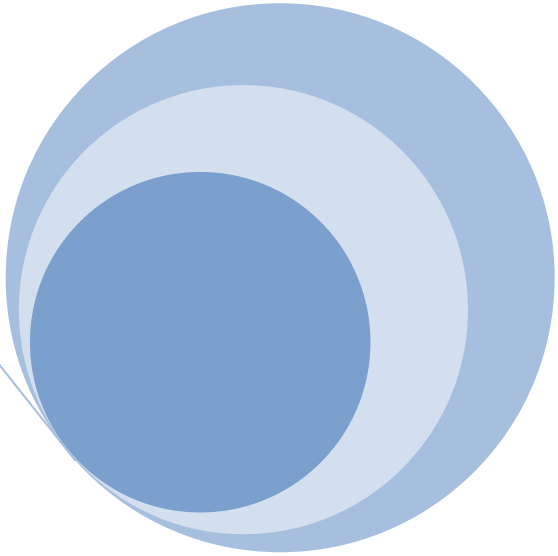
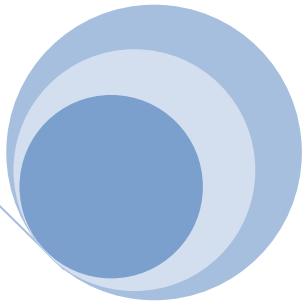
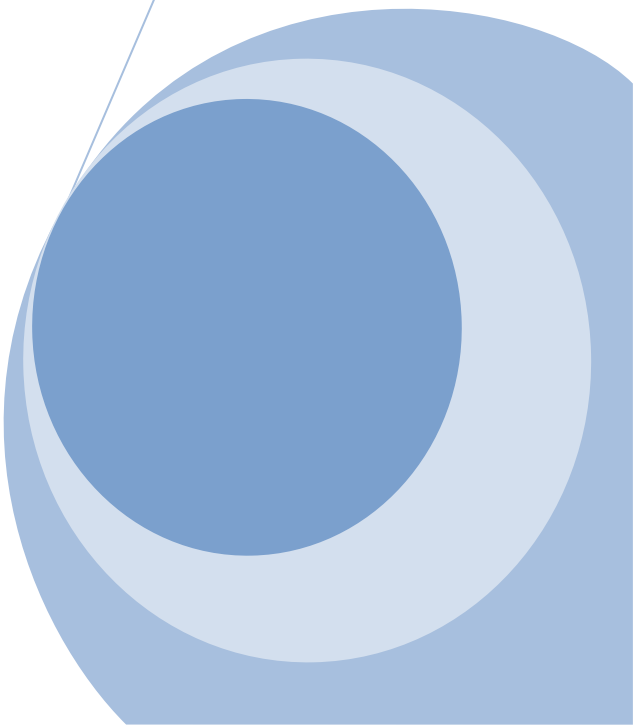




CLEVERFILTER 3.0

User Manual

Devrace



Contents

Introduction

Very often the end users of your programs want to have a tool for additional processing of data. They want to have an easy to use users' filter and they want to control the data processing. By building CleverFilter into your applications you'll give the users of your programs an opportunity to control the data.

CleverFitler allows you to apply additional conditions to a ready query. The users don't waste time learning the inner structure of the database, and you only need to select the fields accessible in the user filter in design time.

By using CleverFilter in your applications you receive a number of serious advantages, e.g.:

- **Support of different database formats and SQL servers** using adapters. The current version is supplied with adapters for FIBPlus, ADO, BDE, IBX, dbExpress, NexusDB, ODAC, UniDAC and IbDAC.
- CleverFilter modifies the WHERE condition in the query, which allows you to **process the data on the server side** and reduce the load on the client computers.
- **Support of parametric queries.** This way, when the field values in the filter are changed, the server doesn't need to prepare the query to execution once again.
- **Support of lookup fields**.
- **Support of the basic field types:** string, numerical, date and time fields, emulating of logical fields by means of using value lists.
- **Using special visual elements in the filter** for user's convenience: possibility to set value ranges (between), lists of possible values (radio button, check list, list box), drop-down calendars for date type fields.
- Simple and intuitive for users **visualization of complex conditions with OR and AND**.
- Possibility to insert filter conditions anywhere in the main query.
- A mechanism of choosing fields accessed by users in the filter is available to the programmers.
- Possibility to use sets of conditions and sorting orders prepared in advance.
- Possibility to load the filter scheme from outside sources (files or databases), which enables you to change dialog parameters and provide different selection opportunities to different users in runtime without recompiling the project.

User Interface

Main dialog window of the filter

The main dialog window of the filter is shown on the Fig. 1. The main part of the window is occupied by pages and elements the filtering will be made by. Elements on the same page are grouped together using the AND condition.

The dialog window can contain several pages with elements. Pages are connected between themselves using the OR condition. Buttons  and  are used to add or delete additional pages and located in the bottom left corner of the dialog window. The primary page cannot be deleted.

The OK button confirms the user's selection and the Cancel button cancels it.

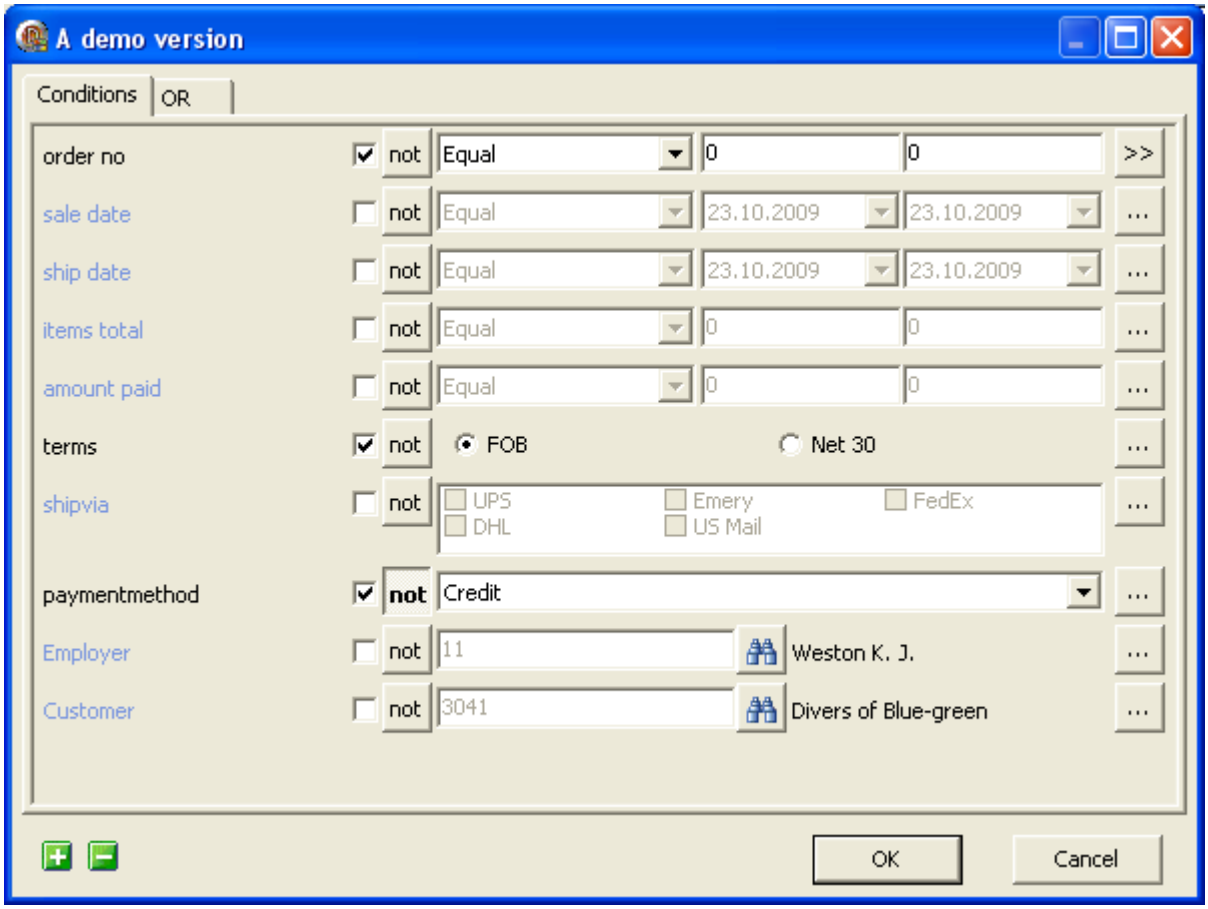


Fig. 1. The main dialog window of the filter

The element structure

Each element consists of 5 components:

- Heading
- Element selection checkbox
- The NOT button to create negation (inversion) of a condition (the logical "not" operation)
- The value entering area. Depends on the element's type
- The button that creates additional conditions for the element.

