Broadview
www.broadview.com.cn

国际技术期刊中文精华合集

Delphi Informant

2002-2003 中文精华合集 [第1辑]

Informant Communications Group 独家授权

国际技术期刊合集编委会 编译

- Delphi 名刊精华
- Delphi 技术荟萃
- Delphi 开发必读

電子工業出版社
PUBLISHING HOUSE OF ELECTRONIC INDUSTRY
http://www.phei.com.cn

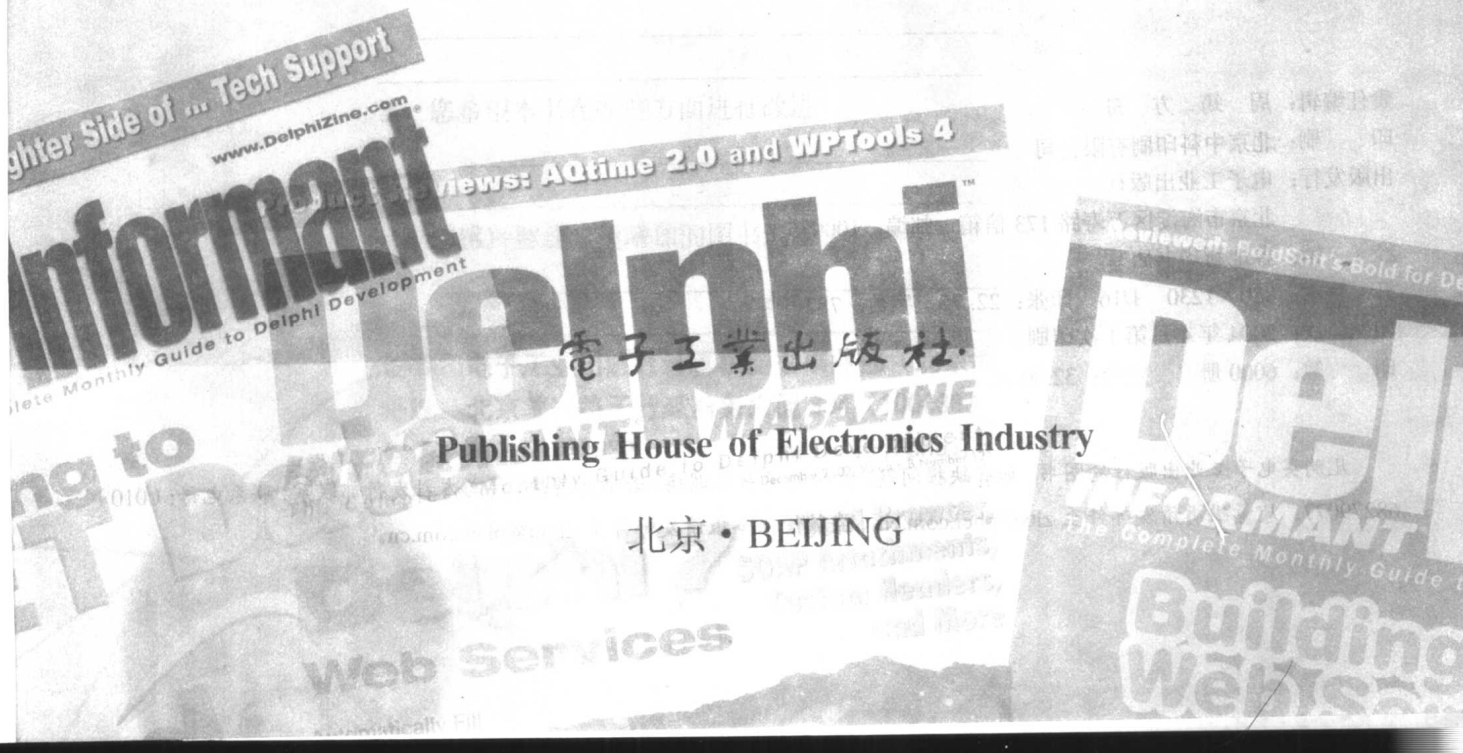
国际技术期刊中文精华合集

Delphi Informant

2002—2003 中文精华合集 [第1辑]

Informant Communications Group 独家授权

国际技术期刊合集编委会 编译



電子工業出版社
Publishing House of Electronics Industry
北京 · BEIJING

内 容 简 介

本书汇集了国际著名技术媒体 Informant Communication Group 旗下著名技术杂志 Delphi Informant 2002—2003 年度精华文章数十篇。主要围绕 Delphi 开发工具，涉及 Windows 开发、组件开发、数据库和 Web 开发，由各自领域内的一流专家撰写，其内容包括了从编程技术到产品配置，从工具使用技巧到新技术剖析的各个方面，技术含量丰富，观点权威，涵盖面广。

本书不仅适合专业软件开发者阅读学习和参考，同时也适合广大技术爱好者、在校学生和教师阅读学习。

图书在版编目（CIP）数据

Delphi Informant 2002—2003 中文精华合集．第 1 辑 / 美国信使通信集团著；国际技术期刊合集编委会编译．—北京：电子工业出版社，2004.4

（国际技术期刊中文精华合集）

书名原文：Delphi Informant Magazine

ISBN 7-5053-9730-3

Ⅰ.D...　Ⅱ.①美...②国...　Ⅲ.软件工具—程序设计—文集　Ⅳ.TP311.56-53

中国版本图书馆 CIP 数据核字（2004）第 016561 号

责任编辑：周　筠　方　舟

印　　刷：北京中科印刷有限公司

出版发行：电子工业出版社

北京市海淀区万寿路 173 信箱　邮编　100036

经　　销：各地新华书店

开　　本：880×1230　1/16　印张：22.75　字数：782 千字

印　　次：2004 年 4 月第 1 次印刷

印　　数：6000 册　　定价：32.00 元

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。联系电话：（010）68279077。质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

