StarOffice™ and Delphi

Accessing StarOffice™ API with Delphi

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
1 (800) 786.7638
1.512.434.1511

Content

Preface....4
History....4
StarOffice™ and the OLE Automation Bridge for Delphi....5
StarOffice™ and OLE....5
StarOffice™ API....7
Services and interfaces of the StarOffice™ API....7
A code example....8
StarOffice™ API services....10
Collections and Container....11
Characteristics....12
Data types....12
Enumerations....12
Structures....13
Optional Parameters....13
Dealing with Errors....14
A complete sample program....15

Preface

In a customer project, the customer wants to access the StarOffice application program interface (API) of the StarOffice 5.2 from the programming language Delphi.

The Object Linking and Embedding (OLE) automation binding of the StarOffice 5.2 allows to access the StarOffice API. In this document, the OLE automation is briefly explained. Furthermore, you will learn from several examples, how to access the StarOffice API from Delphi.

The following specification, examples and representations are based on version 5.2 of StarOffice and version 5 of Delphi.

For additional information, please refer to the following sources:

- For details on the OpenOffice API, refer to the StarOffice API Reference document (http://api.openoffice.org/).
- Basically, you can learn how the OpenOffice API works using the StarOffice Programmer's Tutorial that comes with the software development kit (see http://api.openoffice.org).
- For details on StarOffice, refer to http://www.sun.com/staroffice.

History

Version	Publication Date	Change Notes	Author
Version 1.0	07.02.01	First version of this paper.	Magdolna Tolnay
Version 1.1	12.11.01	First version of this paper in English.	Bertram Nolte

CHAPTER 2

StarOffice™ and the OLE Automation Bridge for Delphi

Since the version 5.2, OLE automation binding is a constituent of StarOffice. The component technology used by StarOffice does not support OLE automation directly. The binding is made by the so called OLE Automation Bridge, which couples the OLE automation interface to the language binding of the StarOffice API.

Delphi 5 enables, to create and control OLE automation objects. Normally, this is only possible in Scripting languages (for example Visual Basic). Therefore, the OLE automation Bridge enables to use the full functionality of the StarOffice API in the programming language Delphi.

StarOffice™ and OLE

Object Linking and Embedding (OLE) is a compound document standard. It enables you to create objects with one application and then link or embed them in a second application. Embedded objects retain their original format and links to the application that created them. Support for OLE is built into the Windows and Macintosh operating systems.

OLE (acronym for Object Linking and Embedding) serves to integrate distributed objects or data into an application in such a way, that they can be processed on the spot with that application (in-place-handling), with which they (e.g. diagrams in a text document) were originally created.

Two possibilities are offered:

- **Linking:** Reference to a file, in which the data are contained. Data can be kept central and merged into several documents.

- **Embedding:** Objects/data are inserted. Then, the document is independent (Exe file of the OLE container).
- The philosophy of OLE is simple: The OLE container is able to represent also data, which it did not create. In order to process these data, the program (OLE server) is called, which produced these data. After this processing, in the OLE container the current OLE object is represented. With it, an OLE container can handle an OLE object, without copying all functions and formats.

In this context, the following terms are used:

- An OLE server is a program, which can create and handle an OLE object. In connection with StarOffice API, StarOffice can be regarded as object server.
- An OLE container is a program, which can contain an OLE object. Since StarOffice API does not support linking and embedding functions, it is here mentioned only marginally, that there are also such programs.
- An OLE object consists of data, which can be used by an OLE server (and if necessary by the container). Such data are typically documents, pictures, data of a spreadsheet calculation program and so on.
- OLE automation is constituent of the COM specification (Component Object Model). COM is part of the ActiveX specification. Thus OLE automation is indirectly also part of the ActiveX specification. For the user, automation provides the possibility of navigating directly in the object model of an application, of using and of manipulating the object model. With the version 2.0 so-called automation objects were integrated into the OLE standard. An automation object is a class instance, which is defined in a program (server), and which provides its properties and methods to other programs (clients). Thus the Client program can call functions of the server program. Now then, OLE automation is a further development of OLE, which enables you to navigate directly in the object model of an application, you can use and manipulate the object model.

CHAPTER 3

StarOffice™ API

The StarOffice API is an application programming interface, which let you use the functionality of StarOffice in other applications. The StarOffice API is based on the component technology UNO (Universal Network Objects). It is an object-oriented and language independent programming model. Currently, there are language bindings for Java, different C++ compilers on different platforms, and scripting languages. Other languages can be integrated by creating other language bindings. Further on, UNO supports conceptional bridging to other component technologies and programming models such as Java or COM.

