Documentation SFBusinessData

TSFConnector

TSFBusinessDataSet/TSFDataSet

TSFBusinessDataWrap/TSFBusinessDataWrapSource

TSFStmt

Contents

Introduction

SFBusinessData is a componentset for Delphi® to

- organize your businesslogic in classes (specialized DataSets) and so to structurize your whole application
- write reusable code by strictly separate businesslogic and GUI
- get database independent SQL queries by using a query builder

Key feature is (beside the query builder) TSFBusinessData, which is the baseclass for own, specialized classes for organizing your businesslogic.
This you can also use as a unspecialized DataSet and as a exclusively buffered DataSet (without a connection to database). When using as a exlusively buffered DataSet, p. e. you can use databounded controls for all data and validate input by the baselogic of Datasetfields.

The componentset has not own logic to connect with a database, to connect with a database they use different components and mechanisms from Delphi®.
The link between TSFBusinessDataSet and the database connection is a connector (TSFConnector).

To integrate a own, specialized TSFBusinessDataSet over the designer (IDE) you need a wrapper (TSFBusinessDataWrap). This wrapper includes on runtime a object form your specialized class (on designtime a unspecialized object from TSFBusinessData).
Furthermore you need a special DataSource (TSFBusinessDataWrapSource) for your wrapper to bind on controls.

Topics

Connection

Connect to database

TSFBusinessData has not own components to connect with a database. The connection to a database will be done by the default components (depends on the acess technology you want).
The component TSFConnector ist the link between the data connection and the DataSets.

To create a data connection, do:

1. Add a TSFConnector
2. Choose the acess technology you want in property *ConnectionType*.
3. Add a connection component. Depending on *ConnectionType* use following component:
   - ctADO = TADOConnection
   - ctDBExpress = TSQLConnection
   - ctFireDac = TFDConnection
   - ctInterbase = TIBDatabase

   When using FireDac (ctFireDac) you also need a *DriverLink*, p. e. when connect to MsAccess you need a component *TFDPhysMsAccessDriverLink*.
4. Set the connection component to property *Connection* from your TSFConnector
5. TSFConnector tries to detect the databasetype automatically. Correct the the databasetype in property *ConnectionDBType*, when it is wrong or could not detected.
6. When using one TSFConnector for the whole application, set the property *CommonConnector* to *true*. Because of this, DataSets will detect and use the TSFConnector automatically (except a DataSet has an own connector).
   Note: An *CommonConnector* can be only found from DataSets, when the *CommonConnector* was created before.

DataSet

Difference between TSFDataSet and TSFBusinessData

TSFBusinessData is the baseclass for own, specialized classes for your businesslogic.
TSFDataSet is unspecialized, you can use this how an regular DataSet.

The basic functionally is the same in both classes.

Create a own, specialized class (TSFBusinessDataSet)

To create a specialized class for your businesslogic, do following (see also UBusinessDataTmpl.pas):

1. Create a new unit (File > New > Unit - Delphi)
2. Define a new class depending on TSFBusinessData and name the class how the referenced table in database.
3. Override constructor
4. In constructor set the properties *TableName*, *CatalogName*, *SchemaName*
5. Add a initialization section for registering your class

Expamle:

```
unit ...;

uses SFBusinessData, SFBusinessDataCustom, Data.DB;

type

	Customer = class(TSFBusinessData)
	public
		constructor Create(Component: TComponent); override;
	end;

implementation

constructor Customer.Create(Component: TComponent);
begin

	inherited;

	TableName := 'customer';
	SchemaName := '';
	CatalogName := '';
end;

initialization
begin
	TSFBusinessClassFactory.RegisterClass(customer, 'customer);
end;
```

Usage of your own, specialized class (TSFBusinessDataSet) on designtime and runtime

To integrate a specialized businessdata class on ***designtime***, you need a wrapper (TSFBuisnessDataWrap).

Add a component TSFBusinessDataWrap to your form/your datamodule and set the name of your businessdata class to property *BusinessClassName* from the TSFBusinessDataWrap component.
Because of this in property *BusinessDataSet* you reach a object from type

TSFBusinessDataSet. There you can configure the properties, call the assitent for building query (see property *Stmt*) and set procedures to events.

During designtime the businessdata object is unspecialized, on runtime you get a object from specialized class (see property *BusinessDataSet*).

To call functions from your class in code, use cast, p. e.

```
Customer(BusinessDataWrapper1.BusinessDataSet).MyFuntion1;
```

To link the *BusinessDataSet* with databounded controls add a component TSFBusinessDataWrapSource and set the TSFBusinessDataWrap to property *BusinessDataWrapper*.

To create on object of your specialized businessdata class, call *constructor* from your class. Your specialzied DataSet you can use how an regular DataSet (p. e. map with a DataSource).

Example:

```
var mySpecDataSet: Customer;
…
mySpecDataSet := Customer.Create;
DataSource1.DataSet := mySpecDataSet;
```

Usage of the DataSet without a specialized class

To use a unspecialized DataSet add a TSFDataSet and set the properties *TableName*, *CatalogName* and *SchemaName*.

The query builder you reach over property *Stmt* or over context menu of the component.

When creating a TSFDataSet on runtime, you also can give *TableName*, *CatalogName* and *SchemaName* to *constructor*.

Example:

```
var myDataSet: TSFDataSet;
…
// Create(TableName, CatalogName, SchemaName: String; Owner: TComponent = nil)
myDataSet:= TSFDataSet.Create('customer', '', '');
DataSource1.DataSet := myDataSet;
```

Processing from data changes

Data changes will be processed automatically. Therefore isn't any query or component required.

With the property *UpdateMode* you can adjust, how UPDATE/DELETE statements will be generated.

Changes on fieldvalues are only possible for fields related to the basetable (table which is identified by *TableName*, *CatalogName*, *SchemaName*). The basetable is also the bastable for the query builder.

AutoInc columns

Depending on access technology autoinc columns will be detected automatically.

To define a autoinc column programmatically use the function AddAutoValueForField. This function returns a object from TSFBDSAutoValueGenerator, which you can adjust (p. e. set the name of a sequenz). To adjust a automatically detected autoinc column use the function GetAutoValueForField.

Furthermore you can define a own class (depending on TSFBDSAutoValueGenerator) to generate autovalues by yourself - see also TSFBDSAutoValueGenerator, GetAutoValueCls.

When inserting a record to DataSet with autoinc columns the autoinc values will be initialized with a temporary (negative) Id.

Formatting fields

To define formats for field classes uses TSFConnector.FormatOptions or TSFCustomBusinessData.FormatOptions.
When no formats are defined, the formats will be detected by the source (DataSet). For the source you perhaps can define formats in connection component.

Transactions

Depending on access technology you can/you have to work with transactions (FireDac, Interbase).

Therfore you can link transaction components in the properties *Transaction* and *UpdateTransaction* in your DataSet. The type of the transaction component depends on the access technology:

- ctFireDac = TFDTransaction
- ctInterbase = TIBTransaction

Also you can link your transaction components with your connection component directly - see TFDConnection.Transaction, TFDConnection.UpdateTransaction, TIBDatabase.DefaultTransaction.

Relations/Master-Detail-Relations

Relations means there is DataSet (detail) that depends from another dataset (master). Details will be synchronized automatically by the master.

To define a relation on designtime use the property ParentRelationDesigner.

On runtime or in specialzed businessdata classes see AddRelation.

Internal sorting

You can sort your DataSet internally. That means only the record buffer will be sorted and no new query will be send to database. For sorting DataSet uses a quicksort algorithm.

To compare records the function SortBuffer calls the function CompareRecords and the event OnCompareRecords.
When use sorting you have to override the function (in your specialized businessdata class) or handle the event.

The function and the event gets 2 records from type TSFBDSCompareRecord, to compare values use the functions from this class.

Example:

…

```
myDataSet.SortBuffer;
```

…

```
function TForm1.TSFBusinessDataCompareRecords(CompareRecordFrom,
            CompareRecordTo: TSFBDSCompareRecord): TSFBDSRecordCompareResult;
begin
   // compare given records (called by SortBuffer)
   if (CompareRecordFrom.GetFieldValByName('Field1') >
      CompareRecordTo.GetFieldValByName('Field1')) then
     Result := compareResultGreater
   else if (CompareRecordFrom.GetFieldValByName('Field1') <
            CompareRecordTo.GetFieldValByName('Field1')) then
     Result := compareResultLess
   else
     Result := compareResultEqual;
end;
```

Internal filtering

With filtering your DataSet you can hide records without sending a new query to database.

To filter records you have to override the function FilterRecord (in your specialized businessdata class) or handle the event TDataSet.OnFilterRecord. The var parameter Accept regulates the current record will be shown or not (true = show record; false = hide record).
The values of the fields you get with the regular functions from TDataSet (p. e. FieldByName).

Example:

…

```
myDataSet.Filtered := true;
```

…

```
procedure TForm1.TSFBusinessDataFilterRecord(var Accept: Boolean);
begin
        Accept := (myDataSet.FieldByName('Field1').AsInteger > 10);
end;
```

Calculated and lookup fields in specialized classes (TSFBusinessData)

On designtime you can add persistant fields, calculatetd fields and lookup fields with the field editor (how in other DataSets). Therfore see helpfile from Delphi®.

In TSFBusinessData you also can add calculated fields and lookup fields dynamically in code. That means that you don't must add all persistent fields in field editor (or code) when using a calculated or a lookup field.
Therfore see the functions AddDynCalcField and AddDynLkpField. This functions you also can use in your specialized businessdata classes - p. e. when a calculated column should available in all objects.

In the fields container (TDataSet.Fields) the added calculated and lookup columns are available after open DataSet (how the other fields).

To set values for your calculated fields use the function TDataSet.DoOnCalcFields or the event TDataSet.OnCalcFields.

Refreshing calculated fields

To refresh calculated fields manually in code use the function RecalcCalculatedFields. Through this the function TDataSet.DoOnCalcFields will be called and the event TDataSet.OnCalcFields will be fired.

Usage without a connection to a database

You also can create objects from TSFDataSet, which are not related to a table in your database and have not a query. That means a DataSet which is only in buffer but can be used how an regular DataSet.

To create a DataSet without data connection call the constructor without *TableName*, *SchemaName* and *CatalogName*. Furthermore do not set this properties later.

```
// constructor Create(pOwner: TComponent); overload; override;
// constructor Create; reintroduce; overload; virtual;
myBufferDataSet := TSFDataSet.Create;
```

In next step add fields to your DataSet. Therefore use the function AddField.

```
myBufferDataSet.AddField ('Field1', ftInteger, 0);
myBufferDataSet.AddField ('Field2', ftString, 20);
```

To copy fields from a specialized businessdata class or from a table use InitFieldsFromBusinessData.

```
myBufferDataSet.InitFieldsFromBusinessData('customer', '', '', True, True, True);
```

Now you can open your DataSet.

```
myBufferDataSet.Open;
```

To insert and edit records use the regular functions from TDataSet.

```
myBufferDataSet.Insert;
myBufferDataSet.FieldByName(‘Field1').AsInteger := 1;
myBufferDataSet.FieldByName(‘Field2').AsString := 'Test';
myBufferDataSet.Post;

myBufferDataSet.Edit;
myBufferDataSet.FieldByName(‘Field2').AsString := 'For testing';
myBufferDataSet.Post;
```

You also can use the functions for internal sorting and internal filtering.

Classes TSFBusinessDataWrap and TSFBusinessDataWrapSource

The class TSFBusinessDataWrap is requiered to add a object from a specialized businessdata class on designtime.

TSFBusinessDataWrapSource is a DataSource (depends on TDataSource), which takes a object/component from TSFBusinessDataWrap instead a TDataSet.

See als Usage of your own, specialized class (TSFBusinessDataSet) on designtime and runtime

Query builder

Add a query on designtime and runtime

On designtime there is a assitant for defining queries. This assistant you get over the property Stmt from a TSFBusinessData component or over context menu from a TSFStmt component.

The structure form the assistent is orientated on class structure:

- General = General options, see properties TSFStmt
- Tables = Tables and Joins
- Attributes = Attributes/Fields and Items

- Conditions = Regular search conditions und EXISTS conditions
- Order = Sorting
- Group = Grouping
- Test = Generates the SQL query for testing, see GetSelectStmt

On runtime there are some functions to define SQL queries, see Generate a SELECT query programmatically.

Generate a SELECT query programmatically

First step to define a query ist to set a basetable. When the statement component is the internal statement from a TSFBusinessData the basetable will be setted automatically. The internal statement from a TSFBusinessData you reach with the property *Stmt*.

```
// SetBaseTable(pTableName, pSchema, pCatalog, pTableAlias: String)
myStmt.SetBaseTable('customer', '', '', '');
```

When you don't set aliases, TSFStmt manages aliases automatically. That means each table/join will get a alias.
With the overloaded function from *SetBaseTable* you also can set another TSFStmt as basetable. Thereby basetable is subselect.

Now you can add joins to your basetable.

```
// SetTableJoin(pTableAlias, pTableName, pSchema, pCatalog, pSourceTableAlias:
// String; const pRelValsDest, pRelValsSource: Array of Variant; const pRelTypesDest,
// pRelTypesSource: Array of TSFStmtJoinRelItemType; pType: TSFStmtJoinType):
// TSFStmtTable; overload;
myStmt.SetTableJoin('', 'customertype', '', '', 'customer', ['customertypeid'],
['customertypeid'], [stmtJoinRelItemAttr], [stmtJoinRelItemAttr], stmtJoinTypeInner);
```

Furthermore you can add joins recursive. That means you can add a join to joined table. For referencing source table you can use the name of the table or the alias.

Next step is to configure the SELECT clause. That means to add attributes - see SetStmtAttr, SetStmtAggr, AddStmtAttr.
When no visible attributes were added, TSFStmt generates a *Select* *.

```
// SetStmtAttr(pAttrName, pAttrAlias, pTableAlias: String; pOnlyForSearch: Boolean);
myStmt.SetStmtAttr('*', '', 'customer', False);
myStmt.SetStmtAttr('customertypedesc', '', 'customertype', False);

// AddStmtAttr(pAttrName: String; pOnlyForSearch: Boolean)
with myStmt.AddStmtAttr('incid', False) do
begin
    AddItemDbFld('customer', 'customerid', '');
    AddItemOperator(stmtAttrItemTypeOpPlus);
    AddItemValue(1);
end;
```

When you want to set a search condition (or a sort) to a field, which should not be shown in SELECT clause, you have to add the field with *OnlyForSearch*.

```
myStmt.SetStmtAttr('customerid', '', 'customer', True);
```

Closing you must set the search conditions, sort and group, see AddConditionVal, AddConditionAttr, AddConditionIsNull, AddConditionIsNotNull, AddConditionType, AddConditionExists, AddOrderAttr

```
// AddConditionIsNotNull(pTableAlias, pAttrName: String; pRestrict: Boolean = False);
myStmt. AddConditionIsNotNull('customer', 'customerid');

// AddOrderAttr(pTableAlias, pAttrName: String; pOrderType: TSFStmtSortType =
// stmtSortTypeAsc);
myStmt.AddOrderAttr('customer', 'customerid');
```

When between search conditions no operator (AND, OR) was added, TSFStmt generates a AND.

The Flag *Restrict* means that this are secure conditions which will not be deleted on ClearConditions. Search conditions which are marked with *Restricted* you have to handle separatly. That means you cannot link this conditions with regular conditions (restricted conditions will be generated separately).

The beforehand defined query, generates following result - see also GetSelectStmt:

```
SELECT t1.*, t2.customertypedesc, t1.customerid + 1 as incid
FROM customer t1 inner join customertype t2 on t1.customertypeid = t2.customertypeid
WHERE t1.customerid is not NULL
ORDER BY t1.customerid
```

Generate a INSERT query programmatically

Set your basetable, see SetBaseTable.

```
// SetBaseTable(pTableName, pSchema, pCatalog, pTableAlias: String)
myStmt.SetBaseTable('customer', '', '', '');
```

Add the values

```
// AddInsertCondition(pAttrName: String; pVal: Variant; pValType:
// TSFStmtAttrItemValueType);
myStmt.AddInsertCondition('customerid', 1, stmtAttrItemTypeValue);
myStmt.AddInsertCondition('customername', 'Test', stmtAttrItemTypeValue);
```

The SQL you will get with GetInsertStmt.

Generate a UPDATE query programmatically

Set your basetable, see SetBaseTable.

```
// SetBaseTable(pTableName, pSchema, pCatalog, pTableAlias: String)
myStmt.SetBaseTable('customer', '', '', '');
```

Define the SET clause

```
// AddSetCondition(pAttrName: String; pVal: Variant; pValType:
// TSFStmtAttrItemValueType);
myStmt.AddSetCondition('customername', 'Test', stmtAttrItemTypeValue);
```

Define the WHERE clause, see also Generate a SELECT query programmatically.

```
myStmt.SetStmtAttr('customerid', '', 'customer', True);
myStmt. AddConditionVal('customer', 'customerid', SFSTMT_OP_EQUAL, 1);
```

The SQL you will get with GetUpdateStmt.

Generate a DELETE query programmatically

Set your basetable, see SetBaseTable.

```
// SetBaseTable(pTableName, pSchema, pCatalog, pTableAlias: String)
myStmt.SetBaseTable('customer', '', '', '');
```

Define the WHERE clause, see also Generate a SELECT query programmatically.

```
myStmt.SetStmtAttr('customerid', '', 'customer', True);
myStmt. AddConditionVal('customer', 'customerid', SFSTMT_OP_EQUAL, 1);
```

The SQL you will get with GetDeleteStmt.