The heading can be located to the left from the selection checkbox, as well as to the right from it. IN the specified area the heading can be left-aligned, right-aligned or centered, which gives us considerable flexibility as far as the customization of the window's look and feel goes.

The NOT button inverts the filtering condition to the opposite. ON the Fig. 1. the NOT button is pressed for the paymentmethod element, and the condition thus created will be interpreted as paymentmethod <> 'Credit'. If the NOT button weren't pressed, the condition would be interpreted as paymentmethod = 'Credit'.

The value entering area will be described in detail below for each element separately.

The button the created additional conditions (Fig. 2.) allows the users to quickly and easily add OR conditions individually for each element, which, together with the pages' Ors, allows the users to create quite complex filtering conditions without writing any code!

If the element has additional conditions, the button looks this way:  ; otherwise it looks like this:  .

New conditions are added using the  button, and deleted using the  button. Additional conditions are grouped with each other and with the main element using the OR condition.

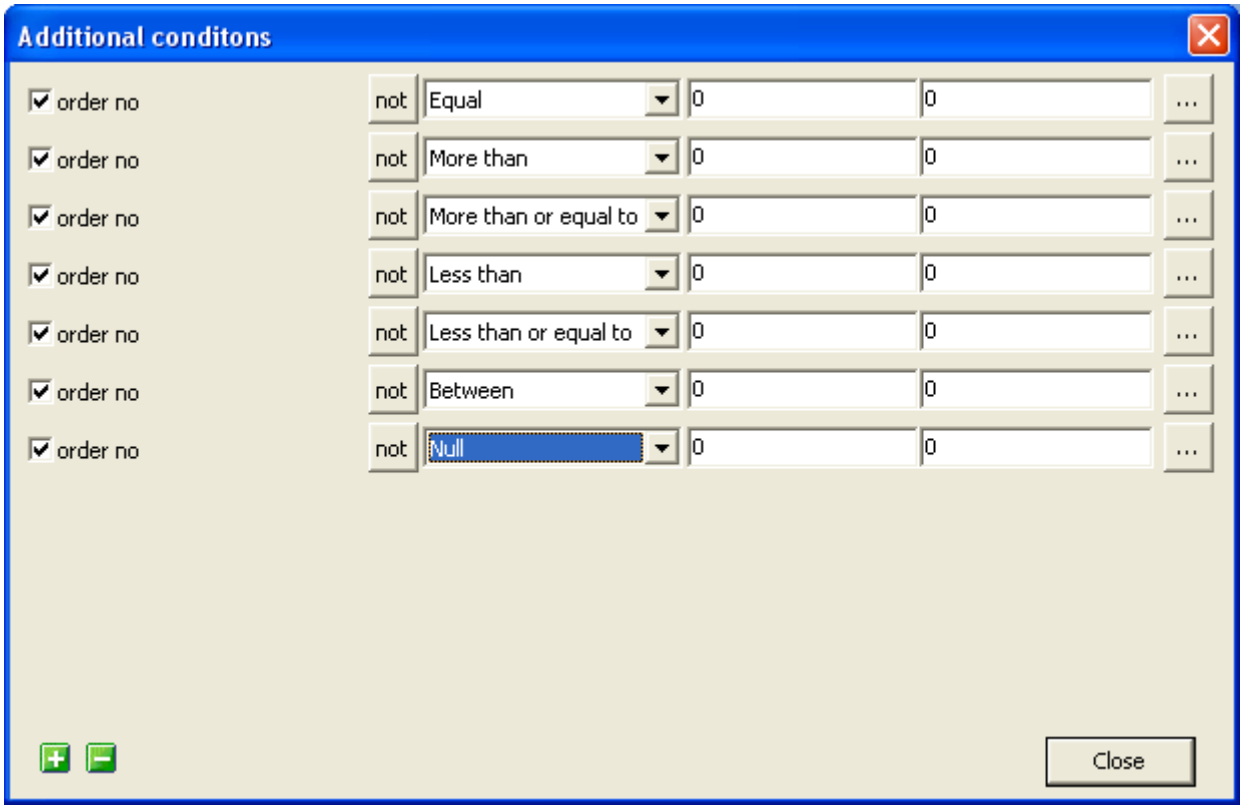


Fig. 2. Additional conditions for elements

The String element

This element is meant for entering conditions on textual data like First/Middle/Last names or the address.

The element supports the following operations:: Equal, Starts with, Ends with, Contains and Null.

- Equal –the field should exactly contain the entered value.
- Starts with – the field's contents should start with the entered value.
- Ends with – the field's contents should end with the entered value.
- Contains – the field's contents should have the entered value somewhere inside them.
- Null – the field should contain the Null value.

The element supports both case-sensitive and case-insensitive search.

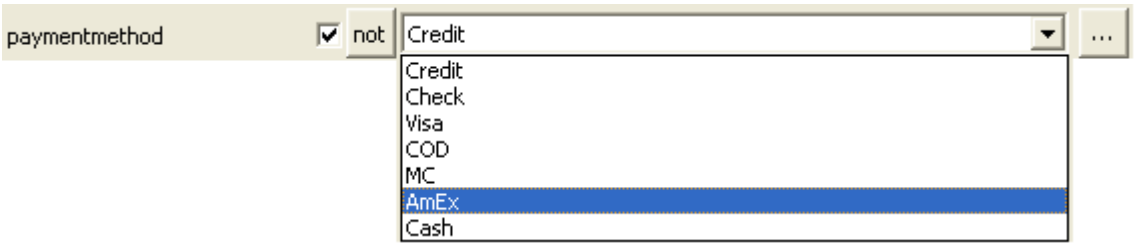
The Integer, Float, Date elements

These elements are meant to create conditions for numerical fields (both integer and real) and for Date type fields.

They support the following operations: Equal, More than, More than or equal to, Less than, Less than or equal to, Between and Null.

- Equal – the field should contain the exact value from the first entry field.
- More than – the field's contents should be greater than the entered value in the first entry field.
- More than or equal to – the field's contents should be greater than or equal to the entered value from the first entry field.
- Less than – the field's contents should be less than the entered value from the first entry field.
- Less than or equal to – the field's contents should be less than or equal to the entered value from the first entry field.
- Between – the field's contents should be between the entered values. The value from the second entry field is used only for this operation. For all other operations it's ignored.
- Null – the field should contain the Null value.

List and DBList elements



These elements are for selecting one value from the list. The difference between the two elements is this: List forms the list of values beforehand, and DBList draws them out of a chosen Dataset.

RadioGroup and DBRadioGroup elements

These elements are similar to List/DBList elements. The difference is in the way they represent the list: as a radio group.

CheckListBox and DBCheckListBox

Unline the List and RadioGroupelements, which allow the user to choose only one value, the CheckListBox element allows choosing several values.

Lookup element

This element is ideally suitable for the fields of the source query, which are actually external keys on the master-table.

The value can be entered in two ways:

1) By entering the primary key of the master-table into the entry field (if it's known); certain information about the record (e.g. first/middle/last names) will be displayed on the right.
2) By clicking the search button:  . This method allows quick and easy (and most importantly, visual) choice of the necessary value. The way it works reminds us about the way the filter works, except a page will be added for displaying the search results and selecting the desired record (see Fig. 3). The search conditions are entered the usual way, after which the desired record can be selected on the Result tab. After selecting the record, the Ok button will be enabled to confirm the final selection, see Fig. 4.