Services and interfaces of the StarOffice™ API

UNO, the component technology used by StarOffice, offers the most important functionalities for the object model, which again offers methods for services, event handling, introspection, and reflection for the StarOffice API.

The StarOffice API is based on two substantial concepts: Services and interfaces.

It does not concern classes in the classical sense, here for example Java or C++ classes, StarOffice API offers only the specification. These classes can be regarded as meta classes or concepts.

In the StarOffice API, services are to be regarded as abstract concepts, which provide interfaces and properties. On each individual server, each implementation must offer the same interfaces. An interface is a collection of methods, which provide most important functionalities.

Let’s use a car to illustrate these concepts. The abstract car provides two concrete interfaces: XAccelerationControl and XDrivingDirection. Both interfaces export methods, the first one for accelerating and slowing down, the second one to turn the car left or right. In addition to these interfaces, the service Car has the properties Color and Seats. This description is generally accepted for all cars.

A code example

This paragraph demonstrates, how you can use a service of the StarOffice API in a Delphi program.

All objects, which are created and addressed with the StarOffice API, must be defined as type Variant. Method calls or property accesses are then checked by the Delphi compiler neither for their existence nor for type safety, but they will be analyzed and interpreted as in a scripting language at run-time. Errors, like falsely written method names or an incorrect number of parameters, are thus only announced at run-time by an exception. StarOffice makes a root object available as OLE server - the service manager. The service manager provides access to all UNO services, which will be available within StarOffice.

The following example shows a Delphi code snippet, which creates a new, empty text document over the OLE automation binding. Here, the example uses the service `com.sun.star.Desktop` and its method `loadComponentFromURL()`.

```
unit SampleCode;
interface

uses
   Windows, Messages, SysUtils, Classes,
   Graphics, Controls, Forms, Dialogs,
   StdCtrls, ComObj, Variants;
type
   TsampleCode = class
      function CreateTextDocument(): Variant;

   private
      { Private declarations }
   public
      { Public declarations }
   end;

implementation
function TsampleCode.CreateTextDocument(): Variant;
var
   ServiceManager: Variant;
   StarDesktop: Variant;
   Document: Variant;
begin
   ServiceManager := CreateOleObject('com.sun.star.ServiceManager');

   StarDesktop := ServiceManager.createInstance(
```

```
                    'com.sun.star.frame.Desktop');

   Document := StarDesktop.LoadComponentFromURL(
                  'private:factory/swriter',
                  '_blank',
                  0,
                  VarArrayCreate([0, - 1], varVariant));
   CreateTextDocument := Document;
end;

end.
```

StarOffice™ API services

A complete representation of the services, which are supported by StarOffice API, would go beyond the scope. Therefore only the most important, available modules or services are introduced:

com.sun.star.chart contains services for charting. The most important one is ChartDocument() that specifies the data to use in the chart and some general characteristics.

com.sun.star.sdb provides data base services. DatabaseEnvironment() is a service, which offers the possibility for a connection with a data base.

com.sun.star.drawing provides all services used for drawing line, rectangles, and circles.

com.sun.star.form provides services to the access control in forms in documents.

com.sun.star.frame contains the Desktop() service. This service is used to open existing documents or create documents. In addition, you can create an instance of these services through the global MultiServiceFactory().

com.sun.star.presentation provides all services to create and work with presentations.

com.sun.star.sheet contains services for spreadsheets. The SpreadheetDocument() service is used to work with spreadsheets.

com.sun.star.table provides all services for tables in text documents and spreadsheets.

com.sun.star.text groups services related to text documents. The TextDocument() service provides all interfaces required to work with text documents.

Collections and Container

In StarOffice API some components are combined into so-called Collections and container. For example, the tables in a spreadsheet are recognized as a collection of tables, like a text document, which is recognized as a collection of paragraphs.

UNO defines interfaces for four different kinds of containers:

1. **Enumeration containers** only allow sequential access to their elements.
2. **Indexed containers** allow direct indexed access to elements. Index containers correspond to the java.util.List interface.
3. **Named access containers** allow programmers to access elements by name. Containers of this type correspond to the java.util.Map interface.
4. **Set containers** allow programmers to insert and remove elements. These containers can also be queried to check for a specified element. Within a set container, no two elements may be equal.

Some containers have several access methods. For example, tables of a spreadsheet program can be addressed both with names and with index.

