queries visually, work with query plans, build charts based on retrieved data quickly and more.


Data Comparer for Oracle
Compare and synchronize the contents of your databases. Automate your data migrations from development to production database.

Scroll to top

IBM DB2


SQL Manager for DB2
Simplify and automate your database development process, design, explore and maintain existing databases, build compound SQL query statements, manage database user rights and manipulate data in different ways.


Data Export for DB2
Export your data to any of 20 most popular data formats, including MS Access, MS Excel, MS Word, PDF, HTML and more.


Data Import for DB2
Import your data from MS Access, MS Excel and other popular formats to database tables via user-friendly wizard interface.


Data Pump for DB2
Migrate from most popular databases (MySQL, PostgreSQL, Oracle, MySQL, InterBase/Firebird, etc.) to DB2


Data Generator for DB2
Generate test data for database testing purposes in a simple and direct way. Wide range of data generation parameters.


DB Extract for DB2
Create database backups in the form of SQL scripts, save your database structure and table data as a whole or partially.


SQL Query for DB2
Analyze and retrieve your data, build your queries visually, work with query plans, build charts

based on retrieved data quickly and more.

Scroll to top

Tools & components

Advanced Data Export for RAD Studio VCL
Advanced Data Export for RAD Studio VCL allows you to save your data in the most popular office programs formats.

Advanced Data Export .NET
Advanced Data Export .NET is a component for Microsoft Visual Studio .NET that will allow you to save your data in the most popular data formats for the future viewing, modification, printing or web publication. You can export data into MS Access, MS Excel, MS Word (RTF), PDF, TXT, DBF, CSV and more! There will be no need to waste your time on tiresome data conversion - Advanced Data Export will do the task quickly and will give the result in the desired format.

Advanced Data Import for RAD Studio VCL
Advanced Data Import for RAD Studio VCL will allow you to import your data to the database from files in the most popular data formats.

Advanced PDF Generator for RAD Studio
Advanced PDF Generator for RAD Studio gives you an opportunity to create PDF documents with your applications written on Delphi or C++ Builder.

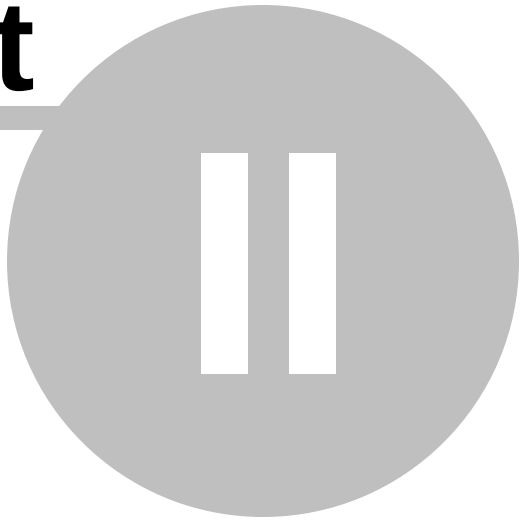
Advanced Query Builder for RAD Studio VCL
Advanced Query Builder for RAD Studio VCL is a powerful component for Delphi and C++ Builder intended for visual building SQL statements for the SELECT, INSERT, UPDATE and DELETE clauses.

Advanced Excel Report for RAD Studio
Advanced Excel Report for RAD Studio is a powerful band-oriented generator of template-based reports in MS Excel.

Advanced Localizer for RAD Studio VCL
Advanced Localizer for RAD Studio VCL is an indispensable component for Delphi for adding multilingual support to your applications.

Scroll to top

Part



2 Advanced Localizer Component

EMS Advanced Localizer for RAD Studio VCL represents a set of tools for efficient localization.

Advanced Localizer for RAD Studio VCL provides a collection of the following components:

Component	Brief description
TQCustomLocalizer	Intended for localizing the string sets
TQFormLocalizer	Localizes the properties of the owner form's components
TQLanguageSource	Reads/writes localized strings from/to the language source
TQCustomLanguageSource	Intended for reading/writing string sets from/to the language files
TQUserLanguageSource	Intended for localizing an application by defining the corresponding event handlers
TQFileLanguageSource	Specifies language source files to write and read localized strings
TQDBLanguageSource	Reads/writes localized strings from/to any instance of the DataSet descendant

2.1 Getting Started

This topic will guide you through the process of quick localizing your Delphi project. Follow the instructions below to learn how to use the **EMS Advanced Localizer for RAD Studio VCL**.

First of all you must install **Advanced Localizer for RAD Studio VCL** to the IDE. See Installation for the proper instructions. If the installation was successful, the **Advanced Localizer for RAD Studio VCL** tab will appear on the Delphi component palette with two components: QFormLocalizer and QLanguageSource. Drop a QLanguageSource component and a QFormLocalizer component onto the form. Set the Source property of QFormLocalizer to the already existing TQLanguageSource component (e.g. QLanguageSource1).
Note: If you don't specify this property, you will not be able to use the QFormLocalizer properly.

Now double-click the QLanguageSource component to activate the Language Source Editor. Click button  to add a language that will be used for localization. Specify the name of the **.lng* file in the 'File Name' edit field, then specify the language name. If no file exists with the name you specified, it will be created. Set this language as active, so that the values from this language file will be applied to the components of the current form (select the language in the list and click button ).

Close the Language Source Editor and double-click the QFormLocalizer component to activate the Form Localizer Editor. In this window you should specify the components and properties to be localized and set their localized values. All the components of the current form are available at the left of the window. Select a component to see the list of its properties, available for localization on the 'Property Names' tab. Check the component properties you want to localize to add them to the grid above. The columns of this grid correspond to the languages from the language list of the QLanguageSource component, the rows to the properties of the currently selected component. Click the cells with the property values twice and enter the new values. Repeat this for each component you want to localize. Click button  to save the changes to the **.lng* file and to apply the property values to the components of the parent form.

2.2 TQCustomLocalizer

2.2.1 TQCustomLocalizer Reference

Unit
QLocal

Description

The *TQCustomLocalizer* is intended for localizing the string sets, i.e. replacing the strings taken from the source, and sending the new strings to the source. This class is abstract, so you can't create an instance of this class - you should use its descendant - TQFormLocalizer, or create your own descendant component.