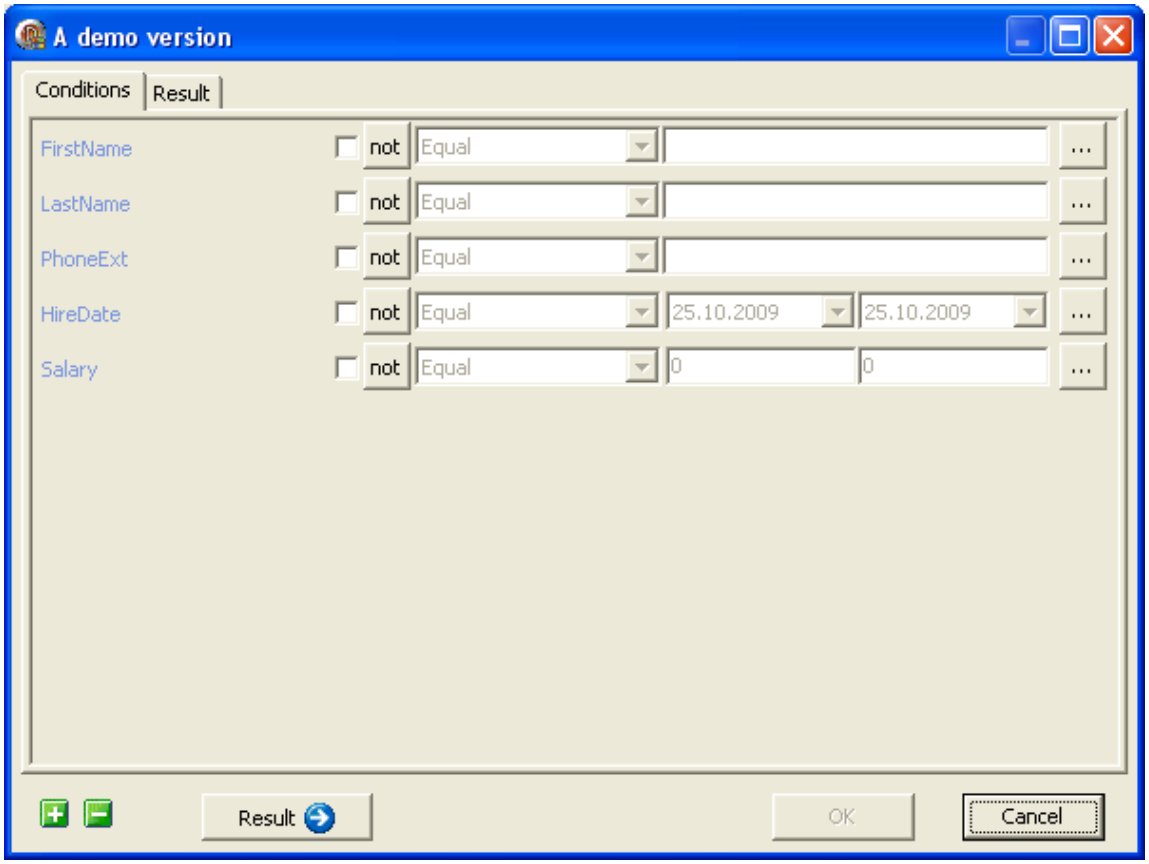


Fig. 3. How the Lookup element works

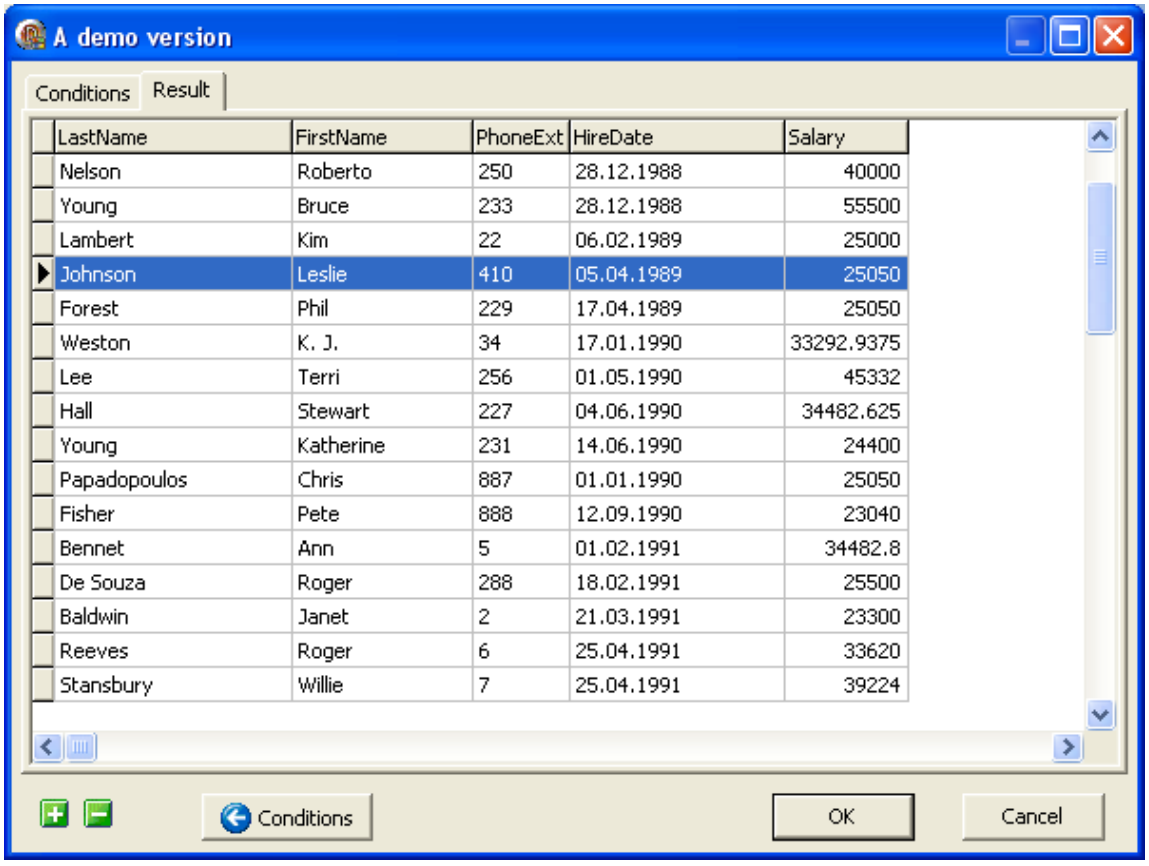


Fig. 4. Selecting a record

The sorting window

The filtering component also supports sorting. The choice of fields for sorting (out of the total number of available fields formed on the stage of designing the application) and the sorting order are done visually without writing a single line of code.

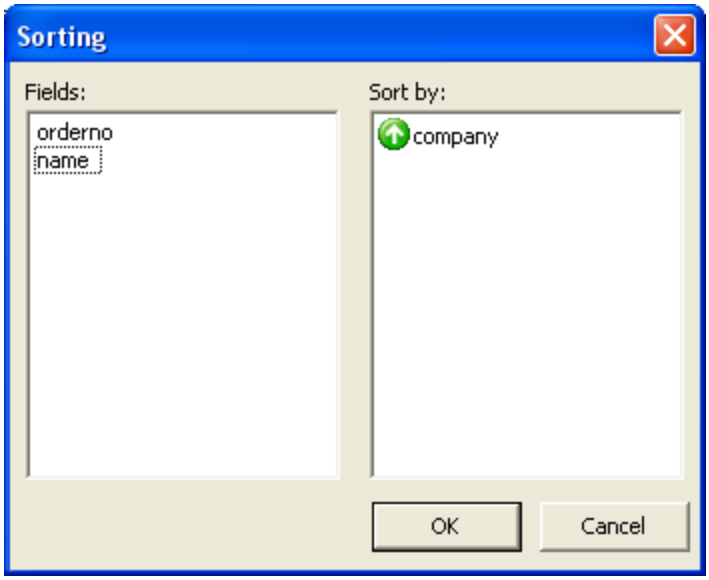


Fig. 5. Sorting window

The list of fields, by which sorting is possible, is displayed in the right Sort by window. Moving fields from one list to another is done using the mouse. It's only necessary to drag and drop the desired field.

To change the order of fields just move the field to the desired location. You'll need to double-click the field.

Components, classes and accompanying types

TcfFilter component

TcfFilter is the main component that is responsible for all the data filtering and sorting functionality.

Methods

function Execute: Boolean;	Displays the filter'a dialog window. Returns True if the user has confirmed the choice. Otherwise, returns False.
function SortExecute: Boolean;	Displays the sorting dialog window. Returns True if the user has confirmed the choice. Otherwise, returns False.
procedure Where;	Generates the filtering sql-code using the conditions selected by the user. The sql code is inserted into the place denoted by markers from the Marker property. If the properties AutoClose/AutoOpen are set True, the dataset will be opened/closed automatically.
procedure ApplyFilter;	A synonym of the Where method.
procedure Refresh;	Displays the filter's dialog window and, if the user has confirmed the selection, applies the filter.
procedure SortAndRefresh;	Displays the filtering window and, if the user has confirmed the selection, sorts the data.
procedure LoadShema;	Executes custom loading of the filter's scheme in run-time. The loading is possible from all sources. All the logic of the loading should be implemented in the OnLoadShema and OnLoadItem events.

