EZDML 脚本配置说明

http://www.ezdml.com

——HUZ20191027

——本文档最后更新于 20200411，基于 EZDML x64 3.01 版

——HUZ2020 添加剪贴板

很多用户没必要用到脚本，但如有可能仍建议你了解一点脚本。不过，建议只看《EZDML 脚本快速上手》，看完估计就差不多够用了，简单的脚本能解决大部分需求了。

大部分人不需要阅读本文档。如果碰到问题了，或想深入脚本自己生成代码控制之类的了，才需要看下去。

本文档前面是一些奇怪的脚本使用姿势，后面基本上就是类定义参考，多是无聊、枯燥、乏味的东西，内容也很乱，用脚本用出问题的人想找救命草的话可以试试，没什么问题就不要往下看了。

本文混杂着 JS 和 PAS 脚本，但所列的方法和属性很多都是 Pascal 的方式，在 JavaScript 中使用时，首字母要小写，如：

```
PASCAL：  CurOut.Add('当前表名：'+CurTable.Name);
JAVASCRIPT：curOut.add("当前表名："+curTable.name);
```

另外，全局函数还需要加下划线，如：

```
PASCAL：     Sleep(500);
JAVASCRIPT：_sleep(500);
```

再次强调，建议只看《EZDML 脚本快速上手》，有问题时来翻一下本文。

目录

一、 快速上手

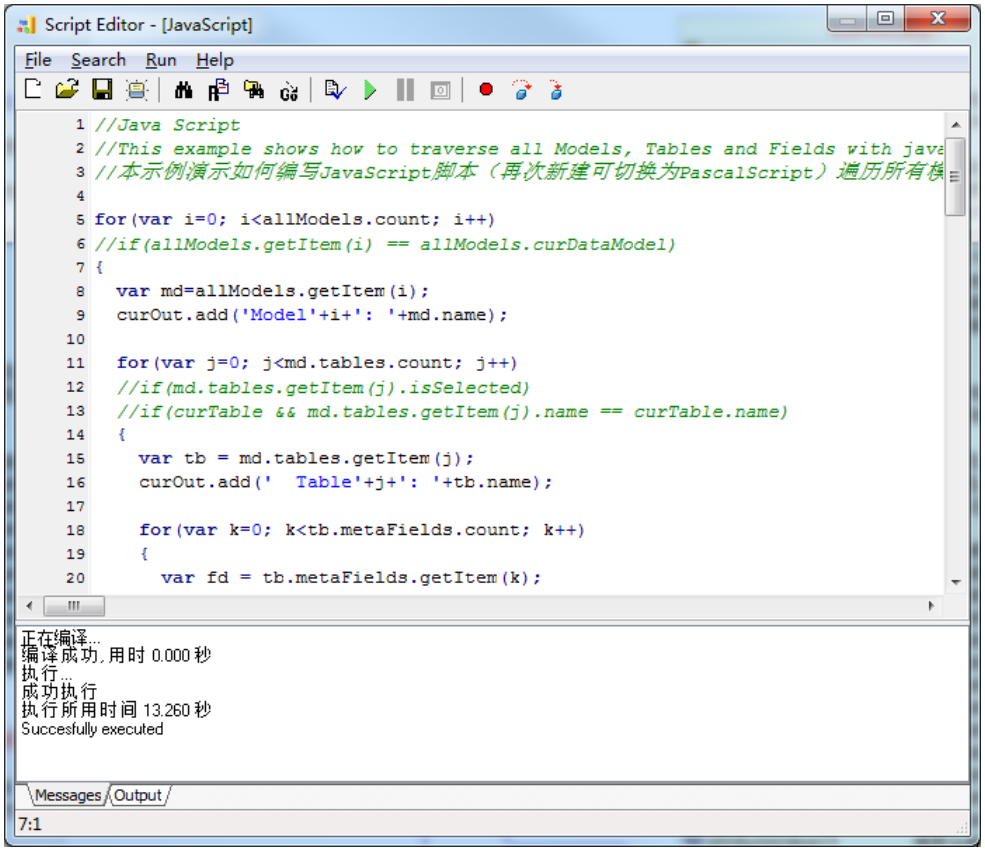
参见《EZDML 脚本快速上手》。

一般情况下，看完快速上手就可以了，没问题就不必看后面的内容了，有问题时再来翻查一下。

二、 脚本窗口

还要继续往下看的话，其实也没什么大不了的，用 EZDML 的很多都是程序员，要阅读本文档并不难。

先说说这个脚本编辑调试窗口，印象中好像是从那个 PAS 脚本的 DEMO 中演化来的，当然可能我作了一些改进：



脚本调试编辑窗口说明：

1. 新建命令（File|New），连续执行会在 JS 与 PAS 之间切换：
   a) 新建结果为一个示例脚本，可能是 JS 也可能会是 PAS
   b) 如果当前是编辑修改过的 JS 脚本，新建后也是 JS 脚本
   c) 新建完 JS 脚本后，如不作任何修改，再次新建，则切换为 PAS 脚本
   d) 新建完 PAS 脚本后，如不作任何修改，再次新建，则切换为 JS 脚本
   e) 如果当前是编辑修改过的 PAS 脚本，则新建后也是 PAS 脚本
2. 打开文件时，按文件扩展名判断脚本类型：
   a) 以下扩展名为 JS：
      i. .js
      ii. .js_
      iii. .js~
   b) 以下扩展名为 PAS：
      i. .pas
      ii. .ps
      iii. .ps_
      iv. .ps~
3. 保存文件（FileSave）：
   ~~a) （此段内容已作废）请注意 Pascal 脚本采用 GBK 或 ASCII 编码，JavaScript 脚本文件则使用了 UTF-8 编码，如果你用外部工具编辑脚本，请遵循此编码规则~~