前　言

这是一本开发技术文集，收录了国际著名开发技术期刊 Delphi Informant 杂志 2002—2003 年发表的一批精华文章。

程序员作为一个职业群体，不但需要系统学习基础和经典理论，在实际项目中不断磨练自己，而且需要与同行充分交流，保持对新技术和新思想的敏感，向同行中的佼佼者取经。在这一点上，技术期刊能够起到别的载体无法替代的作用，而国外的高水平技术期刊，在这方面更是有着辉煌的传统。技术期刊中的文章，一方面能紧跟最新的技术趋势，另一方面来自名家之作，高屋建瓴，能够反映比较高的技术水平和崭新的视角。在 IT 产业发展的历史上，技术期刊起到了重要的推动作用。很多重要的新技术和新思想，最初都是发表在期刊上，进而对整个产业界产生深刻影响。因此，作为一名不断追求进步的程序员，应当经常地阅读高水平的国际技术期刊。

创立于 1990 年的 ICG（ICG Informant Communications Group, Inc.）是美国著名的 IT 技术媒体和出版集团，旗下拥有 asp.netPRO、C# PRO、Delphi Informant、Smart Solutions 等著名技术期刊。ICG 集团的技术期刊拥有一大批世界知名的技术作家，如.NET 专家 Dino Esposito、ASP.NET 专家 Jonathan Goodyear、微软 C#产品族（C#语言以及 Visual C# .NET）项目经理 Scott Wiltamuth、Delphi 专家 Alan Moore 等，一向以文章质量高、观点权威新颖、技术含量丰富著称。

Delphi 是著名开发工具厂商 Borland 公司出品的著名开发工具，也是最早的几个成熟的 RAD 开发环境。自推出以来，Delphi 以其优秀的开发环境、强大的功能、广泛的适应性和不断的发展，赢得了全世界开发者的青睐，成为 Windows 环境下最受欢迎的开发工具之一。特别是在我国，Delphi 是使用最普遍、最成功的开发工具之一，拥有广大的使用者和用户基础。随着.NET 时代的到来，Delphi 也不失时机地开启了面向.NET 的新纪元。广大 Delphi 开发者，急需通过阅读高水平、国际化的 Delphi 技术刊物迅速提升自己的技术水平。

Delphi Informant 自创刊以来，一路发展壮大，在国际 Delphi 技术界享有盛誉。此次我们本着为广大专业开发人员和学习者服务的宗旨，引进 Delphi Informant 杂志，将其精华内容集结成书，翻译出版。在选编时，我们尽可能注意技术期刊内容的特点，既选择一些最新、最前沿的技术文章，也选择一些立意深远、具有长期意义的文章。希望这本精华合集既能够满足开发者平时学习工作的要求，也能够具有一定的保存价值。这是我们的一个尝试，希望得到广大读者的积极反馈。

全书共 16 篇，分别收录了 Delphi Informant 杂志 2002 年 3 月—2003 年 2 月各期的内容和 2003 年 3—9 月的文章精选。

本书由“国际技术期刊合集编委会”主持编译，由徐舒、葛鑫、漆小洛进行选编和翻译。

国际技术期刊的选、编、译，对于我们来说是一个崭新的尝试，限于经验和水平，失误之处在所难免，恳请读者不吝赐教。

国际技术期刊合集编委会

2004 年 2 月

目 录

Delphi Informant

第一篇

- Delphi 条条大道通 Excel
- HTML 语言样式表
- 从 XML 到面向对象技术之一
- 微分方程

Delphi 条条大道通 Excel

利用 ADO、ODBC 或者 Automation 处理电子数据表中的数据。

● Bill Todd

Delphi 提供了 3 种处理 Excel 电子数据表中数据的途径，你可以根据自己使用 Excel 数据的需要和应用程序所使用的数据存取组件来选择最好的一条途径。如果你所需要的是读取、添加或者更新 Excel 文件中的表列数据，你可以使用 ADO 或者 Excel ODBC 驱动器和 BDE。如果你需要执行更为复杂的操作，你可以通过 Automation 来使用 Excel。

对于每一项技术，都各有其优缺点。ADO 和 ODBC 只允许读取、更新以及向工作表中插入数据，无法删除数据行。不过，ADO 和 ODBC 易于操作，并且允许像使用数据库表格那样使用工作表。这样的结果是，你可以很容易地通过使用 Delphi 的数据感知控件（data-aware control）来查看和编辑 Excel 工作表中的数据。

Automation 有 3 个缺点。首先，只有安装了 Excel 的计算机才能正常运行应用程序。其次，没有什么好的途径可以演示工作表中的数据，更谈不上允许用户进行方便的编辑了。你将不得不把数据复制到一个 ClientDataSet 或者 StringGrid 中，让用户对这些数据做出修改，然后再把修改后的数据复制回工作表中。再次，熟练掌握如何使用 Automation 操作 Excel 数据的难度远远超过另外两种方法，并且对于 Delphi 开发者而言，还缺乏很好的文档说明。不过除此之外，Automation 提供了强大的功能。使用 Automation，可以做到一个对 Excel 用户接口的熟练用户所能做到的一切。

使用 ADO

如果你的应用程序已经使用了 ADO 来进行数据存取操作，这种方法无疑是最好的选择。为了操作一个已有的 Excel 文件，首先在数据模块上创建一个 ADOConnection 组件，将它的 LoginPrompt 属性设置为 False。然后，选择 ConnectionString 属性，单击省略号按钮来打开 Connection String 对话框。单击 Use Connection String 单选按钮，接着再单击 Build 按钮，这时会弹出一个 Data Link Properties 对话框（如图 1 所示）。

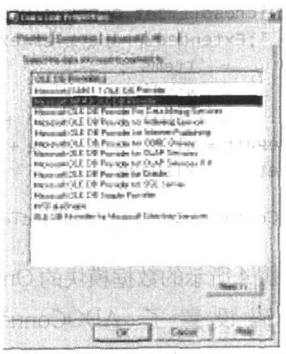


图 1 Data Link Properties 对话框

在 Provider 标签上，选择 Microsoft Jet 4.0 OLE DB Provider。在 Connection 标签上，输入 Excel 文件的路径和管理员密码，并且核对 Blank Password 复选框。接着，单击 All 标签，选择 Extended Properties，然后单击 Edit Value 按钮并输入 Excel 8.0。单击 OK 按钮关闭所有的对话框。如果不想对 Excel 文件名进行硬编码，你可以将刚刚创建的连接字符串（connection string）作为模板，运行时设置 ConnectionString 属性取代连接字符串中正确的文件名。