Search with LIKE/NOT LIKE

Searching with LIKE differs between LIKE single and LIKE many.

LIKE single means the count of chars in the searchstring are known. You only set your wildcard for single chars inside the searchstring, p. e. when searching for names with different spellings how Mayer, Meyer, Maier, Meier.
For searching with LIKE single you have to set your wildcard in the searchstring directly. The wildcard you can detect with TSFCustomBusinessData.GetLikeWildcardSingle or TSFStmt.GetDBDialectLikeWildcardSingle.

```
…
    var wildcard, search: String;
…
    wildcard := myStmt.GetDBDialectWildcardSingle;
    search := 'M' + wildcard + wildcard + 'er';
    myStmt. AddConditionVal('customer', 'customername', SFSTMT_OP_LIKE, search);
```

Searching with LIKE many means search for a substring. When using AutoEscapeLike the wildcards will be added automacially. Wildcards inside the searchstring will be "escaped" (when the used database supports ESCAPE syntax).

"Escaped" means, wildcards inside the searchstring will be marked with a ESCAPE char. Thereby the database knows that this isn't a wildcard.

When using AutoEscapeLike add the search condition without wildcard.

```
myStmt. AddConditionVal('customer', 'customernotice', SFSTMT_OP_LIKE, '100%');
```

When don't using AutoEscapeLike, add the wildcards to the searchstring.

```
…
  var wildcard, search: String;
…
  myStmt.AutoEscapeLike := false;
  wildcard := myStmt.GetDBDialectWildcardMany;
  search := wildcard + '100' + wildcard;
  myStmt. AddConditionVal('customer', 'customernotice', SFSTMT_OP_LIKE, search);
```

With LikeEscapeChar you can "escape" manually. LikeEscapeChar works only when don't using AutoEscapeLike and the database supports ESCAPE syntax.

```
…
  var wildcard, escape, search: String;
…
  escape := '#';
  myStmt.AutoEscapeLike := false;
  myStmt.LikeEscapeChar := escape;
  wildcard := myStmt.GetDBDialectWildcardMany;
  search := wildcard + '100' + escape + '%' + wildcard;
  myStmt. AddConditionVal('customer', 'customernotice', SFSTMT_OP_LIKE, search);
```

See also Generate a SELECT query programmatically.

Subselects

You can add subselects as

- Attributeitem for the SELECT clause
- Table
- (NOT) EXISTS search condition

To add a subselect on designtime, you have to add a separted component for the subselect (TSFStmt) and configure your subselect (add tables, attributes, etc.) with the assistant (s. Add a query on designtime and runtime).
Now you must referenceing your subselect in the basequery (p. e. add as table).

To add a subselect in, create first a new object from type TSFStmt.

```
mySubselect := TSFStmt.Create(nil);
```

Set the tables/joins, attributes/fields and search conditions for your subselect - see Generate a SELECT query programmatically.

Add your subselect to the basequery,

as a attributeitem for SELECT clause

```
with myStmt.AddAttr('mysub', False) do
    AddItemStmt(mySubselect);
```

as basetable

```
myStmt.SetBaseTable(mySubselect,'');
```

as join

```
myStmt.SetTableJoin('', 'customer', mySubselect, ['subselcustomerid'],
['customerid'], [stmtJoinRelItemAttr], [stmtJoinRelItemAttr], stmtJoinTypeInner);
```

as (NOT) EXISTS search condition

```
myStmt.AddConditionExists(mySubselect, 'customer', 'subtab1', SFSTMT_OP_EXISTS,
['subselcustomerid'], ['customerid'], [stmtJoinRelItemAttr], [stmtJoinRelItemAttr]);
```

Don’t free your subselects, this will be freed automatically by basequery.

Joins

As join you can add a table or a subselect.

See also:

Add a query on designtime and runtime
Generate a SELECT query programmatically
Subselects
TSFStmt.SetTableJoin

Calculated columns in SQL queries

To add calculated columns to the SELECT clause, you have to add the elements as items.

```
with myStmt.AddStmtAttr('incid', False) do
begin
    AddItemDbFld('customer', 'customerid', '');
    AddItemOperator(stmtAttrItemTypeOpPlus);
    AddItemValue(1);
end;
```

The name of the attribute represents the name of the column in result (… AS incid).

Aggregates

To add a simple aggregate for tablefield use

```
myStmt.SetStmtAggr(SFSTMTAGGR_SUM, 'customerid', 'sumid', 'customer');
```

Also you can combine aggregates with calculated columns

```
with myStmt.AddStmtAttr('sumidinc', False) do
begin
    AddItemAggrFunc(SFSTMTAGGR_SUM);
    AddItemBracket(stmtAttrItemTypeBracketOpen);
    AddItemDbFld('customer', 'customerid', '');
    AddItemBracket(stmtAttrItemTypeBracketClose);
    AddItemOperator(stmtAttrItemTypeOpPlus);
    AddItemValue(1);
end;
```

Userdefined text in SELECT clause

For SELECT clause you also can add userdefined attributes - p. e. for using database functions.
You are responsibly for the querytext inside the attribute by yourself. You should consider your code maybe isn't database independent - especially when using database functions.

```
with myStmt.AddStmtAttr('userdefattr', False) do
    AddItemDynamic ('substr(t1.customername, 1, 5)');
```

Usage of parameters in a query

To add a parameter use

```
with myBusinessDataObj.Stmt.AddStmtAttr('myprm1', True) do
    AddItemParam ('myprm1');
```

Now link your parameter with a search condition

```
myBusinessDataObj.Stmt.AddConditionAttr('customer', 'customerid',
SFSTMT_OP_EQUAL, '', ' myprm1');
```

On Execute your query inside a TSFBusinessData (*myBusinessDataObj.Open*) you have to set a value for your parameter.

Therefore use the event OnSetParams or set the values by using the property StmtParamValues directly.

Reference of classes and functions

TSFConnector

Description

Is the link between the database connection (p. e. TSQLConnection) and the objects from TSFBusinessData/TSFDataSet.

When setting *CommonConnector* to *true* you can use a single connector for your whole application.
Therby the objects from TSFBusinessData/TSFDataSet will detect the connector by theirself - without there is a connector referenced.
Hereby you should consider the connector is created before the first object accessing to the connector (p. e. place it to a automatically created datamodul).

Index

Functions

GetNewQuery

Notation:

function GetNewQuery(pTransaction: TComponent; pActionType: TSFQueryActionType; pCanUniDir: Boolean = True): TDataSet;

Visibility:

Public

Description:

Creates a internal query depending on *ConnectionType*. After creating query the event *OnDataSetCreated* will be fired (p. e. to configure settings).
In applications it should not be neccessary to use this function directly.

GetNewTable

Notation:

function GetNewTable(pTransaction: TComponent; pActionType: TSFQueryActionType; pTableName, pCatalog, pSchema: String): TDataSet;

Visibility:

Public

Description:

Creates a internal table depending on *ConnectionType*. After creating table the event *OnDataSetCreated* will be fired (p. e. to configure settings).
In applications it should not be neccessary to use this function directly.

GetKeyFields

Notation:

function GetKeyFields(pTableName, pCatalog, pSchema: String): String;

Visibility:

Public

Description:

Detects key fields for the given table.

In applications it should not be neccessary to use this function directly.

GetFieldNames

Notation:

function GetFieldNames(pTableName, pCatalog, pSchema: String): TStringList;

Visibility:

Public

Description:

Detects the fields for the given table.

In applications it should not be neccessary to use this function directly.

SetSQLToQuery

Notation:

function SetSQLToQuery(pSQL: String; pDataSet: TDataSet): TCollection;

Visibility:

Public

Description:

Sets the given SQL to the given internal query.

In applications it should not be neccessary to use this function directly.

QueryExecSQL

Notation:

function QueryExecSQL(pDataSet: TDataSet): LongInt;

Visibility:

Public

Description:

Executes an *ExecSQL* for the given dataset.

In applications it should not be neccessary to use this function directly.

SetQueryParamValue

Notation:

procedure SetQueryParamValue(pParam: TCollectionItem; pValue: Variant; pDataType: TFieldType = ftUnknown);

Visibility:

Public

Description:

Sets a value for parameter internally.

In applications it should not be neccessary to use this function directly.

GetQueryParamName

Notation:

function GetQueryParamName(pParam: TCollectionItem): String;

Visibility:

Public

Description:

Detects the name of a parameter.

In applications it should not be neccessary to use this function directly.

SequenceNameForField

Notation:

function SequenceNameForField(pField: TField): String;

Visibility:

Public

Description:

Detects the name of the sequence for given field. Whether a sequence can be detected depends on the *ConnectionType*.

In applications it should not be neccessary to use this function directly.

GetConnectionDBType

Notation:

function GetConnectionDBType: TSFConnectionDBType;

Visibility:

Public

Description:

Detects the used database with help of the given *Connection*.

In applications it should not be neccessary to use this function directly.

CheckTransaction

Notation:

function CheckTransaction(pTransaction: TComponent; pSilent: Boolean = False): Boolean;

Visibility:

Public

Description:

Validates the given transaction, p. e. when using FireDac you only can use TFDTransaction.

In applications it should not be neccessary to use this function directly.

HasDataSetTransaction

Notation:

function HasDataSetTransaction(pDataSet: TDataSet; pTransaction: TComponent): Boolean;

Visibility:

Public

Description:

Checks the given DataSet references a transaction.

In applications it should not be neccessary to use this function directly.

StartTransactionForDataSet

Notation:

procedure StartTransactionForDataSet(pDataSet: TDataSet);

Visibility:

Public

Description:

Starts a transaction for the given DataSet.

In applications it should not be neccessary to use this function directly.

CommitTransactionForDataSet

Notation:

procedure CommitTransactionForDataSet(pDataSet: TDataSet; pRetain: Boolean);

Visibility:

Public

Description:

Commits the transaction for the given DataSets.

In applications it should not be neccessary to use this function directly.

ActiveTransactionForDataSet

Notation:

function ActiveTransactionForDataSet(pDataSet: TDataSet): Boolean;

Visibility:

Public

Description:

Checks the given DataSet has an active transaction.

In applications it should not be neccessary to use this function directly.

CanDBInsertion

Notation:

function CanDBInsertion(pDataSet: TDataSet): Boolean;

Visibility:

Public

Description:

Checks if it is possible to do an insert with the given DataSet.

In applications it should not be neccessary to use this function directly.

AddConnectorMsgNotification

Notation:

procedure AddConnectorMsgNotification(pProc: TSFBDSMessageProc);

Visibility:

Public

Description:

Used internally to react on changes at the connector.

In applications it should not be neccessary to use this function directly.

RemoveConnectorMsgNotification

Notation:

procedure RemoveConnectorMsgNotification(pProc: TSFBDSMessageProc);

Visibility:

Public

Description:

Used internally to react on changes at the connector.

In applications it should not be neccessary to use this function directly.

GetCommonConnector

Notation:

class function GetCommonConnector: TSFConnector;

Visibility:

Public

Description:

Detectes the *CommonConnector*.

In applications it should not be neccessary to use this function directly.

AddCommonConnectedProc

Notation:

class procedure AddCommonConnectedProc(pProc: TSFBDSMessageProc);

Visibility:

Public

Description:

Used internally to react on changes at the connector.

In applications it should not be neccessary to use this function directly.

RemoveCommonConnectedProc

Notation:

class procedure RemoveCommonConnectedProc(pProc: TSFBDSMessageProc);

Visibility:

Public

Description:

Used internally to react on changes at the connector.

In applications it should not be neccessary to use this function directly.

Properties

ConnectionType

Notation:

property ConnectionType: TSFConnectionType read mConnectionType write setConnectionType;

Visibility:

Published

Description:

The type of the connection, p. e. FireDac, dbExpress, ADO. See also TSFConnectionType.

Connection

Notation:

property Connection: TCustomConnection read mConnection write setConnection;

Visibility:

Published

Description:

The referenced connection depending on *ConnectionType* (p. e. TFDConnection, TSQLConnection, TADOConnection).

CommonConnector

Notation:

property CommonConnector: Boolean read mCommonConnector write setCommonConnector;

Visibility:

Published

Description:

Defines the connector is the global connector. The objects from TSFBusinessData/TSFDataSet will detect the *CommonConnector* by theirself - without there is a connector referenced.

Hereby you should consider the *CommonConnector* is created before the first object accessing to the connector (p. e. place it to a automatically created datamodul).

ConnectionDBType

Notation:

property ConnectionDBType: TSFConnectionDBType read getDBType write mDBType;

Visibility:

Published

Description:

The used database, which was detected with help of connection.

In some cases (p. e. ODBC connections) the database cannot be detected with help of connection. Then you have to set the database by yourself. See also TSFConnectionDBType.

FormatOptions

Notation:

property FormatOptions: TSFBDSFormatOptions read mFormatOptions write setFormatOptions;

Visibility:

Published

Description:

Settings for formatting datevalues, floatvalues, etc. When no options are setted (in connector and in the object from TSFBusinessData/TSFDataSet), the options for formatting will be detected by the *Connection*.

For more information to see TSFBDSFormatOptions.

Events

OnDataSetCreated

Notation:

property OnDataSetCreated: TSFConnectorDSCreatedEvt read mOnDataSetCreated write mOnDataSetCreated;

Visibility:

Published

Description:

Will be fired after a new internal query or table was created. See als GetNewQuery and GetNewTable.

TSFCustomBusinessData/TSFBusinessData/TSFDataSet

Description

TSFBusinessData is the basclass for own, specialized classes to manage businesslogic. To create and use a own, specialized businessdata class see the correspondent topics.

TSFDataSet is a component with the same logic but without specialized classes. Also you can use TSFDataSet as a only buffered DataSet.

To use own, specialized businessdata classes on designtime consider that there are the classes TSFBusinessDataWrap and TSFBusinessDataWrapSource necessary.

Index

FilterRecord
FormatOptions
FullRefresh
GetAutoValueCls
GetAutoValueForField
GetAutoValueOptionsForDBType
GetBaseTableFields
GetCanSelectWithoutTable
GetCanSubSelectInFrom
GetKeyFields
GetLikeWildcardMany
GetLikeWildcardSingle
GetNameInBaseFieldsList
GetNeedTableOnSubSelectInFrom
GetSupportsLikeEscape
HasDynCalcField
HasDynLkpField
HasPassKeysRel
InitFieldsFromBusinessData
LocateNext
MappedStmtDBDialect
NotifyCurrentRecModified
OnAfterDBDeleteRow
OnAfterDBEditRow
OnAfterDBInsertRow
OnAfterPassKeyToObj
OnAfterRefreshFull
OnAfterRefreshRow
OnAfterSyncRelObj
OnBeforeDBDeleteRow
OnBeforeDBEditRow
OnBeforeDBInsertRow
OnBeforePassKeyToObj
OnBeforeRefreshFull
OnBeforeRefreshRow
OnBeforeSyncRelObj
OnCompareRecords
OnFieldChange
OnGetAutoValCls
OnRecordChange
OnSetParams
ParentRelationDesigner
PassKeysOnCachedUpdates
Prepare
QueryQuoteType
RecalcCalculatedFields
Refilter
RefreshMode
RefreshRelations
RefreshStmtParamValues
RemoveRelation
SchemaName
SelectNameForIdentifier
SetDisableSyncRel
SetPassKeysRel
SetQueryParams

SortBuffer
Stmt
StmtParamValues
SyncDisabled
TableName
Transaction
UpdateMode
UpdatesPending
UpdateTransaction

Functions

GetKeyFields

Notation:

function GetKeyFields: String; virtual;

Visibility:

Protected

Description:

Detects the key fields (primary keys) of the basetable.

GetBaseTableFields

Notation:

function GetBaseTableFields: TStringList; overload; virtual;

function GetBaseTableFields(pTableName, pSchemaName, pCatalogName: String): TStringList; overload;

function GetBaseTableFields(pStmtTable: TSFStmtTable): TStringList; overload;

Visibility:

Protected

Description:

Detects the fields of the basetable.

GetNameInBaseFieldsList

Notation:

function GetNameInBaseFieldsList(pName: String; pList: TStringList): Boolean;

Visibility:

Protected

Description:

Checks the given name (fieldname) exists in given list. Therfore the function also checks the name is quoted (regular quotes and quotes depending on used database).

NotifyCurrentRecModified

Notation:

procedure NotifyCurrentRecModified;

Visibility:

Protected

Description:

Used internally to notify DataSet that data was changed.

In applications it should not be neccessary to use this function directly.

GetAutoValueCls

Notation:

function GetAutoValueCls(pFieldName: String; pAutoDetected: Boolean): TSFBDSAutoValueGeneratorCls; virtual;

Visibility:

Protected

Description:

Detects the class which generates the values for autoinc fields. The parameter *pFieldName* is the name of the field, *pAutoDetected* defines the autoinc field was detected and added automatically.

See also TSFBDSAutoValueGenerator, AddAutoValueForField, GetAutoValueForField

GetAutoValueOptionsForDBType

Notation:

function GetAutoValueOptionsForDBType(pDBType: TSFConnectionDBType; pMode: TSFBDSAutoValueGetMode): TSFBDSAutoValueOptions; virtual;

Visibility:

Protected

Description:

Detects options for generating autoinc values depending on the database.