   20200310：从 2.41 版起，Pascal 脚本和 JavaScript 脚本文件均使用 UTF-8 编码，如果你用外部工具编辑脚本，请遵循此编码规则。

b) 手动保存脚本咱就不说了，说说自动保存。退出脚本窗口时，可能会提示保存，也可能不会，取决于当前脚本的场景，比如右键运行脚本，退出时是自动保存，不会提示的

c) 不管会不会提示，程序都会自动将脚本内容保存到临时文件

d) 每次编译或运行脚本时，程序都会自动将脚本内容保存到临时文件（如果有修改的话）

e) 执行 File|Show Temporary File，可查看当前脚本的临时文件目录内容，可在这找回旧的历史版本文件

4. PAS 脚本支持断点、单步调试、变量查看等功能；JS 不支持调试，只能编译、运行
5. 编译运行结果在下方显示
6. 特殊复制——编辑器里选择一段文本，鼠标右键弹出菜单，可以复制内容，有以下几种姿势：

a) 直接复制，结果为选中的内容的纯文本

b) 执行特殊复制，可复制出用于 JAVA、C、C++、C#、PASCAL、SQL 的字符串代码片断

c) 按住 SHIFT 键复制，结果为所有内容（注：不是选中内容）的带格式（类似 HTML）的富文本（这文档里的代码都是这么复制出来的）

三、 高级功能

下面我要开始我的装 X 表演了。

1. “JSP”和“PSP”页面模板

用过 JSP、ASP、PHP 的可能猜到了，这两个是在页面模板中嵌脚本，运行前自动将其翻译成真正的脚本。下面举例说明。

比如有这么一个表：

```
admin(管理员)
--------------------------------
id(ID)                      PKInteger   //<<关联:users.id>>
department(部门)            String(255)
email(E-mail)               String(255) //<<唯一索引,非空>>
encodedPassword(加密密码)   String(255) //<<非空>>
name(姓名)                  String(255)
username(用户名)            String(255) //<<唯一索引,非空>>
```

admin(管理员)	
id	ID
department	部门
email	E-mail
encodedPassword	加密密码
name	姓名
username	用户名

我想输出一个简单的 HTML 表格：

admin: 管理员	
id	ID
department	部门
email	E-mail
encodedPassword	加密密码
name	姓名
username	用户名

以 JS 为例，正常情况下，我们会这样编写脚本：

```
curOut.add("<html>");
curOut.add("<body>");
curOut.add("");
curOut.add("<table border=1>");
curOut.add("    <tr>");
curOut.add("        <td colspan=\"2\">");
curOut.add("          "+curTable.name+": "+curTable.caption);
curOut.add("        </td>");
curOut.add("    </tr>");
curOut.add("");

for(vari=0;i<curTable.metaFields.count;i++)
{
varf=curTable.metaFields.getItem(i);

curOut.add("    <tr>");
curOut.add("        <td>");
curOut.add(f.name);
curOut.add("        </td>");
curOut.add("        <td>");
curOut.add(f.displayName);
curOut.add("        </td>");
curOut.add("    </tr>");
}

curOut.add("");
curOut.add("</table>");
curOut.add("</body>");
curOut.add("</html>");
```

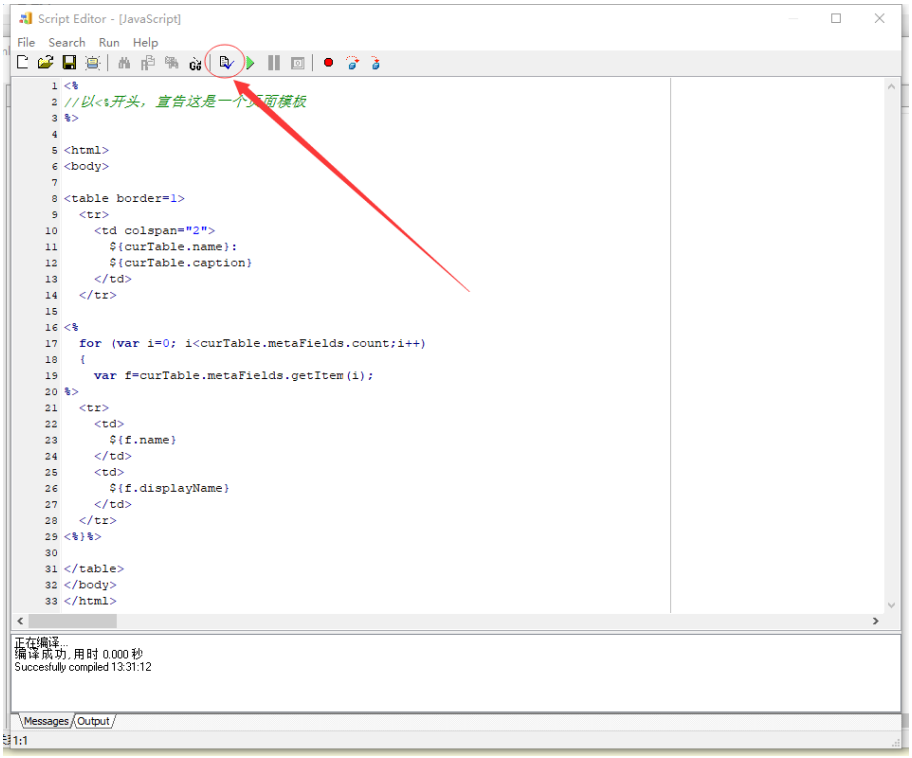
比较麻烦和反人类，下面我们换一种姿势，用 JSP 的方式重写，代码如下：

```
<%
//以<%开头，宣告这是一个页面模板
%>

<html>
<body>

<tableborder=1>
<tr>
```