接下来，在数据模块上创建一个 ADODataSet，并将其 Connection 属性设置为上述的 ADOConnection 组件。将 ConnectionType 设置为 cmdTableDirect。CommandText 属性则必须被设置为 Excel 文件中的工作表的名称，该文件包含了想要进行存取操作的数据。文件名前必须加上一个美元符号（$），或者必须被设置为一个包含这些数据的指定单元格范围。如果你使用的是工作表名，同样可以制定一个单

元格范围。例如，在例程应用程序中，如果你使用了Items$A1:C10，那么就只能看到前三列和前十条记录。

图2显示了本文附录中的例程ExcelADO应用程序(下载细节见文末）的数据模块单位中的implementation部分的const声明。例程Excel文件的名称是ITEMS.XLS，包含有待处理数据的工作表的名称是Items。常量ConStr1和ConStr2中包含了连接字符串，这个字符串会使得程序在需要插入Excel文件名的地方暂停。XlsName是该Excel文件和包含数据的工作表共同的根名。XlsDir含有该Excel文件的目录的路径。你需要更改这个值来运行程序。XlsPath既是Excel文件的路径，也是它的文件名。XlsTemplatePath用于创建空Excel文件所需要的模板文件的路径名和文件名。XlsWorksheet则是添加有美元符号（$）的工作表名。

```
const
  ConnStr1 =
  'Provider=Microsoft.Jet.OLEDB.4.0;Data Source=';
  ConnStr2 = ';Extended Properties=excel 8.0;"";';
  XlsName        = 'Items';
  XlsDir         = 'w:\di\excel\';
  XlsPath        = XlsDir + XlsName + '.xls';
  XlsTemplatePath = XlsDir + XlsName + 'Template.xls';
  XlsWorksheet    = XlsName + '$';
```

图2 用于创建ConnectionString和CommandText属性值的常量

图3显示了图4所示的数据模块的OnCreate事件处理器。这段代码设置了ADOConnection组件的ConnectionString属性和ADODataSet组件的CommandText属性。它接着打开了ADODataSet。

```
procedure TMainDm.DataModuleCreate(Sender: TObject);
begin
  ExcelConn.ConnectionString :=
    ConnStr1 + XlsPath + ConnStr2;
  ExcelDs.CommandText := XlsWorksheet;
  ExcelDs.Open;
end;
```

图3 数据模块的OnCreate事件处理器

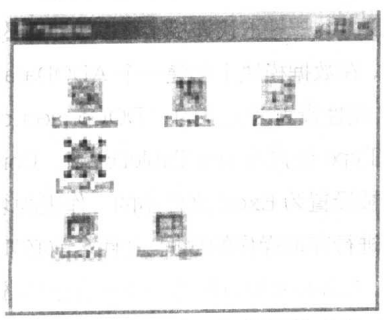


图4 ADO应用程序的数据模块

数据模块中还包含了一个DataSource组件——ExcelSrc，与ExcelDs ADODataSet组件相连接。图5所示的是该应用程序的主窗口，其中使用DBGrid显示了Excel数据。

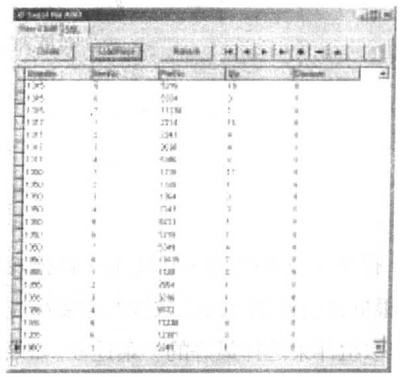


图5 ADO应用程序的主窗口

没有办法用ADO来创建一个Excel文件，因为提供给Jet Engine的OLE DB只能支持Access文件类型的ADO扩展（ADOX）。如果你的应用程序需要新建一个Excel文件并向其中添加数据，最简单的办法是使用Excel创建一个空文件，在每列的上方添加列名，然后保存该文件，当你需要新建一个Excel文件时直接复制它就可以了。

主窗口中的Create按钮调用了数据模块的CreateExcelFile方法，如图6所示。

```
procedure TMainDm.CreateExcelFile;
var
  TempFs: TFileStream;
  ExcelFs: TFileStream;
begin
  ExcelConn.Close;
  TempFs := TFileStream.Create(XlsTemplatePath,
                               fmOpenRead);
  try
    ExcelFs := TFileStream.Create(XlsPath, fmCreate);
    try
      ExcelFs.CopyFrom(TempFs, 0);
    finally
      ExcelFs.Free;
    end;
  finally
    TempFs.Free;
  end;
end;
```

图6 CreateExcelFile方法

这个方法创建了两个FileStream：一个用于打开模板文件进行读取，另一个则新建了一个空的ITEMS.XLS文件。对ExcelFs.CopyFrom方法的调用实际上是将模板文

件复制到新建的文件中去。

数据模块中还含有一个 BDE Database 和一个 Table 组件，用于将 DBDemos 例程数据库的 Items 表格中的记录装载入工作表中。Load Recs 按钮的 OnClick 事件处理器调用了数据模块的 LoadData 方法，如图 7 所示。将记录复制到 Excel 文件中的方法和将其复制到任何一个离线缓存（dataset）中的方法完全一样。该方法使用了一个 while 循环来遍历 Items 表格中的所有记录。对于每一个记录，调用 ExcelDs.Insert 向 Excel 文件中添加一条新的记录。然后 for 循环将 ItemsTable 中各个域的数值复制到 ExcelDs 中。向 Excel 工作表中插入行是比较慢的。虽然例程应用程序只插入了不到 1,000 行数据，在 1GHz 主频的计算机上仍然需要运行好几秒钟。这是 Excel 的特性，而不是 ADO 的特性；你可以发现使用 ODBC 和 Automation 插入行也是如此之慢。

```
procedure TMainDm.LoadData;
var
  I : Integer;
begin
  MainForm.StatusBar.SimpleText := 'Loading...';
  ExcelDs.DisableControls;
   try
    ItemsTable.Open;
     while not ItemsTable.EOF do begin
       ExcelDs.Insert;
       for I := 0 to ItemsTable.Fields.Count - 1 do
ExcelDs.Fields[I].Assign(ItemsTable.Fields[I]);
       ExcelDs.Post;
       ItemsTable.Next;
     end;
   finally
    ExcelDs.EnableControls;
    MainForm.StatusBar.SimpleText := '';
   end;
end;
```