Properties

❖ property DataSource: TDataSource	The dataset with which the component works.
❖ property AutoOpen: Boolean	Indicates whether to open the dataset automatically after changing the query or not.
❖ property AutoClose: Boolean	Indicates whether to close the dataset automatically before changing the quesry or not.
❖ property OnBeforeWhere: TcfFilterWhereEvent	The event takes place before forming the sql code of the condition.
❖ property OnAfterWhere: TcfFilterWhereEvent	The event takes place after forming the sql code of the condition. Together with OnBeforeWhere it allows carrying out any actions not accessible by the component, e.g., changing the text of bound queries or correcting the resulting query.
❖ property SortFields: TStrings	A list of sorting fields in the format of a <Caption>=<Field> pair. <Field> is given in the format of an 'order by' list.
❖ property Marker: TcfFilterMarker	Sets markers (syntax of a comment or a valid sql expression) to define areas in which the component changes the code.
❖ property EnabledComents: Boolean	Defines whether comments will be inserted while adding conditions to the SQL code. These comments make it easier to define which page of the filter and which element has generated the condition. This feature can be useful for analyzing the formed SQL code in the onAfterWhere event. If the database management system doesn't support comments in sql queries, this property allows the user to turn off the generation of special comments.
property Sort: TcfSort	Pointer to an inner component responsible for the sorting

functionality.

property EnableWhere: Boolean — Defines whether the sql code for filtering will be formed. Can be used for temporarily turning off the filtering functionality.

property EnableOrderBy: Boolean — Defines whether the sql code for sorting will be formed. Can be used for temporarily turning off the sorting functionality.

❖ property Items: TcfFilterCollection — The list of the filter's elements. Defines the structure of the filter.

property Storage: TcfStorage — Gives access to the internal data array. Allows reading/setting of any element's value.

❖ property Section: AnsiString — A section's name in the file of the filter's settings. Defines whether it's necessary to save user's settings in an external xml file and the section where the information will be saved. If there's no value defined, the data won't be saved.

❖ property Caption: string — Defines the caption of the filter's dialog window.

❖ property Width: Integer — Defines the width of the filter's dialog window. The default value is 600 pixels.

❖ property MinHeight: Integer — Sets the minimum height of the filter's dialog window. The final height would be defined as the largest of the two numbers: the minimum height and the sum of the heights of all elements plus the height of the working area.

❖ property OnLoadShema: TcfFilterLoadSheMaEvent — The event is meant for loading the filter's scheme from external sources. In the handler it will be necessary to read the list of the element types, which define the structure of the table. Every element will be read in the OnLoadItem event.

❖ property OnLoadItem: TcfFilterLoadItemEvent — The event is meant for loading the properties of each filter's element. The type of every element is set in the OnLoadShema event.

property Settings: AnsiString — Presents the internal state of the filter as a string (xml format), which allows creating the sets of conditions beforehand and loading them when the program is running.

❖ - published properties

TcfSubFilter component

TcfSubFilter component realizes the functionality pof the Lookup element, namely allows visual selection of the record for further work. The component can be used separately, as it's not connected directly with other components.

The component had two restrictions:

- It doesn't publish the SortFields property, so, the sorting is hard-coded into the query;
- The first field in the query has to be the record's ID, which will become the result of the component's work.

Methods

function Execute: Boolean; — Display's the filter's dialog window. Returns True if the user has confirmed the choice. Otherwise, returns False.

procedure LoadShema; — Executes custom loading of the filter's scheme in runtime. The loading is possible from all sources. The logic of loading has to be realized in the OnLoadShema and OnLoadItem events.

Note. Though TcfSubFilter is a descendant of TcfFilter, the rest of the methods are available, but make no sense.

Properties

❖ property DataSource: TDataSource	The dataset with which the component works.
❖ property AutoOpen: Boolean	Indicates whether to open the dataset automatically after changing the query or not.
❖ property AutoClose: Boolean	Indicates whether to close the dataset automatically before changing the quesry or not.
❖ property OnBeforeWhere: TcfFilterWhereEvent	The event takes place before forming the sql code of the condition.
❖ property OnAfterWhere: TcfFilterWhereEvent	The event takes place after forming the sql code of the condition. Together with OnBeforeWhere it allows carrying out any actions not accessible by the component, e.g., changing the text of bound queries or correcting the resulting query.
❖ property Marker: TcfFilterMarker	Sets markers (syntax of a comment or a valid sql expression) to define areas in which the component changes the code.
❖ property EnabledComents: Boolean	Defines whether comments will be inserted while adding conditions to the SQL code. These comments make it easier to define which page of the filter and which element has generated the condition. This feature can be useful for analyzing the formed SQL code in the onAfterWhere event. If the database management system doesn't support comments in sql queries, this property allows the user to turn off the generation of special comments.
property Sort: TcfSort	Pointer to an inner component responsible for the sorting functionality.
property EnableWhere: Boolean	Defines whether the sql code for filtering will be formed. Can be used for temporarily turning off the filtering functionality.
property EnableOrderBy: Boolean	Defines whether the sql code for sorting will be formed. Can be used for temporarily turning off the sorting functionality.
❖ property Items: TcfFilterCollection	The list of the filter's elements. Defines the structure of the filter.
property Storage: TcfStorage	Gives access to the internal data array. Allows reading/setting of any element's value.
❖ property Section: AnsiString	A section's name in the file of the filter's settings. Defines whether it's necessary to save user's settings in an external xml file and the section where the information will be saved. If there's no value defined, the data won't be saved.
❖ property Caption: string	Defines the caption of the filter's dialog window.
❖ property Width: Integer	Defines the width of the filter's dialog window. The default value is 600 pixels.
❖ property MinHeight: Integer	Sets the minimum height of the filter's dialog window. The final height would be defined as the largest of the two numbers: the minimum height and the sum of the heights of all elements plus the height of the working area.
❖ property OnLoadShema: TcfFilterLoadShemaE-	The event is meant for loading the filter's scheme from

vent	external sources. In the handler it will be necessary to read the list of the element types, which define the structure of the table. Every element will be read in the OnLoadItem event.
❖ property OnLoadItem: TcfFilterLoadItemEvent	The event is meant for loading the properties of each filter's element. The type of every element is set in the OnLoadShema event.
property Settings: AnsiString	Presents the internal state of the filter as a string (xml format), which allows creating the sets of conditions beforehand and loading them when the program is running.
❖ property ID: Variant	If the user has selected a record, it contains the value of the first field of the selected record (usually its ID).
❖ property Columns: TStrings	Sets the caption for the grid columns on the result page. It's also possible to set the width in pixels for each column. The width value is separated from the caption by a *semicolon* (;)
❖ - published properties	

TcfSimpleFilter component

It often happens that it's necessary to filter the query result by one or two fields, and it would be far more convenient to the user to enter the conditions directly on the main form rather than in an additional window. For such cases the TcfSimpleFilter component is intended. Each element of the filter can generate only one predefined condition. The elements of the graphic interface for entering conditions are no longer generated by the component, and you have complete freedom regarding their placement and visual look.

Methods

function SortExecute: Boolean;	Display's the sorting dialog window. Returns True if the user has confirmed the choice. Otherwise, returns False.
procedure Where;	Generates the filtering sql-code using the conditions selected by the user. The sql code is inserted into the place denoted by markers from the Marker property. If the properties AutoClose/AutoOpen are set True, the dataset will be closed/opened automatically.
procedure ApplyFilter;	A synonym of the Where method.
procedure SortAndRefresh;	Displays the filter's dialog window and, if the user has confirmed the selection, applies the filter.

Properties

❖ property DataSource: TDataSource	The dataset with which the component works.
❖ property AutoOpen: Boolean	Indicates whether to open the dataset automatically after changing the query or not.
❖ property AutoClose: Boolean	Indicates whether to close the dataset automatically before changing the quesry or not.
❖ property OnBeforeWhere: TcfFilterWhereEvent	The event takes place before forming the sql code of the condition.
❖ property OnAfterWhere: TcfFilterWhereEvent	The event takes place after forming the sql code of the condition. Together with OnBeforeWhere it allows carrying out any actions not accessible by the component, e.g., changing the text of bound queries or correcting the resulting query.
❖ property SortFields: TStrings	A list of sorting fields in the format of a <Caption>=<Field> pair. <Field> is given in the format of an 'order by' list.