2.2.2 Properties

Run-time only Key properties

- ClearBeforeSave
- FileName
- LanguageName
- Source

2.2.2.1 ClearBeforeSave

Applies to
TQCustomLocalizer component

Declaration
```
property ClearBeforeSave: boolean;
```

Description

If the *ClearBeforeSave* property is true then the strings associated with the instance of this class via Source property are cleared before saving.

See also:
Source property
Save method

2.2.2.2 FileName

Applies to
TQCustomLocalizer component

Declaration
```
property FileName: string;
```

Description

The *FileName* property contains the name of the language file where the localized strings are stored. This property is read-only.

See also:
LanguageName property
Save method

2.2.2.3 LanguageName

Applies to
TQCustomLocalizer component

Declaration
```
property LanguageName: string;
```

Description

The *LanguageName* property contains the name of the language, currently active. This property is read-only.

See also:
FileName property

2.2.2.4 Source

Applies to
TQCustomLocalizer component

Declaration
```
property Source: TQCustomLanguageSource;
```

Description

The *Source* property determines the TQCustomLanguageSource component, which provides the current TQCustomLocalizer component with the localized strings. This property is used in receiving the localized strings from the language file or saving strings to the file.

See also:
TQCustomLanguageSource component

2.2.3 Methods

Key methods

- LanguageChanged
- LanguageChanging
- Localize
- Save

2.2.3.1 LanguageChanged

Applies to
TQCustomLocalizer component

Declaration
```
procedure LanguageChanged; virtual;
```

Description

The *LanguageChanged* method evokes the OnLanguageChanged event. This method can be overridden in the descendant classes to define the actions taken after changing the active language.

See also:
LanguageChanging method
OnLanguageChanged event

2.2.3.2 LanguageChanging

Applies to
TQCustomLocalizer component

Declaration
```
procedure LanguageChanging; virtual;
```

Description

The *LanguageChanging* method evokes the OnLanguageChanging event. This method can be overridden in the descendant classes to define the actions taken before changing the active language.

See also:
LanguageChanged method
OnLanguageChanging event

2.2.3.3 Localize

Applies to
TQCustomLocalizer component

Declaration
```
procedure Localize; virtual; abstract;
```

Description

The *Localize* method localizes the strings associated with the instance of this class via Source property. This method is abstract and should be overridden in the descendant classes.

See also:
Source property
Save method

2.2.3.4 Save

Applies to
TQCustomLocalizer component

Declaration
```
procedure Save; virtual;
```

Description

The *Save* method saves the localized strings associated with the instance of this class via Source property. To use this method you should override it in the descendant classes.

See also:
Source property
Localize method

2.2.4 Events

Key events

- OnLanguageChanged
- OnLanguageChanging

2.2.4.1 OnLanguageChanged

Applies to
TQCustomLocalizer component

Declaration
```
property OnLanguageChanged: TNotifyEvent;
```

Description

The *OnLanguageChanged* event takes place after the active language is changed. It is invoked by the LanguageChanged method.

See also:
OnLanguageChanging event
LanguageChanged method

2.2.4.2 OnLanguageChanging

Applies to
TQCustomLocalizer component

Declaration
```
property OnLanguageChanging: TQLangChangingEvent;
```

Description

The *OnLanguageChanging* event takes place before the active language is changed. It is invoked by the LanguageChanging method.

See also:
OnLanguageChanged event
LanguageChanging method

2.3 TQFormLocalizer

2.3.1 TQFormLocalizer Reference

Unit
QFormLocal

Description

The *TQFormLocalizer* component localizes the properties of the owner form's components, using the source language file, specified by the TQLanguageSource component.

2.3.2 Properties

Run-time only Key properties

- ClearBeforeSave
- Dependencies
- Excluded
- FileName
- LanguageName
- PropNames
- SaveOptions
- Source

2.3.2.1 Dependencies

Applies to
TQFormLocalizer component

Declaration
```
property Dependencies: TStrings;
```

Description

The *Dependencies* property is used when two properties of the owner form's component have the same value. In such case you can set the dependency between these properties, so that only one property would be saved to the file and the other one would be set automatically when localizing, e.g. if Dependencies = 'Caption=Hint' then all the captions of the owner's form will have the same localized values as the hints; the hints will be saved to the language file, but the captions will be not.

See also:
Excluded property
PropNames property

2.3.2.2 Excluded

Applies to
TQFormLocalizer component

Declaration
```
property Excluded: TStrings;
```

Description

The *Excluded* property contains the names of owner's form components that should not be localized. The localized property values from the source language file will be not applied to these components.

See also:
Dependencies property
PropNames property

2.3.2.3 PropNames

Applies to
TQFormLocalizer component

Declaration
```
property PropNames: TStrings;
```

Description

The *PropNames* property contains the names of the properties to be localized. These properties are common for all the form's components (including the form itself). If one of the components doesn't have some property from this list, then this property will be ignored for this component when localizing.

See also:
Dependencies property
Excluded property

2.3.2.4 SaveOptions

Applies to
TQFormLocalizer component

Declaration
```
property SaveOptions: TQSaveOptions;
```

Description

The *SaveOptions* property is a set of TQSaveOptions properties, which define the different parameters of saving property values to the language file.

See also:
TQSaveOptions type

2.3.3 Methods

Key methods

LanguageChanged
LanguageChanging
Localize
Save

2.3.3.1 Save

Applies to
TQFormLocalizer component

Declaration
```
procedure Save; override;
```

Description

The *Save* method saves the localized strings associated with the instance of this class via Source property to the language file which is currently active (see ActiveLanguage property of the TQCustomLanguageSource component).

See also:
Source property
Localize method

2.3.3.2 Localize

Applies to
TQFormLocalizer component

Declaration
```
procedure Localize; override;
```

Description

The *Localize* method applies the localized strings associated with the instance of this class via Source property to the elements of the currrent form.

See also:
Source property
Save method

2.3.4 Events

Key events

OnLanguageChanged
OnLanguageChanging

2.3.4.1 OnPropertyLocalized

Applies to
TQFormLocalizer component

Declaration
```
property OnPropertyLocalized: TQPropProcessedEvent;
```

Description

The *OnPropertyLocalized* event takes place after a property of some component on a form is localized.