图 7　数据模块的 LoadData 方法

有两个地方 Excel 文件与其他离线缓存表现得不太一样。首先，你不能够删除记录，如果你试着这样做，就会产生一个异常。其次，调用 Excel 离线缓存的 Refresh 方法会导致一个异常，因为 Excel 文件中没有主代码，也没有其他任何可以惟一标识各行的标识符。例程应用程序使用了图 8 所示的代码，用来保存当前的记录编号，关闭并且重新打开离线缓存，然后返回到同样的记录编号处。

```
procedure TMainDm.RefreshExcelData;
var
  CurrentRec : Integer;
begin
  CurrentRec := ExcelDs.RecNo;
  ExcelDs.Close;
  ExcelDs.Open;
  ExcelDs.RecNo := CurrentRec;
end;
```

图 8　ExcelADO 工程中的 RefreshExcel 数据方法

你可以在 Excel 文件中使用 SQL 的 INSERT，UPDATE 和 SELECT 语句。数据模块中含有一个 ADOCommand 组件，该组件是通过主窗口中的 SQL 标签上的 Insert 和 Update 按钮来使用的，其作用是插入新的行，并且更新 Insert 按钮插入的行的 Discount 栏。

使用 BDE

使用 BDE 和 Excel ODBC 驱动器的第一步是为你的 Excel 文件创建一个 ODBC DSN（数据源名称 Data Source Name）。首先打开 ODBC 数据源管理器（Data Source Administrator），如图 9 所示，然后根据你所想创建的 DSN 的类型单击用户 DSN（User DSN）或者系统 DSN（System DSN）标签。接着，单击 Add 按钮，选择 Excel 驱动器，再单击 Finish 按钮。在 ODBC Microsoft Excel Setup 对话框中，输入数据源的名称和说明，然后从下拉列表中选择你想要使用的 Excel 版本。接着单击 Select Workbook 按钮，输入你的 Excel 文件的路径。单击每一个对话框中的 OK 按钮退出。

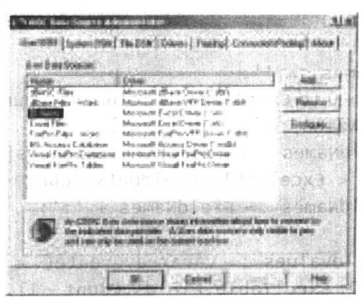


图 9　ODBC 数据源管理器

在你的应用程序中，在数据模块上创建一个 BDE Database 组件，并将它的 AliasName 属性设置为 ODBC DSN 的名称。将 LoginPrompt 属性设置为 False，将 DatabaseName 属性设置为你想要在应用程序中使用的数值。向数据模块中添加一个 BDE Table 组件，并将其 DatabaseName 设置为你在 Database 组件中使用的那个数

值。将 TableName 属性设置为加有美元符号（$）前缀的工作表名，你也可以使用一个指定的范围来代替工作表名。如果你使用的是工作表名，则可以使用与 ADO 相同的语法来添加一个单元格范围。你现在可以把一个 DataSourse 组件和一个栅格或者别的数据感知控件连接到 Table 组件上，这样就可以查看和编辑 Excel 文件中的数据了。

文末附录中的 ExcelBDE 例程应用程序与 ExcelADO 例程应用程序一样有些异常。使用 ODBC 可以修改 Excel 文件中的数值，插入新的数据行，但是不能删除行。和使用 ADO 时一样，Refresh 方法不起作用。不过，BDE 例程应用程序中的 Refresh 按钮调用的代码与图 8 所示的 ADO 应用程序的更新代码不同。ODBC 应用程序的 Refresh 按钮调用的是图 10 所示的 RefreshExcelData 方法。更新数据的惟一途径是关闭然后再次打开 Table 组件。问题是当 Table 再次打开后，如何回到相同的记录上。ADO 例程使用了 RecNo 属性，但是对于 BDE/ODBC 组合体而言它不起任何作用，因为 RecNo 返回的值将始终是-1。代替的方案是调用 SaveCurrentRec 方法，传递一个字符串变量 FieldNames 和一个变量数组 FieldValues。SaveCurrentRec 将域名作为一个使用逗号分隔符的列表保存在 FieldNames 中，将各域的数值保存在变量数组中。

```
procedure TMainDm.SaveCurrentRec(var FieldNames:
string;
    var FieldValues: Variant);
  var
    I : Integer;
  begin
    FieldNames := '';
    for I := 0 to ExcelTable.FieldDefs.Count - 1 do begin
     FieldNames :=
       FieldNames + ExcelTable.FieldDefs[I].Name;
      if I < ExcelTable.FieldDefs.Count - 1 then
       FieldNames := FieldNames + ';';
    end;
    // FieldValues := VarArrayCreate(
    //    [0, ExcelTable.Fields.Count - 1], varVariant);
    for I := 0 to ExcelTable.Fields.Count - 1 do
     FieldValues[I] := ExcelTable.Fields[I].Value;
  end;

  function TMainDm.IsThisTheRightRow(FieldValues:
Variant):
    Boolean;
  var
    I : Integer;
  begin
    Result := True;
    for I := 0 to ExcelTable.Fields.Count -1 do
      if ExcelTable.Fields[I].Value <> FieldValues[I]
then
       begin
        Result := False;
        Break;
       end;
  end;

  procedure TMainDm.RefreshExcelData;
  var
    FieldNames : string;
    FieldValues : Variant;
    S : string;
    I : Integer;
  begin
    FieldValues := VarArrayCreate(
      [0, ExcelTable.Fields.Count - 1], varVariant);
    SaveCurrentRec(FieldNames, FieldValues);
    ExcelTable.Close;
    ExcelTable.Open;
    ExcelTable.Locate(FieldNames, FieldValues, []);
    {出于某种奇怪的原因，有时进行定位时会将光标置于所要查找的记
录之后的记录上。所以，需要比较当前记录中的数值和数组中的数值；如果
它们不相等，就向后移动一个记录。}
    if not IsThisTheRightRow(FieldValues) then
     ExcelTable.Prior;
    { 如果你还是没有到达正确的行，显示一个出错消息。 }
    if not IsThisTheRightRow(FieldValues) then
     raise Exception.Create(
       'Locate failed to find the right row.');
    // for I := 0 to 4 do
    //   S := S + VarToStr(FieldValues[I]) + ' | ';
    // ShowMessage(S);
  end;
```

