Delphi Advanced Programming Technology

CHAPTER 8 USING DATA ACCESS COMPONENTS AND TOOLS

Professor Zhaoyun Sun

INTRODUCTION

- This chapter describes how to use key Delphi features and tools when building database applications, including:

  - The *TSession* component.
  - Dataset components (*TTable* and *TQuery*), their properties, and their methods.
  - *TDataSource* components, their properties, and their methods.
  - *TField* objects, their properties, and their methods.
  - The Fields Editor to instantiate and control *TField* objects.
  - *TReport* and *TBatchMove* components.

INTRODUCTION

- This chapter provides an overview and general description of data access components in the context of application development.

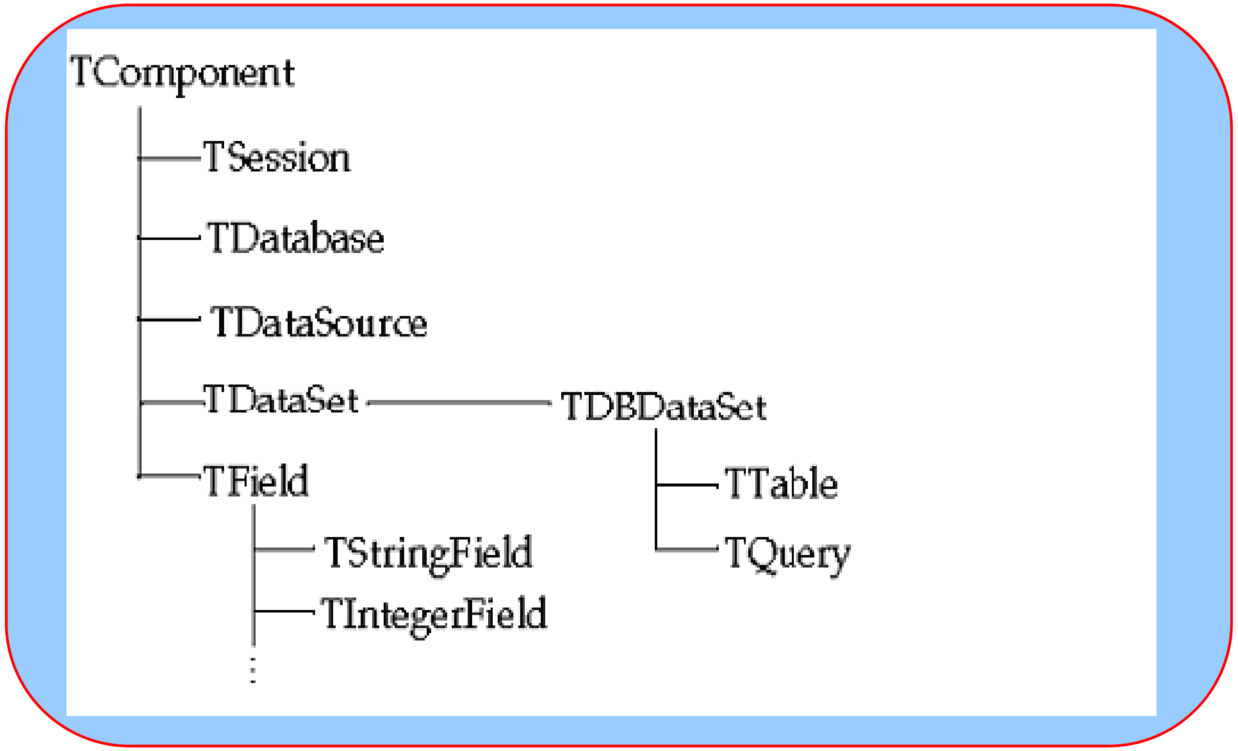
8.1 Database components hierarchy

- The Delphi database component hierarchy is important to show the properties, methods, and events inherited by components from their ancestors.

- The most important database components are forms either at design time or run time.

8.1 Database components hierarchy

Delphi Data Access components hierarchy



8.2 Using the TSession component

Controlling database connections

TSession **provides global control over database connections for an application.**

Getting database information

- *TSession* has a number of methods that enable an application to get database-related information.

- Each method takes a *TStrings* component as its parameter and returns into a *TStrings* the specified information.