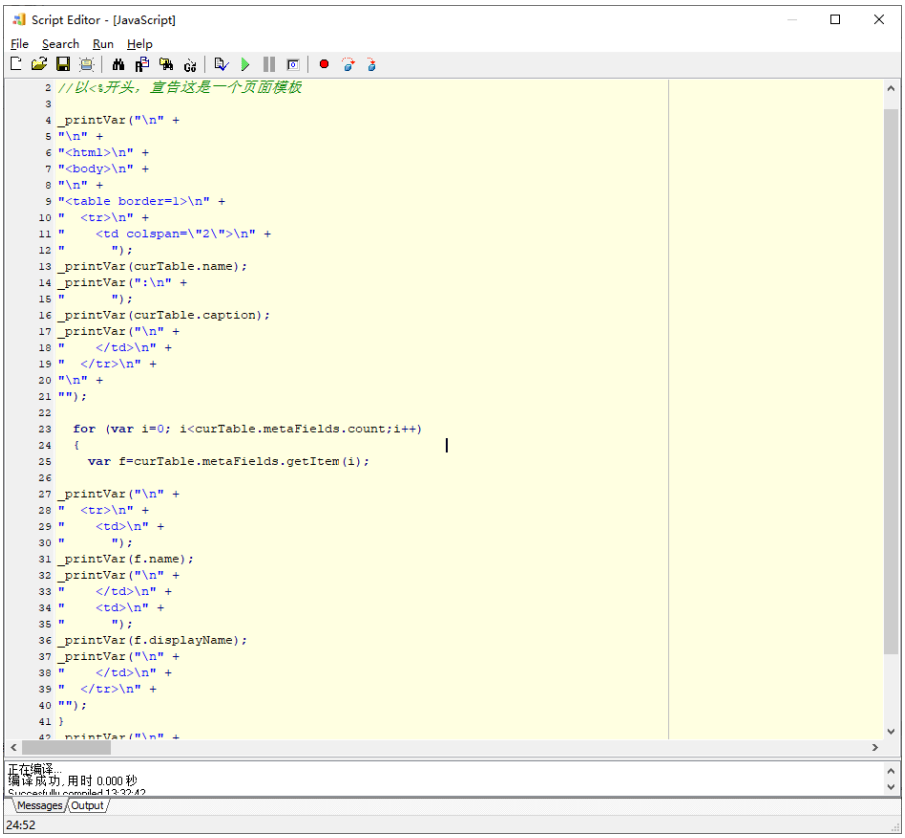
```
<tdcolspan="2">
${curTable.name}:
${curTable.caption}
</td>
</tr>

<%
for(vari=0;i<curTable.metaFields.count;i++)
{
varf=curTable.metaFields.getItem(i);
%>
<tr>
<td>
${f.name}
</td>
<td>
${f.displayName}
</td>
</tr>
<%}%>

</table>
</body>
</html>
```

是不是感觉好一些？

写好后，先不要运行，点击编译工具按钮：



这时会发现，程序会自动预编译处理，将“JSP”转成 JS，脚本框变成只读，背景会变成浅黄色：



运行结果：

点右侧的“用浏览器打开”（文件名以 Html 开头的脚本会显示这按钮）：

下面是 PSP（PASCAL）的版本：

```
<%
begin
%>

<html>
<body>

<table border=1>
<tr>
<td colspan="2">
${CurTable.Name}:
${CurTable.Caption}
</td>
</tr>

<%
gvar I:Integer; //PASCAL 要求预先定义 var 所有变量，在输出页面文件时很麻烦，所以我搞了个 gvar
gvar F: TCtMetaField; //PSP 里 gvar 开头的行，执行前会自动挪到最顶
for I:=0to CurTable.MetaFields.Count-1do
begin
    F:=CurTable.MetaFields.Items[I];
%>
<tr>
<td>
${F.Name}
</td>
<td>
${F.DisplayName}
</td>
</tr>
<%end;%>

</table>
</body>
</html>

<%
end.
%>
```

运行效果是一样的。

2. 包含引用文件

包含引用文件，程序员都懂的，就是脚本太多了，全放一起麻烦，要分开；或者有公共函数库，不想拷来拷去的，希望一次编写多次引用，要独立出来，方便维护。

Pascal 脚本包含引用文件：**{$INCLUDE 文件名}** 或 **{$I 文件名}**

JavaScript 脚本包含引用文件：**#include "文件名"**

包含文件名一般都是以宿主文件所在目录为参考的相对目录的文件名。

JS 脚本包含是我自己做的，简单说就是执行前把包含文件内容拼进来（CTRL+F9 编译时能看到所有 Include 文件拼进来后的最终脚本内容）；PAS 脚本包含由脚本引擎完成，不过调试时我也会将内容拼一块。

3. 创建和显示窗体

一般来说没有必要自己写窗体，写这主题我感觉我好像有点是非要告诉别人茴香豆的茴字的第 N 种写法似的。正常看不惯的话请跳过吧；但如果你碰上了不正常的情况，或者闲得荡藤想了解一下茴字的第 N+1 种写法的话，可以瞅一瞅。