See also:
OnPropertyLocalizing event
OnPropertySaved event
OnPropertySaving event
TQPropProcessedEvent type

2.3.4.2 OnPropertyLocalizing

Applies to
TQFormLocalizer component

Declaration
```
property OnPropertyLocalizing: TQPropProcessingEvent;
```

Description

The *OnPropertyLocalizing* event takes place before a property of some component on a form is localized.

See also:
OnPropertyLocalized event
OnPropertySaved event
OnPropertySaving event
TQPropProcessingEvent type

2.3.4.3 OnPropertySaved

Applies to
TQFormLocalizer component

Declaration
```
property OnPropertySaved: TQPropProcessedEvent;
```

Description

The *OnPropertySaved* event takes place after a property of some component on a form is saved to the language source.

See also:
OnPropertyLocalized event
OnPropertyLocalizing event
OnPropertySaving event
TQPropProcessedEvent type

2.3.4.4 OnPropertySaving

Applies to
TQFormLocalizer component

Declaration
```
property OnPropertySaving: TQPropProcessingEvent;
```

Description

The *OnPropertySaving* event takes place before a property of some component on a form is saved to the language source.

See also:
OnPropertyLocalized event
OnPropertyLocalizing event
OnPropertySaved event
TQPropProcessingEvent type

2.4 TQLanguageSource

2.4.1 TQLanguageSource Reference

Unit
QSource

Description

The *TQLanguageSource* component provides the TQFormLocalizer component with the localized strings.

2.4.2 Properties

Run-time only Key properties

ActiveLanguage
DefaultFileExt
FormSection
IsUpdating
LanguageFile
LanguageName
Languages
OriginalName
ReadOnly

2.4.3 Methods

Key methods

BeginUpdate
Clear
ClearAll
CloseLanguage
GetFormSection
GetReadOnly
EndUpdate
LanguageChanged
LanguageChanging
LoadString
LoadStrings
Localize
OpenLanguage
Save
SaveString
SaveStrings
SetFormSection

2.4.4 Events

Key events

OnLanguageChanged
OnLanguageChanging

2.5 TQCustomLanguageSource

2.5.1 TQCustomLanguageSource Reference

Unit
QLocal

Description

The *TQCustomLocalizer* is intended for reading (writing) string sets from (to) the language files, localizing all the instances of the TQCustomLocalizer components where TQCustomLanguageSource component is set as Source and saving all the string sets received from these components.
This class is abstract, so you can't create an instance of this class - you should use its descendant - TQLanguageSource, or create your own descendant component.
The structure of the language file is not defined, but you can define it yourself, if you create your own descendant component and override such methods as OpenLanguageFile, CloseLanguageFile, LoadString, SaveString, SaveStrings and (optionally) GetFormSection and SetFormSection.

2.5.2 Properties

Run-time only Key properties

- ActiveLanguage
- FormSection
- IsUpdating
- LanguageName
- Languages
- OriginalName
- ReadOnly

2.5.2.1 ActiveLanguage

Applies to
TQCustomLanguageSource component

Declaration
```
property ActiveLanguage: integer;
```

Description

The *ActiveLanguage* property determines the currently active language. The default property value is -1, that means no language is selected (all instances of TQCustomLocalizer attached to the TQCustomLanguageSource use their own string sets). The value of this property is the language index of the Languages property.

See also:
LanguageName property
Languages property

2.5.2.2 FormSection

Applies to
TQCustomLanguageSource component

Declaration
```
property FormSection: string;
```

Description

The *FormSection* property determines the current section of the language file. This property is used only when localizing properties, taken from some definite form, e.g. some properties of the form components, etc. The methods, used for processing this property - GetFormSection and SetFormSection can be overridden in the descendant classes. Thus you can define the way of processing this property yourself.

See also:
LanguageValue property
Localize method
Save method

2.5.2.3 IsUpdating

Applies to
TQCustomLanguageSource component

Declaration
```
property IsUpdating: boolean;
```

Description

The *IsUpdating* property is read-only and it becomes true when the BeginUpdate method is envoked and returns to false on applying EndUpdate method.

See also:
BeginUpdate method
EndUpdate method

2.5.2.4 LanguageName

Applies to
TQCustomLanguageSource component

Declaration
```
property LanguageName[Index: integer]: string;
```

Description

The *LanguageName* property is read-only. It is used to receive the name of the language by its index.

See also:
LanguageValue property
Languages property

2.5.2.5 LanguageValue

Applies to
TQCustomLanguageSource component

Declaration
```
property LanguageValue: string;
```

Description

The *LanguageValue* property contains a string, describing the language source for the language, defined by the LanguageName property.

See also:
LanguageName property

2.5.2.6 Languages

Applies to
TQCustomLanguageSource component

Declaration
```
property Languages: TStrings;
```

Description

The *Languages* property contains all the available languages and the corresponding files with the localized strings. The correspondence between the languages and the language files is set in the following format:
<LanguageName1>=<LanguageFile1>, e.g. English=C:.lng.

See also:
LanguageValue property
LanguageName property

2.5.2.7 OriginalName

Applies to
TQCustomLanguageSource component

Declaration
```
property OriginalName: string;
```

Description

The *OriginalName* contains the name of the original program language (e.g. 'English'). This property doesn't influence anything except the default value of the ActiveLanguage property, so it is possible not to use it at all. The default property value is 'Original'.

See also:
LanguageName property

2.5.2.8 ReadOnly

Applies to
TQCustomLanguageSource component

Declaration
```
property ReadOnly: boolean;
```

Description

The *ReadOnly* property shows whether the source language file is read-only or not. This property is read-only, but you can define its property yourself if you create a descendant component and override the GetReadOnly method.

See also:
LanguageValue property
GetReadOnly method

2.5.2.9 ActiveSettings

Applies to
TQCustomLanguageSource component

Declaration
```
property ActiveSettings: TQLanguageSettings;
```

Description

The *ActiveSettings* property allows you to specify the bi-directional mode and font character set for each localization language separately.