图 10 ExcelBDE 应用程序的更新代码

RefreshExcelData 接着会关闭和重新打开 Table 组件，并且调用 Locate 方法，传递的参数是 FieldNames 和 FieldValues 变量值。这应该定位到具有与表格关闭时当前记录中的数值相等的数值的第一条记录上。然而，出于某种原因，有时光标将会落到 Locate 所要寻找的记录之后的一个记录上。为了解决这个奇怪的问题，RefreshExcelData 调用了 IsThisTheRightRow 函数，传递的参数是 FieldValues 数组。如果当前行中的数值与数组中的数值相等，IsThisTheRightRow 就返回 True，否则返回 False。如果数值不相等，RefreshExcelData 将光标移到前一个记录上，并且再次调用 IsThisTheRightRow。如果数值还是不相等，就抛出一个异常。

ODBC 驱动器同样允许你使用 SQL 的 SELECT、UPDATE 和 INSERT 语句来处理 Excel 数据。ExcelBDE 例程应用程序使用了数据模块上的 ExcelQry BDE Query 组件来运行和 ExcelADO 应用程序中相同的 INSERT 和 UPDATE 语句。

使用 Automation

当你通过 Automation 来控制 Excel 时，你可以做到 Excel 用户接口所能做到的一切。所以，通过 Automation 使用 Excel 比前面的方法功能上更为强大，但也更为复杂。在你能够通过 Automation 对 Excel 进行有效的运用之前，需要了解 Excel 对象模块的基础知识。最好的信息来源是 Microsoft Office 自带的 Excel Visual Basic for Applications（VBA）帮助文档。对于 Office 2000，这个文件叫做 VBAXL9.CHM。图 11 显示了打开帮助文档后看到的对象树状结构图的第一页。

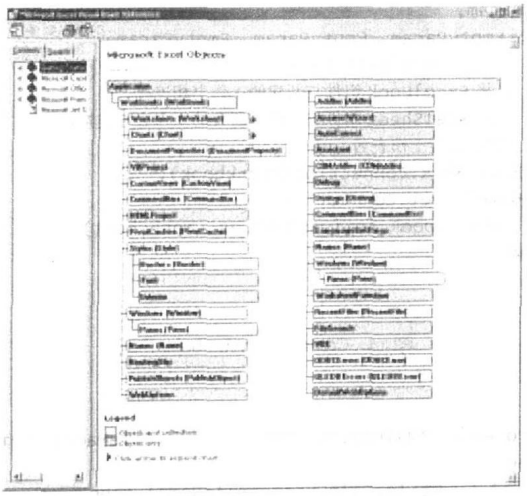


图 11　Excel VBA 帮助文档

另外一个比较好的信息和例子来源是 Microsoft Office 2000/Visual Basic Programmer's Guide（Microsoft Press，1999）一书。虽然所有的例子都是用 Visual Basic 编写的，但对于我们同样有用。另外一个很有用的使用 Delphi 语言编写的 Office Automation 例子来源自 Deborah Pate 的站点 www.djpate.freeserve.co.uk/Automation.htm。最后，如果你从组件选择板（Component Palette）的服务器（Servers）标签中选择使用 Excel 组件的话，别忘了 Delphi\OCX\Servers 目录中还有一个 Excel2000.pas 文件。

使用 Delphi 有 3 种不同的方法可以通过 Automation 来使用 Excel。其中一个方法是使用延迟绑定（Late Binding），其第一步是创建一个 Excel 实例，调用：

```
CreateOleObject('Excel. Application')
```

第二种方法是导入 Excel 类型库，直接运用其中的接口。第三种方法是使用组件选择板的服务器标签中的组件。这最后一种方法也是本文中将要说明的方法。

理解对象模块。Application 对象位于 Excel 对象的最顶层。Application 对象包含了 Workbooks 属性，该属性收集了所有当前打开的工作书（文件）。这个对象还包含了一个 Workbook 属性，该属性是对处于激活状态中的工作书的引用。Workbook 对象则包含了一个 Worksheets 属性，该属性收集了该工作书中的所有工作表。这个对象还包含了一个 Worksheet 属性，该属性是对处于激活状态中的工作表的引用。

处理 Excel 数据时，使用最多的对象将会是 Range 对象。Range 对象表示了工作表中一个或更多的单元格的范围。其他许多具有 Range 属性的 Excel 对象返回的是对 Range 对象的引用。通过 Range 对象，你可以在单个单元格中获取数值和设置数值，也可以同时改变该范围中的所有单元格的属性。Range 对象的 Cells 属性可以让你遍历该范围中的所有单元格，或者通过行号和列号访问单个的单元格。

处理 Excel 数据。当你打开 ExcelAuto 例程应用程序时，将会看到图 12 所示的主窗口。Connect 按钮打开了一个与 Excel 相连接的 Automation，它会在后台启动一个 Excel 实例。Creat 按钮新建一个叫做 ITEMS.XLS 的 Excel 工作书文件，并将其保存。Load 按钮从 DBDemos Items 表格读取数据装载到新建的工作书文件的工作表中，并将其保存。Open 按钮打开一个已有的 ITEMS.XLS 文件，并将其数据装载到一个与窗口中的 DBGrid 相连接的 ClientDataSet 中。