❖ property Marker: TcfFilterMarker	Sets markers (syntax of a comment or a valid sql expression) to define areas in which the component changes the code.
❖ property EnabledComents: Boolean	Defines whether comments will be inserted while adding conditions to the SQL code. These comments make it easier to define which page of the filter and which element has generated the condition. This feature can be useful for analyzing the formed SQL code in the onAfterWhere event. If the database management system doesn't support comments in sql queries, this property allows the user to turn off the generation of special comments.
property Sort: TcfSort	Pointer to an inner component responsible for the sorting functionality.
property EnableWhere: Boolean	Defines whether the sql code for filtering will be formed. Can be used for temporarily turning off the filtering functionality.
property EnableOrderBy: Boolean	Defines whether the sql code for sorting will be formed. Can be used for temporarily turning off the sorting functionality.
❖ property Items: TcfSimpleFilterCollection	The list of the filter's elements. Defines the structure of the filter.
❖ - published properties	

TcfFilterMarker class

The TcfFilterMarker class defines the so-called markers, which define the areas where the filter works. Markers are created in the form of comments, and if the Database Management System doesn't support comments, it's necessary to create unique valid expressions.

Properties

❖ property StartFilter: Ansistring	Defines the marker of the beginning of the area, into which the filter will insert the conditions of the "where" section of the query. By default, '/*filter*/'.
❖ property EndFilter: Ansistring	Defines the marker of the end of the area, into which the filter will insert the conditions of the "where" section of the query. By default, '/*end filter*/'.
❖ property Order: Ansistring	Defines the place where the filter will insert the previously generated "order by" clause.
❖ property ContinuesStart: Ansistring	Defines the marker of the beginning of the area, into which the filter will insert the conditions of the "where" section of the query. By default, '/*filter+*/'. This marker is used in the queries that already contain conditions in **where.**
❖ - published properties	

There can be several pairs of markers StartFilter/EndFilter or ContinueStart/EndFilter.

For a StartFilter/EndFilter pair the query should not contain the **where** keyword, as the filter will insert it automatically immediately after the StartFilter marker. For the ContinuesStart marker, an **and** operator will be inserted.

TcfFilterCollection class

The TcfFilterCollection class is a collection of the filter elements. The main difference between this collection and the standard one is the fact that TcfFilterCollection can store elements of different types, which descend from TCollectionItem. And, unlike with the Fields collection, no components will

be created. It's necessary to pay attention to the fact that the standard collection editor will work incorrectly.

TcfSimpleFilterCollection class

Standard collection of the filter's elements of the TcfSimpleFilter component.

TcfFilterItem class

The basis class of the filter's element; defines behaviour, which is the same for all element types. The TcfFilterItem class is meant for determining the structure of a page in design-time. In run-time the real storage structure is created based on the scheme and the TcfFilterItemStorage class. Access to these structures can be gained in run-time via the Strorage property. For displaying the element graphically on the pages of the dialog window there's the special hierarchy of mediator-type classes: TcfFilterItemPanel.

Properties

property GUID: string	Unique element identifier. Used for internal purposes to identify an element.
property Height: Integer	The element's height in pixels. By default, 29.
property CaseInsensitive: Boolean	Defines if the search for strings will be case sensitive (True) or not (False). By default, False.
❖ property FieldName: string	Defines the name of the field or the expression for which the condition will be generated. A sub-query can act as the expression!
❖ property Caption: string	Defines the caption of the element.
❖ property CaptionWidth: Integer	Sets the width of the caption in pixels. By default, 150.
❖ property BevelInner: TPanelBevel	Defines the type of the inner border around the element.
❖ property BevelOuter: TPanelBevel	Defines the type of the outer border around the element. Together with the BevelInner allows visual highlighting of the element.
❖ property Alignment: TAlignment	Defines the alignment of the caption relating to its area – right, left or center.
❖ property DirectValue: Boolean	Determines if during forming the condition a user-defined value will be inserted (True), or the value will be passed via a parameter (False), which increases the performance of the Database Management System when the searches are of the same type.
❖ property FieldType: TFieldType	Defines the type of the field in the query. For simple fields is defined automatically, but for expressions it has to be specified manually.
❖ property Enabled: Boolean	Defines if the element will be accessible to the user (True) or not (False). By default, True.
❖ property Visible: Boolean	Defines if the element will be displayed in the dialog window (True) or not (False). By default, True.
❖ property ActivatePosition: TActivatePosition	Defines the location of the caption in relation to the selection checkbox.
❖ property UpperFunc: string	Defines the sql function of making the capitalization of strings consistent. Used when CaseInsensitive = True. By default, 'upper'.
❖ property UsingDoubleQuotes: Boolean	Defines whether the name of the field will be in double quotes (True) or not (False). By default, False
❖ property OnWhere: TcfFilterItemWhereEvent	This event allows redefining the standard procedure of generating sql-code for an element and writing custom sql-code for conditions, as complex as necessary, which considerably enhances the flexibility of the component.
❖ - published properties	

Methods

function GetFilterClass: TcfFilterItemStorageClass	Returns the pointer of the mediator-type class meant for storing data.

TcfFilterItemStorage class

It's the basis class for storing the user's choice and generating the sql code of the conditions. It can also read settings from a file and write setting into a file. Bound to the corresponding TcfFilterItemPanel class responsible for displaying the element.

Properties

VisualElement: TcfFilterItemPanel	Pointer to the element responsible for the visualization of the element.
property GUID: string	Unique element identifier. Used for internal purposes to identify an element in the file of settings.
property FieldName: string	Defines the name of the field or the expression for which the condition will be generated. A sub-query can act as the expression!
property Caption: string	Defines the caption of the element.
property CaptionWidth: Integer	Sets the width of the element in pixels.
property Height: Integer	Sets the height of the element in pixels.
property BevelInner: TPanelBevel	Defines the type of the inner border around the element.
property BevelOuter: TPanelBevel	Defines the type of the outer border around the element. Together with the BevelInner allows visual highlighting of the element.
property Alignment: TAlignment	Defines the alignment of the caption relating to its area – right, left or center.
property DirectValue: Boolean	Determines if during forming the condition a user-defined value will be inserted (True), or the value will be passed via a parameter (False), which increases the performance of the Database Management System when the searches are of the same type.
property FieldType: TFieldType	Defines the type of the field in the query. For simple fields is defined automatically, but for expressions it has to be specified manually.
property Enabled: Boolean	Defines if the element will be accessible to the user (True) or not (False). By default, True.
property Visible: Boolean	Defines if the element will be displayed in the dialog window (True) or not (False). By default, True.
property ActivatePosition: TActivatePosition	Defines the location of the caption in relation to the selection checkbox.
property UpperFunc: string	Defines the sql function of making the capitalization of strings consistent. Used when CaseInsensitive = True. By default, 'upper'.
property UsingDoubleQuotes: Boolean	Defines whether the name of the field will be in double quotes (True) or not (False). By default, False
property OnWhere: TcfFilterItemWhereEvent	This event allows redefining the standard procedure of generating sql-code for an element and writing custom sql-code for conditions, as complex as necessary, which considerably enhances the flexibility of the component.
property Checked: Boolean	Defines whether the user has selected an element or not.
property Down: Boolean	Defines the condition of the NOT button. True if the button is pressed.

Methods

procedure AssignEx(Data: TcfFilterItem)	Copies the design-time settings from the corresponding TFilterItem element.
procedure CreateVisual(Page: TWinControl)	Creates the visual image of the element.

procedure DoWhere(SQL: TStrings; Params: TParams; PageNum, LineNum, ItemNum: Integer)	Procedure of the standard generation of conditions. SQL – contains the sql-code of the generated conditions. Params – contains parameters of the conditions. FieldName – contains the name of the field (or expression), to which the condition is applied. PageNum – contains the number of the page that has triggered the event. LineNum – contains the number of the element on the page that has triggered the event. ItemNum – contains the number of the additional element that has triggered the event. If the element is located on one of the pages, the number equals 0. If the element is additional, the property contains its index (starting from 1).
function GetParam1: Variant	Returns the value of the first parameter. Applies to all element types.
function GetParam2: Variant	Returns the value of the second parameter. Applies to elements that allow two parameters (numbers and dates).
procedure GetVisualState	Reads user's settings from VisualElement (whether the element is activated or not, whether the NOT button is pressed or not, the kind of the operation (if applicable), parameters' values).
procedure LoadData(Root: IXmlNode)	Reads the condition of the element from the xml-file with settings.
procedure SaveData(Root: IXmlNode)	Writes the current condition into the xml-file with settings.
procedure SetDefaultValues	Sets default values (if there is no file with stored settings).
procedure SetVisualState;	Sets the visual condition of the element (whether the element is activated or not, whether the NOT button is pressed or not, the kind of the operation, parameters' values).
procedure CancelUpdates	Reverts the settings to the previous values.
function Where(PageNum, LineNum, ItemNum: Integer; q: TcfFilterQuery): Boolean;	Generates conditions. The conditions are generated only if the element is activated (Checked = True), available for changing (Enabled = True) and displays on the page (Visible = True). If the OnWhere handler is defined, its code is executed; otherwise the standard DoWhere handler is called. PageNum – the number of the page containing the element. LineNum – the number of the element on the page. ItemNum – the number of the element in the series (there's at least one element in the series on the main page).