2.5.3 Methods

Key methods

- GetFormSection
- GetReadOnly
- LoadString
- BeginUpdate
- Clear
- ClearAll
- CloseLanguage
- EndUpdate
- LanguageChanged
- LanguageChanging
- LoadString
- LoadStrings
- Localize
- OpenLanguage
- Save
- SaveString
- SaveStrings
- SetFormSection

2.5.3.1 BeginUpdate

Applies to
TQCustomLanguageSource component

Declaration
```
procedure BeginUpdate;
```

Description

The *BeginUpdate* method turns the isUpdating property to TRUE. This method can be used only along with the EndUpdate method. All the changes you make after the BeginUpdate is invoked and before the EndUpdate do not affect the application.

See also:
IsUpdating property
EndUpdate method

2.5.3.2 Clear

Applies to
TQCustomLanguageSource component

Declaration
```
procedure Clear; virtual; abstract;
```

Description

The *Clear* method is used in all the descendant components of TQCustomLanguageSource to clear the current section of the language source. The current section is specified in the FormSection property. If the section is not specified then it clears all the source.

See also:
FormSection property
ClearFile method

2.5.3.3 ClearAll

Applies to
TQCustomLanguageSource component

Declaration
```
procedure ClearAll; virtual; abstract;
```

Description

The *ClearAll* method is used in all the descendant components of TQCustomLanguageSource to clear the contents of the language source.

See also:
Clear method

2.5.3.4 CloseLanguage

Applies to
TQCustomLanguageSource component

Declaration
```
procedure CloseLanguage; virtual; abstract;
```

Description

The *CloseLanguage* method is used in all the descendant components of TQCustomLanguageSource to finish working with the language source.

See also:
OpenLanguage property

2.5.3.5 EndUpdate

Applies to
TQCustomLanguageSource component

Declaration
```
procedure EndUpdate;
```

Description

The *EndUpdate* method turns the isUpdating property to FALSE. This method can be used only along with the BeginUpdate method. All the changes you make after the BeginUpdate is invoked and before the EndUpdate do not affect the application.

See also:
IsUpdating property
BeginUpdate method

2.5.3.6 GetFormSection

Applies to
TQCustomLanguageSource component

Declaration
```
function GetFormSection: string; virtual;
```

Description

The *GetFormSection* method is used in all the descendant components of TQCustomLanguageSource to receive information about the current section of the source language file (see FormSection property).

See also:
FormSection property
SetFormSection method

2.5.3.7 GetReadOnly

Applies to
TQCustomLanguageSource component

Declaration
```
function GetReadOnly: boolean; virtual;
```

Description

The *GetReadOnly* method is used in all the descendant components of TQCustomLanguageSource to receive the value of ReadOnly property.

See also:
ReadOnly property

2.5.3.8 LanguageChanged

Applies to
TQCustomLanguageSource component

Declaration
```
procedure LanguageChanged; virtual;
```

Description

The *LanguageChanged* method evokes the OnLanguageChanged event. This method can be overridden in the descendant classes.

See also:
LanguageChanging method
OnLanguageChanged event

2.5.3.9 LanguageChanging

Applies to
TQCustomLanguageSource component

Declaration
```
procedure LanguageChanging; virtual;
```

Description

The *LanguageChanging* method evokes the OnLanguageChanging event. This method can be overridden in the descendant classes.

See also:
LanguageChanged method
OnLanguageChanging event

2.5.3.10 LoadString

Applies to
TQCustomLanguageSource component

Declaration

```
function LoadString(const StringName, DefaultValue: string): string; virtual;
```

Description

The *LoadString* method is used in all the descendant components of TQCustomLanguageSource to load a string from the source language file.

See also:
LoadStrings method
SaveString method

2.5.3.11 LoadStrings

Applies to
TQCustomLanguageSource component

Declaration
```
procedure LoadStrings(Strings: TStrings); virtual; abstract;
```

Description

The *LoadStrings* method is used in all the descendant components of TQCustomLanguageSource to load a number of strings (e.g. strings of the specified section) from the source language file.

See also:
LoadString method
SaveStrings method

2.5.3.12 Localize

Applies to
TQCustomLanguageSource component

Declaration
```
procedure Localize;
```

Description

The *Localize* method localizes all the instances of the TQCustomLocalizer components, where TQCustomLanguageSource component is set as Source.

See also:
Save method

2.5.3.13 OpenLanguage

Applies to
TQCustomLanguageSource component

Declaration
```
procedure OpenLanguage; virtual; abstract;
```

Description

The *OpenLanguage* method is used in all the descendant components of TQCustomLanguageSource to start working with the language source.

See also:
CloseLanguage method

2.5.3.14 Save

Applies to
TQCustomLanguageSource component

Declaration
```
procedure Save;
```

Description

The *Save* method saves all the localized string sets, received from the TQCustomLocalizer components, where TQCustomLanguageSource component is set as Source.

See also:
LanguageValue property
Localize method

2.5.3.15 SaveString

Applies to
TQCustomLanguageSource component

Declaration

```
procedure SaveString(const StringName, Value: string); virtual;
```

Description

The *SaveString* method is used in all the descendant components of TQCustomLanguageSource to save a string to the source language file.

See also:
SaveStrings method
LoadString method

2.5.3.16 SaveStrings

Applies to
TQCustomLanguageSource component

Declaration
```
procedure SaveStrings(Strings: TStrings); virtual;
```

Description

The *SaveStrings* method is used in all the descendant components of TQCustomLanguageSource to save a number of strings (e.g. strings corresponding to the current form) to the source language file.

See also:
SaveString method
LoadStrings method

2.5.3.17 SetFormSection

Applies to
TQCustomLanguageSource component

Declaration
```
procedure SetFormSection(const Value: string); virtual;
```

Description

The *SetFormSection* method is used in all the descendant components of TQCustomLanguageSource to set the current section of the source language file (see FormSection property).

See also:
FormSection property
GetFormSection method

2.5.4 Events

Key events

- OnLanguageChanged
- OnLanguageChanging

2.5.4.1 OnLanguageChanged

Applies to
TQCustomLanguageSource component

Declaration
```
property OnLanguageChanged: TNotifyEvent;
```

Description

The *OnLanguageChanged* event takes place after the active language is changed. It is invoked by the LanguageChanged method.