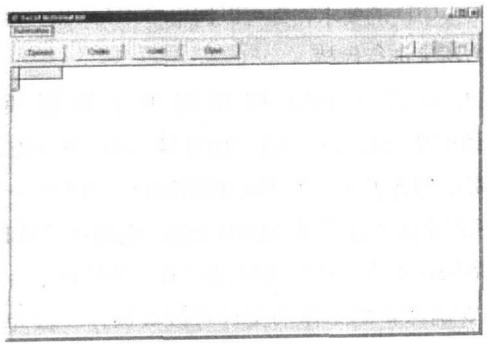


图 12　ExcelAuto 例程应用程序的主窗口

图 13 显示了 ExcelAuto 应用程序的数据模块。数据模块中的前 3 个组件分别是 ExcelApplication、

ExcelWorkbook 和 ExcelWorksheet。这些组件是用来连接 Excel 和操作 Excel 数据的。第二行含有一个 BDE Database 组件和一个 BDE Table 组件。它们是用来提供 Delphi 自带的 DBDemos 例程数据库的 Items 表格中的数据的。第三行包含一个 ClientDataSet 和 DataSource，用于显示 Excel 工作表中的数据。

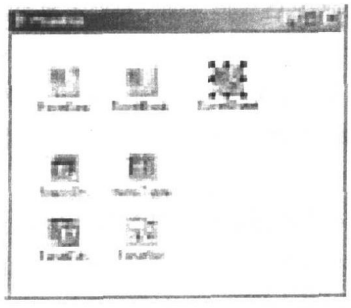


图 13　ExcelAuto 应用程序的数据模块

主窗口的 Connect 按钮调用数据模块的 ConnectToExcel 方法，这是个只有一行代码的方法，它调用了 ExcelApplication 组件的 Connect 方法：

```
procedure TMainDm.ConnectToExcel;
begin
  ExcelApp.Connect;
   // ExcelApp.Visible[GetUserDefaultLCID] := True;
end;
```

如果你想知道当你运行例程应用程序时 Excel 发生了什么，可以去掉代码中将 ExcelApplication 组件的 Visible 属性设置为 True 的那一行的注释符。你将要通过 Automation 用到的许多属性和方法都会要求一个本地 ID，在属性和方法声明中以 LCID 的形式显示出来。不论什么时候出现对 LCID 的要求，都可以调用 GetUserDefaultLCID 函数的方法来进行代替。

主窗口的 Create 按钮调用了数据模块的 CreateExcelWorkbook 方法，如图 14 所示。这个方法只有一个参数，包含了文件名（没有路径名）。该方法一开始就将应用程序的路径添加到文件名之前。它调用了图 15 所示的 DeleteExcelFile 方法，如果文件存在则将其删除。这个步骤需要确保新的工作书文件可以被保存，因为 Excel Workbook 对象的 SaveAs 方法不会覆盖一个已存在的文件。

```
{ 新建一个含有一个工作表的 Excel 工作书文件，
  然后将其保存在应用程序的目录中。 }
  procedure TMainDm.CreateExcelWorkbook(FileName:
string);
  var
    XlWorkbook : _Workbook;
  begin
    FileName :=
      ExtractFilePath(Application.ExeName) + FileName;
     { 如果文件存在则将其删除，因为工作书的
      SaveAs 不会覆盖一个已存在的文件。 }
    DeleteExcelFile(FileName);

ExcelApp.SheetsInNewWorkbook[GetUserDefaultLCID] := 1;
    XlWorkbook :=
      ExcelApp.Workbooks.Add(EmptyParam,
GetUserDefaultLCID);
     (XlWorkbook.ActiveSheet as _Worksheet).Name :=
      WORKSHEET_NAME;
    XlWorkbook.SaveAs(FileName, EmptyParam,
EmptyParam,
      EmptyParam, EmptyParam, False, xlExclusive,
EmptyParam,
      EmptyParam, EmptyParam, EmptyParam,
      GetUserDefaultLCID);
     { 将 ExcelBook 组件与新建的工作书相连接，
      将 ExcelSheet 组件与工作书所含的工作表相连接，
      然后设置工作表名。 }
    ExcelBook.ConnectTo(XlWorkbook);
    ExcelSheet.ConnectTo(
      XlWorkbook.ActiveSheet as _Worksheet);
  end;
```

图 14　CreateExcelWorkbook 方法

```
procedure TMainDm.DeleteExcelFile(FileName: string);
begin
  if SysUtils.FileExists(FileName) then
    if not SysUtils.DeleteFile(FileName) then
      RaiseLastOsError;
end;
```

图 15　DeleteExcelFile 方法

接着，代码将 ExcelApp 组件的 SheetsInNewWorkbook 属性设置为 1，然后调用 Workbooks 属性的 Add 方法将新建的工作书文件添加到 Workbooks 的收集中。这个方法返回一个对新建的工作书文件的引用，该引用被赋值给 type_Workbook 的本地变量 XlWorkbook。最后，使用 Workbook 对象的 ActiveSheet 属性设置工作表名。WORKSHEET_NAME 是在数据模块单元的 implementation 部分声明的一个常量。如果你顺便看看 Delphi 的 Excel2000.pas 单元的话，你可以发现 ActiveSheet 属性返回的是一个对 IDispatch 接口的引用，所以 ActiveSheet 属性必须被赋值给 type _Worksheet 以访

问 Worksheet 对象的各个属性。

既然工作书已经被创建出来了，它就必须通过调用 Workbook 对象的 SaveAs 方法进行保存。绝大多数参数不是必需的，所以传递的是变量（Variants）单元中声明的空变量 EmptyParam。要求的参数是文件名；第 6 个参数 CreateBackup，用于确定文件的备份是否需要保存；第 7 个参数 AccessMode，作为一个系统定义的常量被传递，确定工作书的存取模式；最后一个参数是本地 ID。剩下的步骤是通过调用工作书和工作表的 ConnectTo 方法，将新建的工作书与数据模块的 ExcelBook 组件相连接，将工作书的工作表与数据模块的 ExcelSheet 组件相连接。传递给 ExcelBook.ConnectTo 方法的参数是 XlWorkbook 变量，指向新建的工作书。传递给 ExcelSheet.ConnectTo 的参数是指派给 type _Worksheet 的 XlWorkbook.ActiveSheet 属性。