Getting database information

TSession methods

Method	Remarks
GetAliasNames	Defined BDE alias names.
GetAliasParams	Parameters for the specified BDE alias.
GetDatabaseNames	Database names and BDE aliases defined.
GetDriverNames	Names of BDE drivers installed.
GetDriverParams	Parameters for the specifie BDE driver.
GetTableNames	All table names in the specified database.

8.3 Using datasets

- *TTable* and *TQuery* component classes are descended from *TDataSet* through *TDBDataSet*.

- These component classes share a number of inherited properties, methods, and events.

Dataset states

Dataset states

State	Description
Inactive	The dataset is closed.
Browse	The default state when a dataset is opened. Records can be viewed but not changed or inserted.
Edit	Enables the current row to be edited.
Insert	Enables a new row to be inserted. A call to Post inserts a new row.
SetKey	Enables FindKey, GoToKey, and GoToNearest to search for values in database tables. These methods only pertain to TTable components. For TQuery, searching is done with SQL syntax.
CalcFields	Mode when the OnCalcFields event is executed; prevents any changes to fields other than calculated fields. Rarely used explicitly.

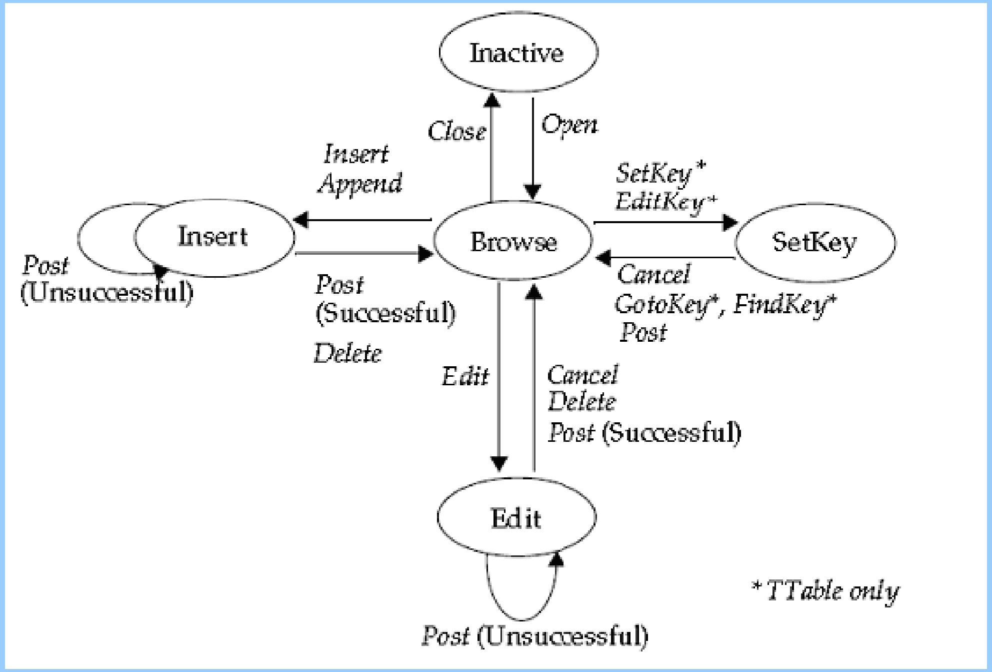
Dataset states

- An application can put a dataset into most states by calling the method corresponding to the state.

- For example, an application can put Table1 in Insert state by calling Table1.Insert or Edit state by calling Table1.Edit.

Dataset states

Dataset state diagram



8.3 Using datasets

- Opening and closing datasets
- Navigating datasets
- The Next and Prior methods
- The First and Last methods
- The BOF and EOF properties
- The MoveBy function

Modifying data in datasets

Methods to insert,update and delete data in data sets

Method	Description
Edit	Puts the dataset into Edit state. If a dataset is already in Edit or Insert state, a call to *Edit* has no effect.
Append	Posts any pending data, moves current record to the end of the dataset, and puts the dataset in Insert state.
Insert	Posts any pending data, and puts the dataset in Insert state.
Post	Attempts to post the new or altered record to the database. If successful, the dataset is put in Browse state; if unsuccessful, the dataset remains in its current state.
Cancel	Cancels the current operation and puts the dataset into Browse state.
Delete	Deletes the current record and puts the dataset in Browse state.

8.3 Using datasets

- Posting data to the database
- Disabling, enabling, and refreshing data-aware controls

– The *DisableControls* method disables all data-aware controls linked to a dataset. This method should be used with caution.