See also:
OnLanguageChanging event
LanguageChanged method

2.5.4.2 OnLanguageChanging

Applies to
TQCustomLanguageSource component

Declaration
```
property OnLanguageChanging: TQLangChangingEvent;
```

Description

The *OnLanguageChanging* event takes place before the active language is changed. It is invoked by the LanguageChanging method.

See also:
OnLanguageChanged event
LanguageChanging method

2.6 TQUserLanguageSource

2.6.1 TQUserLanguageSource Reference

Unit
QLocalUserSource

Description

The *TQUserLanguageSource* component allows you to localize your application by defining the corresponding event handlers.

2.6.2 Properties

Run-time only Key properties

Languages

2.6.2.1 Languages

Applies to
TQCustomLanguageSource component

Declaration
```
property Languages: TStrings;
```

Description

The *Languages* property contains all the available languages and the corresponding files with the localized strings. The correspondence between the languages and the language files is set in the following format:
<LanguageName1>=<LanguageFile1>, e.g. English=C:.lng.

See also:
LanguageValue property
LanguageName property

2.6.3 Methods

Key methods

DesignTime
GetReadOnly

2.6.3.1 DesignTime

Applies to
TQCustomLanguageSource component

Declaration
```
function DesignTime: boolean; override;
```

Description

The *DesignTime* method indicates if the controls can be localized in the design-time.

2.6.3.2 GetReadOnly

Applies to
TQCustomLanguageSource component

Declaration
```
function GetReadOnly: boolean; virtual;
```

Description

The *GetReadOnly* method is used in all the descendant components of TQCustomLanguageSource to receive the value of ReadOnly property.

See also:
ReadOnly property

2.6.4 Events

Key events

OnClear
OnClearAll
OnGetBiDiMode
OnGetFontCharSet
OnGetFormSection
OnLanguageChanged
OnLanguageChanging
OnSetBiDiMode
OnSetFontCharSet

2.6.4.1 OnClear

Applies to
TQCustomLanguageSource component

Declaration
```
property OnClear;
```

Description

The *OnClear* event takes place when the Clear method is called. Define the event handler if you want to take some actions after calling this method.

2.6.4.2 OnClearAll

Applies to
TQCustomLanguageSource component

Declaration
```
property OnClearAll;
```

Description

The *OnClearAll* event takes place when the ClearAll method is called. Define the event handler if you want to take some actions after calling this method.

2.6.4.3 OnGetBiDiMode

Applies to
TQCustomLanguageSource component

Declaration
```
property OnGetBiDiMode;
```

Description

The *OnGetBiDiMode* event takes place when the GetBiDiMode method is called. Define the event handler if you want to take some actions after calling this method.

2.6.4.4 OnGetFontCharSet

Applies to
TQCustomLanguageSource component

Declaration
```
property OnGetFontCharSet;
```

Description

The *OnGetFontCharSet* event takes place when the GetFontCharSet method is called. Define the event handler if you want to take some actions after calling this method.

2.6.4.5 OnGetFormSection

Applies to
TQCustomLanguageSource component

Declaration
```
property OnGetFormSection;
```

Description

The *OnGetFormSection* event takes place when the GetFormSection method is called. Define the event handler if you want to take some actions after calling this method.

2.6.4.6 OnSetBiDiMode

Applies to
TQCustomLanguageSource component

Declaration
```
property OnSetBiDiMode: TQSetBiDiMode;
```

Description

The *OnSetBiDiMode* event takes place when the SetBiDiMode method is called. Define the event handler if you want to take some actions after calling this method.

2.6.4.7 OnSetFontCharSet

Applies to
TQCustomLanguageSource component

Declaration
```
property OnSetFontCharSet: TQSetFontCharSet;
```

Description

The *OnSetFontCharSet* event takes place when the SetFontCharSet method is called. Define the event handler if you want to take some actions after calling this method.

2.6.4.8 OnLanguageChanged

Applies to
TQCustomLanguageSource component

Declaration
```
property OnLanguageChanged: TNotifyEvent;
```

Description

The *OnLanguageChanged* event takes place after the active language is changed. It is invoked by the LanguageChanged method.

See also:
OnLanguageChanging event
LanguageChanged method

2.6.4.9 OnLanguageChanging

Applies to
TQCustomLanguageSource component

Declaration
```
property OnLanguageChanging: TQLangChangingEvent;
```

Description

The *OnLanguageChanging* event takes place before the active language is changed. It is invoked by the LanguageChanging method.

See also:
OnLanguageChanged event
LanguageChanging method

2.7 TQFileLanguageSource

2.7.1 TQFileLanguageSource Reference

Unit
QLocal

Description

The *TQFileLanguageSource* component is intended for working with the resources stored in file on the disc. It contains specific properties for working with files: DefaultFileExt and LanguageFile, and the OnGetFileName event which allows you to edit the LanguageFile property on receiving its value, e.g. add path to the filename, etc.

2.7.2 Properties

Run-time only Key properties

- DefaultFileExt
- LanguageFile

2.7.2.1 DefaultFileExt

Applies to
TQFileLanguageSource component

Declaration
```
property DefaultFileExt: string;
```

Description

The *DefultFileExt* property determines the default extension of the language file which contains the localized strings. The default property value is 'lng'.

See also:
LanguageFile property

2.7.2.2 LanguageFile

Applies to
TQFileLanguageSource component

Declaration
```
property LanguageFile[Index: integer]: string;
```

Description

The *LanguageFile* property is read-only. It is used to receive the name of the language file by its index.

See also:
DefaultFileExt property
OnGetFileName event

2.7.3 Events

- Key events
  - OnGetFileName

2.7.3.1 OnGetFileName

Applies to
TQFileLanguageSource component

Declaration
```
property OnGetFileName: TQLangFileNameEvent;
```

Description

The *OnGetFileName* event takes place before the LanguageFile property is set. Depending on the property value, you can process it in the way you need, e.g. add the full path to the filename, etc.

See also:
LanguageFile property
TQLangFileNameEvent type

2.8 TQDBLanguageSource

2.8.1 TQDBLanguageSource Reference

Unit
QLocalDBSource

Description

The *TQDBLanguageSource* component allows you to save and load localized strings to/from any instance of the TDataSet descendant.