单击主窗口的 Load 按钮调用数据模块的 LoadWorksheetData 方法，如图 16 所示。这个方法首先使用 Worksheet 的 Cells 属性将栏标题放置在工作表的首行。Cells 对象的 Item 属性允许你使用一个二维数组在工作表中定位单元格，其中第一维是行号，第二维是列号。所以，ExcelSheet.Cells.Item[1，2]指的是位于第一行第二列的单元格，即惯例中电子表格符号的 B1 所指的单元格。

```
{ 从 DBDemos Items 表格中读取数据装载到空的工作表中。}
procedure TMainDm.LoadWorksheetData;
var
  Row : Integer;
  Col : Integer;
begin
  MainForm.StatusBar.SimpleText := 'Loading Excel
data...';
  { 将栏名放置在工作表的首行。 }
  ExcelSheet.Cells.Item[1, 1].Value := 'OrderNo';
  ExcelSheet.Cells.Item[1, 2].Value := 'ItemNo';
  ExcelSheet.Cells.Item[1, 3].Value := 'PartNo';
  ExcelSheet.Cells.Item[1, 4].Value := 'Qty';
  ExcelSheet.Cells.Item[1, 5].Value := 'Discount';
  { 循环扫描 Items 表格，将各个记录的域复制到工作表中的相应行
中。 }
  ItemsTable.Open;
  Row := 1;
  while not ItemsTable.Eof do begin
    Inc(Row);
    for Col := 1 to 5 do
      ExcelSheet.Cells.Item[Row, Col].Value :=
        ItemsTable.Fields[Col - 1].Value;
    ItemsTable.Next;
  end;
  { 关闭表格，将改动保存到工作书中。 }
  ItemsTable.Close;
  ExcelBook.Save;
  MainForm.StatusBar.SimpleText := '';
end;
```

图 16 LoadWorksheetData 方法

while 循环遍历 Items 表格中的所有记录。对于每一个记录又用 for 循环将表格中的各个域值复制到工作表的相应单元格中。由于工作表各行的第一个单元格的编号是 1 而 Table 组件的 Fields 数组中的第一个域的编号是 0，所以 Fields 数组的下标实际上是 Col - 1。所有数据都复制到工作表中之后，ItemsTable 被关闭，调用工作书的 Save 方法来将改动保存到 Excel 文件中。

单击 Open 按钮是将 Excel 工作表中的数据复制到 ExcelCds ClientDataSet 组件中，并在主窗口的 DBGrid 中进行显示。完成这项功能的代码是数据模块的 CopyExcelDataToClientDataSet 方法（如图 17 所示）。这是一个通过 Automation 处理 Excel 数据的缺点的例子。没有什么便捷的方法可以在你的 Delphi 应用程序中显示数据。最简单的方法是将工作表中的数据复制到一个 ClientDataSet 中去，这样你就可以使用 Delphi 的数据感知控件来显示数据了。

```
{将当前 Excel 工作表中的数据复制到 ExcelCds ClientDataSet
中，这样就可以显示数据了。 }
procedure TMainDm.CopyExcelDataToClientDataSet;
var
  Row, Col : Integer;
  DataRange : Range;
  FirstRow, LastRow: Integer;
begin
  { 获取含有 A1 单元格的单元格范围，然后算出该范围的首行和末
行的编号。首行编号加 1 以跳过含有栏标题而不是数据的第一行。 }
  DataRange := ExcelSheet.Range['A1',
'A1'].CurrentRegion;
  FirstRow := DataRange.Row + 1;
  LastRow := FirstRow + DataRange.Rows.Count - 2;
  { 从一个新的空 ClientDataSet 开始。 }
  ExcelCds.Close;
  ExcelCds.CreateDataSet;
  { 将 ClientDataSet 的 ReadOnly 属性设置为 False 然后显示
一个状态消息。 }
  MainForm.StatusBar.SimpleText := 'Loading Excel
data...';
  ExcelCds.ReadOnly := False;
  ExcelCds.DisableControls;
  { 在电子表格中逐行下移，直到出现一个栏 1 为空单元格的行为止。
每遇到一行时都在 ClientDataSet 中加入一个新行，然后复制数值。 }
  try
    for Row := FirstRow to LastRow do begin
      ExcelCds.Append;
```

```
      for Col := 1 to 5 do
       ExcelCds.Fields[Col - 1].AsInteger :=
         ExcelSheet.Cells.Item[Row, Col].Value;
      ExcelCds.Post;
     end;
    ExcelCds.First;
   finally
    ExcelCds.EnableControls;
    ExcelCds.ReadOnly := True;
    MainForm.StatusBar.SimpleText := '';
   end;
end;
```

图 17 CopyExcelDataToClientDataSet 方法

从工作表向一个 ClientDataSet 中复制数据时存在一个问题，那就是你不得不确定工作表中有多少行数据。有两个方法可以做到这一点。一个是按行循环，直到遇到一个本该含有数据而事实上为空的单元格所在的行。不过，还有一种技术允许你同时确定行号和栏号。Excel Range 对象有一个叫做 CurrentRegion 的只读属性。CurrentRegion 返回一个 Range 对象，该对象包含了 Range 对象所定义的范围周围的所有非空单元格。换一句话说，如果你有一个 Range 对象，定义了单个单元格范围 E5:E5，CurrentRegion 属性将沿四个方向向外搜索，直到遇到一个空的单元格或者工作表的边沿位置。接着它会返回一个 Range 对象，定义这些边界。CopyExcelDataToClientDataSet 方法声明了一个 Range 类型的变量 DataRange，然后将其赋值为 ExcelSheet.Range['A1','A1'].CurrentRegion 返回的 Range 对象。

代码接着使用范围的 Row 属性确定第一个数据行的编号。由于第一行包含的是栏标题而你实际想要使用的是从第二行开始，所以该数值需要加 1。Range 对象的 Rows 属性返回 Range 中的行数，所以可以通过首行的编号和总行数计算出末行的编号。例子中的栏编号是已知的，但如果不是这样，你可以使用 Range 对象的 Col 和 Cols 属性通过同样的方法算出首栏和末栏的编号。