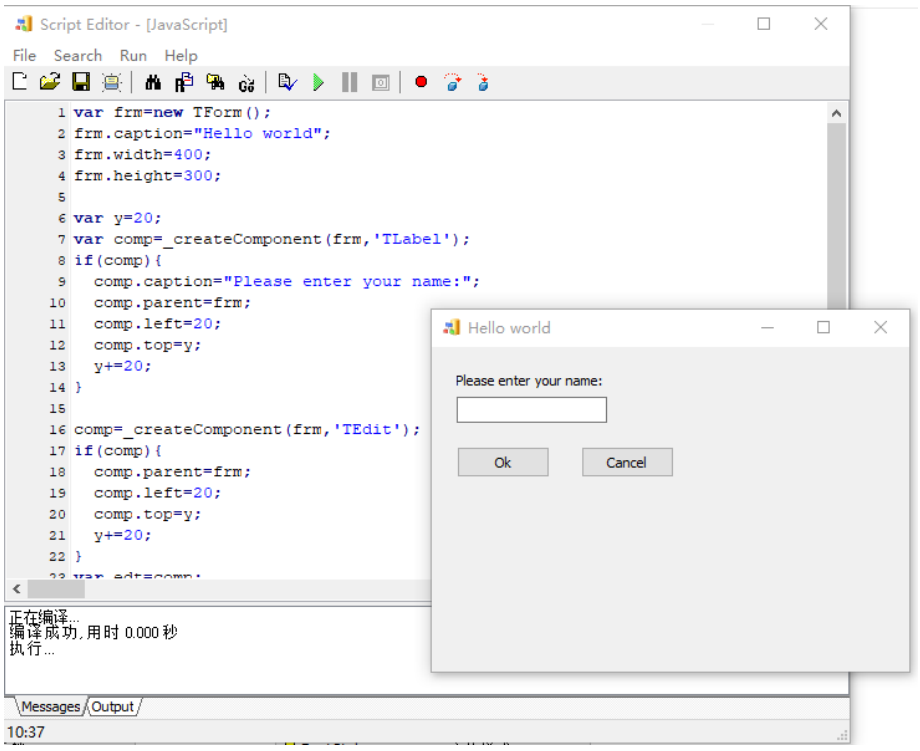
再强调一下，用脚本创建使用窗体很麻烦，简单的还好，复杂的建议还是写个 DLL 更快。

示例开始，假设你要显示个界面让用户输入一段内容，示例 JS 脚本如下：

```
varfrm=newTForm();
frm.caption="Hello world";
frm.width=400;
frm.height=300;

vary=20;
varcomp=_createComponent(frm,'TLabel');
if(comp){
comp.caption="Please enter your name:";
comp.parent=frm;
comp.left=20;
comp.top=y;
y+=20;
}

comp=_createComponent(frm,'TEdit');
if(comp){
comp.parent=frm;
comp.left=20;
comp.top=y;
y+=20;
}
varedt=comp;
```

```
y+=20;
comp=_createComponent(frm,'TButton');
if(comp){
comp.parent=frm;
comp.left=20;
comp.top=y;
comp.caption='Ok';
_setCompPropValue(comp,'ModalResult',IDOK)
}

comp=_createComponent(frm,'TButton');
if(comp){
comp.parent=frm;
comp.left=120;
comp.top=y;
comp.caption='Cancel';
_setCompPropValue(comp,'ModalResult',IDCANCEL)
y+=20;
}

if(frm.showModal()==IDOK)
alert("your name is: "+edt.text);
```