2.8.2 Properties

Run-time only Key properties

- ActiveLanguage
- DataFields
- DataSet
- Languages
- OriginalName

2.8.2.1 ActiveLanguage

Applies to
TQCustomLanguageSource component

Declaration
```
property ActiveLanguage: integer;
```

Description

The *ActiveLanguage* property determines the currently active language. The default property value is -1, that means no language is selected (all instances of TQCustomLocalizer attached to the TQCustomLanguageSource use their own string sets). The value of this property is the language index of the Languages property.

See also:
LanguageName property
Languages property

2.8.2.2 DataFields

Applies to
TQDBLanguageSource component

Declaration
```
property DataFields: TQDBLanguageFields;
```

Description

The *DataFields* property defines the fields of the dataset, specified by the DataSet property which contain languages, sections, property names and values.

See also:
DataSet property
Languages property

2.8.2.3 DataSet

Applies to
TQDBLanguageSource component

Declaration
```
property DataSet: TDataSet;
```

Description

The *DataSet* property defines the instanse of the TDataSet descendant which contains the localized properties.

See also:
DataFields property
Languages property

2.8.2.4 Languages

Applies to
TQCustomLanguageSource component

Declaration
```
property Languages: TStrings;
```

Description

The *Languages* property contains all the available languages and the corresponding dataset fields with the localized strings. The correspondence between the languages and the dataset fields is set in the following format:
<LanguageName1>=<LanguageField1>, e.g. English=English.

See also:
LanguageValue property
LanguageName property

2.8.2.5 OriginalName

Applies to
TQCustomLanguageSource component

Declaration
```
property OriginalName: string;
```

Description

The *OriginalName* contains the name of the original program language (e.g. 'English'). This property doesn't influence anything except the default value of the ActiveLanguage property, so it is possible not to use it at all. The default property value is 'Original'.

See also:
LanguageName property

2.8.3 Methods

Key methods

BeginUpdate
Clear
ClearAll
CloseLanguage
EndUpdate
GetFormSection
GetReadOnly
LanguageChanged
LanguageChanging
LoadString
LoadStrings
Localize
OpenLanguage
Save
SaveString
SaveStrings
SetFormSection

2.8.3.1 LoadString

Applies to
TQCustomLanguageSource component

Declaration

```
function LoadString(const StringName, DefaultValue: string): string; virtual;
```

Description

The *LoadString* method is used in all the descendant components of TQCustomLanguageSource to load a string from the source language file.

See also:
LoadStrings method
SaveString method

2.8.3.2 Clear

Applies to
TQCustomLanguageSource component

Declaration
```
procedure Clear; virtual; abstract;
```

Description

The *Clear* method is used in all the descendant components of TQCustomLanguageSource to clear the current section of the language source. The current section is specified in the FormSection property. If the section is not specified then it clears all the source.

See also:
FormSection property
ClearFile method

2.8.3.3 ClearFile

Applies to
TQCustomLanguageSource component

Declaration
```
procedure ClearAll; virtual; abstract;
```

Description

The *ClearAll* method is used in all the descendant components of TQCustomLanguageSource to clear the contents of the language source.

See also:
Clear method

2.8.3.4 LoadStrings

Applies to
TQCustomLanguageSource component

Declaration
```
procedure LoadStrings(Strings: TStrings); virtual; abstract;
```

Description

The *LoadStrings* method is used in all the descendant components of TQCustomLanguageSource to load a number of strings (e.g. strings of the specified section) from the source language file.

See also:
LoadString method
SaveStrings method

2.8.3.5 SaveString

Applies to
TQCustomLanguageSource component

Declaration
```
procedure SaveString(const StringName, Value: string); virtual;
```

Description

The *SaveString* method is used in all the descendant components of TQCustomLanguageSource to save a string to the source language file.

See also:
SaveStrings method
LoadString method

2.8.3.6 SaveStrings

Applies to
TQCustomLanguageSource component

Declaration
```
procedure SaveStrings(Strings: TStrings); virtual;
```

Description

The *SaveStrings* method is used in all the descendant components of TQCustomLanguageSource to save a number of strings (e.g. strings corresponding to the current form) to the source language file.

See also:
SaveString method
LoadStrings method

2.8.4 Events

Key events

OnLanguageChanged
OnLanguageChanging

2.8.4.1 OnLanguageChanged

Applies to
TQCustomLanguageSource component

Declaration
```
property OnLanguageChanged: TNotifyEvent;
```

Description

The *OnLanguageChanged* event takes place after the active language is changed. It is invoked by the LanguageChanged method.

See also:
OnLanguageChanging event
LanguageChanged method

2.8.4.2 OnLanguageChanging

Applies to
TQCustomLanguageSource component

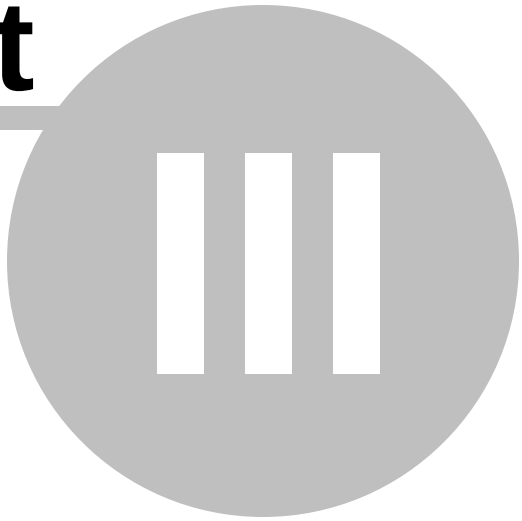
Declaration
```
property OnLanguageChanging: TQLangChangingEvent;
```

Description

The *OnLanguageChanging* event takes place before the active language is changed. It is invoked by the LanguageChanging method.

See also:
OnLanguageChanged event
LanguageChanging method

Part



3 How to...

3.1 Generate the template of the language file

To generate the template of the language file you should create the file using the TQLanguageSource component (use Language Source Editor to fasten this process). Then you should select components and their properties, and define the localized property values using the TQFormLocalizer component (use Form Localizer Editor to fasten this process).

See Language Source Editor and Form Localizer Editor for details.