在设计阶段，使用 FieldDefs 属性的属性编辑器向 ClientDataSet 添加了 5 个整型域，分别是 OrderNo，ItemNo、PartNo、Qty 和 Discount。代码关闭了 ClientDataSet，然后调用它的 CrcateDataSet 方法新建一个空的内存（in-memory）离线缓存并将其打开。接着，将 ClientDataSet 的 ReadOnly 属性设置为 False，调用它的 DisableControls 方法。外层 for 循环遍历所有含有数据的行。每遇到一行时就在 ClientDataSet 中追加一个新的记录，而内层 for 循环则将行中各单元格的数值复制到新记录相应的域中。最后，公布新的记录。当所有的数据都复制完成后，调用 ClientDataSet 的 First 方法回到第一行，调用 EnableControls 使得数据在栅格中可见，设置 ReadOnly 属性为 True。

如果你需要允许用户修改栅格中显示的数据，将 ClientDataSet 的 ReadOnly 属性设置为 False，LogChanges 属性也设置为 False。用户修改数据完成后，使用把数据由 Items 数据库表格装载到工作表中的那一段代码将数据复制回工作表。然后用 ClientDataSet 替代 BDE Table。

数据模块的 OnDestroy 事件处理器调用了 DisconnectFromExcel 方法，如图 18 所示。这段代码调用了工作书的 Close 方法，传递的参数值是 True，以确保所有尚未保存的改动都被保存了。这个调用是封装在 try..except 语句块中的，以隐藏数据模块限于工作书打开而被删除时抛出的异常。最后，这个方法调用了 Worksheet，Workbook 和 Application 对象的 Disconnect 方法。

```
{ 保存和关闭工作书，断开与所有 Excel 对象的连接 }
procedure TMainDm.DisconnectFromExcel;
begin
  try
   ExcelBook.Close(True);
  except
  end;
 ExcelSheet.Disconnect;
 ExcelBook.Disconnect;
 ExcelApp.Disconnect;
end;
```

图 18 DisconnectFromExcel 方法.

结 论

Delphi 允许使用 3 种数据存取技术来方便地实现应用程序和 Excel 数据的一体化。使用 Excel 提供数据分析能力可以降低系统的成本和复杂度，并且允许用户在一个他们可能已经知道的产品中分析数据。

Bill Todd 是 Database Group，Inc 的董事长，该公司位于 Phonix 附近，经营数据库咨询和开发业务。他是四本数据库编程书籍的合著者，90 多篇文章的作者，Delphi Informant Magazine 的执行编辑和 Team B（为 Borland Internet 新闻组提供技术支持）的成员。Bill 是一著名的培训师，也是一个经常出席美国和欧洲的 Borland 开发者大会的演讲者。他在世界各地教授 Delphi 编程课程。读者可以通过 mailto:bill@dbginc.com 联系到他。

HTML 语言样式表

● Rick Spence

如果想写互联网应用程序，就要首先理解网页制作语言HTML，并且需要理解如何利用样式表使用 HTML 语言。HTML 语言，顾名思义，是一种为在一个单独文件中组合文本与格式化信息的自身注明结构或逻辑关系设计的一组标签的语言。这其中就蕴藏着它要解决的最大问题：HTML 语言需要同时完成组合文本与制版的任务，而不是分开这两项工作。

HTML 语言经过了几代版本变迁，目前广泛使用的版本是 4.01 版。HTML 4.0 版由世界互联网协会（W3C）推荐并发布于 1997 年 12 月 18 号，其中包括了有关样式表的定义。这样就能够支持用户把对文件的制版工作同内容的制定工作分开来实现。样式表同样支持用户针对 HTML 单元使用在其他环境下无法识别的特殊格式。

下面我们要展示如何在 Delphi CGI 应用程序默认的 HTML 语言中使用样式表来实现相关功能。在脱离样式表系统支持的状态下，用户可以使用 HTML 语言改变样式表的内容——即改变了相应 HTML 文件的版面格式，与此同时并未改变应用程序代码。事实上，也可根据不同用户需要使用不同的样式表来实现功能。但归根到底，如果纯手工实现，将会是一项有些烦琐的反反复复的创建，测试样式表工序，所以我们要展示如何借助 Microsoft 的 Visual InterDev 软件在 WYSIWYG 方式下开发自己的样式表系统。

什么是样式表，需要它的原因

在 HTML 样式表之中蕴藏的理念同 Microsoft Word 字处理软件下的文本方式背后的思路如出一辙。在 Word 字处理软件下，你可以为文件的各个部分创建自己需要的文本样式，比如标题样式和主体文本样式。这种文本样式可以定义一系列格式特性，例如设置字体、选择字号大小以及对齐方式。Word 字处理软件还支持用户创建文件模板，即一些文本样式的简单集合。你可以利用这种模板创建一个或多个文件，而所有创建文件自然共享同样的文本样式。

HTML 样式表，也被叫做级联样式表(后面你会明白为什么可以这么称呼)，可以为你的 HTML 文件实现同样的功能。而样式表系统能够为 HTML 语言提供相同的服务。一个 HTML 样式表包括一系列特定 HTML 元素的定义。举个例子，你可以把 HTML 文本定义为标题 1 模式即代表 20 号 Arial 黑体字。你也可以同样定义标题 2 模式代表 14 号 Times New Roman 字。由此可见一个样式表是一些定义 HTML 元素状态显示的文本样式的集合。

当然你也可以不必在 HTML 语言中使用样式表系统。并且根据我们教授 Delphi 程序开发者的经验，也不是很多人会使用样式表系统。HTML 语言支持用户使用正规 HTML 标识符和属性来规划单个文本格式。这样就为用户提供了支配文本格式的权限，尽管不如借助样式表系统时那么自如和丰富。例如，在下面的 HTML 程序中就包括了标题 1 模式，相应文本就会以蓝色报刊字体显示，同时包含一些以红色 Arial 字体显示的主题段落。

```
<body>
<h1><font face="courier" color="Blue">
Heading 1 here</font></h1>
<p><font face="Arial" color="red">
This is the paragraph</font></p>
</body>
```

如果你想让所有的文本以标题 1 模式一致地显示出来，那么你在所有需要使用<h1>标识符的地方都不得不重复这种格式化，这可是算是一个繁琐枯燥的过程。此外，当你