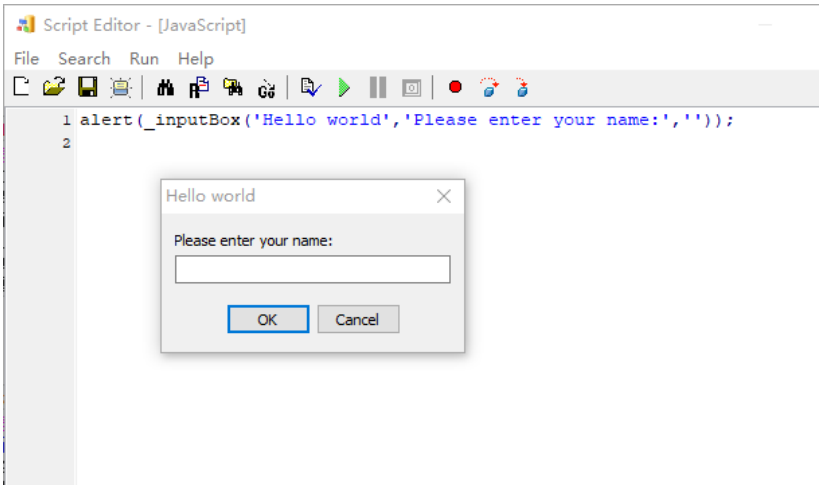
运行结果如下：



当然了，只是示例。真要输入的话，用下面这一句就行了：

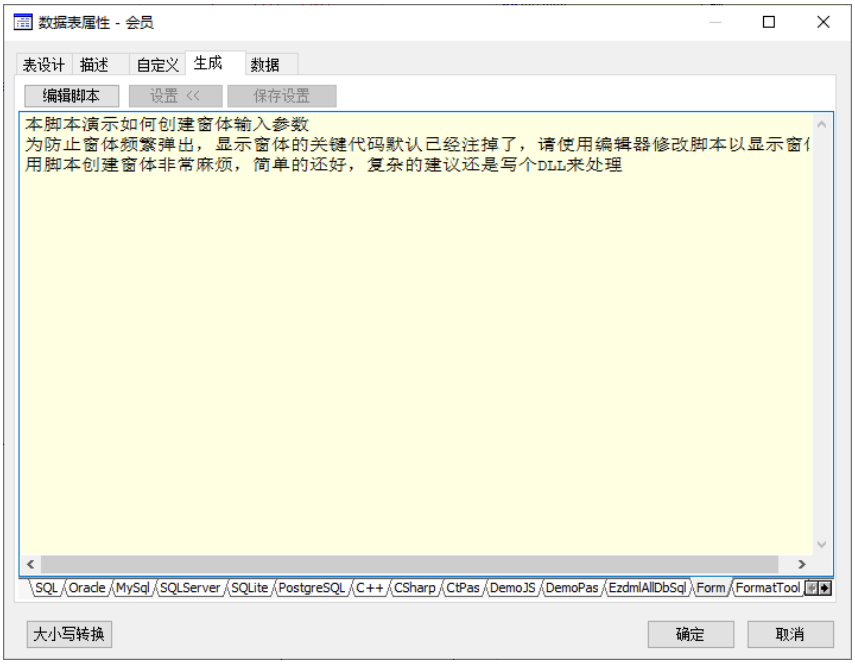
```
alert(_inputBox('Hello world','Please enter your name:',''));
```

笑果是一样的，或者更好：

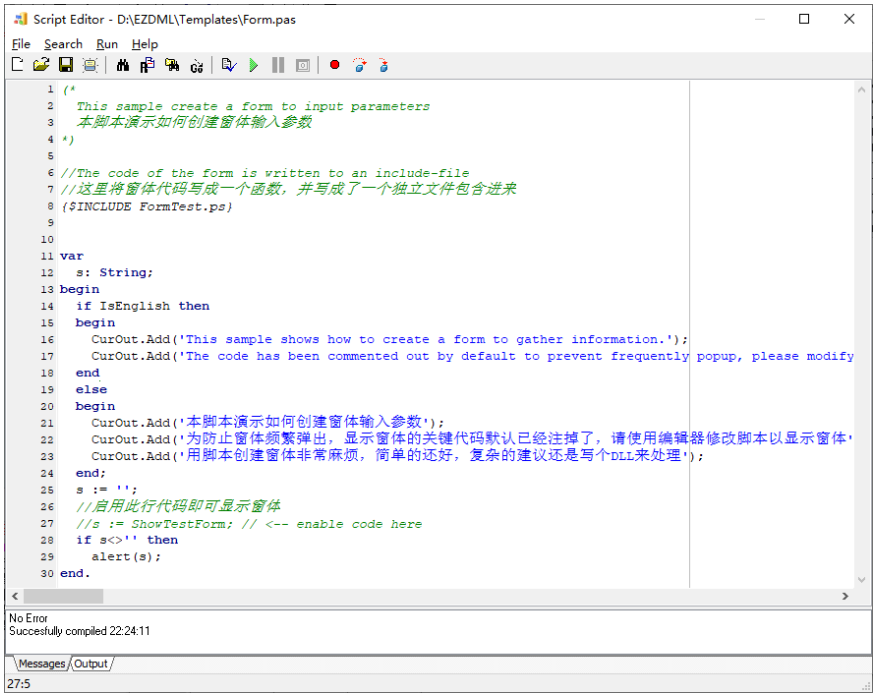


接下来我们来看下 PAS 的窗体示例，它跟 JS 是不一样的机制。JS 无法拦截事件，因此只能做简单的界面，各控件也需要用反射的方法动态创建和赋值；PASCAL 脚本要强很多，~~可以拦截事件~~（注：x64 版目前不支持 PASCAL 脚本拦截事件），很多控件都可以直接创建。