See also TSFBDSAutoValueGenerator, TSFBDSAutoValueGetMode, TSFBDSAutoValueOption, TSFBDSAutoValueOptions

BeforeDBEditRow

Notation:

procedure BeforeDBEditRow; virtual;

Visibility:

Protected

Description:

Called before data changes will be written in database by the internal UPDATE statement.

See also OnBeforeDBEditRow

AfterDBEditRow

Notation:

procedure AfterDBEditRow; virtual;

Visibility:

Protected

Description:

Called after data changes has been written in database by the internal UPDATE statement.

See also OnAfterDBEditRow

BeforeDBInsertRow

Notation:

procedure BeforeDBInsertRow; virtual;

Visibility:

Protected

Description:

Called before a new record will be written in database by the internal INSERT statement.

See also OnBeforeDBInsertRow

AfterDBInsertRow

Notation:

procedure AfterDBInsertRow; virtual;

Visibility:

Protected

Description:

Called after a new record has been written in database by the internal INSERT statement.

See also OnAfterDBInsertRow

BeforeDBDeleteRow

Notation:

procedure BeforeDBDeleteRow; virtual;

Visibility:

Protected

Description:

Called before a record will be deleted from database by the internal DELETE statement.

See also OnBeforeDBDeleteRow

AfterDBDeleteRow

Notation:

procedure AfterDBDeleteRow; virtual;

Visibility:

Protected

Description:

Called after a record has been deleted from database by the internal DELETE statement.

See also OnAfterDBDeleteRow

BeforeRefreshRow

Notation:

procedure BeforeRefreshRow; virtual;

Visibility:

Protected

Description:

Called before a record will be refreshed by the internal REFRESH statement.

See also OnBeforeRefreshRow

AfterRefreshRow

Notation:

procedure AfterRefreshRow; virtual;

Visibility:

Protected

Description:

Called after a record has been refreshed by the internal REFRESH statement.

See also OnAfterRefreshRow

BeforeRefreshFull

Notation:

procedure BeforeRefreshFull; virtual;

Visibility:

Protected

Description:

Called before the whole DataSet will be refreshed by the internal REFRESH statement.

See also OnBeforeRefreshFull

AfterRefreshFull

Notation:

procedure AfterRefreshFull; virtual;

Visibility:

Protected

Description:

Called after the whole DataSet has been refreshed by the internal REFRESH statement.

See also OnAfterRefreshFull

FilterRecord

Notation:

procedure FilterRecord(var pAccept: Boolean); virtual;

Visibility:

Protected

Description:

Function for filtering records. The var parameter *pAccept* regulates the current record will be shown or not (true = show record; false = hide record).

See also TDataSet.Filtered, TDataSet.OnFilterRecord, Refilter

CompareRecords

Notation:

function CompareRecords(CompareRecordFrom, CompareRecordTo: TSFBDSCompareRecord): TSFBDSRecordCompareResult; virtual;

Visibility:

Protected

Description:

Called on internal sorting with SortBuffer to compare records. When using the function SortBuffer you have to override this function or handle the event OnCompareRecords.

See also SortBuffer, OnCompareRecords, TSFBDSCompareRecord

SetQueryParams

Notation:

procedure SetQueryParams(pType: TSFBDSExecParamsType; pParams: TCollection); virtual;

Visibility:

Protected

Description:

In this function you can set values for the pararmeters from you SQL query on runtime. The parameter *pType* means the type of the query, the parameter *pParams* ist the list with the parameters. The type from *pParams* depends on used data connection, p. e. when using FireDac the type of *pParams* is TFDParams.

See also OnSetParams, StmtParamValues, TSFBDSExecParamsType

MappedStmtDBDialect

Notation:

function MappedStmtDBDialect: TSFStmtDBDialect;

Visibility:

Protected

Description:

Detects the SQL dialect from the query builder with the help of the ConnectionDBType from connector.

LocateNext

Notation:

function LocateNext(const KeyFields: string; const KeyValues: Variant;

Options: TLocateOptions): Boolean;

Visibility:

Public

Description:

Position the DataSet to the next record, which hits the search conditions (*KeyFields* = search fields; *KeyValues* = search values). In difference to locate the search starts on current record (and not on first record).

See also TDataSet.Locate

Prepare

Notation:

procedure Prepare;

Visibility:

Public

Description:

Prepares the internal SELECT query and creates field definitions.

SortBuffer

Notation:

procedure SortBuffer;

Visibility:

Public

Description:

Sorts the record buffer. That means there will not be send a new query to database. For sorting the function uses a QuickSort algorithm. For comparsion records the function use the function CompareRecords and the event OnCompareRecords.

ApplyUpdates

Notation:

procedure ApplyUpdates;

Visibility:

Public

Description:

Writes bufferd data changes in database.

See also CachedUpdates, CancelUpdates, UpdatesPending.

CancelUpdates

Notation:

procedure CancelUpdates;

Visibility:

Public

Description:

Faults buffered data changes.

See also ApplyUpdates, CachedUpdates, UpdatesPending.

FullRefresh

Notation:

procedure FullRefresh;

Visibility:

Public

Description:

Refreshes whole DataSet with sending a new query to database.

Refilter

Notation:

procedure Refilter;

Visibility:

Public

Description:

Refreshes the filter. Hereby all records will be checked again.

See also FilterRecord, TDataSet.OnFilterRecord

AddAutoValueForField

Notation:

function AddAutoValueForField(pFieldName: String; pAutoValueClass: TSFBDSAutoValueGeneratorCls = nil): TSFBDSAutoValueGenerator;

Visibility:

Public

Description:

Call this function to define a fiel das autoinc column. The parameter *pFieldName* is the name of the field, *pAutoValueClass* is the class which generate the autoinc values. When parameter *pAutoValueClass* is nil the default class TSFBDSAutoValueGenerator will be used.

See also GetAutoValueCls

GetAutoValueForField

Notation:

function GetAutoValueForField(pFieldName: String): TSFBDSAutoValueGenerator;

Visibility:

Public

Description:

Detects the object which generates auto values for the given field.

Siehe auch TSFBDSAutoValueGenerator, AddAutoValueForField, GetAutoValueCls

AddField

Notation:

function AddField(pFieldName: String; pDataType: TFieldType; pSize: Integer; pPrecision: Integer = 0; pRequired: Boolean = False; pReadOnly: Boolean = False): TField;

Visibility:

Public

Description:

Adds a field to the DataSet programmatically. Especially this you can use for buffered DataSets (DataSet without connection to a database, no referencing a table and no query).

When you want to add all fields from a specialized TSFBusinessData to a buffered DataSet use InitFieldsFromBusinessData.

InitFieldsFromBusinessData

Notation:

procedure InitFieldsFromBusinessData(pTabObjName: String; pCatalog: String = '';
pSchema: String = ''; pPreventAutoValues: Boolean = False; pPreventReadOnly: Boolean =
False; pPreventRequired: Boolean = False); overload;

procedure InitFieldsFromBusinessData(pObj: TSFCustomBusinessData;
pPreventAutoValues: Boolean = False; pPreventReadOnly: Boolean = False;
pPreventRequired: Boolean = False); overload;

Visibility:

Public

Description:

Adds all fields from a specialized TSFBusinessData to a buffered DataSet programmatically.

See also AddField

AllBaseFieldsToStmt

Notation:

procedure AllBaseFieldsToStmt(pOnlySearch: Boolean = False);

Visibility:

Public

Description:

Adds all fields from the basetable to the internal query builder. The parameter pOnlySearch means the fields will be added only for search conditions which will not be listed in SELECT clause.

AddDynCalcField

Notation:

procedure AddDynCalcField(pFieldName: String; pDataType: TFieldType; pSize: Integer;
pPrecision: Integer = 0);

Visibility:

Public

Description:

Adds a definition for a calulated field to the DataSet. The field himself will be created when opening the DataSet.

AddDynLkpField

Notation:

procedure AddDynLkpField(pFieldName: String; pDataType: TFieldType; pLkpDs: TDataSet; pKeyFlds, pLkpKeyFlds, pLkpRsltFld: String; pCached: Boolean; pSize: Integer; pPrecision: Integer = 0);

Visibility:

Public

Description:

Adds a definition for a lookup field to the DataSet. The field himself will be created when opening the DataSet.

HasDynCalcField

Notation:

function HasDynCalcField(pFieldName: String): Boolean;

Visibility:

Public

Description:

Checks the DataSet has definitions for calculated fields which have been added programmatically.

HasDynLkpField

Notation:

function HasDynLkpField(pFieldName: String): Boolean;

Visibility:

Public

Description:

Checks the DataSet has definitions for lookup fields which have been added programmatically.

RecalcCalculatedFields

Notation:

procedure RecalcCalculatedFields;

Visibility:

Public

Description:

Recalculates calculated fields.

DatabaseNameForFieldName

Notation:

function DatabaseNameForFieldName(pFieldName: String; var pTableAlias, pTableName, pTableSchema, pTableCatalog, pAttrName: String): Boolean;

Visibility:

Public

Description:

Detects the source (table, field, etc.) for given name (name of field in DataSet) in the query builder. On sucess the function returns true, the results will be written in the var parameters.

Especially this function is helpful when using aliases in your query.

SelectNameForIdentifier

Notation:

function SelectNameForIdentifier(pIdentifier: String; var pTableAlias, pTableName, pTableSchema, pTableCatalog, pAttrName: String): String;

Visibility:

Public

Description:

Detects the displayname for given identifier. The identifier is the name of the field in a table or an alias.

On success the function returns the displayname of the identifier, the source (tablealias, tablename, etc.) will be written in the var parameters.

FieldNameForDBField

Notation:

function FieldNameForDBField(pDBFieldName: String; pOnlyBaseFields: Boolean): String;

Visibility:

Public

Description:

Detects fieldname in DataSet for given fieldname from a table. The parameter pOnlyBaseFields means only fields from the basetable are to be searched.

GetLikeWildcardSingle

Notation:

function GetLikeWildcardSingle: String;

Visibility:

Public

Description:

Detects the wildcard single for searching with LIKE with help of the query builder. The char for wildcard single depends on used database normally the char is "_".

GetLikeWildcardMany

Notation:

function GetLikeWildcardMany: String;

Visibility:

Public

Description:

Detects the wildcard many for searching with LIKE with help of the query builder. The char for wildcard many depends on used database normally the char is "%".

GetSupportsLikeEscape

Notation:

function GetSupportsLikeEscape: Boolean;

Visibility:

Public

Description:

Checks the used database supports ESCAPE syntax on searching with LIKE. Through ESCAPE syntax you can search for strings which includes the wildcard char.

ExchangeRecordPositions

Notation:

procedure ExchangeRecordPositions(pFrom, pTo: Integer);

Visibility:

Public

Description:

Exchanges the postions from 2 records inside the DataSet.

RefreshStmtParamValues

Notation:

procedure RefreshStmtParamValues;

Visibility:

Public

Description:

Refreshes the definition of the query parameters with help of the query builder.

See also StmtParamValues.

GetCanSelectWithoutTable

Notation:

function GetCanSelectWithoutTable(var pDummyTable: String): Boolean;

Visibility:

Public

Description:

Detects the used database supports SELECT syntax without a table.

GetCanSubSelectInFrom

Notation:

function GetCanSubSelectInFrom: Boolean;

Visibility:

Public

Description:

Detects the used database supports SELECT syntax with an subselect in the FROM clause.

GetNeedTableOnSubSelectInFrom

Notation:

function GetNeedTableOnSubSelectInFrom: Boolean;

Visibility:

Public

Description:

Detects the used database supports SELECT syntax with an subselect in the FROM clause which (the subselect) doesn't referencing a table.

AddRelation

Notation:

procedure AddRelation(pDestObj: TSFBusinessData; pSourceAttrs, pDestAttrs: Variant; pPassKeys: Boolean = False); overload;

procedure AddRelation(pDestObj: TSFBusinessData; pSourceAttrs, pDestAttrs: String; pPassKeys: Boolean = False); overload;

Visibility:

Public

Description:

Adds a Master-Detail-Relation. The parameter *pPassKeys* means the master maps data changes on keyfields to the detail automatically.

The fields for the relation (*pSourceAttrs*, *pDestAttrs*) you can give as an VarArray or as string (use semicolon to split names). The names of the fields can be names from tablecolumns or the names from fields is DataSet.

See also RemoveRelation, RefreshRelations, DisableSync, EnableSync

RemoveRelation

Notation:

procedure RemoveRelation(pDestObj: TSFBusinessData);

Visibility:

Public

Description:

Deletes the Master-Detail-Relation for the given object.

See also AddRelation, RefreshRelations, DisableSync, EnableSync

RefreshRelations

Notation:

procedure RefreshRelations;

Visibility:

Public

Description:

Refreshs all Master-Detail-Relations if necessary.

See also AddRelation, RemoveRelation, DisableSync, EnableSync, ExplicitSyncRel

DisableSync

Notation:

procedure DisableSync;

Visibility:

Public

Description:

Deactivates synchronization of details (Master-Detail-Relations).

See also AddRelation, RemoveRelation, EnableSync, RefreshRelations, ExplicitSyncRel

EnableSync

Notation:

procedure EnableSync;

Visibility:

Public

Description:

Activates synchronization of details (Master-Detail-Relations).

See also AddRelation, RemoveRelation, DisableSync, RefreshRelations, ExplicitSyncRel

SyncDisabled

Notation:

function SyncDisabled: Boolean;

Visibility:

Public

Description:

Checks synchronization of details (Master-Detail-Relations) is active.

See also AddRelation, RemoveRelation, EnableSync, DisableSync, RefreshRelations, ExplicitSyncRel

SetDisableSyncRel

Notation:

procedure SetDisableSyncRel(pObj: TSFBusinessData; pDisabled: Boolean);

Visibility:

Public

Description:

Activates/Deactivates synchronization of the given detail (Master-Detail-Relation).

See also AddRelation, RemoveRelation, EnableSync, DisableSync, RefreshRelations, ExplicitSyncRel

ExplicitSyncRel

Notation:

procedure ExplicitSyncRel(pObj: TSFBusinessData);

Visibility:

Public

Description:

Refreshs the given detail (Master-Detail-Relation) if necessary.

See also RefreshRelations

SetPassKeysRel

Notation:

procedure SetPassKeysRel(pObj: TSFBusinessData; pPassKeys: Boolean);

Visibility:

Public

Description:

Sets *PassKeys* for the given detail (Master- Detail-Relation). When *PassKeys* is set, master maps data changes on keyfields to detail.

See also AddRelation

HasPassKeysRel

Notation:

function HasPassKeysRel: Boolean;

Visibility:

Public

Description:

Checks *PassKeys* is set (in one ore more relations).

See also AddRelation, SetPassKeysRel

DeleteByStmtConditions

Notation:

procedure DeleteByStmtConditions(pParamValues: Variant; pWithRefresh: Boolean);

Visibility:

Public

Description:

Executes a DELETE statement with the search conditions added in internal query builder (Stmt). In the parameter *pParamValues* you can add parametervalues for the query, when the parameter *pWithRefresh* is setted the DataSet will be refreshed after execution.

DeleteDepended

Notation:

procedure DeleteDepended(pTableName, pCatalog, pSchema, pSrcAttr, pDestAttr: String); overload;

procedure DeleteDepended(pTableName, pCatalog, pSchema, pDestAttr: String; pSrcVal: Variant); overload;

Visibility:

Public

Description:

Executes a DELETE statement for a related table.

Properties

TableName

Notation:

property TableName: String read mTableName write setTableName;

Visibility:

Protected/Published (TSFDataSet)

Description:

The name of the basetable.

See also CatalogName, SchemaName

CatalogName

Notation:

property CatalogName: String read mCatalogName write setCatalogName;

Visibility:

Protected/Published (TSFDataSet)

Description:

The catalog of the basetable.

See also TableName, SchemaName

SchemaName

Notation:

property SchemaName: String read mSchemaName write setSchemaName;

Visibility:

Protected/Published (TSFDataSet)

Description:

The schema of the basetable.

Siehe auch TableName, CatalogName

QueryQuoteType

Notation:

property QueryQuoteType: TSFBDSQuoteType read getQueryQuoteType;

Visibility:

Protected

Description:

Regulates identifiers will be quoted when generating query.

See also TSFBDSFormatOptions.QuoteType

Connector

Notation:

property Connector: TSFConnector read mConnector write setConnector;

Visibility:

Published

Description:

The mapped Connector. When using *CommonConnector* you don't need to set a connector.

ConnectorUsed

Notation:

property ConnectorUsed: TSFConnector read getConnectorUsed;

Visibility:

Public

Description:

The used Connector.

DBTableIdentifier

Notation:

property DBTableIdentifier: String read getDBTableIdentifier;

Visibility:

Public

Description:

Generated Identifier with help of the *TableName*, *SchemaName* and *CatalogName*.

UpdateMode

Notation:

property UpdateMode: TUpdateMode read mUpdateMode write mUpdateMode;

Visibility:

Published

Description:

Regulates on which fields are to be searched on internal UPDATE and DELETE statements - only on keyfields or on all fields.

See also TDataSet.Edit, TDataSet.Post, TDataSet.Delete

CachedUpdates

Notation:

property CachedUpdates: Boolean read mCachedUpdates write setCachedUpdates;

Visibility:

Published

Description:

When set to *true* data changes will be buffered.