Example 1 :

The method `getElementNames()` returns the name of elements from a table.

```
...
sTableNames := oDoc.getTextTables.getElementNames();
iHighestNumber := 0;
for iTableCounter := VarArrayLowBound(sTableNames, 1)
   to VarArrayHighBound(sTableNames, 1) do
...
```

Example 2 :

The method `getByName()` returns the element, which has the given name.

```
...
GetBookmarkFromDBPointer :=
   oDoc.Bookmarks.getByName(sBookNames[iBookCounter]);
...
```

Characteristics

StarOffice API was not developed for the use with OLE automation. You can only access the StarOffice API with a bridge. Therefore, you should consider some special features

Data types

StarOffice API specifies its own data types, which are not available in the programming language Delphi as types. In particular, enumerations and structures are affected. In order to create objects of those types, you have to use the CoreReflection() service.

Enumerations

An enumeration is a collection of designated constants. For example, the vertical adjustment of a text can be LEFT, MIDDLE or RIGHT. These constants are summarized in the enumeration com.sun.star.text.HorizontalAdjust. The following example shows, how you can access the constant value of an enumeration in Delphi.

```
...
var
   CoreReflection: Variant;
   i: Integer;

...
StarOffice := CreateOleObject('com.sun.star.ServiceManager');
CoreReflection := StarOffice.createInstance(
                      'com.sun.star.reflection.CoreReflection');
...
i := CoreReflection.forName('com.sun.star.text.HorizontalAdjust')
                       .getField('RIGHT')
                          .get(null);
...
```

Structures

The following example demonstrates, how you can create a structure with a given type:

```
...
var
   PropertyValue  : Variant;
...
CoreReflection := StarOffice.createInstance(
                     'com.sun.star.reflection.CoreReflection');
CoreReflection.forName('com.sun.star.beans.PropertyValue')
                  .createObject(PropertyValue);
PropertyValue.Name := 'ReadOnly';
PropertyValue.Value := true;
...
```

Optional Parameters

In the StarOffice API, optional parameters are not allowed. That means, that a call of a StarOffice API function requires each parameter, even if this parameter is null (for type Variant) or empty (for type array). For example the function loadComponentFromURL() expects an array of variants as the fourth parameter. The following code example shows, how you can create an empty array.

```
...
LoadParams := VarArrayCreate([0, -1], varVariant);
...
Document := StarDesktop.LoadComponentFromURL(
               'private:factory/swriter', '_blank', 0, LoadParams);
...
```

Dealing with Errors

In order to produce more robust programs, you should apply exception handling. Exception handling wires error handling directly into the programming language. Exceptions provide a way to reliably recover from a bad situation. Instead of just exiting you are often able to set things right and continue the execution of a program, which produces much more robust programs.

The StarOffice API uses exceptions as concept of the error handling. Exceptions, which occur during a call over the OLE Bridge, are illustrated on a generic error code. In Delphi such errors can be intercepted and treated as before with `try...except`. Thereby, the detailed information about the error cause are lost however.

Example:

```
...
try
   if Sample.CreateDocument(true) then
   begin
      Button4.Enabled := true;
      Button5.Enabled := true;
      Button6.Enabled := true;
   end;
   StatusBar1.SimpleText := 'Ready';
except
   StatusBar1.SimpleText := 'Error';
end;
...
```

A complete sample program

The following Delphi program is an application, which creates a connection to the StarOffice server and an empty text document. Further on, you can insert tables specified by the table name and the database pointer prefix, get the database pointer specified by the table name and the cell address, and get the content of a cell specified by the database pointer.

The program consists of two modules: SampleUI defines the user interface, SampleCode is the StarOffice API implementation.