窗体的 PAS 脚本 EZDML 自带了一个，随便打开一个表的属性窗口，切到“生成”页，在下方找到 Form：

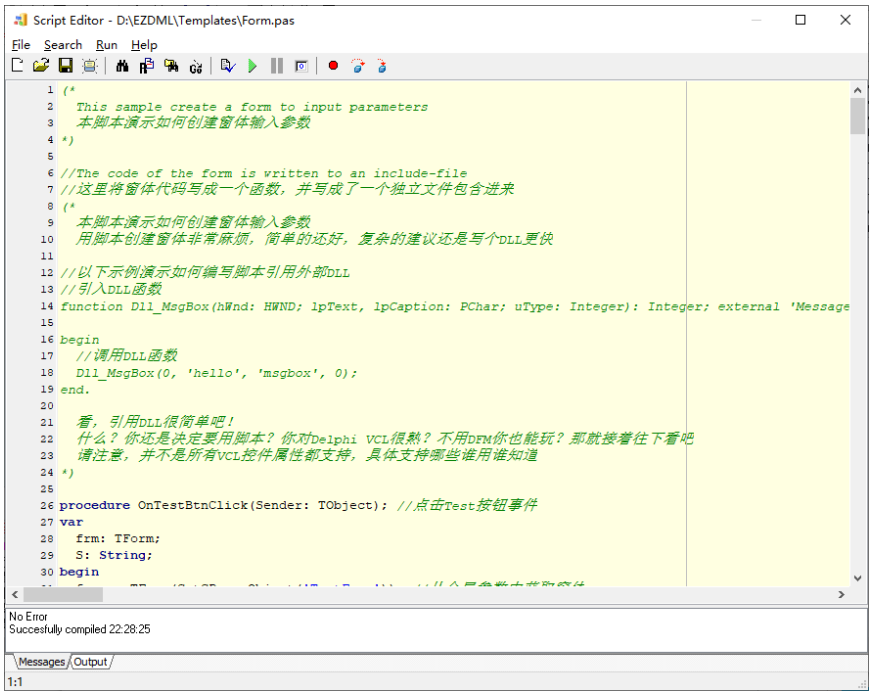


点“编辑脚本”，显示示例脚本：

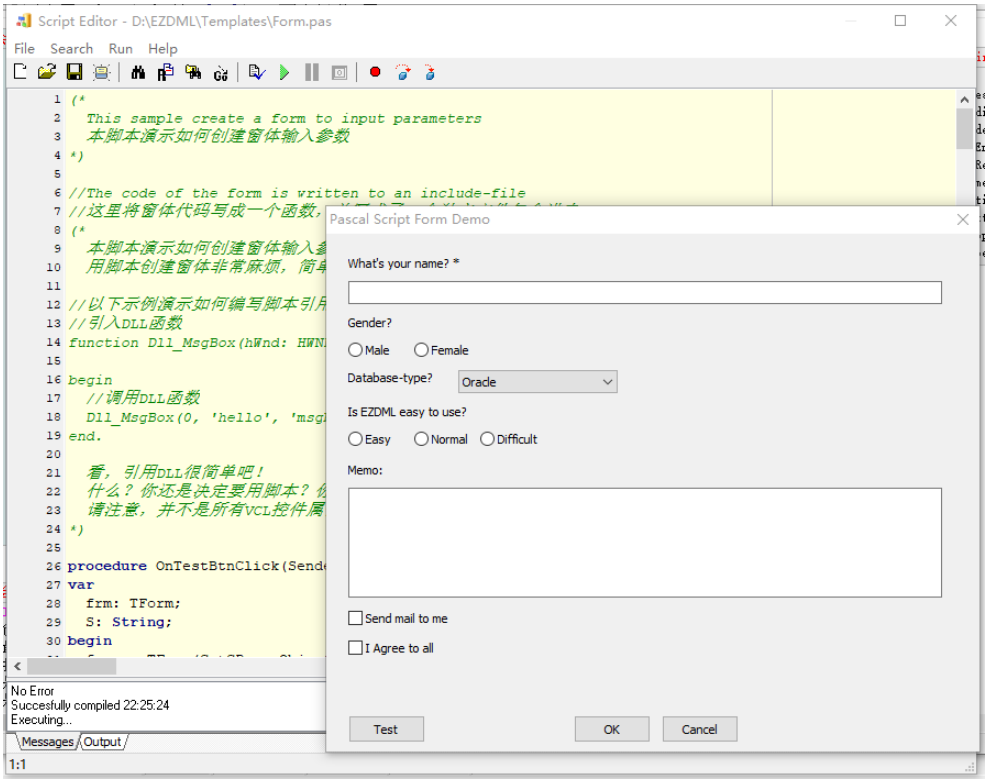


可以看到第 8 行脚本是从另一个文件“FormTest.ps”包含引用进来的。

启用第 27 行的注释代码，按 Ctrl+F9 编译，编辑器底色变淡黄，这时包含引用的文件内容被加载进来了：



按 F9 运行，窗体显示出来了：



脚本代码如下（红色高亮代码是拦截按钮点击事件的示例）（注：x64 版目前不支持 PASCAL 脚本拦截事件）：

```
procedureOnTestBtnClick(Sender: TObject); //点击 Test 按钮事件（注：x64 版目前不支持 PASCAL 脚本拦截事件）
var
   frm: TForm;
   S: String;
begin
   frm := TForm(GetGParamObject('TestForm')); //从全局参数中获取窗体

//名字为空？
   S := TEdit(FindChildComp(frm, 'edtName')).Text;
if Trim(S)='then
begin
if Application.MessageBox('Name is empty, fill TestName to continue?',
'Test button clicked', MB_YESNOCANCEL or MB_ICONWARNING) <> IDYES then
         Exit;
      TEdit(FindChildComp(frm, 'edtName')).Text := GetGParamValue('TestName');
end;

//frm.Close; //关闭窗体
   frm.ModalResult := mrOk; //关闭窗体并返回 ok
end;

function ShowTestForm: String; //执行测试窗体代码
var
   frm: TForm;
   pnl: TPanel;
   lbTip: TLabel;
   fx, fy, fw, dy: Integer;
```

```
  S, res: String;
begin
  Result := '';
  frm:=TForm.Create(nil); //创建窗体
try
    SetGParamValue('TestName', 'Somebody'); //全局参数在 EXE 运行期间一直有效
    SetGParamObject('TestForm', frm); //在全局参数中记录窗体对象
with frm do//设置窗体属性
begin
      BorderStyle := bsDialog; //默认为 bsSizeable
      Position := poScreenCenter; //默认为 poDesigned
//AutoScroll := True;
      Caption := 'Pascal Script Form Demo';
      Width := 600;
      Height := 500;
end;

    fx := 20;
    fy := 20;
    fw := 540;
    dy := 10;

with TLabel.Create(frm) do//名字
begin
      Parent := frm;
      Left := fx;
      Top := fy;
      Caption := 'What''s your name? *';
      fy := fy + Height + dy;
end;

with TEdit.Create(frm) do
begin
      Name := 'edtName'; //我们在后面通过名字来获取值，就不需要用变量记录控件指针了
      Text := '';
      Parent := frm;
      Left := fx;
      Top := fy;
      Width := fw;
      fy := fy + Height + dy;
end;

。。。此处省略几百行。。。

with TButton.Create(frm) do//测试
begin
      Parent := frm;
//Name := 'btnTest';
      Caption := 'Test';
OnClick := @OnTestBtnClick; //点此按钮将执行 OnTestBtnClick 函数
      Left := fx;
      Top := fy;
      Width := 70;
//fy := fy + Height + dy;
end;

with TButton.Create(frm) do//确定
```