Siehe auch ApplyUpdates, CancelUpdates, UpdatesPending

UpdatesPending

Notation:

property UpdatesPending: Boolean read mUpdatesPending;

Visibility:

Public

Description:

Is *true* when buffered data changes are existent.

See also ApplyUpdates, CancelUpdates, CachedUpdates

RefreshMode

Notation:

property RefreshMode: TSFBDSRefreshMode read mRefreshMode write mRefreshMode;

Visibility:

Published

Description:

Means what is to do when calling the function *Refresh*. Refresh the whole DataSet or Refresh only the current record.

See also TSFBDSRefreshMode

Stmt

Notation:

property Stmt: TSFStmt read mStmt;

Visibility:

Published

Description:

The internal query builder.

StmtParamValues

Notation:

property StmtParamValues: TCollection read getStmtParamValues write setStmtParamValues;

Visibility:

Published

Description:

Definitions of parameters in the query builder.

See also RefreshStmtParamValues

Transaction

Notation:

property Transaction: TComponent read mTransaction write setTransaction;

Visibility:

Published

Description:

Linked transaction (object) for SELECT query. The type of the transaction depends on the ConnectionType from the used Connector. That means p. e. when using FireDac the transaction have to be from type TFDTransaction.

See also TSFConnector.CheckTransaction, UpdateTransaction

FormatOptions

Notation:

property FormatOptions: TSFBDSFormatOptions read mFormatOptions write setFormatOptions;

Visibility:

Published

Description:

Settings for formatting datevalues, floatvalues, etc. When no options are setted (in connector and in the object from TSFBusinessData/TSFDataSet), the options for formatting will be detected by the *TSFConnector.Connection*.

For more information to see TSFBDSFormatOptions.

UpdateTransaction

Notation:

property UpdateTransaction: TComponent read mUpdateTransaction write setUpdateTransaction;

Visibility:

Published

Description:

Linked transaction (object) for UPDATE, INSERT and DELETE statements. The type of the transaction depends on the ConnectionType from the used Connector. That means p. e. when using FireDac the transaction have to be from type TFDTransaction.

See also TSFConnector.CheckTransaction, Transaction

ParentRelationDesigner

Notation:

property ParentRelationDesigner: TSFBusinessDataRelationDesigner read mParentRelationDesigner;

Visibility:

Published

Description:

Definition of Master-Detail-Relations on designtime.

See also AddRelation, RemoveRelation, RefreshRelations

PassKeysOnCachedUpdates

Notation:

property PassKeysOnCachedUpdates: Boolean read mPassKeysOnCachedUpdates write mPassKeysOnCachedUpdates;

Visibility:

Published

Description:

When true master maps changes on keyfield to details as well when changes on data will be buffered.

See also AddRelation, SetPassKeysRel

Events

OnBeforeDBEditRow

Notation:

property OnBeforeDBEditRow: TDataSetNotifyEvent read mOnBeforeDBEditRow write mOnBeforeDBEditRow;

Visibility:

Published

Description:

Will be fired before data changes will be written in database by the internal UPDATE statement.

See also BeforeDBEditRow

OnAfterDBEditRow

Notation:

property OnAfterDBEditRow: TDataSetNotifyEvent read mOnAfterDBEditRow write mOnAfterDBEditRow;

Visibility:

Published

Description:

Will be fired after data changes has been written in database by the internal UPDATE statement.

See also AfterDBEditRow

OnBeforeDBInsertRow

Notation:

property OnBeforeDBInsertRow: TDataSetNotifyEvent read mOnBeforeDBInsertRow write mOnBeforeDBInsertRow;

Visibility:

Published

Description:

Will be fired before a new record will be written in database by the internal INSERT statement.

See also BeforeDBInsertRow

OnAfterDBInsertRow

Notation:

property OnAfterDBInsertRow: TDataSetNotifyEvent read mOnAfterDBInsertRow write mOnAfterDBInsertRow;

Visibility:

Published

Description:

Will be fired after a new record has been written in database by the internal INSERT statement.

See also AfterDBInsertRow

OnBeforeDBDeleteRow

Notation:

property OnBeforeDBDeleteRow: TDataSetNotifyEvent read mOnBeforeDBDeleteRow write mOnBeforeDBDeleteRow;

Visibility:

Published

Description:

Will be fired before a record will be deleted from database by the internal DELETE statement.

See also BeforeDBDeleteRow

OnAfterDBDeleteRow

Notation:

property OnAfterDBDeleteRow: TDataSetNotifyEvent read mOnAfterDBDeleteRow write mOnAfterDBDeleteRow;

Visibility:

Published

Description:

Will be fired after a record has been deleted from database by the internal DELETE statement.

See also AfterDBDeleteRow

OnBeforeRefreshRow

Notation:

property OnBeforeRefreshRow: TDataSetNotifyEvent read mOnBeforeRefreshRow write mOnBeforeRefreshRow;

Visibility:

Published

Description:

Will be fired before a record will be refreshed by the internal REFRESH statement.

See also BeforeRefreshRow

OnAfterRefreshRow

Notation:

property OnAfterRefreshRow: TDataSetNotifyEvent read mOnBeforeRefreshRow write mOnBeforeRefreshRow;

Visibility:

Published

Description:

Will be fired after a record has been refreshed by the internal REFRESH statement.

See also AfterRefreshRow

OnBeforeRefreshFull

Notation:

property OnBeforeRefreshFull: TDataSetNotifyEvent read mOnBeforeRefreshFull write mOnBeforeRefreshFull;

Visibility:

Published

Description:

Will be fired before the whole DataSet will be refreshed by the internal REFRESH statement.

See also BeforeRefreshFull

OnAfterRefreshFull

Notation:

property OnAfterRefreshFull: TDataSetNotifyEvent read mOnAfterRefreshFull write mOnAfterRefreshFull;

Visibility:

Published

Description:

Will be fired after the whole DataSet has been refreshed by the internal REFRESH statement.

See also AfterRefreshFull

OnSetParams

Notation:

property OnSetParams: TSFBDSSetParamsEvt read mOnSetParams write mOnSetParams;

Visibility:

Published

Description:

Handling this event you can set values for query paremeters on runtime. The parameter *pType* is the type of the SQL query, the parameter *pParams* is the list with the parameters. The type from *pParams* depends on used database connection, p. e. when using FireDac the type of *pParams* is TFDParams.

See also SetQueryParams, StmtParamValues, TSFBDSExecParamsType, TSFBDSSetParamsEvt

OnCompareRecords

Notation:

property OnCompareRecords: TSFBSDRecordCompareEvent read mOnCompareRecords write mOnCompareRecords;

Visibility:

Published

Description:

Will be fired on internal sorting with SortBuffer to compare records. When using the function SortBuffer you have to handle this event or override the function CompareRecords.

See also SortBuffer, CompareRecords, TSFBDSCompareRecord, TSFBSDRecordCompareEvent

OnGetAutoValCls

Notation:

property OnGetAutoValCls: TSFBDSGetAutoValueCls read mOnGetAutoValCls write mOnGetAutoValCls;

Visibility:

Public

Description:

Handling this event you can set the class which generates values for autoinc columns. The parameter *pFieldName* is the name of the autoinc field, the parameter *pAutoDetected* notes the autoinc field was detected automatically or added manually.

See also TSFBDSAutoValueGenerator, AddAutoValueForField, GetAutoValueForField, GetAutoValueCls, TSFBDSGetAutoValueCls

OnFieldChange

Notation:

property OnFieldChange: TDataChangeEvent read mOnFieldChange write mOnFieldChange;

Visibility:

Published

Description:

Will be fired when value of a field was changed.

OnRecordChange

Notation:

property OnRecordChange: TDataSetNotifyEvent read mOnRecordChange write mOnRecordChange;

Visibility:

Published

Description:

Will be fired when current record was changed.

OnBeforeSyncRelObj

Notation:

property OnBeforeSyncRelObj: TDataSetNotifyEvent read mOnBeforeSyncRelObj write mOnBeforeSyncRelObj;

Visibility:

Published

Description:

Will be fired before a detail (Master-Detail-Relation) will be synchronized.

See also AddRelation, RefreshRelations, ExplicitSyncRel

OnAfterSyncRelObj

Notation:

property OnAfterSyncRelObj: TDataSetNotifyEvent read mOnAfterSyncRelObj write mOnAfterSyncRelObj;

Visibility:

Published

Description:

Will be fired after a detail (Master-Detail-Relation) has been synchronized.

See also AddRelation, RefreshRelations, ExplicitSyncRel

OnBeforePassKeyToObj

Notation:

property OnBeforePassKeyToObj: TDataSetNotifyEvent read mOnBeforePassKeyToObj write mOnBeforePassKeyToObj;

Visibility:

Published

Description:

Will be fired before master (Master-Detail-Relation) will map changes on keyfields to detail.

See also AddRelation, SetPassKeysRel, HasPassKeysRel

OnAfterPassKeyToObj

Notation:

property OnAfterPassKeyToObj: TDataSetNotifyEvent read mOnAfterPassKeyToObj write mOnAfterPassKeyToObj;

Visibility:

Published

Description:

Will be fired after master (Master-Detail-Relation) has mapped changes on keyfields to detail.

See also AddRelation, SetPassKeysRel, HasPassKeysRel

TSFBDSAutoValueGenerator

Description

TSFBDSAutoValueGenerator is the class which generates values for autoinc fields.

Wenn you need a own logic for generating values for autoinc fields you have to create a class which depends on TSFBDSAutoValueGenerator. Your class you can set in TSFBusinessDataSet.GetAutoValueCls oder TSFBusinessDataSet.OnGetAutoValCls.

Index

Functions

GetGeneratorValue

Notation:

function GetGeneratorValue(pMode: TSFBDSAutoValueGetMode): Variant; virtual;

Visibility:

Protected

Description:

Detects next autovalue.

See also TSFBDSAutoValueGetMode

Properties

SequenceName

Notation:

property SequenceName: String read mSequenceName write mSequenceName;

Visibility:

Public

Description:

When using generators (p. e. Interbase) or sequences here you can set the name of the generator or sequence.

Depending on connection type sequences maybe will be detected automatically.

DataSet

Notation:

property DataSet: TSFCustomBusinessData read mDataSet;

Visibility:

Public

Description:

Reference to the DataSet for which autovalues will be generated.

FieldName

Notation:

property FieldName: String read mFieldName;

Visibility:

Public

Description:

Name of field for which autovalues will be generated.

AutoDetected

Notation:

property AutoDetected: Boolean read mAutoDetected;

Visibility:

Public

Description:

Notes the autoinc field was detected automatically.

ExplicitInsertByDBMS

Notation:

property ExplicitInsertByDBMS: Boolean read mExplicitInsertByDBMS write mExplicitInsertByDBMS;

Visibility:

Public

Description:

Notes values for the autoinc field will be inserted by the database.

Options

Notation:

property Options[pMode: TSFBDSAutoValueGetMode]: TSFBDSAutoValueOptions read getOptions write setOptions;

Visibility:

Public

Description:

Options for generating autovalues.

See also TSFBDSAutoValueOptions, TSFBDSAutoValueGetMode

TSFBDSCompareRecord

Description

Class to compare values from records when sorting DataSet with TSFBusinessDataSet.SortBuffer

Index

GetBlobFieldValByIdx
GetBlobFieldValByName
GetFieldValByIdx
GetFieldValByName

Functions

GetFieldValByName

Notation:

function GetFieldValByName(pFieldName: String): Variant;

Visibility:

Public

Description:

Detects the value for given fieldname.

GetFieldValByIdx

Notation:

function GetFieldValByIdx(pFieldIdx: Integer): Variant;

Visibility:

Public

Description:

Detects the value for given fieldindex.

GetBlobFieldValByName

Notation:

function GetBlobFieldValByName(pFieldName: String): TArray<Byte>;

Visibility:

Public

Description:

Detects the value of a blob-field for given fieldname.

GetBlobFieldValByIdx

Notation:

function GetBlobFieldValByIdx(pFieldIdx: Integer): TArray<Byte>;

Visibility:

Public

Description:

Detects the value of a blob-field for given fieldindex.

TSFBusinessDataRelation

Description

Class to manage relations (Master-Detail-Relations).

Index

Properties

SrcObj

Notation:

property SrcObj: TSFBusinessData read mSrcObj write mSrcObj;

Visibility:

Public

Description:

The Master-Object.

See also AddRelation

DestObj

Notation:

property DestObj: TSFBusinessData read mDestObj write mDestObj;

Visibility:

Public

Description:

The Detail-Object

SrcAttrs

Notation:

property SrcAttrs: Variant read mSrcAttrs write mSrcAttrs;

Visibility:

Public

Description:

The given fieldnames in the Master-Object. A fieldname can be the fieldname in database/table or the fieldname in DataSet (p. e. differences when using aliases).

DestAttrs

Notation:

property DestAttrs: Variant read mDestAttrs write mDestAttrs;

Visibility:

Public

Description:

The given fieldnames in the Detail-Object. A fieldname can be the fieldname in database/table or the fieldname in DataSet (p. e. differences when using aliases).

SrcFieldNames

Notation:

property SrcFieldNames: Variant read mSrcFieldNames write mSrcFieldNames;

Visibility:

Public

Description:

The fieldnames in DataSet from *SrcAttrs* (p. e. differences when using aliases in SQL query).

DestFieldNames

Notation:

property DestFieldNames: Variant read mDestFieldNames write mDestFieldNames;

Visibility:

Public

Description:

The fieldnames in DataSet from *DestAttrs* (p. e. differences when using aliases in SQL query).

SrcDBIdent

Notation:

property SrcDBIdent: Variant read mSrcDBIdent write mSrcDBIdent;

Visibility:

Public

Description:

The identifiers in database/table from *SrcAttrs* (p. e. differences when using aliases in SQL query).

DestDBIdent

Notation:

property DestDBIdent: Variant read mDestDBIdent write mDestDBIdent;

Visibility:

Public

Description:

The identifiers in database/table from *DestAttrs* (p. e. differences when using aliases in SQL query).

SyncDisabled

Notation:

property SyncDisabled: Boolean read mSyncDisabled write mSyncDisabled;

Visibility:

Public

Description:

When *true* the synchronization of the relation is deactivated.

See also SetDisableSyncRel,

PassKeys

Notation:

property PassKeys: Boolean read mPassKeys write mPassKeys;

Visibility:

Public

Description:

When *true* master maps changes on keyfields to detail automatically.

See also SetPassKeysRel

TSFBusinessDataRelationDesigner

Description

Manages relations (Master-Detail-Relations) on designtime (IDE).

See also TSFBusinessDataRelation

Index

Properties

SrcObj

Notation:

property SrcObj: TSFBusinessData read mSrcObj;

Visibility:

Public

Description:

The Master-Object.

See also AddRelation

DestWrapper

Notation:

property DestWrapper: TSFBusinessDataWrap read mDestWrapper write setDestWrapper;

Visibility:

Published

Description:

The Detail-Object which is included in a wrapper. A Wrapper is necessary when using specialized classes from TSFBusinessData on designtime.

See also TSFBusinessDataWrap

DestObj

Notation:

property DestObj: TSFBusinessData read getDestObj write setDestObj;

Visibility:

Published

Description:

The Detail-Object. The Detail-Object can be included in a wrapper or linked directly.

See also TSFBusinessDataWrap

SrcAttrs

Notation:

property SrcAttrs: String read mSrcAttrs write setSrcAttrs;

Visibility:

Published

Description:

The given fieldnames in the Master-Object. A fieldname can be the fieldname in database/table or the fieldname in DataSet (p. e. differences when using aliases).

DestAttrs

Notation:

property DestAttrs: String read mDestAttrs write setDestAttrs;

Visibility:

Published

Description:

The given fieldnames in the Detail-Object. A fieldname can be the fieldname in database/table or the fieldname in DataSet (p. e. differences when using aliases).

PassKeys

Notation:

property PassKeys: Boolean read mPassKeys write setPassKeys;

Visibility:

Published

Description:

When *true* master maps changes on keyfields to detail automatically.

See also SetPassKeysRel

TSFBusinessDataWrap

Description

Class to manage objects from specialized classes depending on TSFBusinessData on runtime (IDE).

Index

Functions

AddDataSetNotification

Notation:

procedure AddDataSetNotification(pProc: TSFBusinessDataChanged);

Visibility:

Public

Description:

Internal used to handle DataSet.

RemoveDataSetNotification

Notation:

procedure RemoveDataSetNotification(pProc: TSFBusinessDataChanged);

Visibility:

Public

Description:

Internal used to handle DataSet.

Properties

BusinessClassName

Notation:

property BusinessClassName: String read mBusinessClassName write setBusinessClassName;

Visibility:

Published

Description:

The name of the class depending on TSFBusinessData.

BusinessDataSet

Notation:

property BusinessDataSet: TSFBusinessData read mBusinessDataSet;

Visibility:

Published

Description:

The inclueded object (TSFBusinessData). Note the object is only on runtime a instance from your specialized class depending on TSFBusinessData. On designtime it's a unspecialized object from baseclass.

TSFBusinessDataWrapSource

Description

A DataSource (depends on TDataSource) which gets the DataSet from a wrapper.