Using dataset events

- Datasets have a number of events that enable an application to perform validation, compute totals, and perform other tasks depending on the method performed by the dataset.

- The events are listed in the following table.

Using dataset events

Dataset events

Event	Description
BeforeOpen, AfterOpen	Called before/after a dataset is opened.
BeforeClose, AfterClose	Called before/after a dataset is closed.
BeforeInsert, AfterInsert	Called before/after a dataset enters Insert state.
BeforeEdit, AfterEdit	Called before/after a dataset enters Edit state.
BeforePost, AfterPost	Called before/after changes to a table are posted.
BeforeCancel, AfterCancel	Called before/after the previous state is canceled.
BeforeDelete, AfterDelete	Called before/after a record is deleted.
OnNewRecord	Called when a new record is created; used to set default values.
OnCalcFields	Called when calculated fields are calculated.

Abort a method

- For example, the following code confirms a delete operation:

```
    procedure TForm1.TableBeforeDelete (Dataset:
     TDataset);
    begin
    if MessageDlg('Delete This Record?',
mtConfirmation, mbYesNoCancel, 0) = mrYes
    then
Abort;
    end;
```

8.4 Using TTable

- *TTable* is one of the most important database component classes.

- Along with the other dataset component class, *TQuery*, it enables an application to access a database table.

- This section describes the most important properties that are unique to *TTable*.

Specifying the database table

- *TableName* specifies the name of the database table to which the *TTable* component is linked.

- You can set this property at design time through the Object Inspector.

Specifying the database table

- The ***DatabaseName*** property specifies where Delphi will look for the specified database table.

- It can be a BDE alias, an explicit specification, or the ***DatabaseName*** defined by any ***TDatabase*** component in the application.

Using Goto functions

- The *GoToKey* and *GoToNearest* methods enable an application to search a database table using a key.

Using Goto functions

- For example, the following code could be used in the *OnClick* event of a button:

```
procedure TSearchDemo.SearchExactClick(Sender:
          TObject);
begin
  Table1.SetKey; {First field is the key}
  Table1.Fields[0].AsString := Edit1.Text;
  Table1.GoToKey;
end;
```

Using Goto functions

- The first line of code after begin puts *Table1* in SetKey state.

- This indicates that the following assignment to the table's *Fields* property specifies a search value.

- The first column in the table, corresponding to *Fields*, is the index.

Using Goto functions

```
Table1.SetKey;
Table1.Fields[0].AsString :=
'Smith';
if not Table1.GotoKey
  then ShowMessage('Record
Not Found');
```

Using Goto functions

```
Table1.SetKey;
Table1.Fields[0].AsString := 'Sm';
Table1.GoToNearest;
```

- If a record exists with "Sm" as the first two characters, the cursor will be positioned on that record.

- Otherwise, the position of the cursor does not change and *GoToNearest* returns False.

Using Goto functions

```
Table1.IndexName := 'CityIndex';
Table1.Open;
Table1.SetKey;
Table1.FieldByName('City').AsStrin
g := Edit1.Text;
Table1.GoToNearest;
```

Using Find functions

- The Find functions, *FindKey* and *FindNearest* provide easy way to search a table.

- They combine the functionality of *SetKey*, field assignment, and Goto functions into a single statement.

Using Find functions

- ***FindKey*** **is similar to** ***GotoKey*****:**
  - **It will put a table in search mode (SetKey state).**
  - **It will find the record in the table that matches the specified values. If a matching record is found, it moves the cursor there, and returns True.**
  - **If a matching record is not found, it does not move the cursor, and returns False.**

Using Find functions

- For example, if Table1 is indexed on its first column, then the statement:

```
Table1.FindKey([Edit1.Text]);
```

- will perform the same function as the three statements:

```
Table1.SetKey; {First field is the key}
Table1.Fields[0].AsString := Edit1.Text;
Table1.GoToKey;
```

Indexes

- An *index* determines how records are sorted when a Delphi application displays data.

- By default, Delphi displays data in ascending order, based on the values of the primary index column(s) of a table.

The Exclusive property

- The Exclusive property indicates whether to open the table with an exclusive lock.

- If True, no other user will be able to access it at the same time. You cannot open a table in Exclusive mode if another user is currently accessing the table.

The ReadOnly and CanModify properties

- Before opening a *TTable*, set *ReadOnly* False to request read and write privileges for the dataset.