See also:
How to manage the localization files
How to specify the components and properties to be localized
How to localize the current form

3.2 Manage the localization files

To manage the localization files you should use the TQLanguageSource component. You can add languages from the existing files or create new language files by clicking button  in the Language Source Editor. You can also remove languages from the language list (button  or modify language name and file (button  . To change the language used by the TQFormLocalizer as active select the language from the list and click button  .

See Language Source Editor and Form Localizer Editor for details.

See also:
How to generate the template of the language file
How to specify the components and properties to be localized
How to localize the current form

3.3 Specify the components and properties to be localized

To specify the components and properties to be localized you should use the TQFormLocalizer Editor component. Double-click the component instance to activate the Form Localizer Editor.
The list of all the owner form's components is available at the left of the editor window. Right-click the component to add it to the Excluded list ('Exclude' item) or, if it already excluded, remove it from the list ('Include' item). If you right-click an object, containing subobjects, you can also exclude all its subobjects or remove all the subobjects from the Excluded list ('Exclude all' and 'Include all' items).
The list of all the properties of the current component is available at the bottom of the window. Check the needed properties to add them to the grid above, where you can edit their string values for different languages.

See Form Localizer Editor for details.

See also:
How to generate the template of the language file
How to manage the localization files
How to localize the current form

3.4 Localize the current form

First of all you should create the language file or add the existing language file to the Languages property of the TQLanguageSource component (see How to manage the localization files). Then you should define set language you need in the ActiveLanguage property of the TQLanguageSource compoent. You can do this using the Language Source Editor. Place the TQFormLocalizer component to the form you want to localize and set the Source property to the TQLanguageSource component instance, created in advance. Specify the components and properties to be localized in the Form Localizer Editor and click the button 'Save'  . The localized property values from the active language file will be applied to the current form.

See Language Source Editor and Form Localizer Editor for details.

See also:
How to generate the template of the language file
How to manage the localization files
How to specify the components and properties to be localized

Part

4 Units

4.1 QFormLocal unit

The QFormLocal unit contains the TQFormLocalizer component which localizes the properties of the owner's form components.

Components
TQFormLocalizer

Types
TQSaveOptions
TQPropProcessedEvent
TQPropProcessingEvent

4.1.1 TQSaveOptions type

Unit
QFormLocal

Declaration
```
type TQSaveOptions = set of TQSaveOption;
```

Description

The TQSaveOptions type is a set of TQSaveOption properties. The following properties are available:

soSaveEmpty - if this property is included then empty property values will be also saved to the language file. The default property value is false.
soIgnoreDependencyIfEmpty - if this property is included then the dependencies of the empty property values will be ignored. The default property value is false.
soIncludeFrames - if this property is included then the property values of the owner's form frames will be also saved to the language file. The default property value is false.

See also:
SaveOptions property

4.1.2 TQPropProcessedEvent type

Unit
QFormLocal

Declaration
```
type TQPropProcessedEvent = procedure(Sender: TObject; Obj: TPersistent; const PropNa
```

Description

The TQPropProcessedEvent type is the type of the OnPropertyLocalized and OnPropertySaved events. This type is similar to the TQPropProcessingEvent type, but it has no Allow variable.

See also:
OnPropertyLocalized event
OnPropertySaved event
TQPropProcessingEvent type

4.1.3 TQPropProcessingEvent type

Unit
QFormLocal

Declaration
```
type TQPropProcessingEvent = procedure(Sender: TObject; Obj: TPersistent; const PropNa
string; var NewValue: string; var Allow: boolean); of object;
```

Description

The TQPropProcessingEvent type is the type of the OnPropertyLocalizing and OnPropertySaving events. Use the following variables to process this event:

Component - the form component, containing the localized property;
PropName - the name of the localized property;
Value - the property value;
Allow - if it is true, the property will be localized (saved). You can set it to false to forbid localizing (saving).

See also:
OnPropertyLocalizing event
OnPropertySaving event
TQPropProcessedEvent type

4.2 QLocal unit

The QLocal unit contains the definitions of TQCustomLocalizer and TQCustomLanguageSource classes. These two classes are the basic classes of the **Advanced Localizer for RAD Studio VCL** Component Suite.

Components
TQCustomLocalizer
TQCustomLanguageSource
TQFileLanguageSource

Types
TQLangChangingEvent
TQLangFileNameEvent

4.2.1 TQLanguageSettings object

Unit
QLocal

Description

The TQBLanguageSettings object defines the settings for each localization settings.

4.2.1.1 Properties

Run-time only Key properties

- BiDiMode
- FontCharSet

4.2.1.1.1 BiDiMode

Applies to
TQLanguageSettings object

Declaration
```
property BiDiMode: TBiDiMode;
```

Description

The BiDiMode specifies the bi-directional mode for localizing controls.

4.2.1.1.2 FontCharSet

Applies to
TQLanguageSettings object

Declaration
```
property FontCharSet: TFontCharSet;
```

Description

The FontCharSet defines the font character set for the localization language.

4.2.1.2 Methods

Key methods

- GetBiDiMode
- GetFontCharSet
- Assign
- SetBiDiMode
- SetFontCharSet

4.2.1.2.1 GetBiDiMode

Applies to
TQLanguageSettings object

Declaration
```
function GetBiDiMode: TBiDiMode; virtual;
```

Description

The GetBiDiMode method calls the OnGetBiDiMode event.

4.2.1.2.2 GetFontCharSet

Applies to
TQLanguageSettings object

Declaration
```
function GetFontCharSet: TFontCharSet; virtual;
```

Description

The GetFontCharSet method calls the OnGetFontCharSet event.

4.2.1.2.3 SetBiDiMode

Applies to
TQLanguageSettings object

Declaration
```
procedure SetBiDiMode(const Value: TBiDiMode); virtual;
```

Description

The SetBiDiMode method calls the OnSetBiDiMode event.

4.2.1.2.4 SetFontCharSet

Applies to
TQLanguageSettings object

Declaration
```
procedure SetFontCharSet(const Value: TFontCharSet); virtual;
```

Description

The SetFontCharSet method calls the OnSetFontCharSet event.