See also TDataSource, TSFBusinessDataWrap

Index

BusinessDataWrapper

DataSet

Properties

BusinessDataWrapper

Notation:

property BusinessDataWrapper: TSFBusinessDataWrap read mWrapper write setWrapper;

Visibility:

Public

Description:

The Wrapper.

DataSet

Notation:

property DataSet: TDataSet read getDataSet;

Visibility:

Public

Description:

The DataSet included in the Wrapper.

TSFStmt

Description

Class to manage and generate SQL queries.

Index

AssignStmt
AssignStmtTo
AttrDatabaseNameForAttrName
AttrDisplayName
AttrExists
AutoEscapeLike
BaseTable
ClearBaseRestrictions
ClearClientRectrictions
ClearConditions
ClearGroup
ClearInsConditions
ClearOrder
ClearSetConditions
ConfigStmtTimeValue
ConvertArrayValueToStr
ConvertValueInType
DBDialect
GenerateLevel
GenerateSubId
GenerateUnionId
GetConvertedValue
GetDBDialectCanSelWithoutTab
GetDBDialectCanSubInFrom
GetDBDialectLikeSupportsEscape
GetDBDialectLikeWildcardMany
GetDBDialectLikeWildcardSingle
GetDBDialectNeedTableOnSubInFrom
GetDeleteStmt
GetInsertStmt
GetLastAutoValueStmt
GetNextAutoValueStmt
GetNextTableNo
GetQuotedIdentifier
GetReferencedStmtByNamePath
GetReferencedStmtForParent
GetReferencedStmtNamePath
GetRelItemsForJoin
GetSelectStmt
GetStmtDatePart
GetStmtTimePart
GetTableAliasForAttr
GetTableForAttr
GetTableNameForAttr
GetTypeForValue
GetUpdateStmt
HasConditions
HasStmtDatePart
HasStmtTimePart
HasUnion
LikeEscapeChar
ListAttributeParams
ListAttributes
ListConditions
ListGroup
ListOrder

Functions

SetBaseTable

Notation:

function SetBaseTable(pTableName, pSchema, pCatalog, pTableAlias: String): TSFStmtTable; overload;

function SetBaseTable(pStmt: TSFStmt; pTableAlias: String): TSFStmtTable; overload;

function SetBaseTable(pStmtName, pTableAlias: String): TSFStmtTable; overload;

Visibility:

Public

Description:

Sets the basetable. The basetable also can be a reference of another statement (subselect).

TSFBusinessData/TSFDataSet sets the basetable for internal statement automatically.

Aliases (for tables) will be managed automatically but you can define own aliases. Therefore set parameter *pTableAlias* only when you want to use own aliases.

ReconfigBaseTable

Notation:

procedure ReconfigBaseTable(pTableName, pSchema, pCatalog, pTableAlias: String);

Visibility:

Public

Description:

Refreshs the basetable.

AddStmtAttr

Notation:

function AddStmtAttr(pAttrName: String; pOnlyForSearch: Boolean): TSFStmtAttr;

Visibility:

Public

Description:

Adds a undefinied attribute or field for the SELECT clause.
When parameter pOnlySearch is true, the attribut is only for search conditions, that means the attribute will not be listed in SELECT clause. P. e. when you want to add search condition for a tablefield (where fieldname = …) you have to add the tablefield as attribute before (as well when the tablefield must not listed in SELECT clause).

For detailed definition you have to add at leaest 1 Item - therefore see TSFStmtAttr.

See also TSFStmtAttr, TSFStmtAttrItemType

SetStmtAttr

Notation:

procedure SetStmtAttr(pAttrName, pAttrAlias, pTableAlias: String; pOnlyForSearch: Boolean); overload;

procedure SetStmtAttr(pAttrName, pAttrAlias: String; pStmtTable: TSFStmtTable; pOnlyForSearch: Boolean); overload;

Visibility:

Public

Description:

Adds a tablefield. The parameter *pAttrName* means the fieldname in database/table, the parameter *pAttrAlias* is optional and means the alias for the field generated in SELECT clause.

The parameter *pTableAlias*/*pStmtTable* references a existing table. Each table gets a alias automatically, in *pTableAlias* either you can set this alias or the name of table (if it's unique).

See also AddStmtAttr

SetStmtAggr

Notation:

procedure SetStmtAggr(pAggr, pAttrName, pAttrAlias, pTableAlias: String); overload;

procedure SetStmtAggr(pAggr, pAttrName, pAttrAlias: String; pStmtTable: TSFStmtTable); overload;

Visibility:

Public

Description:

Adds a tablefield with an aggregate.

See also SetStmtAttr, Constants

SetTableJoin

Notation:

function SetTableJoin(pTableAlias, pTableName, pSchema, pCatalog, pSourceTableAlias: String; pRelItems: TSFStmtJoinRelItems; pType: TSFStmtJoinType): TSFStmtTable; overload;

function SetTableJoin(pTableAlias, pTableName, pSchema, pCatalog, pSourceTableAlias: String; const pRelValsDest, pRelValsSource: Array of Variant; const pRelTypesDest, pRelTypesSource: Array of TSFStmtJoinRelItemType; pType: TSFStmtJoinType): TSFStmtTable; overload;

function SetTableJoin(pTableAlias, pSourceTableAlias: String; pDestStmt: TSFStmt; pRelItems: TSFStmtJoinRelItems; pType: TSFStmtJoinType): TSFStmtTable; overload;

function SetTableJoin(pTableAlias, pSourceTableAlias: String; pDestStmt: TSFStmt; const pRelValsDest, pRelValsSource: Array of Variant; const pRelTypesDest, pRelTypesSource: Array of TSFStmtJoinRelItemType; pType: TSFStmtJoinType): TSFStmtTable; overload;

function SetTableJoin(pTableAlias, pTableName, pSchema, pCatalog: String; pSourceTable: TSFStmtTable; pRelItems: TSFStmtJoinRelItems; pType: TSFStmtJoinType): TSFStmtTable; overload;

function SetTableJoin(pTableAlias, pTableName, pSchema, pCatalog: String; pSourceTable: TSFStmtTable; const pRelValsDest, pRelValsSource: Array of Variant; const pRelTypesDest, pRelTypesSource: Array of TSFStmtJoinRelItemType; pType: TSFStmtJoinType): TSFStmtTable; overload;

function SetTableJoin(pTableAlias: String; pSourceTable: TSFStmtTable; pDestStmt: TSFStmt; pRelItems: TSFStmtJoinRelItems; pType: TSFStmtJoinType): TSFStmtTable; overload;

function SetTableJoin(pTableAlias: String; pSourceTable: TSFStmtTable; pDestStmt: TSFStmt; const pRelValsDest, pRelValsSource: Array of Variant; const pRelTypesDest, pRelTypesSource: Array of TSFStmtJoinRelItemType; pType: TSFStmtJoinType): TSFStmtTable; overload;

function SetTableJoin(pTableAlias: String; pSourceTable: TSFStmtTable; pDestStmtName: String; pRelItems: TSFStmtJoinRelItems; pType: TSFStmtJoinType): TSFStmtTable; overload;

function SetTableJoin(pTableAlias: String; pSourceTable: TSFStmtTable; pDestStmtName: String; const pRelValsDest, pRelValsSource: Array of Variant; const pRelTypesDest, pRelTypesSource: Array of TSFStmtJoinRelItemType; pType: TSFStmtJoinType): TSFStmtTable; overload;

Visibility:

Public

Description:

Adds a join for the with pSourceTable/pSourceTableAlias defined table. The first table you add with SetBaseTable. TSFBusinessData/TSFDataSet adds the basetable for the internal statement automatically.

When the join/table should be an subselect set the object from type TSFStmt to this function. When using subselects don't free the object, this will be freed automatically (only when the object hasn't a owner).

The attributes/fields for the relation you can give with structures from type TSFStmtJoinRelItems or with Arrays. A attribute for a relation needn't be a tablefield this also can be a value.

Aliases (for tables) will be managed automatically but you can define own aliases. Therefore set Alias only when you want to use own aliases.

See also SetBaseTable, TSFStmtJoinRelItems, TSFStmtJoinRelItem, TSFStmtJoinRelItemType, TSFStmtJoinType

ModfiyTableJoinType

Notation:

procedure ModfiyTableJoinType(pDestTableAlias, pSourceTableAlias: String; pTypeFrom, pTypeTo: TSFStmtJoinType); overload;

procedure ModfiyTableJoinType(pDestTable, pSourceTable: TSFStmtTable; pTypeFrom, pTypeTo: TSFStmtJoinType); overload;

Visibility:

Public

Description:

With this function you can change the type of a join.

Siehe auch TSFStmtJoinType

TableJoinAliasesForAttr

Notation:

function TableJoinAliasesForAttr(pSourceTableAlias, pAttr: String): Variant; overload;

function TableJoinAliasesForAttr(pSourceTable: TSFStmtTable; pAttr: String): Variant; overload;

Visibility:

Public

Description:

Detects joins/tables for which the given attribute/field is definied in relation.

GetRelItemsForJoin

Notation:

function GetRelItemsForJoin(pSourceTable, pDestTable: TSFStmtTable): TSFStmtJoinRelItems; overload;

function GetRelItemsForJoin(pSourceTableAlias, pDestTableAlias: String): TSFStmtJoinRelItems; overload;

Visibility:

Public

Description:

Detects the attributes/fields for the relation in a join.

See also TSFStmtJoinRelItems

SetRelItemsForJoin

Notation:

procedure SetRelItemsForJoin(pSourceTable, pDestTable: TSFStmtTable; pRelItems: TSFStmtJoinRelItems); overload;

procedure SetRelItemsForJoin(pSourceTableAlias, pDestTableAlias: String; pRelItems: TSFStmtJoinRelItems); overload;

Visibility:

Public

Description:

Changes the attributes/fields for the relation in a join.

See also TSFStmtJoinRelItems

GetNextTableNo

Notation:

function GetNextTableNo: Integer;

Visibility:

Public

Description:

Detects next unique table number. Tables/Joins inside the statement will get a unique number and a unique alias automatically.

AddConditionVal

Notation:

procedure AddConditionVal(pTableAlias, pAttrName, pOp: String; pVal: Variant; pRestrict: Boolean = False);

Visibility:

Public

Description:

Adds a condition for the WHERE clause. The searchvalue for the condition is a value. The parameters *pTableAlias* and *pAttrName* describes the attribute/field for the condition, for Alias you also can set the tablename.

When *pRestrict* = *true* the condition will not be deleted by the function ClearConditions. Hereby the condition is defined as a saved condition (ClientRestriction), this conditions you can delete by the function ClearClientRestrictions.

When the attribute/field for the condition isn't a tablefield the parameter *pTableAlias* should be empty, in this case the parameter *pAttrName* means the alias from the attribut.

To add a condition for a field/attribute you have to add the attribute before.

See also SetStmtAttr, AddStmtAttr, Constants

AddConditionAttr

Notation:

procedure AddConditionAttr(pSrcTabAlias, pSrcAttrName, pOp, pDestTabAlias, pDestAttrName: String; pRestrict: Boolean = False);

Visibility:

Public

Description:

Adds a condition for the WHERE clause. The searchvalue for the condition is a another attribute. The parameters *pSrcTabAlias/pDestTabAlias* and *pSrcAttrName/pDestAttrName* describes the attribute/field for the condition, for Alias you also can set the tablename.

When *pRestrict* = *true* the condition will not be deleted by the function ClearConditions. Hereby the condition is defined as a saved condition (ClientRestriction), this conditions you can delete by the function ClearClientRestrictions.

When a attribute/field for the condition isn't a tablefield the parameter *pSrcTabAlias/pDestTabAlias* should be empty, in this case the parameter *pSrcAttrName/pDestAttrName* means the alias from the attribut.

To add a condition for a field/attribute you have to add the attribute before.

See also SetStmtAttr, AddStmtAttr, Constants

AddConditionIsNull

Notation:

procedure AddConditionIsNull(pTableAlias, pAttrName: String; pRestrict: Boolean = False);

Visibility:

Public

Description:

Adds a condition for the WHERE clause which checks the value of a field is NULL. The parameters *pTableAlias* and *pAttrName* describes the attribute/field for the condition, for Alias you also can set the tablename.

When *pRestrict* = *true* the condition will not be deleted by the function ClearConditions. Hereby the condition is defined as a saved condition (ClientRestriction), this conditions you can delete by the function ClearClientRestrictions.

When the attribute/field for the condition isn't a tablefield the parameter *pTableAlias* should be empty, in this case the parameter *pAttrName* means the alias from the attribut.

To add a condition for a field/attribute you have to add the attribute before.

See also SetStmtAttr, AddStmtAttr

AddConditionIsNotNull

Notation:

procedure AddConditionIsNotNull(pTableAlias, pAttrName: String; pRestrict: Boolean = False);

Visibility:

Public

Description:

Adds a condition for the WHERE clause which checks the value of a field is NOT NULL. The parameters *pTableAlias* and *pAttrName* describes the attribute/field for the condition, for Alias you also can set the tablename.

When *pRestrict* = *true* the condition will not be deleted by the function ClearConditions. Hereby the condition is defined as a saved condition (ClientRestriction), this conditions you can delete by the function ClearClientRestrictions.

When the attribute/field for the condition isn't a tablefield the parameter *pTableAlias* should be empty, in this case the parameter *pAttrName* means the alias from the attribut.

To add a condition for a field/attribute you have to add the attribute before.

See also SetStmtAttr, AddStmtAttr

AddConditionType

Notation:

procedure AddConditionType(pType: TSFStmtConditionType; pRestrict: Boolean = False);

Visibility:

Public

Description:

Adds a item of specified type to the WHERE clause (p. e. bracket, AND, OR).

See also TSFStmtConditionType

AddConditionExists

Notation:

procedure AddConditionExists(pDestStmt: TSFStmt; pTableAlias, pDestTableAlias, pOp: String; pRelItems: TSFStmtJoinRelItems; pRestrict: Boolean = False); overload;

procedure AddConditionExists(pDestStmt: TSFStmt; pTableAlias, pDestTableAlias, pOp: String; const pRelValsDest, pRelValsSource: Array of Variant; const pRelTypesDest, pRelTypesSource: Array of TSFStmtJoinRelItemType; pRestrict: Boolean = False); overload;

procedure AddConditionExists(pDestStmtName, pTableAlias, pDestTableAlias, pOp: String; pRelItems: TSFStmtJoinRelItems; pRestrict: Boolean = False); overload;

procedure AddConditionExists(pDestStmtName, pTableAlias, pDestTableAlias, pOp: String; const pRelValsDest, pRelValsSource: Array of Variant; const pRelTypesDest, pRelTypesSource: Array of TSFStmtJoinRelItemType; pRestrict: Boolean = False); overload;

Visibility:

Public

Description:

Adds a EXISTS conditions to the WHERE clause. The subselect for the EXISTS condition references another instance/object from TSFStmt which have to be setted in *pDestStmt*.

When using subselects don't free the object, this will be freed automatically (only when the object hasn't a owner).

The attributes/fields for the relation you can give with structures from type TSFStmtJoinRelItems or with Arrays. A attribute for a relation needn't be a tablefield this also can be a value.

When *pRestrict = true* the condition will not be deleted by the function ClearConditions. Hereby the condition is defined as a saved condition (ClientRestriction), this conditions you can delete by the function ClearClientRestrictions.

See also TSFStmtJoinRelItems, TSFStmtJoinRelItem, TSFStmtJoinRelItemType, Constants

AddOrderAttr

Notation:

procedure AddOrderAttr(pTableAlias, pAttrName: String; pOrderType: TSFStmtSortType = stmtSortTypeAsc);

Visibility:

Public

Description:

Defines a added attribute as orderattribut (ORDER BY).

The parameters *pTableAlias* and *pAttrName* describes the field/attribute, instead of alias you also can set the tablename in parameter *pTableAlias*.

When the attribute/field isn't a tablefield the parameter *pTableAlias* should be empty, in this case the parameter *pAttrName* means the alias from the attribut.

See also SetStmtAttr, AddStmtAttr, TSFStmtSortType

AddGroupAttr

Notation:

procedure AddGroupAttr(pTableAlias, pAttrName: String);

Visibility:

Public

Description:

Defines a added attribute as groupattribut (GROUP BY).

The parameters *pTableAlias* and *pAttrName* describes the field/attribute, instead of alias you also can set the tablename in parameter *pTableAlias*.

When the attribute/field isn't a tablefield the parameter *pTableAlias* should be empty, in this case the parameter *pAttrName* means the alias from the attribut.

See also SetStmtAttr, AddStmtAttr

AddSetCondition

Notation:

procedure AddSetCondition(pAttrName: String; pVal: Variant; pValType: TSFStmtAttrItemValueType);

Visibility:

Public

Description:

Sets a value for a attribut/field to the SET claus of a UPDATE statement.

See also TSFStmtAttrItemValueType

AddInsertCondition

Notation:

procedure AddInsertCondition(pAttrName: String; pVal: Variant; pValType: TSFStmtAttrItemValueType);

Visibility:

Public

Description:

Sets a value for a attribut/field to a INSERT statement.

See also TSFStmtAttrItemValueType

Reset

Notation:

procedure Reset;

Visibility:

Public

Description:

Resets the whole statement. That means all tables/joins, attributes, conditions, etc. will be deleted.