Unit SampleCode:

```
unit SampleCode;
interface

uses
   Windows, Messages, SysUtils, Classes, Graphics, Controls,
   Forms, Dialogs, StdCtrls, ComObj, Variants;
   type
   TSampleCode = class
      function Connect(): boolean;
      procedure Disconnect();
      function CreateDocument(): boolean;
      procedure InsertTable(sTableName: String; dbPointer: String);
      procedure InsertDatabaseTable(oDoc: Variant; sTableName: String;
         oCursor: Variant; iRows: Integer; iColumns: Integer;
         dbPointer: String);
      function CreateTextTable(oDoc: Variant; oCursor: Variant;
         sName: String; iRow: Integer; iColumn: Integer): Variant;
      function getCellContent(sBookmarkName: String): Variant;
      function getDatabasePointer(sTableName: String; sCellname: String):
         String;
      procedure InsertBookmark(oDoc: Variant; oTextCursor: Variant;
         sBookmarkName: String);
      function CreateBookmarkName(sTableName: String; sCellName: String;
         sDatabasepointer: String): String;
      procedure ChangeCellContent(oDoc: Variant; sTableName: String;
         sCellName: String; dValue: Double);
      function GetBookmarkFromDBPointer(oDoc: Variant;
         sBookmarkName: String): Variant;
      function GetBookmarkFromAdress(oDoc: Variant; sTableName: String;
```

```
            sCellAdress: String): Variant;
         function JumpToBookmark(oBookmark: Variant): Variant;
         function CreateUniqueTablename(oDoc: Variant): String;
      private
         StarOffice: Variant;
         Document: Variant;

         { Private declarations }
      public
         { Public declarations }
      end;

implementation

{ Insert a table texttable and insert in each cell a Bookmark with the
  address of the cell and database pointer }
function TSampleCode.Connect(): boolean;
begin
   if  VarIsEmpty(StarOffice) then
      StarOffice := CreateOleObject('com.sun.star.ServiceManager');
   Connect := not (VarIsEmpty(StarOffice) or VarIsNull(StarOffice));
end;

procedure TSampleCode.Disconnect();
begin
   StarOffice := Unassigned;
end;

function TSampleCode.CreateDocument(): boolean;
var
   StarDesktop: Variant;
begin
   StarDesktop := StarOffice.createInstance('com.sun.star.frame.Desktop');
   Document := StarDesktop.LoadComponentFromURL(
                  'private:factory/swriter', '_blank', 0,
                  VarArrayCreate([0, - 1], varVariant));
   CreateDocument := not (VarIsEmpty(Document) or VarIsNull(Document));
end;

function TSampleCode.getCellContent(sBookmarkName: String): Variant;
var
   oBookmark: Variant;
   oTextCursor: Variant;
begin
   oBookmark := GetBookmarkFromDBPointer(Document, sBookmarkName);
   oTextCursor := JumpToBookmark(oBookmark);
   getCellContent := oTextCursor.Cell.Value;
end;

function TSampleCode.getDatabasePointer(sTableName: String;
   sCellname: String): String;
var
   oBookmark: Variant;
   sBookmarkName: String;
   iPos: Integer;
begin
   oBookmark := GetBookmarkFromAdress(Document, sTableName, sCellName);
   sBookmarkName := oBookmark.getName();
   iPos := Pos('/%', sBookmarkName);
   while Pos('/%', sBookmarkName) > 0 do
   begin
      iPos := Pos('/%', sBookmarkName);
      sBookmarkName[iPos] := '%';
```

```
   end;
   Delete(sBookmarkName, 1, iPos + 1);
   getDatabasePointer := sBookmarkName;
end;

procedure TSampleCode.InsertTable(sTableName: String; dbPointer: String);
var
   oCursor: Variant;
begin
   { create a cursor object on the current position in the document }
   oCursor := Document.Text.CreateTextCursor();

   { Create for each table a unique database name }
   if (sTableName = '') then
      sTableName := createUniqueTablename(Document);

   InsertDatabaseTable(Document, sTableName, oCursor, 4, 2, dbPointer);
   ChangeCellContent(Document, sTableName, 'B2', 1.12);
end;

procedure TSampleCode.InsertDatabaseTable(oDoc: Variant;
   sTableName: String; oCursor: Variant; iRows: Integer;
   iColumns: Integer; dbPointer: String);
var
   oTable: Variant;
   sCellnames: Variant;
   iCellcounter: Integer;
   oCellCursor: Variant;
   oTextCursor: Variant;
   sCellName: String;
begin
   oTable := CreateTextTable(oDoc, oCursor, sTableName, iRows, iColumns);
   sCellnames := oTable.getCellNames();
   for iCellcounter := VarArrayLowBound(sCellnames, 1) to
                          VarArrayHighBound(sCellnames, 1) do
   begin
      sCellName := sCellnames[iCellcounter];
      oCellCursor := oTable.getCellByName(sCellName);