- Set *ReadOnly* to True to request read-only privileges for the dataset.

- Depending on the characteristics of the underlying table, the request for read and write privileges may or may not be granted by the database.

The ReadOnly and CanModify properties

- *CanModify* is a read-only property of datasets that reflects the actual rights granted for the dataset. When *ReadOnly* is True, *CanModify* will automatically be set to False.

- When *ReadOnly* is False, *CanModify* will be True if the database allows read and write privileges for the dataset and the underlying table. When *CanModify* is False, then the table is read-only, and the dataset cannot be put into Edit or Insert state.

8.5 Creating master-detail forms

- The *MasterSource* and *MasterFields* are used to define one-to-many relationships between two tables.

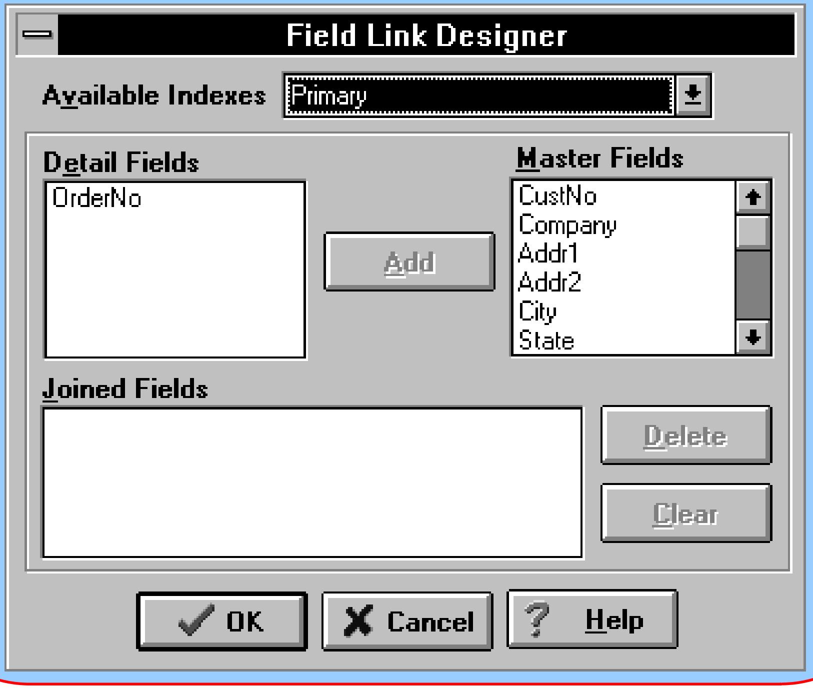
- The *MasterSource* property is used to specify a data source from which the table will get data for the master table.

The Field Link Designer

- At design time, when you double-click (or click on the ellipsis button) on the *MasterFields* property in the Object Inspector, the Field Link Designer dialog box opens.

The Field Link Designer

Field Link designer



The Field Link Designer

- The Field Link Designer provides a visual way to link master and detail tables.

- The Available Indexes combo box shows the currently selected index by which to join the two tables.

The Field Link Designer

- For Paradox tables, this will be “Primary” by default, indicating that the primary index of the detail field will be used.

- Any other named indices defined on the table will be shown in the drop-down list. Select the field you want to use to link the detail table in the Detail Fields list, the field to link the master table in the Master Fields list, and then choose Add.

The Field Link Designer

- The selected fields will be displayed in the Joined Fields list box.

8.6 Using TDataSource

- *TDataSource* acts as a conduit between datasets and data-aware controls. Often the only thing you will do with a *TDataSource* component is to set its *DataSet* property to an appropriate dataset object.

- Then you will set data controls' *DataSource* property to the specific *TDataSource*.

The DataSet property

- Using TDataSource properties

- The *DataSet* property specifies the name of the dataset from which the *TDataSource* will get its data.

- You can also set the *DataSet* property to a dataset on another form to synchronize the data controls on the two forms.

The DataSet property

- For example,

```
procedure TForm2.FormCreate (Sender : TObject);
begin
      DataSource1.Dataset := Form1.Table1;
end;
```

The Enabled property

- The *Enabled* property can temporarily disconnect a *TDataSource* from its *TDataSet*.

- When set to False, all data controls attached to the data source will go blank and become inactive until *Enabled* is set to True.