GetSelectStmt

Notation:

function GetSelectStmt(pLevel: Integer = 0; pSubId: Integer = 0; pUnionId: Integer = 0): String;

Visibility:

Public

Description:

Generates the SELECT statment. The parameters *pLevel*, *pSubId* and *pUnionId* are for internal generation of subselects.

GetDeleteStmt

Notation:

function GetDeleteStmt: String;

Visibility:

Public

Description:

Generates the DELETE statement. Conditions for the DELETE statement will be added as well as generating a SELECT statement.

GetUpdateStmt

Notation:

function GetUpdateStmt: String;

Visibility:

Public

Description:

Generates the UPDATE statement. Conditions for the DELETE statement will be added as well as generating a SELECT statement. To set values for the SET clause see AddSetCondition.

GetInsertStmt

Notation:

function GetInsertStmt: String;

Visibility:

Public

Description:

Generates the INSERT statement.

See also AddInsertCondition

GetNextAutoValueStmt

Notation:

function GetNextAutoValueStmt(pRefName: String = ''): String;

Visibility:

Public

Description:

Generates (depending on DBDialect) the statement to detect next autovalue.

See also TSFStmtDBDialectConv

GetLastAutoValueStmt

Notation:

function GetLastAutoValueStmt(pRefName: String = ''): String;

Visibility:

Public

Description:

Generates (depending on DBDialect) the statement to detect last inserted autovalue.

See also TSFStmtDBDialectConv

GetDBDialectCanSelWithoutTab

Notation:

function GetDBDialectCanSelWithoutTab(var pTableName: String): Boolean;

Visibility:

Public

Description:

Detects the used database (depending on DBDialect) supports SELECT syntax without a table

GetDBDialectCanSubInFrom

Notation:

function GetDBDialectCanSubInFrom: Boolean;

Visibility:

Public

Description:

Detects the used database (depending on DBDialect) supports SELECT syntax with an subselect in the FROM clause.

GetDBDialectNeedTableOnSubInFrom

Notation:

function GetDBDialectNeedTableOnSubInFrom: Boolean;

Visibility:

Public

Description:

Detects the used database (depending on DBDialect) supports SELECT syntax with an subselect in the FROM clause which (the subselect) doesn't referencing a table.

GetDBDialectLikeWildcardSingle

Notation:

function GetDBDialectLikeWildcardSingle: String;

Visibility:

Public

Description:

Detects the wildcard single for searching with LIKE. The char for wildcard single depends on used DBDialect normally the char is "_".

GetDBDialectLikeWildcardMany

Notation:

function GetDBDialectLikeWildcardMany: String;

Visibility:

Public

Description:

Detects the wildcard many for searching with LIKE. The char for wildcard many depends on used DBDialect normally the char is "%".

GetDBDialectLikeSupportsEscape

Notation:

function GetDBDialectLikeSupportsEscape: Boolean;

Visibility:

Public

Description:

Checks the used DBDialect supports ESCAPE syntax on searching with LIKE. Through ESCAPE syntax you can search for strings which includes the wildcard char.

AddBaseRestriction

Notation:

procedure AddBaseRestriction(pTableAlias, pAttrName: String; pVal: Variant; pVisible, pPreventNull: Boolean);

Visibility:

Public

Description:

Used to set internal conditions. Do not use this function in your applications use AddConditionVal instead.

ClearBaseRestrictions

Notation:

procedure ClearBaseRestrictions;

Visibility:

Public

Description:

Deletes internal conditions.

See also AddBaseRestriction

ClearConditions

Notation:

procedure ClearConditions;

Visibility:

Public

Description:

Deletes all search conditions which are not *Restricted*.

See also AddConditionVal, AddConditionAttr, AddConditionIsNull, AddConditionIsNotNull, AddConditionType, AddConditionExists

ClearClientRectrictions

Notation:

procedure ClearClientRectrictions;

Visibility:

Public

Description:

Deletes all search conditions which are *Restricted*.

See also AddConditionVal, AddConditionAttr, AddConditionIsNull, AddConditionIsNotNull, AddConditionType, AddConditionExists

ClearOrder

Notation:

procedure ClearOrder;

Visibility:

Public

Description:

Deletes the sorting (ORDER BY) inside the query.

See also AddOrderAttr

ClearGroup

Notation:

procedure ClearGroup;

Visibility:

Public

Description:

Deletes the grouping (GROUP BY) inside the query.

See also AddGroupAttr

ClearSetConditions

Notation:

procedure ClearSetConditions;

Visibility:

Public

Description:

Delete the definitions in the SET clause from a UPDATE statement.

See also AddSetCondition

ClearInsConditions

Notation:

procedure ClearInsConditions;

Visibility:

Public

Description:

Deletes the definitions from a INSERT statement.

See also AddInsertCondition

AttrExists

Notation:

function AttrExists(pAttrName, pTableAlias, pAggr: String): Boolean;

Visibility:

Public

Description:

Checks the given attribute/field was added before.

The parameter pAttrName can be the name of the tablefield or the alias of the attribut. The parameter *pTableAlias* is optional, when setted it should be the tablealias or the tablename.

See also AddStmtAttr, SetStmtAttr, SetStmtAggr

AttrDisplayName

Notation:

function AttrDisplayName(pAttrName, pTableAlias: String): String;

Visibility:

Public

Description:

Detectes the displayname of the given attribute/field. The displayname is also the columnname in the result. If defined a alias for the attribut the result is the alias, otherwise the result is the name of the tablefield.

The parameter *pAttrName* can be the name of the tablefield or the alias of the attribut. The parameter *pTableAlias* is optional, when setted it should be the tablealias or the tablename.

See also AddStmtAttr, SetStmtAttr, SetStmtAggr

GetTableNameForAttr

Notation:

function GetTableNameForAttr(var pAttrName: String; pIncludeInvisible: Boolean): String;

Visibility:

Public

Description:

Detects the tablename for the attribute/field described with *pAttrName*. The parameter *pAttrName* can be the name of the tablefield or the alias of the attribut. If the attribute/field was found the name of the tablefield will be setted in *pAttrName*.

The parameter *pIncludeInvisible* means that also fields will be searched which are *OnlyForSearch*.

See also AddStmtAttr, SetStmtAttr, SetStmtAggr

GetTableAliasForAttr

Notation:

function GetTableAliasForAttr(var pAttrName: String; pIncludeInvisible: Boolean): String;

Visibility:

Public

Description:

Detects the tablealias for the attribute/field described with *pAttrName*. The parameter *pAttrName* can be the name of the tablefield or the alias of the attribut. If the attribute/field was found the name of the tablefield will be setted in *pAttrName*.

The parameter *pIncludeInvisible* means that also fields will be searched which are *OnlyForSearch*.

See also AddStmtAttr, SetStmtAttr, SetStmtAggr

GetTableForAttr

Notation:

function GetTableForAttr(var pAttrName: String; pIncludeInvisible: Boolean): TSFStmtTable;

Visibility:

Public

Description:

Detects the tableobject for the attribute/field described with *pAttrName*. The parameter *pAttrName* can be the name of the tablefield or the alias of the attribut. If the attribute/field was found the name of the tablefield will be setted in *pAttrName*.

The parameter *pIncludeInvisible* means that also fields will be searched which are *OnlyForSearch*.

See also AddStmtAttr, SetStmtAttr, SetStmtAggr

HasConditions

Notation:

function HasConditions: Boolean;

Visibility:

Public

Description:

Checks the query has search conditions

See also AddConditionVal, AddConditionAttr, AddConditionIsNull, AddConditionIsNotNull, AddConditionType, AddConditionExists

GetConvertedValue

Notation:

function GetConvertedValue(pValue: Variant; pExplicitCast: Boolean = False; pEscapeLike: Boolean = False): String;

Visibility:

Public

Description:

Internal used to convert added values (p. e. with AddConditionVal) for the query.

See also TSFStmtDBDialectConv.ConvertValue

GetTypeForValue

Notation:

function GetTypeForValue(pValue: Variant): TSFStmtValueType;

Visibility:

Public

Description:

Internal used to identify the type of added values (p. e. with AddConditionVal).

See also TSFStmtDBDialectConv.ValueTypeForValue

ConvertValueInType

Notation:

function ConvertValueInType(var pValue: Variant; pType: TSFStmtValueType; pHandleArray: Boolean = False): Boolean;

Visibility:

Public

Description:

Internal used to convert values for import.

ConvertArrayValueToStr

Notation:

function ConvertArrayValueToStr(pValue: Variant): String;

Visibility:

Public

Description:

Internal used to convert arrays for export.

GetReferencedStmtByNamePath

Notation:

function GetReferencedStmtByNamePath(pNamePath: String): TSFStmt;

Visibility:

Public

Description:

Searches a instance from TSFStmt with help of the componentname.

Internal used when generate a imported query.

GetReferencedStmtForParent

Notation:

function GetReferencedStmtForParent(pNamePath: String; pParent: TComponent): TSFStmt;

Visibility:

Public

Description:

Searches a instance from TSFStmt with help of the componentname.

Internal used when generate a imported query.

GetReferencedStmtNamePath

Notation:

function GetReferencedStmtNamePath(pComp: TComponent = nil): String;

Visibility:

Public

Description:

Detectes name and path (parents) for a instance of TSFStmt.

Internal used when export query.

GetQuotedIdentifier

Notation:

function GetQuotedIdentifier(pIdentifier: String): String;

Visibility:

Public

Description:

Quotes a identifier if necessary.

See also QuoteType

SetUnion

Notation:

procedure SetUnion(pStmt: TSFStmt);

Visibility:

Public

Description:

Adds a UNION.

When using subselects don't free the object, this will be freed automatically (only when the object hasn't a owner).

HasUnion

Notation:

function HasUnion: Boolean;

Visibility:

Public

Description:

Checks statement has a UNION.

AssignStmt

Notation:

function AssignStmt: TSFStmt;

Visibility:

Public

Description:

Assigns the statement.

AssignStmtTo

Notation:

procedure AssignStmtTo(pDest: TSFStmt);

Visibility:

Public

Description:

Assigns the statement in *pDest*.

AttrDatabaseNameForAttrName

Notation:

function AttrDatabaseNameForAttrName(pTableAlias, pAttrName: String): String; overload;

function AttrDatabaseNameForAttrName(pAttrName: String; var pTable: TSFStmtTable): String; overload;

Visibility:

Public

Description:

Detects the name of the tablefield for the attribute/field described with *pAttrName*. The parameter *pAttrName* can be the name of the tablefield or the alias of the attribut. The parameter *pTableAlias* is optional, when setted it should be the tablealias or the tablename.

ListTables

Notation:

function ListTables: TObjectList<TSFStmtTable>;

Visibility:

Public

Description:

Collects all tableobjects.

See also SetBaseTable, SetTableJoin

ListAttributes

Notation:

function ListAttributes: TObjectList<TSFStmtAttr>;

Visibility:

Public

Description:

Collects all attributeobjects.

See also AddStmtAttr, SetStmtAttr, SetStmtAggr

ListAttributeParams

Notation:

function ListAttributeParams: TStrings;

Visibility:

Public

Description:

Lists all parameternames in query.

See also AddStmtAttr, TSFStmtAttr.AddItemParam

ListConditions

Notation:

function ListConditions: TObjectList<TSFStmtCondition>;

Visibility:

Public

Description:

Collects all conditionobjects which are not *Restricted*.

See also AddConditionVal, AddConditionAttr, AddConditionIsNull, AddConditionIsNotNull, AddConditionType, AddConditionExists

ListRestrictions

Notation:

function ListRestrictions: TObjectList<TSFStmtCondition>;

Visibility:

Public

Description:

Collects all conditionobjects which are *Restricted*.

See also AddConditionVal, AddConditionAttr, AddConditionIsNull, AddConditionIsNotNull, AddConditionType, AddConditionExists

ListOrder

Notation:

function ListOrder: TObjectList<TSFStmtAttr>;

Visibility:

Public

Description:

Collects all attributeobjects which are added for sorting.

See also AddOrderAttr

ListGroup

Notation:

function ListGroup: TObjectList<TSFStmtAttr>;

Visibility:

Public

Description:

Collects all attributeobjects which are added for grouping.

See also AddGroupAttr

ConfigStmtTimeValue

Notation:

function ConfigStmtTimeValue(pTime: TTime): TDateTime;

Visibility:

Public

Description:

Converts the given timevalue to a value which can be added to query.

See also TSFStmtAttr.AddItemValueTime, TSFStmtAttr.AddItemValueDate, TSFStmtAttr.AddItemValueDateTime

HasStmtDatePart

Notation:

function HasStmtDatePart(pDate: TDateTime): Boolean;

Visibility:

Public

Description:

Checks the given datetime-values has a date.

See also ConfigStmtTimeValue

HasStmtTimePart

Notation:

function HasStmtTimePart(pDate: TDateTime): Boolean;

Visibility:

Public

Description:

Checks the given datetime-values has a time.

See also ConfigStmtTimeValue

GetStmtDatePart

Notation:

function GetStmtDatePart(pDate: TDateTime): TDate;

Visibility:

Public

Description:

Detects the date from the given datetime-value.

See also ConfigStmtTimeValue

GetStmtTimePart

Notation:

function GetStmtTimePart(pDate: TDateTime): TTime;

Visibility:

Public

Description:

Detects the time from the given datetime-value.

See also ConfigStmtTimeValue

SaveToXmlDoc

Notation:

function SaveToXmlDoc: IXmlDocument;

Visibility:

Public

Description:

Saves the query in a Xml-Document.

On export for subselects only will be saved a reference, see GetReferencedStmtNamePath.

SaveToXmlStr

Notation:

procedure SaveToXmlStr(var pXmlStr: String);

Visibility:

Public

Description:

Saves the query in a Xml-String.

On export for subselects only will be saved a reference, see GetReferencedStmtNamePath.

LoadFromXml

Notation:

procedure LoadFromXml(pXmlStr: String; pSuspendRefs: Boolean = True);

Visibility:

Public

Description:

Loads the query from a Xml-String.

On import for subselects only will be added a reference, see GetReferencedStmtByNamePath. When generating the query the objects have to be available.

LoadFromXmlDoc

Notation:

procedure LoadFromXmlDoc(pXmlDoc: IXmlDocument; pSuspendRefs: Boolean = True);

Visibility:

Public

Description:

Loads the query from a Xml-Document.

On import for subselects only will be added a reference, see GetReferencedStmtByNamePath. When generating the query the objects have to be available.

Properties

BaseTable

Notation:

property BaseTable: TSFStmtTable read mBaseTable;

Visibility:

Public

Description:

The basetable, see SetBaseTable

GenerateLevel

Notation:

property GenerateLevel: Integer read mGenerateLevel;

Visibility:

Public

Description:

Internal used to generate subselects.

GenerateSubId

Notation:

property GenerateSubId: Integer read mGenerateSubId;

Visibility:

Public

Description:

Internal used to generate subselects.

GenerateUnionId

Notation:

property GenerateUnionId: Integer read mGenerateUnionId;

Visibility:

Public

Description:

Internal used to generate subselects.

QuoteType

Notation:

property QuoteType: TSFStmtQuoteType read mQuoteType write mQuoteType;

Visibility:

Public

Description:

The type how identifiers will be quoted.

See also TSFStmtQuoteType

StmtGenInfos

Notation:

property StmtGenInfos: TSFStmtGenInfos read mStmtGenInfos;

Visibility:

Public

Description:

Informations about generating last query. This informations are available after a query was generated.

See also TSFStmtGenInfos

UseDistinct

Notation:

property UseDistinct: Boolean read mUseDistinct write mUseDistinct;

Visibility:

Public

Description:

Defines query uses DISTINCT on SELECT clause.

LikeEscapeChar

Notation:

property LikeEscapeChar: String read mLikeEscapeChar write mLikeEscapeChar;

Visibility:

Public

Description:

When *AutoEscapeLike* is off, with *LikeEscapeChar* you can define the char which should be used for ESCAPE syntax (only if DBDialect supports ESCAPE). In this case all LIKE conditions will be escaped with the given char.

AutoEscapeLike

Notation:

property AutoEscapeLike: Boolean read mAutoEscapeLike write mAutoEscapeLike;

Visibility:

Public

Description:

When using *AutoEscapeLike* all LIKE conditons will be checked an escaped automatically (only if DBDialect support ESCAPE).

DBDialect

Notation:

property DBDialect: TSFStmtDBDialect read mDBDialect write mDBDialect;

Visibility:

Public

Description:

The dialect or database which will be used for generation queries.

See also TSFStmtDBDialect

Union

Notation:

property Union: TSFStmt read getUnion write SetUnion;

Visibility:

Public

Description:

References the instance for UNION.

When using subselects don't free the object, this will be freed automatically (only when the object hasn't a owner).

Events

OnBeforeGenSelect

Notation:

property OnBeforeGenSelect: TSFStmtGenSelectEvent read mOnBeforeGenSelect write mOnBeforeGenSelect;

Visibility:

Public

Description:

Will be fired before a SELECT query will be generated. When using this event consider that this event also will be used internally from TSFBusinessData/TSFDataSet (for their internal statement).

OnAfterGenSelect

Notation:

property OnAfterGenSelect: TSFStmtGenSelectEvent read mOnAfterGenSelect write mOnAfterGenSelect;

Visibility:

Public

Description:

Will be fired after a SELECT query has been generated. When using this event consider that this event also will be used internally from TSFBusinessData/TSFDataSet (for their internal statement).