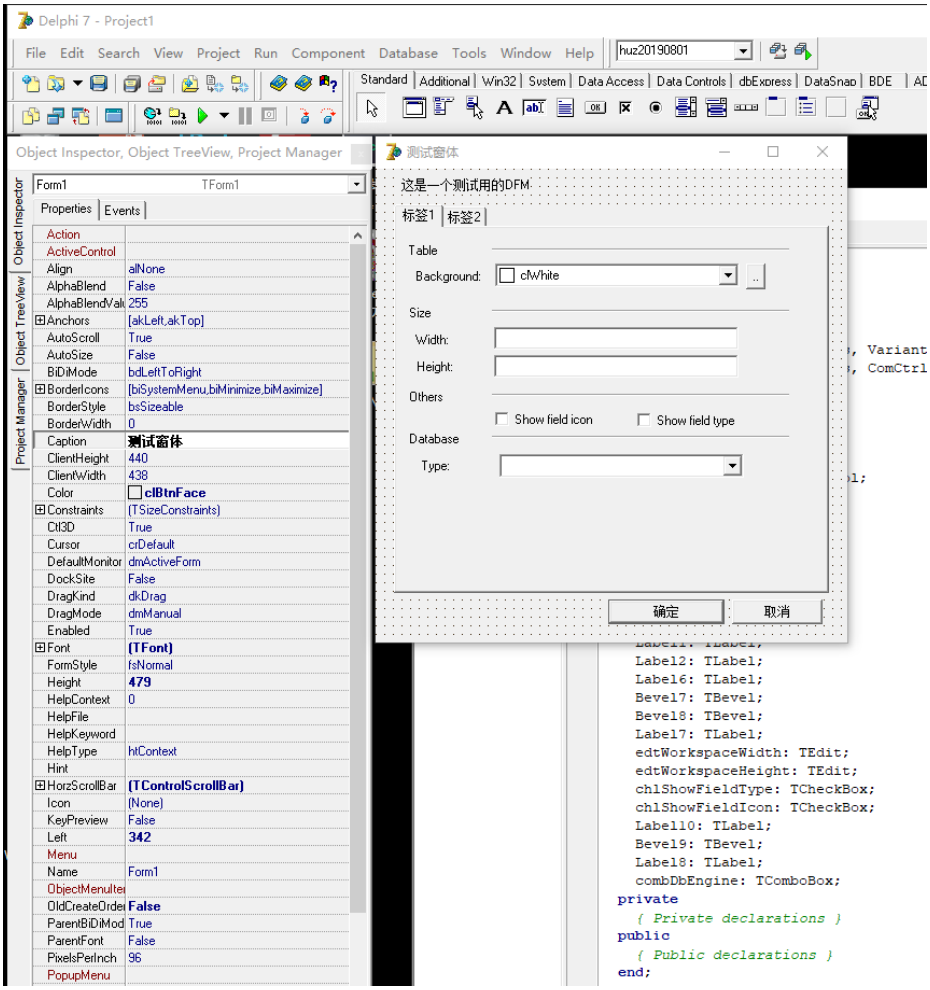
```
begin
        Parent := frm;
//Name := 'btnOK';
        Caption := 'OK';
        ModalResult := mrOk; //点此按钮将关闭窗口并返回 mrOk
        Left := (frm.Width -150) div2;
        Top := fy;
        Width := 70;
//fy := fy + Height + dy;
end;

with TButton.Create(frm) do//取消
begin
        Parent := frm;
//Name := 'btnCancel';
        Caption := 'Cancel';
        Cancel := True; //表示按 ESC 时执行此按钮
        ModalResult := mrCancel; //点此按钮将关闭窗口并返回 mrCancel
        Left := (frm.Width -150) div2+80;
        Top := fy;
        Width := 70;
//fy := fy + Height + dy;
end;


while frm.ShowModal=mrOk do//如果输入结果不符合要求，将一直循环
begin

        res := 'Form result:'#10; //获取结果

//名字
        S := TEdit(FindChildComp(frm, 'edtName')).Text;
if Trim(S)='' then
begin
//名字为空，不允许
with lbTip do
begin//显示警告
            Caption := 'Name needed!';
            Visible := True;
end;
          frm.ActiveControl := TEdit(FindChildComp(frm, 'edtName'));
          Continue; //继续循环
end;
        res := res + 'Name: '+ S + #10;
。。。此处省略几十行。。。
        Result :=res;   //返回结果
        Break; //退出循环
end;
finally
     frm.Free;
end;
end;
```