      oCellCursor.Value := iCellcounter;
      oTextCursor := oCellCursor.getEnd();
      InsertBookmark(oDoc,  oTextCursor,
         createBookmarkName(sTableName, sCellName, dbPointer));
   end;
end;

{ Change the content of a cell}
procedure TSampleCode.ChangeCellContent(oDoc: Variant;
   sTableName: String; sCellName: String; dValue: Double);
var
   oBookmark: Variant;
   oTextCursor: Variant;
   sBookmarkName: String;
begin
   oBookmark := GetBookmarkFromAdress(oDoc, sTableName, sCellName);
   oTextCursor := JumpToBookmark(oBookmark);
   oTextCursor.Cell.Value := dValue;

   { create a new bookmark for the new number }
   sBookmarkName := oBookmark.getName();
   oBookmark.dispose();
   InsertBookmark(oDoc, oTextCursor, sBookmarkName);
end;
```

```
{Jump to Bookmark and return for this position the cursor }
function TSampleCode.JumpToBookmark(oBookmark: Variant): Variant;
begin
   JumpToBookmark := oBookmark.Anchor.Text.createTextCursorByRange(
                        oBookmark.Anchor);
end;

{ create a Texttable on a Textdocument }
function TSampleCode.CreateTextTable(oDoc: Variant; oCursor: Variant;
   sName: String; iRow: Integer; iColumn: Integer): Variant;
var
   ret: Variant;
begin
   ret := oDoc.createInstance('com.sun.star.text.TextTable');
   ret.setName(sName);
   ret.initialize(iRow, iColumn);
   oDoc.Text.InsertTextContent(oCursor, ret, False);
   CreateTextTable := ret;
end;

{ create a unique name for the Texttables }
function TSampleCode.CreateUniqueTablename(oDoc: Variant): String;
var
   iHighestNumber: Integer;
   sTableNames: Variant;
   iTableCounter: Integer;
   sTableName: String;
   iTableNumber: Integer;
   i: Integer;
begin
   sTableNames := oDoc.getTextTables.getElementNames();
   iHighestNumber := 0;
   for iTableCounter := VarArrayLowBound(sTableNames, 1) to
                           VarArrayHighBound(sTableNames, 1) do
   begin
      sTableName := sTableNames[iTableCounter];
      i := Pos('$$', sTableName);
      iTableNumber := strtoint(Copy(sTableName, i + 2,
                                    Length(sTableName) - i - 1));

      if iTableNumber > iHighestNumber then
         iHighestNumber := iTableNumber;
   end;
   createUniqueTablename := 'DBTable$$' + inttostr(iHighestNumber + 1);
end;

{' Insert a Bookmark on the cursor }
procedure TSampleCode.InsertBookmark(oDoc: Variant; oTextCursor: Variant;
   sBookmarkName: String);
var
   oBookmarkInst: Variant;
begin
   oBookmarkInst := oDoc.createInstance('com.sun.star.text.Bookmark');
   oBookmarkInst.Name := sBookmarkName;
   oTextCursor.gotoStart(true);
   oTextCursor.text.InsertTextContent(oTextCursor, oBookmarkInst, true);
end;

function TSampleCode.CreateBookmarkName(sTableName: String;
   sCellName: String; sDatabasepointer: String): String;
begin
   createBookmarkName := '//' + sTableName + '/%' + sCellName + '/%' +
      sDatabasePointer + ':' + sCellName;
```

```
end;

{ Returns the Bookmark the Tablename and Cellname }
function TSampleCode.GetBookmarkFromAdress(oDoc: Variant;
   sTableName: String; sCellAdress: String): Variant;
var
   sTableAddress: String;
   iTableNameLength: Integer;
   sBookNames: Variant;
   iBookCounter: Integer;
begin
   sTableAddress := '//' + sTableName + '/%' + sCellAdress;
   iTableNameLength := Length(sTableAddress);

   sBookNames := oDoc.Bookmarks.getElementNames;

   for iBookCounter := VarArrayLowBound(sBookNames, 1) to
                          VarArrayHighBound(sBookNames, 1) do
   begin
      if sTableAddress = Copy(sBookNames[iBookCounter], 1,
                              iTableNameLength) then
      begin
         GetBookmarkFromAdress := oDoc.Bookmarks.getByName(
                                     sBookNames[iBookCounter]);
         exit;
      end;
   end;
end;

{ Returns the Bookmark the Tablename and Cellname }
function TSampleCode.GetBookmarkFromDBPointer(oDoc: Variant;
   sBookmarkName: String): Variant;
var
   sBookNames: Variant;
   iBookCounter: Integer;
begin
   sBookNames := oDoc.Bookmarks.getElementNames;
   for iBookCounter := VarArrayLowBound(sBookNames, 1) to
                       VarArrayHighBound(sBookNames, 1) do
   begin
      if Pos(sBookmarkName, sBookNames[iBookCounter]) =
         (1 + Length(sBookNames[iBookCounter]) -
              Length(sBookmarkName)) then
      begin
         GetBookmarkFromDBPointer := oDoc.Bookmarks.getByName(
                                        sBookNames[iBookCounter]);
         exit;
      end;
   end;
end;

end.
```