OnBeforeGenDelete

Notation:

property OnBeforeGenDelete: TSFStmtGenSelectEvent read mOnBeforeGenDelete write mOnBeforeGenDelete;

Visibility:

Public

Description:

Will be fired before a DELETE statement will be generated.

OnAfterGenDelete

Notation:

property OnAfterGenDelete: TSFStmtGenSelectEvent read mOnAfterGenDelete write mOnAfterGenDelete;

Visibility:

Public

Description:

Will be fired after a DELETE statement has been generated.

OnBeforeGenUpdate

Notation:

property OnBeforeGenUpdate: TSFStmtGenSelectEvent read mOnBeforeGenUpdate write mOnBeforeGenUpdate;

Visibility:

Public

Description:

Will be fired before a UPDATE statement will be generated.

OnAfterGenUpdate

Notation:

property OnAfterGenUpdate: TSFStmtGenSelectEvent read mOnAfterGenUpdate write mOnAfterGenUpdate;

Visibility:

Public

Description:

Will be fired after a UPDATE statement has been generated.

OnBeforeGenInsert

Notation:

property OnBeforeGenInsert: TSFStmtGenSelectEvent read mOnBeforeGenInsert write mOnBeforeGenInsert;

Visibility:

Public

Description:

Will be fired before a INSERT statement will be generated.

OnAfterGenInsert

Notation:

property OnAfterGenInsert: TSFStmtGenSelectEvent read mOnAfterGenInsert write mOnAfterGenInsert;

Visibility:

Public

Description:

Will be fired after a INSERT statement has been generated.

OnGetDBDialectCls

Notation:

property OnGetDBDialectCls: TSFStmtGetDialectConvEvent read mOnGetDBDialectCls write mOnGetDBDialectCls;

Visibility:

Public

Description:

With this event you can integrate your own converter.

See also TSFStmtGetDialectConvEvent, TSFStmtDBDialectConv

TSFStmtTable

Description

Class to manage tables inside the query builder

Index

Functions

SetTableJoin

Notation:

function SetTableJoin(pTableAlias, pTableName, pSchema, pCatalog: String; pTableNo: Integer; pRelItems: TSFStmtJoinRelItems; pType: TSFStmtJoinType): TSFStmtTable; overload;

function SetTableJoin(pTableAlias: String; pStmt: TSFStmt; pTableNo: Integer; pRelItems: TSFStmtJoinRelItems; pType: TSFStmtJoinType): TSFStmtTable; overload;

function SetTableJoin(pTableAlias, pStmtName: String; pTableNo: Integer; pRelItems: TSFStmtJoinRelItems; pType: TSFStmtJoinType): TSFStmtTable; overload;

Visibility:

Public

Description:

Adds a join to a table - see TSFStmt.SetTableJoin.

GetJoinTableByAlias

Notation:

function GetJoinTableByAlias(pAlias: String; pSearchType: TSFStmtTableSearchType = stmtTableSearchAll): TSFStmtTable;

Visibility:

Public

Description:

Detects a join (or his table) with help of the alias or the tablename.

GetJoinTableAliasesForAttr

Notation:

function GetJoinTableAliasesForAttr(pAttr: String): Variant;

Visibility:

Public

Description:

Detects joins/tables for which the given attribute/field is definied in relation.

See also TSFStmt.TableJoinAliasesForAttr

ResetJoins

Notation:

procedure ResetJoins;

Visibility:

Public

Description:

Resets all joins.

See also TSFStmt.Reset

HasJoins

Notation:

function HasJoins: Boolean;

Visibility:

Public

Description:

Checks table has joins.

GetMaxTableNo

Notation:

function GetMaxTableNo: Integer;

Visibility:

Public

Description:

Detects last used table number.

See also TSFStmt.GetNextTableNo

GetTableDef

Notation:

function GetTableDef(pWithAlias: Boolean = True): String;

Visibility:

Public

Description:

Generates the SQL syntax for the table and her joins.

See also TSFStmt.GetSelectStmt, TSFStmt.GetDeleteStmt, TSFStmt.GetUpdateStmt, TSFStmt.GetInsertStmt

AssignStmtTable

Notation:

function AssignStmtTable(pDestStmt: TSFStmt): TSFStmtTable;

Visibility:

Public

Description:

Assigns the table (with joins).

See also AssignStmtTableJoins, TSFStmt.AssignStmt, TSFStmt.AssignStmtTo

AssignStmtTableJoins

Notation:

procedure AssignStmtTableJoins(pDest: TSFStmtTable);

Visibility:

Public

Description:

Assigns the joins of a table.

See also AssignStmtTable, TSFStmt.AssignStmt, TSFStmt.AssignStmtTo

GetJoinType

Notation:

function GetJoinType(pDest: TSFStmtTable): TSFStmtJoinType;

Visibility:

Public

Description:

Detects the type of the join for given table.

See also TSFStmtJoinType

ModifyJoinType

Notation:

procedure ModifyJoinType(pDest: TSFStmtTable; pTypeFrom, pTypeTo: TSFStmtJoinType);

Visibility:

Public

Description:

Changes the type of the join for given table.

See also TSFStmt.ModfiyTableJoinType

GetRelItemsForJoin

Notation:

function GetRelItemsForJoin(pDest: TSFStmtTable): TSFStmtJoinRelItems;

Visibility:

Public

Description:

Detects the attributes/fields for the relation in the join for given table.

See also TSFStmt.GetRelItemsForJoin

SetRelItemsForJoin

Notation:

procedure SetRelItemsForJoin(pDest: TSFStmtTable; pRelItems: TSFStmtJoinRelItems);

Visibility:

Public

Description:

Changes the attributes/fields for the relation in the join for given table.

Siehe auch TSFStmt.SetRelItemsForJoin

QuotedTableIdentifier

Notation:

function QuotedTableIdentifier: String;

Visibility:

Public

Description:

Quotes the identifier of the table if necessary.

See also QuotedTableName, QuotedTableSchema, QuotedTableCatalog

QuotedTableName

Notation:

function QuotedTableName: String;

Visibility:

Public

Description:

Quotes the name of the table if necessary.

Siehe auch QuotedTableIdentifier, QuotedTableSchema, QuotedTableCatalog

QuotedTableSchema

Notation:

function QuotedTableSchema: String;

Visibility:

Public

Description:

Quotes the schema of the table if necessary.

Siehe auch QuotedTableIdentifier, QuotedTableName, QuotedTableCatalog

QuotedTableCatalog

Notation:

function QuotedTableCatalog: String;

Visibility:

Public

Description:

Quotes the catalog of the table if necessary.

Siehe auch QuotedTableIdentifier, QuotedTableName, QuotedTableSchema

ListJoinTables

Notation:

procedure ListJoinTables(pLst: TObjectList<TSFStmtTable>; pRecursive: Boolean = True);

Visibility:

Public

Description:

Collects all joins of a table.

See also TSFStmt.ListTables

SaveToXmlTable

Notation:

procedure SaveToXmlTable(pXmlTable: TSFStmtTableXML);

Visibility:

Public

Description:

Saves/Exports the table to XML.

See also TSFStmt.SaveToXmlDoc, TSFStmt.SaveToXmlStr

LoadFromXmlTable

Notation:

procedure LoadFromXmlTable(pXmlTable: TSFStmtTableXML; pSuspendRefs: Boolean);

Visibility:

Public

Description:

Loads/Imports the table from a XML.

See also TSFStmt.LoadFromXml, TSFStmt.LoadFromXmlDoc

Properties

TableName

Notation:

property TableName: String read mTableName;

Visibility:

Public

Description:

The name of the table if references a table inside database.

TableStmt

Notation:

property TableStmt: TSFStmt read getTableStmt;

Visibility:

Public

Description:

References the subselect if it is a statement/query.

TableAlias

Notation:

property TableAlias: String read getTableAlias;

Visibility:

Public

Description:

The alias for the table.

TableAliasNested

Notation:

property TableAliasNested[pLevel, pSubId, pUnionId: Integer]: String read getTableAliasNested;

Visibility:

Public

Description:

The alias for the table on gerating as subselect.

TableNo

Notation:

property TableNo: Integer read mTableNo;

Visibility:

Public

Description:

The unique number inside the statement.

ParentStmt

Notation:

property ParentStmt: TSFStmt read mParentStmt;

Visibility:

Public

Description:

References the statement.

Schema

Notation:

property Schema: String read mSchema;

Visibility:

Public

Description:

The schema of the table if references a table inside database.

Catalog

Notation:

property Catalog: String read mCatalog;

Visibility:

Public

Description:

The catalog of the table if references a table inside database.

TableIdentifier

Notation:

property TableIdentifier: String read getTableIdentifier;

Visibility:

Public

Description:

The identifier of the table if references a table inside database.

TSFStmtTableJoin

Description

Class to manage joins inside the query builder

Index

Functions

GetJoinDef

Notation:

function GetJoinDef: String;

Visibility:

Public

Description:

Generates the SQL syntax for the join.

See also TSFStmt.GetSelectStmt, TSFStmt.GetDeleteStmt, TSFStmt.GetUpdateStmt, TSFStmt.GetInsertStmt, TSFStmtTable.GetTableDef

SaveToXmlRelation

Notation:

procedure SaveToXmlRelation(pXmlRelation: TSFStmtTableRelationXML);

Visibility:

Public

Description:

Saves/Exports the join in XML.

See also TSFStmt.SaveToXmlDoc, TSFStmt.SaveToXmlStr, TSFStmtTable.SaveToXmlTable

Properties

DestTable

Notation:

property DestTable: TSFStmtTable read mDestTable;

Visibility:

Public

Description:

Reference to the destination from the join.

TSFStmtAttr

Description

Class to manage attributes inside query builder. Attributes are the parts of a query which will be listed in SELECT clause (beside they are defined as *OnlyForSearch*). A attribute can include 1 to n items with different types.

Index

Functions

GetSelectDef

Notation:

function GetSelectDef: String;

Visibility:

Public

Description:

Generates the SQL syntax for attribute.

See also TSFStmt.GetSelectStmt

GetAttrDef

Notation:

function GetAttrDef(pWithSortType: Boolean = False; pWithAliases: Boolean = True; pExplicitCast: Boolean = False; pEscapeLike: Boolean = False): String;

Visibility:

Public

Description:

Generates the SQL syntax for attribute independent from position inside the query (p. e. for WHERE clause as search condition).

See also TSFStmt.GetSelectStmt, TSFStmt.GetDeleteStmt, TSFStmt.GetUpdateStmt, TSFStmt.GetInsertStmt

HasItems

Notation:

function HasItems: Boolean;

Visibility:

Public

Description:

Checks attribute has items.

IsSingleItem

Notation:

function IsSingleItem: Boolean;

Visibility:

Public

Description:

Checks attribute has only 1 item.

IsSingleDBFieldItem

Notation:

function IsSingleDBFieldItem: Boolean;

Visibility:

Public

Description:

Checks attribute has only 1 item which is a databasefield.

IsSingleDBFieldUndefined

Notation:

function IsSingleDBFieldUndefined: Boolean;

Visibility:

Public

Description:

Checks attribute has only 1 item which is a undefined databasefield (means = *).

AddItem

Notation:

procedure AddItem(pType: TSFStmtAttrItemType; pTable: TSFStmtTable; pItemValue: Variant; pAggr: String);

Visibility:

Public

Description:

Adds a item from given type.

AddItemDbFld

Notation:

procedure AddItemDbFld(pTable: TSFStmtTable; pAttrName, pAggr: String);

Visibility:

Public

Description:

Adds a item from type *databasefield*.

AddItemValue

Notation:

procedure AddItemValue(pValue: Variant);

Visibility:

Public

Description:

Adds a item from type *value*.

AddItemValueDateTime

Notation:

procedure AddItemValueDateTime(pValue: TDateTime);

Visibility:

Public

Description:

Adds a item from type *datetime*.

AddItemValueDate

Notation:

procedure AddItemValueDate(pValue: TDate);

Visibility:

Public

Description:

Adds a item from type *date*.

AddItemValueTime

Notation:

procedure AddItemValueTime(pValue: TTime);

Visibility:

Public

Description:

Adds a item from type *time*.

AddItemStmt

Notation:

procedure AddItemStmt(pStmt: TSFStmt); overload;

procedure AddItemStmt(pStmtName: String); overload;

Visibility:

Public

Description:

Adds a item from type *stmt (subselect)*.

AddItemAggrFunc

Notation:

procedure AddItemAggrFunc(pAggrFunc: string);

Visibility:

Public

Description:

Adds a item from type *aggregate* (p. e. COUNT, SUM, MIN, MAX, etc.).

AddItemParam

Notation:

procedure AddItemParam(pParamName: String);

Visibility:

Public

Description:

Adds a item from type *parameter*.

AddItemOperator

Notation:

procedure AddItemOperator(pType: TSFStmtAttrItemOperatorType);

Visibility:

Public

Description:

Adds a item from type *operator*.

See also TSFStmtAttrItemOperatorType

AddItemBracket

Notation:

procedure AddItemBracket(pType: TSFStmtAttrItemBracketType);

Visibility:

Public

Description:

Adds a item from type *bracket*.

See also TSFStmtAttrItemBracketType

AddItemDynamic

Notation:

procedure AddItemDynamic(pValue: String);

Visibility:

Public

Description:

Adds a item from type *dynamic*. The parameter *pValue* defines the userdefined text.

AssignStmtAttr

Notation:

function AssignStmtAttr(pDestStmt: TSFStmt): TSFStmtAttr;

Visibility:

Public

Description:

Assigns the attribut (with items).

See also TSFStmt.AssignStmt, TSFStmt.AssignStmtTo

SetItemParamNamesToList

Notation:

procedure SetItemParamNamesToList(pLst: TStrings);

Visibility:

Public

Description:

Writes the names of all items from type *parameter* in given list.

See also TSFStmt.ListAttributeParams

SaveToXmlAttr

Notation:

procedure SaveToXmlAttr(pXmlAttr: TSFStmtAttrXML);

Visibility:

Public

Description:

Saves/Exports the attribute to XML.

See also TSFStmt.SaveToXmlDoc, TSFStmt.SaveToXmlStr

LoadFromXmlAttr

Notation:

procedure LoadFromXmlAttr(pXmlAttr: TSFStmtAttrXML; pSuspendRefs: Boolean);

Visibility:

Public

Description:

Loads/Imports the attribute from XML.

See also TSFStmt.LoadFromXml, TSFStmt.LoadFromXmlDoc

Properties

ParentStmt

Notation:

property ParentStmt: TSFStmt read mParentStmt;

Visibility:

Public

Description:

Reference to the statement.

AttrName

Notation:

property AttrName: String read mAttrName;

Visibility:

Public

Description:

The name of the attribute which is used as alias when generating for SELECT clause.

DBAttrName

Notation:

property DBAttrName: String read getDBAttrName;

Visibility:

Public

Description:

If attribute has only 1 item from type *databasefield*, with this property you can get the name of the databasefield.

TSFStmt.SetStmtAttr

DBAttrTable

Notation:

property DBAttrTable: TSFStmtTable read getDBAttrTable;

Visibility:

Public

Description:

If attribute has only 1 item from type *databasefield*, with this property you can get a reference to the table.

TSFStmt.SetStmtAttr

DBAttrAggr

Notation:

property DBAttrAggr: String read getDBAttrAggr;

Visibility:

Public

Description:

If attribute has only 1 item from type *databasefield*, with this property you can get the aggregate if setted.

See also TSFStmt.SetStmtAggr

SortType

Notation:

property SortType: TSFStmtSortType read mSortType write mSortType;

Visibility:

Public

Description:

The sorttype for the attribute.

See also TSFStmt.AddOrderAttr, TSFStmtSortType

OnlyForSearch

Notation:

property OnlyForSearch: Boolean read mOnlyForSearch;

Visibility:

Public

Description:

Means the attribute was only added for search conditions. For thus attributs you can add search conditions but they will not be listed in SELECT clause.

See also TSFStmt.AddConditionVal, TSFStmt.AddConditionAttr, TSFStmt.AddConditionIsNull, TSFStmt.AddConditionIsNotNull, TSFStmt.AddStmtAttr, TSFStmt.SetStmtAttr, TSFStmt.SetStmtAggr

Items

Notation:

property Items: TObjectList<TSFStmtAttrItem> read mItems;

Visibility:

Public

Description:

Reference to the items from the attribute.

TSFStmtAttrItem

Description

Class to manage items for a attribute. P. e. with items you also can include arithmetic operations (field1 + field2 + 5)

Index

Functions

GetAttrItemDef

Notation:

function GetAttrItemDef(pWithAlias: Boolean; pExplicitCast, pEscapeLike: Boolean): String;

Visibility:

Public

Description:

Geneates the SQL syntax for the item.

See also TSFStmt.GetSelectStmt, TSFStmt.GetDeleteStmt, TSFStmt.GetUpdateStmt, TSFStmt.GetInsertStmt, TSFStmtAttr.GetAttrDef

SaveToXmlAttrItem

Notation:

procedure SaveToXmlAttrItem(pXmlAttrItem: TSFStmtAttrItemXML);

Visibility:

Public

Description:

Saves/Exports the item to XML.

See also TSFStmt.SaveToXmlDoc, TSFStmt.SaveToXmlStr, TSFStmtAttr.SaveToXmlAttr

Properties

Attr

Notation:

property Attr: TSFStmtAttr read mAttr;

Visibility:

Public

Description:

Reference to the attribute.