TcfFilterItemCustomList class

This is an intermediary class for list-type elements. Defines common behaviour for all lists regardless of their visual format.

Properties

property Columns: Integer	Sets the number of columns to display the list of values.
property Items: TStrings	Defines the list of values, from which the user will select.
property KeyItems: TStrings	Defines the list of value parameters.

	If the list is not empty and the number of its elements is the same as the number of elements in Items, the query parameters' values for generating the sql-code will be taken from this list.
property OnKeyList: TcfFilterItemListEvent	The event allows filling in the KeyItems list by the program in run time.
property OnList: TcfFilterItemListEvent	The event allows filling in the Items list by the program in run time.

❖ - published properties

See the description of inherited properties and methods in TcfFilterItem

TcfFilterItemCustomListStorage class

It's an intermediary class for storing settings and generating the sql code for all list-type elements.

Properties

property Columns: Integer	Sets the number of columns to display the list of values.
property Items: TStrings	Defines the list of values, from which the user will select.
property KeyItems: TStrings	Defines the list of value parameters. If the list is not empty and the number of its elements is the same as the number of elements in Items, the query parameters' values for generating the sql-code will be taken from this list.
property OnKeyList: TcfFilterItemListEvent	The event allows filling in the KeyItems list by the program in run time.
property OnList: TcfFilterItemListEvent	The event allows filling in the Items list by the program in run time.
property ItemIndex: Integer	Contains the number of the selected element.

See the description of inherited properties and methods in TcfFilterItemStorage

TcfFilterItemList class

This is an element of the List filter, which is a list with only one selection of values.

The list of the displayed values is defined in Items. If KeyItems has set parameters and the number of elements in Items and KeyItems is the same, the values of parameters will be taken from KeyItems to generate conditions; otherwise they will be taken from Items.

For example, if 1 corresponds to the value "other" in the table, 2 corresponds to "software" and 3 to "hardware", then Items should contain

other
software
hardware
and KeyItems -
1
2
3

If the handlers of the OnKeyList/OnList events are defined, these lists can be generated in runtime.

Publishes these properties: Items, KeyItems, OnList, OnKeyList. For the description of the rest of the properties see TcfFilterItemCustomList.

TcfFilterItemListStorage class

This class realizes the work of the List element. For the description of all properties and methods see TcfFilterItemCustomListStorage.

TcfFilterItemRadioGroup class

It's an element of the RadioGroup filter, which is a list with a multiple selection of values.

The list of displayed values is definted in Items. If KeyItems has set parameters and the number of elements in Items and KeyItems is the same, the values of parameters will be taken from KeyItems to generate conditions; otherwise they will be taken from Items.

For example, if 1 corresponds to the value "y" in the table and 2 corresponds to "n", then Items should contain

y
n
and KeyItems -
1
2

If the handlers of the OnKeyList/OnList events are defined, these lists can be generated in run-time.

Columns sets the number of columns in which the values will be placed. Height sets the height of the element.

Publishes the following properties: Items, KeyItems, OnList, OnKeyList, Height, Columns. For the description of the rest of the properties see TcfFilterItemCustomList.

Класс TcfFilterItemRadioGroupStorage class

This class realizes the work of the RadioGroup element. For the description of all properties and methods see TcfFilterItemCustomListStorage.

TcfFilterItemCheckListBox class

It's an element of the CheckListBox filter meant to fine-tune the scheme of the filter in design time.

The element is a list with multiple selection of values.

The list of displayed values is definted in Items. If KeyItems has set parameters and the number of elements in Items and KeyItems is the same, the values of parameters will be taken from KeyItems to generate conditions; otherwise they will be taken from Items.

For example, if 1 corresponds to the value "shipped" in the table, 2 corresponds to "open" and 3 to "waiting", then Items should contain

shipped
open
waiting
and KeyItems -
1
2

3

If the handlers of the OnKeyList/OnList events are defined, these lists can be generated in run-time.

Columns sets the number of columns in which the values will be placed. Height sets the height of the element.

Publishes the following properties: Items, KeyItems, OnList, OnKeyList, Height, Columns. For the description of the rest of the properties see TcfFilterItemCustomList.

TcfFilterItemCheckListBoxStorage class

This class realizes the work of the CheckListBox element. For the description of all properties and methods see TcfFilterItemCustomListStorage.

TcfFilterItemEdit class

It's an element of the String filter meant to fine-tune the scheme of the filter in design time. The element allows filtering data by String-type fields (expressions).

Properties

- property MaxLength: Integer — The maximum number of characters to enter into the editor. 0 means there is no restriction.
- property CharCase: TEditCharCase — The case of the entered data.
- property Language: TLanguage — The language of the keyboard when we enter the editor. By default, Cyrillic.
- - published properties

For the description of the rest of the properties and methods see TcfFilterItem.

TcfFilterItemEditStorage class

This class realizes the work of the Edit element. Supports the following operations: Equal, Starts with, Ends with, Contains and Null.

Свойства

property MaxLength: Integer	The maximum number of characters to enter into the editor. 0 means there is no restriction.
property CharCase: TEditCharCase	The case of the entered data.
property Language: TLanguage	The language of the keyboard when we enter the editor. By default, Cyrillic.
property Text: Ansistring	Contains the text entered by the user.

For the description of the rest of the properties and methods see TcfFilterItemStorage.

TcfFilterItemInteger class

It's an element of the Integer filter meant to fine-tune the scheme of the filter in design time. The element allows filtering data by integer fields (expressions). For the description of the properties and methods see TcfFilterItem.

TcfFilterItemIntegerStorage class

This class realizes the work of the Integer element. Supports the following operations: Equal, More than, More than or equal to, Less than, Less than or equal to, Between and Null.

Properties

property Value1: Integer	Contains the first number entered by the user.
property Value2: Integer	Contains the second number entered by the user.

For the description of the rest of the properties and methods see TcfFilterItemStorage.

TcfFilterItemFloat class

It's an element of the Float filter meant to fine-tune the scheme of the filter in design time. The element allows filtering data by float fields (expressions). For the description of the properties and methods see TcfFilterItem.

TcfFilterItemFloatStorage class

This class realizes the work of the Float element. Supports the following operations: Equal, More than, More than or equal to, Less than, Less than or equal to, Between and Null.

Properties

property Value1: Extended	Contains the first number entered by the user.
property Value2: Extended	Contains the second number entered by the user.

For the description of the rest of the properties and methods see TcfFilterItemStorage.

TcfFilterItemDate class

It's an element of the Date filter meant to fine-tune the scheme of the filter in design time. The element allows filtering data by fields (expressions) containing dates. For the description of the properties and methods see TcfFilterItem.

Класс TcfFilterItemDateStorage

This class realizes the work of the Date element. Supports the following operations: Equal, More than, More than or equal to, Less than, Less than or equal to, Between and Null.

Properties

property Value1: TDateTime	Contains the first date entered by the user.
property Value2: TDateTime	Contains the second date entered by the user.

For the description of the rest of the properties and methods see TcfFilterItemStorage.

TcfFilterItemLookup

It's an element of the Lookup filter meant for selecting an ID of a record from another table. A filter associated with the element allows selecting the desired record easily and visually.

Properties

❖ property MaxLength: Integer	The maximum number of characters to enter into the editor. 0 means there is no restriction.
❖ property CharCase: TEditCharCase	The case of the entered data.
❖ property Language: TLanguage	The language of the keyboard when we enter the editor. By default, Cyrillic.
❖ property KeyField: string	The name of the field in the Table table, by which the search is carried out when a value is entered into the entry field.
❖ property InfoField: string	The name of the field or expression in the Table table, from which the additional displayed information is taken.
❖ property IdField: string	The name of the field in the Table table, from which the ID of the record is taken.
❖ property Table: string	The name of the table, in which the search is carried out.
❖ property TmpDataSet: TDataSet	The dataset; required for the work of the element.
❖ property SubFilter: TcfSubFilter-Custom	Pointer to the TcfSubFilter component, that executes visual search.
❖ property KeyFieldType: TFieldType	Defines the type of the KeyField field.
❖ property OnSearch: TcfFilterItemSearchEvent	Sets the handler that allows implementing custom search algorithm on clicking the search button  .
❖ - published properties	

For the description of the rest of the properties and methods see TcfFilterItem

TcfFilterItemLookupStorage class

This class realizes the work of the Lookup element. It supports only one operation: Equal.