The Enabled property

- In general, it is recommended to use datasets’ ***DisableControls*** **and** ***EnableControls*** **methods to perform this function, because they affect all attached data sources.**

What are TField components?

- All Delphi data-aware components rely on an underlying object class, *TField*.

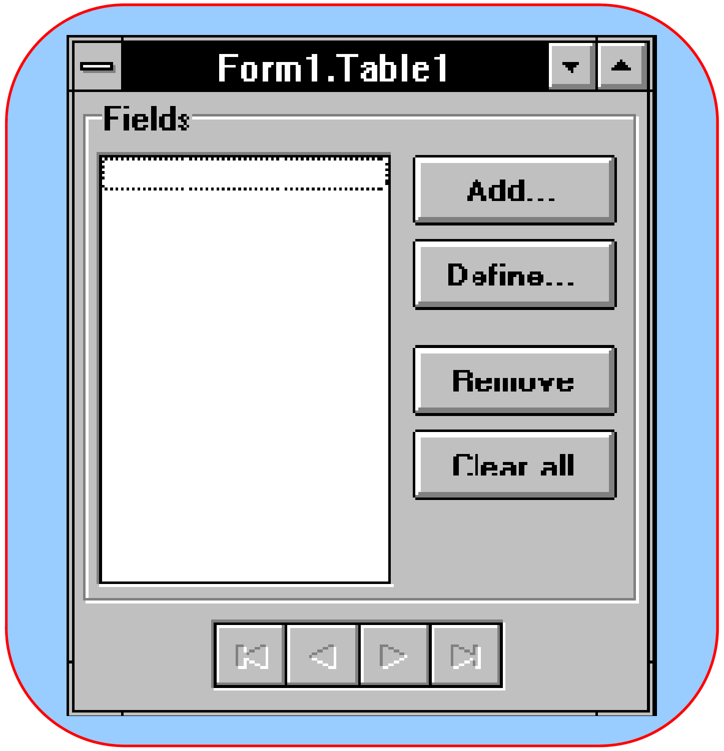
- Although not visible on forms, *TField* components are important because they provide an application a direct link to a database column.

Using the Fields Editor

- The Fields Editor enables you to:
  - Generate a persistent list of *TField* components.
  - Modify the display properties of persistent *TField* components.
  - Remove *TField* components from the list of persistent components.
  - Add new *TField* components based on existing columns in a table.
  - Define calculated *TField* components that behave just like physical data columns, except that their values are computed programmatically.

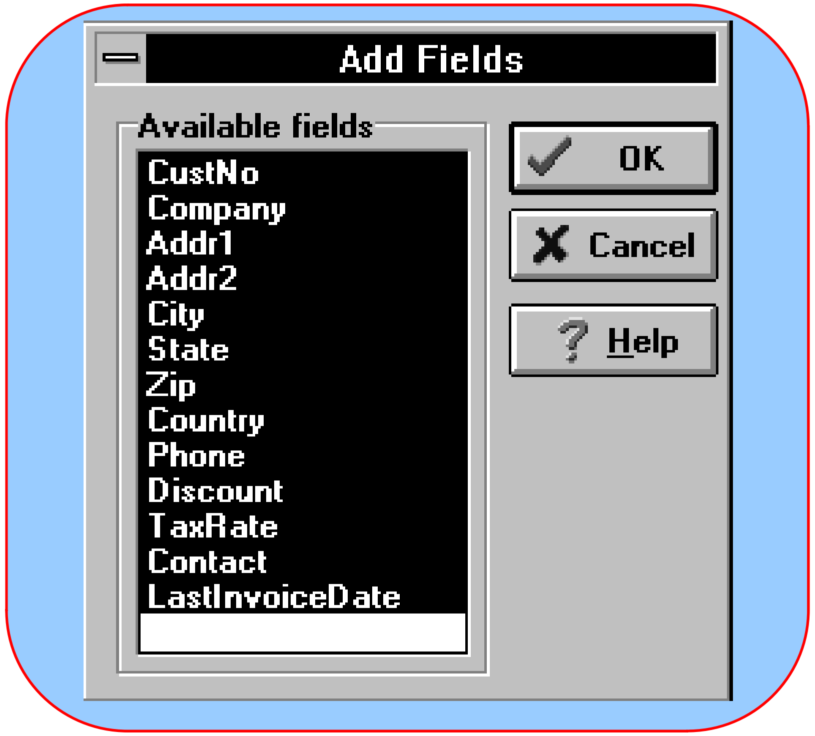
Starting the Fields Editor

Fields Editor



Adding a TField component

Fields Editor Add Fields dialog box



CustNo
Company
Addr1
Addr2
City
State
Zip
Country
Phone
Discount
TaxRate
Contact
LastInvoiceDate