ItemType

Notation:

property ItemType: TSFStmtAttrItemType read mType;

Visibility:

Public

Description:

Type of the item. See TSFStmtAttrItemType

Table

Notation:

property Table: TSFStmtTable read mTable;

Visibility:

Public

Description:

If item is from type *databasefield*, with this property you can get a reference to the table.

ItemRef

Notation:

property ItemRef: Variant read getItemRef;

Visibility:

Public

Description:

Depending on type this property stores the variabel identifier. P. e. when item is a databasefield ItemRef stores the name of the databasefield.

Aggr

Notation:

property Aggr: String read mAggr;

Visibility:

Public

Description:

If item is from type *databasefield*, with this property you can get the aggregate (if setted). For more komplex definitions you also can add aggregates with help of a special type.

TSFStmtCondition

Description

Class to manage search conditions (WHERE-Klausel) inside a query builder.

Index

Functions

GetConditionDef

Notation:

function GetConditionDef(pWithAliases: Boolean): String; virtual;

Visibility:

Public

Description:

Generates the SQL syntax for the condition.

See also TSFStmt.GetSelectStmt, TSFStmt.GetDeleteStmt, TSFStmt.GetUpdateStmt

AssignStmtCondition

Notation:

function AssignStmtCondition(pDestStmt: TSFStmt): TSFStmtCondition; virtual;

Visibility:

Public

Description:

Assigns the condition.

See also TSFStmt.AssignStmt, TSFStmt.AssignStmtTo

SaveToXmlCondition

Notation:

procedure SaveToXmlCondition(pXmlCond: TSFStmtCondXML); virtual;

Visibility:

Public

Description:

Saves/Exports the condition to XML.

See also TSFStmt.LoadFromXml, TSFStmt.LoadFromXmlDoc

Properties

CondType

Notation:

property CondType: TSFStmtConditionType read mType;

Visibility:

Public

Description:

The type of the condition.

See TSFStmtConditionType

StmtAttr

Notation:

property StmtAttr: TSFStmtAttr read mStmtAttr;

Visibility:

Public

Description:

Depending on type (stmtCondTypeAttribute, stmtCondTypeValue, stmtCondTypeIsNull, stmtCondTypeIsNotNull) with this property you can get a reference to the attribute.

CondValue

Notation:

property CondValue: Variant read mValue;

Visibility:

Public

Description:

The value for the condition. Depending on type p. e. this property also can store a reference to a attribute.

CondOperator

Notation:

property CondOperator: String read mOperator;

Visibility:

Public

Description:

The operator for the condition.

See also Constants

TSFStmtConditionExists

Description

Class to manage EXISTS conditions (WHERE clause) inside a query builder.

Index

Functions

GetConditionDef

Notation:

function GetConditionDef(pWithAliases: Boolean): String; override;

Visibility:

Public

Description:

Generates SQL syntax for the EXISTS condition.

See also TSFStmt.GetSelectStmt, TSFStmt.GetDeleteStmt, TSFStmt.GetUpdateStmt

AssignStmtCondition

Notation:

function AssignStmtCondition(pDestStmt: TSFStmt): TSFStmtCondition; override;

Visibility:

Public

Description:

Assigns the EXISTS condition.

See also TSFStmt.AssignStmt, TSFStmt.AssignStmtTo

SaveToXmlCondition

Notation:

procedure SaveToXmlCondition(pXmlCond: TSFStmtCondXML); override;

Visibility:

Public

Description:

Saves/Exports the EXISTS condition to XML.

See also TSFStmt.LoadFromXml, TSFStmt.LoadFromXmlDoc

Properties

DestStmt

Notation:

property DestStmt: TSFStmt read getDestStmt;

Visibility:

Public

Description:

Reference to statement which is the destination for the condition (subselect).

SrcTable

Notation:

property SrcTable: TSFStmtTable read mSrcTable;

Visibility:

Public

Description:

Reference to table (inside own statement) which is the source for the condition.

DestTable

Notation:

property DestTable: TSFStmtTable read getDestTable;

Visibility:

Public

Description:

Table inside *DestStmt* for linking with *SrcTable*.

RelItems

Notation:

property RelItems: TSFStmtJoinRelItems read mRelItems;

Visibility:

Public

Description:

Attributes/Fields form linking *SrcTable* with *DestTable*.

See also TSFStmtJoinRelItem, TSFStmtJoinRelItemType

TSFStmtDBDialectConv

Description

Class for database dependent convertions. The creation from objects of this class depends on the DBDialect, p. e. when using Oracle a object of TSFStmtDBDialectConvOra will be created.

Furthermore you also can write own converters (depends on TSFStmtDBDialectConv). A own converter you can integrate with help of the event TSFStmt.OnGetDBDialectCls.

Index

Functions

ConvertValue

Notation:

function ConvertValue(pValue: Variant; pUsedDecSeparator: String; pExplicitCast, pEscapeLike: Boolean): String;

Visibility:

Public

Description:

Converts a value for a SQL query.

See also TSFStmt.GetConvertedValue

ValueTypeForValue

Notation:

function ValueTypeForValue(pValue: Variant): TSFStmtValueType;

Visibility:

Public

Description:

Detects type of the given value.

See also TSFStmt.GetTypeForValue, TSFStmtValueType

EscapeLike

Notation:

function EscapeLike(var pValue: String): String;

Visibility:

Public

Description:

Checks and escapes LIKE conditions (if DBDialect supports ESCAPE syntax).

See also TSFStmt.AutoEscapeLike

GetNextAutoValue

Notation:

function GetNextAutoValue(pSeqName: String = ''): String; virtual;

Visibility:

Public

Description:

Generates query to get next autovalue.

See also TSFStmt.GetNextAutoValueStmt

GetLastAutoValue

Notation:

function GetLastAutoValue(pSeqName: String = ''): String; virtual;

Visibility:

Public

Description:

Generates query to get last inserted autovalue.

See also TSFStmt.GetLastAutoValueStmt

GetCanSelectWithoutTable

Notation:

class function GetCanSelectWithoutTable(var pTableName: String): Boolean; virtual;

Visibility:

Public

Description:

Detects SELECT syntax without a table is supported.

See also TSFStmt.GetDBDialectCanSelWithoutTab

GetCanSelectInFrom

Notation:

class function GetCanSelectInFrom(pDBDialect: TSFStmtDBDialect): Boolean; virtual;

Visibility:

Public

Description:

Detects SELECT syntax with a subselect in FROM clause is supported.

See also TSFStmt.GetDBDialectCanSubInFrom

GetNeedTableOnSubInFrom

Notation:

class function GetNeedTableOnSubInFrom: Boolean; virtual;

Visibility:

Public

Description:

Detects SELECT syntax with a subselect in FROM clause which (the subselect) doesn't referencing a table is supported.

See also TSFStmt.GetDBDialectNeedTableOnSubInFrom

GetStartQuote

Notation:

class function GetStartQuote: String; virtual;

Visibility:

Public

Description:

The database dependent quote using at first.

GetEndQuote

Notation:

class function GetEndQuote: String; virtual;

Visibility:

Public

Description:

The database dependent quote using at last.

GetLikeWildcardSingle

Notation:

class function GetLikeWildcardSingle: String; virtual;

Visibility:

Public

Description:

Defines the wildcard single for searching with LIKE. In baseclass this char is "_".

See also TSFStmt.GetDBDialectLikeWildcardSingle

GetLikeWildcardMany

Notation:

class function GetLikeWildcardMany: String; virtual;

Visibility:

Public

Description:

Defines the wildcard many for searching with LIKE. In baseclass this char is "%".

See also TSFStmt.GetDBDialectLikeWildcardMany

SupportsLikeEscape

Notation:

class function SupportsLikeEscape: Boolean; virtual;

Visibility:

Public

Description:

Defines ESCAPE syntax for searching with LIKE is supported. Through ESCAPE syntax you can search for strings which includes the wildcard char.

See also TSFStmt.GetDBDialectLikeSupportsEscape

Properties

Stmt

Notation:

property Stmt: TSFStmt read mStmt;

Visibility:

Protected

Description:

Reference to statement.

TSFBDSFormatOptions

Description

Class with options for formatting values.

Index

DisplayFmtCurrency

DisplayFmtDate
DisplayFmtDateTime
DisplayFmtFloat
DisplayFmtTime
EditFmtCurrency
EditFmtFloat
EditMaskDate
EditMaskDateTime
EditMaskTime
QuoteType

Properties

DisplayFmtDateTime

Notation:

property DisplayFmtDateTime: String read mDisplayFmtDateTime write mDisplayFmtDateTime;

Visibility:

Published

Description:

Displayformat for datetime values.

DisplayFmtDate

Notation:

property DisplayFmtDate: String read mDisplayFmtDate write mDisplayFmtDate;

Visibility:

Published

Description:

Displayformat for date values (without time).

DisplayFmtTime

Notation:

property DisplayFmtTime: String read mDisplayFmtTime write mDisplayFmtTime;

Visibility:

Published

Description:

Displayformat for time values (without date).

DisplayFmtFloat

Notation:

property DisplayFmtFloat: String read mDisplayFmtFloat write mDisplayFmtFloat;

Visibility:

Published

Description:

Displayformat for float values.

DisplayFmtCurrency

Notation:

property DisplayFmtCurrency: String read mDisplayFmtCurrency write mDisplayFmtCurrency;

Visibility:

Published

Description:

Displayformat for currency values.

EditMaskDateTime

Notation:

property EditMaskDateTime: TEditMask read mEditMaskDateTime write mEditMaskDateTime;

Visibility:

Published

Description:

Mask for changing datetime values.

EditMaskDate

Notation:

property EditMaskDate: TEditMask read mEditMaskDate write mEditMaskDate;

Visibility:

Published

Description:

Mask for changing date (without time) values.

EditMaskTime

Notation:

property EditMaskTime: TEditMask read mEditMaskTime write mEditMaskTime;

Visibility:

Published

Description:

Mask for changing time (without date) values.

EditFmtFloat

Notation:

property EditFmtFloat: String read mEditFmtFloat write mEditFmtFloat;

Visibility:

Published

Description:

Editformat for float values.

EditFmtCurrency

Notation:

property EditFmtCurrency: String read mEditFmtCurrency write mEditFmtCurrency;

Visibility:

Published

Description:

Editformat for currency values.

QuoteType

Notation:

property QuoteType: TSFBDSQuoteType read mQuoteType write mQuoteType;

Visibility:

Published

Description:

The type how identifiers will be quoted.

Types/Constants

Index

Constants

```
SFSTMT_OP_EQUAL = '=';
SFSTMT_OP_NOTEQUAL = '<>';
SFSTMT_OP_LESSEQUAL = '<=';
SFSTMT_OP_GREATEREQUAL = '>=';
SFSTMT_OP_LESS = '<';
SFSTMT_OP_GREATER = '>';
SFSTMT_OP_LIKE = 'LIKE';
SFSTMT_OP_NOT_LIKE = 'NOT LIKE';
SFSTMT_OP_IN = 'IN';
SFSTMT_OP_NOT_IN = 'NOT IN';
SFSTMT_OP_EXISTS = 'EXISTS';
SFSTMT_OP_NOT_EXISTS = 'NOT EXISTS';

SFSTMTAGGR_COUNT = 'count';
SFSTMTAGGR_MIN = 'min';
SFSTMTAGGR_MAX = 'max';
SFSTMTAGGR_AVG = 'avg';
SFSTMTAGGR_SUM = 'sum';
```

Types

```
TSFConnectionType =
   (ctFireDac,
     ctDBExpress,
     ctInterbase,
     ctADO
   );



TSFConnectionDBType =
   (dbtDB2,
     dbtFB,
     dbtIB,
     dbtMSSQL,
     dbtMySQL,
     dbtOra,
     dbtSQLLite,
     dbtPG,
     dbtMSAcc,
     dbtAdvantage,
     dbtInformix,
     dbtAnywhere,
     dbtSybase,
     dbtUnknown
   );
```

```
TSFQueryActionType =
   (
     atSelect,
     atModify
   );


TSFStmtJoinType =
   (stmtJoinTypeInner,
     stmtJoinTypeOuter,
     stmtJoinTypeROuter,
     stmtJoinTypeNone);


TSFStmtJoinRelItemType =
   (stmtJoinRelItemAttr,
     stmtJoinRelItemValue);


TSFStmtJoinRelItem = record
   riSrcType: TSFStmtJoinRelItemType;
   riSrcValue: Variant;
   riDestType: TSFStmtJoinRelItemType;
   riDestValue: Variant;
 end;


TSFStmtJoinRelItems = Array of TSFStmtJoinRelItem;


TSFStmtAttrItemType =
   (stmtAttrItemTypeDbField,
     stmtAttrItemTypeValue,
     stmtAttrItemTypeParameter,
     stmtAttrItemTypeStmt,
     stmtAttrItemTypeAggrFunc,
     stmtAttrItemTypeOpPlus,
     stmtAttrItemTypeOpMinus,
     stmtAttrItemTypeOpMultiply,
     stmtAttrItemTypeOpDivide,
     stmtAttrItemTypeBracketOpen,
     stmtAttrItemTypeBracketClose,
     stmtAttrItemTypeDynamic);
```

```
TSFStmtAttrItemOperatorType = stmtAttrItemTypeOpPlus..stmtAttrItemTypeOpDivide;


TSFStmtAttrItemBracketType =
stmtAttrItemTypeBracketOpen..stmtAttrItemTypeBracketClose;


TSFStmtAttrItemValueType = stmtAttrItemTypeValue..stmtAttrItemTypeParameter;


TSFStmtConditionType =
   (stmtCondTypeValue,
     stmtCondTypeAttribute,
     stmtCondTypeOpen,
     stmtCondTypeClose,
     stmtCondTypeAnd,
     stmtCondTypeOr,
     stmtCondTypeIsNull,
     stmtCondTypeIsNotNull,
     stmtCondTypeUndefined);


TSFStmtValueType = (
   stmtValTypeNumeric,
   stmtValTypeDate,
   stmtValTypeTime,
   stmtValTypeDateTime,
   stmtValTypeBool,
   stmtValTypeString,
   stmtValTypeOther
 );


TSFStmtSortType =
   (stmtSortTypeAsc,
     stmtSortTypeDesc);


TSFStmtGenInfo =
   (stmtGenSelect,
     stmtGenFrom,
     stmtGenWhere,
     stmtGenGroup,
     stmtGenOrder);
```

```
TSFStmtGenInfos = set of TSFStmtGenInfo;


TSFStmtDBDialect =
   (stmtDBDDflt,
     stmtDBDOra,
     stmtDBDDB2,
     stmtDBDIfx,
     stmtDBDAcc,
     stmtDBIB,
     stmtDBFB,
     stmtDBDSQLite,
     stmtDBDPG,
     stmtDBDMySQL,
     stmtDBDMSSQL,
     stmtDBDAdvantage,
     stmtDBDAnywhere,
     stmtDBDSybase);


TSFStmtTableSearchType =
   (stmtTableSearchAll,
     stmtTableSearchOnlyAlias,
     stmtTableSearchOnlyIdentifier,
     stmtTableSearchOnlyName);


TSFStmtTableSingleSearchTypes = stmtTableSearchOnlyAlias..stmtTableSearchOnlyName;


TSFStmtQuoteType = (
   stmtQuoteTypeAuto,
   stmtQuoteTypeAll,
   stmtQuoteTypeNone
 );


TSFBDSRecordUpdateState = (
     usUnmodified,
     usInserted,
     usModified,
     usDeleted
   );
```

```
TSFBDSRecordCompareResult = (
    compareResultLess,
    compareResultEqual,
    compareResultGreater,
    compareResultUndefined
  );


TSFBDSRefreshMode = (
    refreshModeRow,
    refreshModeFull
  );


TSFBDSAutoValueOption = (
    avoExecute,
    avoNeedSequence,
    avoNeedTable,
    avoExecWhenAuto,
    avoPreventWhenAuto,
    avoExecWhenExplicitByDBMS,
    avoPreventWhenExplicitByDBMS
  );


TSFBDSAutoValueOptions = set of TSFBDSAutoValueOption;


TSFBDSAutoValueGetMode = (
    avGMAfterInsert,
    avGMBeforePost,
    avGMAfterPost
  );


TSFBDSExecParamsType =
  (exPrmsTypeSelect,
    exPrmsTypeDelete);
```

Functions/Events

```
TSFConnectorDSCreatedEvt = procedure(pDataSet: TDataSet; pActionType:
TSFQueryActionType) of object;
```

TSFStmtGenSelectEvent = procedure(pStmt: TSFStmt; pLevel, pSubId, pUnionId: Integer) of object;

TSFStmtGetDialectConvEvent = function(pDBDialect: TSFStmtDBDialect): TSFStmtDBDialectConvCls of object;

TSFBDSSetParamsEvt = procedure(pType: TSFBDSExecParamsType; pParams: TCollection) of object;

TSFBSDRecordCompareEvent = function(CompareRecordFrom, CompareRecordTo: TSFBDSCompareRecord): TSFBDSRecordCompareResult of object;

TSFBDSGetAutoValueCls = function(pFieldName: String; pAutoDetected: Boolean): TSFBDSAutoValueGeneratorCls;

TSFBusinessDataChanged = procedure(pOldDS, pNewDS: TSFBusinessData) of object;