Properties

property MaxLength: Integer	The maximum number of characters to enter into the editor. 0 means there is no restriction.
property CharCase: TEditCharCase	The case of the entered data.
property Language: TLanguage	The language of the keyboard when we enter the editor. By default, Cyrillic.
property KeyField: string	The name of the field in the Table table, by which the search is carried out when a value is entered into the entry field.
property InfoField: string	The name of the field or expression in the Table table, from which the additional displayed information is taken.
property IdField: string	The name of the field in the Table table, from which the ID of the record is taken.
property Table: string	The name of the table, in which the search is carried out.
property TmpDataSet: TDataSet	The dataset; required for the work of the element.
property SubFilter: TcfSubFilterCustom	Pointer to the TcfSubFilter component, that executes visual search.
property KeyFieldType: TFieldType	Defines the type of the KeyField field.
property OnSearch:	Sets the handler that allows implementing custom search algorithm on

TcfFilterItemSearchEvent	clicking the search button  .
property ID: Variant	Contains the ID of the selected record.

Methods

function GetParam1: Variant	Returns the value of the first parameter. Applies to all element types.

For the description of the rest of the properties and methods see TcfFilterItemStorage.

TcfFilterItemDBCustomList class

This is an intermediary class for list-type elements. Defines behaviours common for all lists regardless of the visual format. Its functionality is similar to the one of the TcfFilterItemCustomList class, except the value for the list is taken from the chosen dataset.

Properties

❖ property KeyField: string	Field that fills in the KeyItems list
❖ property ListField: string	Field that fills in the Items list
❖ property ListDataSource: TDataSource	The data source from which the list is built.
❖ - published properties	

For the description of the rest of the properties see TcfFilterItemCustomList.

TcfFilterItemDBCustomListStorage class

This is an intermediary class for storing settings and generating sql code for all DB-ware list elements.

Properties

property KeyField: string	Field that fills in the KeyItems list
property ListField: string	Field that fills in the Items list
property ListDataSource: TDataSource	The data source from which the list is built.

For the description of the rest of the properties and methods see TcfFilterItemCustomListStorage.

Класс TcfFilterItemDBCheckListBox

It's an element of the DBCheckListBox filter meant to fine-tune the scheme of the filter in design time.

The element is a list with multiple selection of values.

Unlike the CheckListBox element, here the list is formed from a dataset named in ListDataSource. The Items list is filled with the values from the ListField field, and keyItems, with the values from the KeyFields field.

Columns sets the number of columns in which the values will be placed. Height sets the height of the element.

Publishes the following properties: Height, Columns. For the description of the rest of the properties see TcfFilterItemCustomList.

TcfFilterItemDBCheckListBoxStorage class

This class realizes the work of the DBCheckListBox element. For the description of the properties and methods see TcfFilterItemCustomListStorage.

TcfFilterItemDBList class

It's an element of the DBList, which is a list with single selection of a value.

Unlike the ListBox element, here the list is formed from a dataset named in ListDataSource. The Items list is filled with the values from the ListField field, and keyItems, with the values from the KeyFields field.

For the description of the properties see TcfFilterItemCustomList.

TcfFilterItemDBListStorage class

This class realizes the work of the DBListBox element. For the description of the properties and methods see TcfFilterItemCustomListStorage.

TcfFilterItemDBRadioGroup class

This is an element of the DBRadioGroup filter, and a list with multiple selection of values.

Unlike the ListBox element, here the list is formed from a dataset named in ListDataSource. The Items list is filled with the values from the ListField field, and keyItems, with the values from the KeyFields field.

Columns sets the number of columns in which the values will be placed. Height sets the height of the element.

Publishes the following properties: Height, Columns. For the description of the rest of the properties see TcfFilterItemCustomList.

TcfFilterItemDBRadioGroupStorage class

This class realizes the work of the DBRadioGroup element. For the description of the properties and methods see TcfFilterItemCustomListStorage.

TcfSimpleFilterItem class

Defines the list of the elements meant for generating conditions. Each element generates a condition set beforehand, and the values are taken from a chosen GUI element located on the same form with the component.

Properties

- property FieldName: string — Defines the name of the field or the expression for which the condition will be generated. A sub-query can act as the expression!
- property CaseInsensitive: Boolean — Defines if the search for strings will be case sensitive (True) or not

(False). By default, False.

- property DirectValue: Boolean — Determines if during forming the condition a user-defined value will be inserted (True), or the value will be passed via a parameter (False), which increases the performance of the Database Management System when the searches are of the same type.
- property FieldType: TFieldType — Defines the type of the field in the query. For simple fields is defined automatically, but for expressions it has to be specified manually.
- property Enabled: Boolean — Defines if the element will be accessible to the user (True) or not (False). By default, True.
- property UpperFunc: string — Defines the sql function of making the capitalization of strings consistent. Used when CaseInsensitive = True. By default, 'upper'.
- property FilterControl: TWinControl — GUI-element, into which the user enters values for the element.
- property BetweenControl: TWinControl — GUI- element, into which the user enters the second value for the 'between' operation.
- property Operation: TcfSimpleFilterOperation — Logical operation.
- property Values: TStrings — Defines values that will be used for such GUI as List, CheckList, RadioGroup, CheckBox, DBCombobox.
- property OnWhere: TcfSimpleFilterItemWhereEvent — This event allows redefining the standard procedure of generating sql-code for an element and writing custom sql-code for conditions, as complex as necessary, which considerably enhances the flexibility of the component.
- property OnItemEnabled: TcfItemEnabledEvent — The event allows redefining the standard logic of determining whether the element is used to generate a condition or not.
- - published properties

Methods

function Where(ItemNum: Integer; q: TcfFilterQuery): Boolean;	Generates sql code for the element with the ItemNum number. The generated string is inserted into the q query.

The standard logic that determines whether a condition is generated by the element is as follows:

- For descendants of TCustomEdit – the FilterControl component should be visible, editable and contain text. For the sfoBetween operation the same conditions are checked for the BetweenControl component.
- For descendants of TCommonCalendar – the FilterControl component should be visible, editable and contain a date that is not 0. For the sfoBetween operation the same conditions are checked for the BetweenControl component.
- For descendants of TCheckListBox – the list should be visible, element selection enabled and at least one of the elements selected.
- For descendants of TCustomListBox – the list should be visible, element selection enabled and one of the elements selected.
- For descendants of TCustomRadioGroup – the list should be visible, element selection enabled and one of the elements selected.
- For descendants of TCheckBox – the list should be visible, element selection enabled and the element selected.
- For descendants of TDBLookupControl – the list should be visible, element selection enabled and one of the elements selected.
- Other types of GUI elements are not supported.

TcfDatasetAdapter class

CleverFilter works with components that provide for access to a Database Management System, but it is not strictly connected with any component. This is achieved by the fact that the core of CleverFilter works with an abstract Query class. To connect CleverFilter to any real components accessing Database Management Systems adapters are used. Bundled with the library, adapters for the following components are supplied: ADO, BDE, DBX, IBX, FibPlus, Nexus, ODAC, IbDAC and UniDAC. If you use other components for database access, you'll need to write your own adapter. The TcfDatasetAdapter class is the base for adapter creation.

To create an adapter you need to implement the following methods in the descendant:

- procedure SetSQL(aDataset: TObject; SQL: string); virtual;

  sets the text of the query in the access component.

- function GetSQL(aDataset: TObject): string; virtual;

  receives the text of the query from the access component.

- procedure SetParams(aDataset, Params: TObject); virtual;

  sets the parameter values in the access component.

- procedure GetParams(aDataset, Params: TObject); virtual;

  reads the parameter values from the access component.

  As an example, you can see the source code of any adapter bundled with the library.

TcfSort class

The TcfSort class realizes the algorithm of sorting data by altering the "order by" part of the target query. The class is not shaped into a component, so, its independent use is impossible. The TcfFilter component has a Sort: TcfFilter property, through which the sorting algorithm can be controlled.

Properties

property Settings: AnsiString	Contain current settings (by which fields and in which order to sort the data) in the XML format. Can be used to create pre-prepared sets of sorting orders and quick switching between them inside the program.
property OrderBy: AnsiString	Contains the generated "order by" string for the target query.
property Section: AnsiString	The section name in the file with filter settings. Defines whether it's necessary to store user's settings in an external xml file and the section into which the recording will be done. If the value is not specified, the data won't be stored.
property Fields: TStrings	Defines the list of fields with which the class works.
property SortedField[Index: Integer]: TSortedFieldItem	The array makes it possible to control the fields in runtime: turn the sorting by field on and off and setting the direction of sorting.