Unit SampleUI:

```
unit SampleUI;
interface
uses Windows, SysUtils, Classes, Graphics, Forms, Controls, StdCtrls,
     Buttons, ExtCtrls, SampleCode, ComCtrls;
type
   TOKBottomDlg = class(TForm)
      Bevel1: TBevel;
```

```
      Button1: TButton;
      Button2: TButton;
      Button3: TButton;
      Button4: TButton;
      Edit1: TEdit;
      Label1: TLabel;
      Edit2: TEdit;
      Label2: TLabel;
      Button5: TButton;
      Button6: TButton;
      Edit3: TEdit;
      Label3: TLabel;
      Label4: TLabel;
      Label6: TLabel;
      Edit6: TEdit;
      Bevel2: TBevel;
      Bevel3: TBevel;
      Bevel4: TBevel;
      StatusBar1: TStatusBar;
      Edit4: TEdit;
      Label7: TLabel;
      procedure OnConnect(Sender: TObject);
      procedure OnDisconnect(Sender: TObject);
      procedure OnCreateDocument(Sender: TObject);
      procedure OnInsertTable(Sender: TObject);
      procedure OnGetDatabasePointer(Sender: TObject);
      procedure OnGetCellContent(Sender: TObject);
   private
      { Private declarations }
   public
      { Public declarations }
   end;
var
   OKBottomDlg: TOKBottomDlg;
   Sample: TSampleCode;
implementation
{$R *.DFM}

procedure TOKBottomDlg.OnConnect(Sender: TObject);
begin
   StatusBar1.SimpleText := 'Connecting to StarOffice ...';
   Sample := TSampleCode.Create();
   if Sample.Connect() then
   begin
      Button1.Enabled := false;
      Button2.Enabled := true;
      Button3.Enabled := true;
      Button4.Enabled := false;
      Button5.Enabled := false;
      Button6.Enabled := false;
   end;
   StatusBar1.SimpleText := 'Ready';
end;

procedure TOKBottomDlg.OnDisconnect(Sender: TObject);
begin
   StatusBar1.SimpleText := 'Disconnecting from StarOffice ...';
   Sample.Disconnect();
   Button1.Enabled := true;
   Button2.Enabled := false;
   Button3.Enabled := false;
   Button4.Enabled := false;
   Button5.Enabled := false;
```

```
   Button6.Enabled := false;
   StatusBar1.SimpleText := 'Ready';
end;

procedure TOKBottomDlg.OnCreateDocument(Sender: TObject);
begin
   StatusBar1.SimpleText := 'Creating new text document ...';
   try
      if Sample.CreateDocument() then
      begin
         Button4.Enabled := true;
         Button5.Enabled := true;
         Button6.Enabled := true;
      end;
      StatusBar1.SimpleText := 'Ready';
   except
      StatusBar1.SimpleText := 'Error';
   end;
end;

procedure TOKBottomDlg.OnInsertTable(Sender: TObject);
begin
   try
      StatusBar1.SimpleText := 'Inserting Table ...';
      Sample.InsertTable(Edit2.Text, Edit1.Text);
      StatusBar1.SimpleText := 'Ready';
   except
      StatusBar1.SimpleText := 'Error';
   end;
end;

procedure TOKBottomDlg.OnGetDatabasePointer(Sender: TObject);
var
   res: String;
begin
   try
      StatusBar1.SimpleText := 'Getting database pointer...';
      res := Sample.getDatabasePointer(Edit4.Text, Edit3.Text);
      Application.MessageBox(PChar('the pointer: ' + res),
                             PChar('Result'), ID_OK);
      StatusBar1.SimpleText := 'Ready';
   except
      StatusBar1.SimpleText := 'Error';
   end;
end;

procedure TOKBottomDlg.OnGetCellContent(Sender: TObject);
var
   res: String;
begin
   try
      StatusBar1.SimpleText := 'Getting cell content...';
      res := Sample.getCellContent(Edit6.Text);
      Application.MessageBox(PChar('the content: ' + res),
                             PChar('Result'), ID_OK);
      StatusBar1.SimpleText := 'Ready';
   except
      StatusBar1.SimpleText := 'Error';
   end;
end;

end.
```