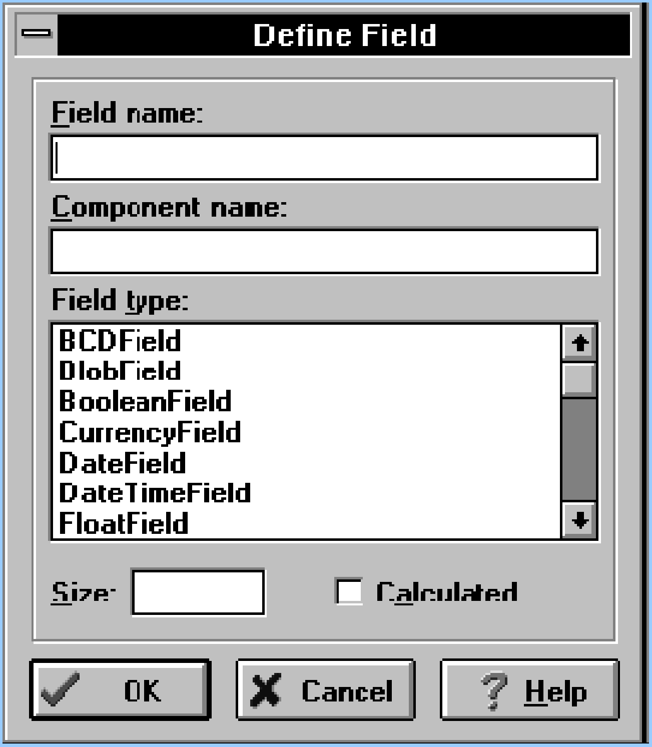
Defining a new TField component

- The Define button of the Fields Editor enables you to create new *TField* components for display.

- You can create a new *TField* based on a column in the underlying table.

Defining a new TField component

Define Field dialog box


Define Field

Field name:

Component name:

Field type:

BCDField
BlobField
BooleanField
CurrencyField
DateField
DateTimeField
FloatField

Size:

☐ Calculated

Defining a calculated field

- A calculated field is used to display values calculated at run time in the dataset’s *OnCalcFields* event handler.

Editing Display properties

TField properties

Property	Purpose
Alignment	Displays contents of field left justified, right justified, or centered within a data-aware component.
Calculated	True, field value can be calculated by a *CalcFields* method at run time. False, field value is determined from the current record.
Currency	True, numeric field displays monetary values. False, numeric field does not display monetary values.
DisplayForma	Specifies the format of data displayed in a data-aware component.
DisplayLabel	Specifies the column name for a field in a *TDBGrid*.
DisplayWidth	Specifies the width, in characters, of a grid column that display this field.
EditFormat	Specifies the edit format of data in a data-aware component.
EditMask	Limits data-entry in an editable field to specified types and ranges of characters, and specifies any special, non-editable characters that appear within the field (hyphens, parentheses, etc.).
FieldName	Specifies the actual name of column in the physical table from which the *TField* component derives its value and data type.
Index	Specifies the order of the field in a dataset.
MaxValue	Specifies the maximum numeric value that can be entered in an editable numeric field.

Editing Display properties

TField properties (continued)

Property	Purpose
MinValue	Specifies the minimum numeric value that can be entered in an editable numeric field.
Name	pecifies the component name of the *TField* component within Delphi.
ReadOnly	True: Field can be displayed in a component, but cannot be edited by a user. False: Field can be displayed and edited.
Size	Specifies the maximum number of characters that can be displayed or entered in a string-based field, or the size of byte and var byte fields.
Tag	Long integer bucket available for programmer use in every component as needed.
Visible	True: Field is displayed by a *TDBGrid* component. User-defined components can also make display decisions based on this property. False: Field is not displayed by a *TDBGrid* component.