TcfStorage class

The TcfStorage class is meant for storing the filter's settings and generating the sql code of the conditions. The information is presented as a list of TcfStoragePage pages.

Properties

property PagesCount: Integer	Returns the number of pages in the list.
property Page[Index: Integer]: TcfStoragePage	Gives access to the page with the Index number. Numbering starts with 0.
property Filter: TComponent	Pointer to the TcfFilter component.

Methods

procedure CreatePages(Area: TPageControl)	Creates visual pages in the filter's dialog window. The dialog consists of at least one page.
function CreateOnePage(Area: TPageControl; Index: Integer): Integer	Creates one visual page in the filter's dialog window. The page's number will be Index.
function AddPage: Integer;	Adds a new page both to the internal data structure and to the filter's dialog window.
procedure DeletePage(Index: Integer);	Deleted the page numbered Index from the internal structure.
procedure LoadFromXML(aXML, aSection: AnsiString);	Reads the filter's setgings from the aSection section of the aXML file. If there is no file or section, only the main page is created with the structure defined in the Filter property.
procedure SaveToXML(aXML, aSection: AnsiString);	Writes the filter's settings into the aSection section of the aXML file.
procedure LoadData(Root: IXmlNode);	Reads the filter's settings from the XML structure. The method can be used to load the settings in runtime from any sources.
procedure SaveData(Root: IXmlNode);	Writes the filter's settings into the XML structures. The method can be used to write the settings in runtime into any sources. Together with the LoadData method it allows creating "standard" sets of conditions to quickly apply of them later on.
procedure GetVisualState;	Reads the data from the filter's dialog window into the internal structures.
procedure Where(q: TcfFilterQuery);	Generates the sql code of conditions.
procedure ClearData;	Clears the internal structures of the data.
procedure CancelUpdates;	Cancels the user's modifications to the filter's dialog.

TcfStoragePage class

The class is meant for storing information from one page of the filter's dialog. The information is presented as a list of TcfStorageItem elements.

Свойства

property ItemsCount: Integer	Returns the number of elements in the list.
property Items[Index: Integer]: TcfStorageItem	Gives access to the element with the Index number. Numbering starts with 0.
property Filter: TComponent	Pointer to the TcfFilter component.

Методы

procedure GetVisualState;	Reads the data from the filter's dialog window into the internal structures.
function Where(PageNum: Integer; q: TcfFilterQuery): Boolean;	Generates the sql code of conditions. PageNum is the number of the page that contains the element.
procedure CancelUpdates;	Cancels the user's modifications to the filter's dialog.

TcfStorageItem class

The class stores the information about one element (along with all its alternative values). The information is presented as a list of TcfFilterItemStorage elements. The list contains at least one element.

Properties

property PointsCount: Integer	Returns the number of elements in the list.
property Points[Index: Integer]: TcfStoragePoint	Gives access to the element with the Index number. Numbering starts with 0. The TcfStoragePoint class is a synonym of the TcfFilterItemStorage class.
property Filter: TComponent	Pointer to the TcfFilter component.

Methods

procedure GetVisualState;	Reads the data from the filter's dialog window into the internal structures.
function Where(PageNum, LineNum: Integer; q: TcfFilterQuery): Boolean;	Generates the sql code of conditions. PageNum is the number of the page that contains the element. LineNum is the number of the element on the page.
procedure CancelUpdates;	Cancels the user's modifications to the filter's dialog.
function CreateOrForm(own: TComponent): TForm;	Creates a form for alternative values of the element.
property OrForm: TForm read FOrForm write FOrForm;	Pointer to the form for alternative values of the element.
procedure CreateVisual;	Creates visual representation of the element.

TcfFilterWhereEvent type

TcfFilterWhereEvent = procedure(Sender: TcfBaseFilter; SQL: TStrings; Params: TParams) of object;

Defines the event that generates the filter before and after generating the conditions.

SQL – current text of the query.
Params – current parameters of the query.

TcfFilterLoadShemaEvent type

TcfFilterLoadShemaEvent = procedure(Sender: TcfBaseFilter; Items: TStrings) of object;

Defines the event of loading the filter's scheme from external sources. The list of element classes should be loaded into Items.

TcfFilterLoadItemEvent type

TcfFilterLoadItemEvent = procedure(Sender: TcfBaseFilter; NumItem: Integer; Item: TcfFilterItem) of object;

Defines the event of loading the element's settings from external sources. NumItem is the element's number.

TcfFilterItemWhereEvent type

TcfFilterItemWhereEvent = procedure(Sender: TObject; SQL: TStrings; Params: TParams; FieldName: string; PageNum, LineNum, ItemNum: Integer) of object;

Defines the event of redefining the standard generation of conditions for the elements of the TcfFilter component.

SQL – current text of the query.
Params – current parameters of the query.
FieldName – the field, for which a condition needs to be generated.
PageNum – the number of the page.
LineNum – the number of the element on the page.
ItemNum – the number of the element in the collection.

Тип TcfSimpleFilterItemWhereEvent

TcfSimpleFilterItemWhereEvent = procedure(Sender: TObject; SQL: TStrings; Params: TParams; FieldName: string; ItemNum: Integer) of object;

Defines the event of redefining the standard generation of conditions for the elements of the TcfSimpleFilter component.

SQL – current text of the query.
Params – current parameters of the query.
FieldName – the field, for which a condition needs to be generated.
ItemNum – the number of the element in the collection.

TcfSimpleFilterItemEnabledEvent type

TcfItemEnabledEvent = procedure(Sender: TcfSimpleFilterItem; var Enabled: Boolean) of object;

Defines the event of determining whether it's necessary to generate a condition by the element or not.

TcfSimpleFilterOperation type

TcfSimpleFilterOperation = (sfoEqual, sfoNotEqual, sfoMore, sfoMoreEqual, sfoLess, sfoLessEqual, sfoBetween, sfoStarts, sfoEnds, sfoContains, sfoNull, sfoNotNull, sfoIn);

Defines the type of the operation for the elements in the TcfSimpleFilter component:

- sfoEqual – condition '='
- sfoNotEqual – condition '<>'
- sfoMore – condition '>'
- sfoMoreEqual – condition '>='
- sfoLess – condition '<'
- sfoLessEqual – condition '<='
- sfoBetween – condition 'between'
- sfoStarts – condition 'begins with'
- sfoEnds – condition 'ends with'
- sfoContains – condition 'contains'
- sfoNull – condition 'is null'
- sfoNotNull – condition 'is not null'

- sfoIn – condition ‘in (…)’

Тип TcfFilterItemListEvent

TcfFilterItemListEvent = procedure(Sender: TObject; Items: TStrings) of object;

Defines the event of filling in the Items list in the following elements: TcfFilterItemList, TcfFilterItemRadioGroup, TcfFilterItemCheckListBox.

TcfFilterItemSearchEvent type

TcfFilterItemSearchEvent = procedure(Sender: TObject; var id: Variant; var Executed: Boolean) of object;

Defines the event of searching for a record in the TcfFilterItemLookup element. Executed shows whether the search has been successful or not. If the search has been successful the ID of the record should be returned in the id.

TactivatePosition type

TActivatePosition = (taRight, taLeft)

Sets the position of the caption of the filter element in relation to the selection checkbox:

- taRight – the caption is placed first, then the checkbox.
- taLeft – the checkbox is placed first, then the caption.

Tlanguage type

TLanguage = (lgEnglish, lgRussian)

Defines the character set for the keyboard for the TcfFilterItemEdit element:

- lgEnglish - Latin
- lgRussian - Cyrillic

TsortedFieldItem type

```
TSortedFieldItem = record

  FieldName: string;

  Enabled: Boolean;

  Direction: TSortedFieldItemDirectin;

end;
```

Defines the structure of the class's internal data management in run time.

- FieldName – the name of the field. Generated by the class independently and shouldn't be altered by the program. Starting from Delphi 2006 it's impossible to change this field by the program.

- Enabled – defines whether it's necessary to sort by this field or not.
- Direction – defines the order of sorting. Makes sense if Enabled = True.

TsortedFieldItemDirectin Type

TSortedFieldItemDirectin = (sfAsc, sfDesc)

Sets the sorting direction:

- sfAsc – asceleration
- sfDesc – desceleration

RegisterSupportDataset procedure

The procedure binds the adapter with the component responsible for access to the Database Management System.

UnRegisterSupportDataset procedure

The procedure unbinds the adapter and the component responsible for access to the Database Management System.