4.2.2 TQLangChangingEvent type

Unit
QLocal

Declaration
```
type TQLangChangingEvent = procedure(Sender: TObject; LanguageIndex: integer; var Allo
```

Description

The TQLanguageEvent type is the OnLanguageChanging event type for the TQCustomLocalizer and TQCustomLanguageSource components.

LanguageIndex variable defines the language to change.
If AllowChange is true, then the language will be changed. Set it to false to forbid changing.

See also:
OnLanguageChanging
OnLanguageChanging

4.2.3 TQLangFileNameEvent type

Unit
QLocal

Declaration
```
type TQLangFileNameEvent = procedure(Sender: TObject; var FileName: string); of object
```

Description

The TQLangFileNameEvent type is the OnGetFileName event type. Processing this event you can change the FileName value which will be the LanguageFile property value.

See also:
OnGetFileName event
LanguageFile property

4.3 QLocalDBSource unit

The QLocalDBSource unit contains the TQDBLanguageSource component which allows you to save/load localization strings to/from the dataset.

Components
TQDBLanguageSource

Objects
TQDBLanguageFields

4.3.1 TQDBLanguageFields object

Unit
QLocalDBSource

Description

The properties of this object define the value of the DataFields property of the TQDBLanguageSource component.

4.3.1.1 Properties

Run-time only Key properties

- LanguageField
- NameField
- SectionField
- ValueField

4.3.1.1.1 LanguageField

Applies to
TQDBLanguageFields object

Declaration
```
property LanguageField: string;
```

Description

Use this property to define the dataset field containing language names (e.g. English, French, etc.).

See also:
NameField
SectionField
ValueField

4.3.1.1.2 NameField

Applies to
TQDBLanguageFields object

Declaration
```
property NameField: string;
```

Description

Use this property to define the dataset field containing names of the localized form properties.

See also:
LanguageField
SectionField
ValueField

4.3.1.1.3 SectionField

Applies to
TQDBLanguageFields object

Declaration
```
property SectionField: string;
```

Description

Use this property to define the dataset field containing section names for the localized properties (e.g. [frmMain]).

See also:
LanguageField
NameField
ValueField

4.3.1.1.4 ValueField

Applies to
TQDBLanguageFields object

Declaration
```
property ValueField: string;
```

Description

Use this property to define the dataset field containing values of the localized properties.

See also:
LanguageField
NameField
SectionField

4.4 QLocalUserSource unit

Components
TQUserLanguageSource

4.5 QSource unit

The QSource unit contains the TQLanguageSource component which allows you to work with language files.

Components
TQLanguageSource

Part V

5 Appendix

5.1 Form Localizer Editor

Form Localizer Editor helps you to set the properties of the TQFormLocalizer component quickly. To activate this window, double-click the component instance or right-click it and choose 'Form Localizer Editor' in the popup menu.

Save - use this button to save all the changes you made in the Form Localizer Editor to the language file. Note that to write strings to the file, you should create an instance of the TQLanguageSource component and specify this instance in the Source property.

Reload - use this button to cancel all the changes you made since last save.

Add language - use this button to add a language to the list of available languages (Languages property of TQCustomLanguageSource component). In the dialog window you must specify the language source name (LanguageValue property) and the language name (LanguageName property). If no file exists with the filename you specified, it will be created. This button is available only if the Source property is specified.

Edit language - use this button to change the name and/or the source file of the selected language. To make this button available you should choose any language column (except 'Original') in the properties/languages grid.

Remove language - use this button to remove a language from the list of the available languages. The language file will not be deleted, but the language will not be used by TQFormLocalizer. To make this button available you should choose any language column (except 'Original') in the properties/languages grid.

At the left of the window there is an object tree of the current application form. Clicking an object displays all its properties, available for localizing, in the 'Property Name' list. Check the properties of the current object to add them to the grid above, where you can edit their string values for different languages.
Right-clicking an object you can add it to the Excluded list ('Exclude' item) or, if it already excluded, remove it from the list ('Include' item). If you right-click an object, containing subobjects, you can also exclude all its subobjects or remove all the subobjects from the Excluded list ('Exclude all' and 'Include all' items).

Set the localized values of the component properties in the proper grid cells. Choose 'Active language' from the drop-down list of available languages to apply the property values of this language file to the current form. Click 'Save' to save all the changes you made to the language file.

Options

On this tab you can specify the following Save Options:

Save empty values - if this option is checked then empty property values will be also saved to the language file.
Include frames - if this option is checked then the property values of the owner's form frames will be also saved to the language file.

Clear section before save - if this option is checked, then the strings associated with the instance of this class via Source property are cleared before save.

See also:
TQFormLocalizer component
Language Source Editor

5.2 Language Source Editor

Form Localizer Editor helps you to set the properties of TQLanguageSource component quickly. To activate this window, double-click the component instance or right-click it and choose 'Edit Language Source' in the popup menu.

 **Save** - use this button to save all the changes you made in the Language Source Editor.

 **Add language** - use this button to add a language to the list of available languages (Languages property). In the dialog window you must specify the language source name (LanguageValue property) and the language name (LanguageName property). If no file exists with the filename you specified, it will be created.

 **Edit language** - use this button to change the name and/or the source file of the selected language. To make this button available you should choose any language (except 'Original') in the language list.

 **Remove language** - use this button to remove a language from the list of the available languages. The language file will not be deleted, but the language will not be used by TQLanguageSource. To make this button available you should choose any language (except 'Original') in the language list.

 **Set As Active Language** - use this button to make the selected language active. Note that to apply the localized property values from the language file to the form components, you should create an instance of TQFormLocalizer and specify the current TQLanguageSource component in the Source property of TQFormLocalizer. If such instance exists, then the property values of the language you set as active will be applied to the form components after you click 'Save'.

See also:
TQLanguageSource component
Form Localizer Editor

Credits

Software Developers:

Michael Kuzevanov

Igor Brynckich

Dmitry Ziborov

Vadim Vinokur

Technical Writers:

Dmitry Doni

Semyon Slobodenyuk

Olga Ryabova

Cover Designer:

Tatyana Makurova

Translators:

Anna Shulkina

Sergey Fominykh

Team Coordinators:

Alexey Butalov

Alexander Chelyadin

Roman Tkachenko