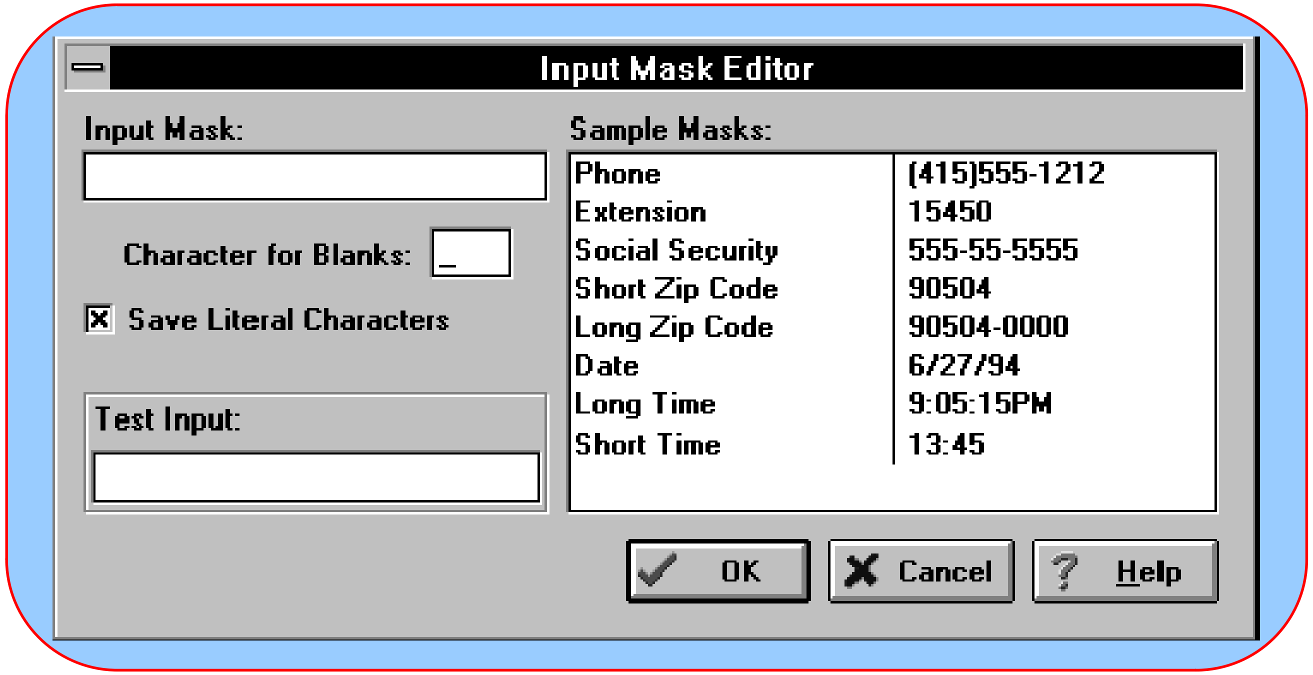
Using the Input Mask Editor

- The ***EditMask*** property provides a way to limit the entries that a user can type into data aware controls tied to a ***TField***.

- You can enter a specific edit mask by hand or use the Input Mask Editor to create a mask.

Using the Input Mask Editor

Input Mask Editor



Sample Masks:

Phone	(415)555-1212
Extension	15450
Social Security	555-55-5555
Short Zip Code	90504
Long Zip Code	90504-0000
Date	6/27/94
Long Time	9:05:15PM
Short Time	13:45

TField conversion functions

TField Type	AsString	AsInteger	AsFloat	AsDate Time	AsBoolean
TStringField	String type by Definition	Convert to Integer if possible	Convert to Float if possible	Convert to Date if possible	Convert to Boolean if possible
TIntegerField TSmallIntField TWordField	Convert to String	Integer type by definition	Convert to Float	Not Allowed	Not Allowed
TFloatField TCurrencyField TBCDField	Convert to String	Round to nearest integer value	Float type by definition	Not Allowed	Not Allowed.
TDateTimeField	Convert to String.	Not Allowed	Convert Date to number of days since 01/01/0001	DateTime type by definition	Not Allowed
TDateField TTimeField	Content depends on DisplayFormat of Field		Convert Time to fraction of 24 hours	Zero date or time If not specified	
TBooleanField	Convert to String "True" or "False"	Not Allowed	Not Allowed	Not Allowed	Boolean type by definition
TBytesField	Convert to String	Not Allowed	Not Allowed	Not Allowed	Not Allowed
TvarBytesField TblobField TmemoField TGraphicField TGraphicField	(Generally only makes sense for TMemoField)				