可以看到，用代码创建窗体很麻烦。

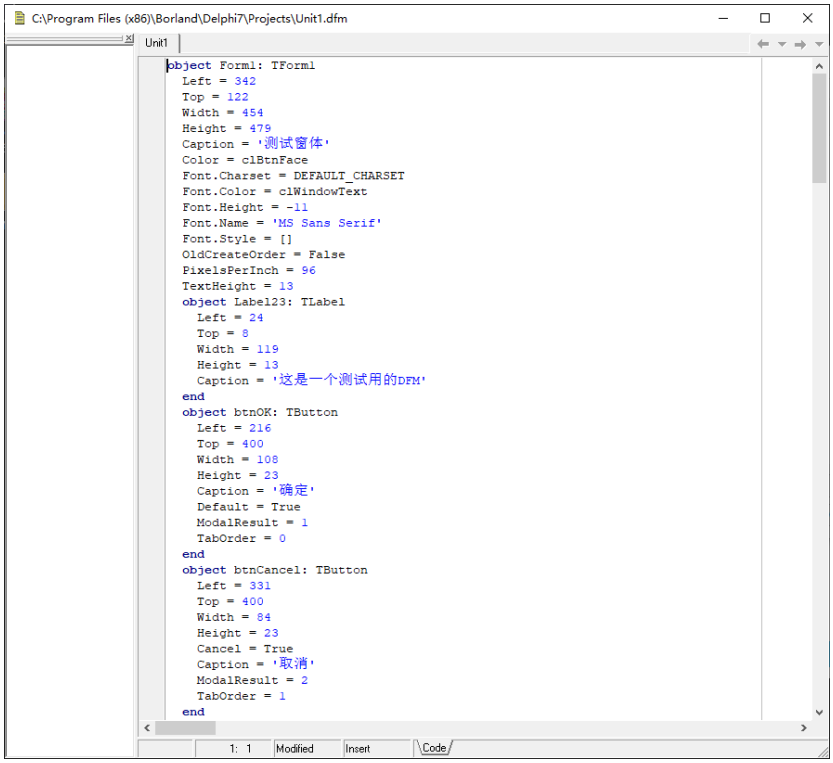
不过，EZDML 还增加了一种新的姿势，是用 ReadDfmComponents 函数加载 DFM：如果你

能用一下 DELPHI，在 DELPHI 上设计好窗体，再把 DFM 拷出来，事情就会简单很多。示例如下：

先在 DELPHI（这示例是 D7）中，新建窗体，并设计如下（请保用常用控件，太高级的控件我没加进来的）：



按 ALT+F12，显示 DFM 文本内容：

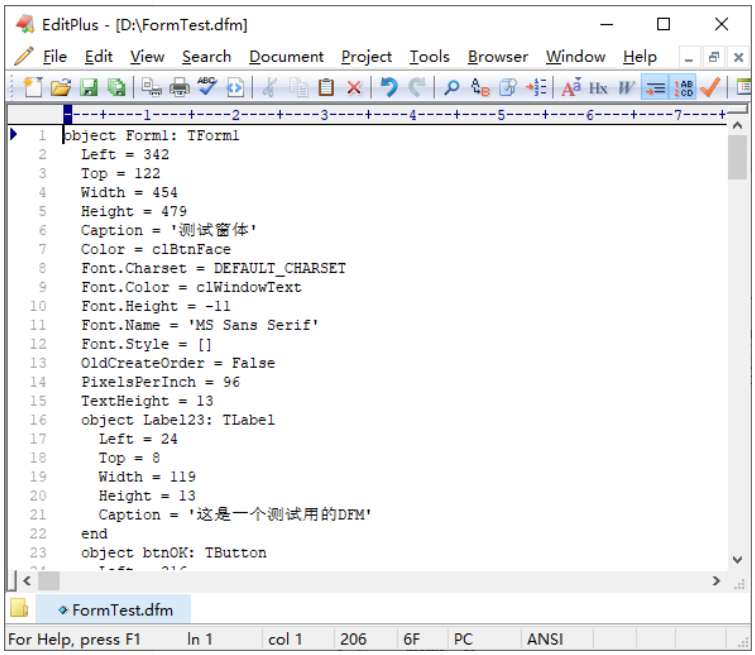


将其复制下来：

```
object Form1: TForm1
  Left = 344
  Top = 127
  Width = 454
  Height = 479
  Caption = '测试窗体'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
object Label23: TLabel
    Left = 24
    Top = 8
    Width = 119
    Height = 13
    Caption = '这是一个测试用的 DFM'
end
object btnOK: TButton
    Left = 216
    Top = 400
    Width = 108
    Height = 23
    Caption = '确定'
    Default = True
    ModalResult = 1
    TabOrder = 0
end
object btnCancel: TButton
    Left = 331
    Top = 400
    Width = 84
    Height = 23
    Cancel = True
    Caption = '取消'
    ModalResult = 2
    TabOrder = 1
end
object PageControl1: TPageControl
    Left = 17
    Top = 33
    Width = 404
    Height = 361
    ActivePage = TabSheet2
    TabOrder = 2
object TabSheet2: TTabSheet
      Caption = '标签 1'
      ImageIndex = 1
object Bevel1: TBevel
        Left = 87
        Top = 17
        Width = 276
        Height = 3
        Shape = bsTopLine
end
object Label12: TLabel
        Left = 16
        Top = 36
        Width = 61
        Height = 13
        Caption = 'Background:'
end
object Label18: TLabel
        Left = 9
        Top = 12
        Width = 27
        Height = 13
        Caption = 'Table'
end
object Label1: TLabel
        Left = 16
        Top = 95
        Width = 31
        Height = 13
        Caption = 'Width:'
end
object Label2: TLabel
        Left = 17
        Top = 120
        Width = 34
        Height = 13
        Caption = 'Height:'
```

```
end
object Label6: TLabel
        Left = 10
        Top = 70
        Width = 20
        Height = 13
        Caption = 'Size'
end
object Bevel7: TBevel
        Left = 87
        Top = 74
        Width = 276
        Height = 2
        Shape = bsTopLine
end
object Bevel8: TBevel
        Left = 87
        Top = 152
        Width = 276
        Height = 2
        Shape = bsTopLine
end
object Label7: TLabel
        Left = 10
        Top = 148
        Width = 31
        Height = 13
        Caption = 'Others'
end
object Label10: TLabel
        Left = 10
        Top = 187
        Width = 46
        Height = 13
        Caption = 'Database'
end
object Bevel9: TBevel
        Left = 87
        Top = 191
        Width = 276
        Height = 2
        Shape = bsTopLine
end
object Label8: TLabel
        Left = 21
        Top = 212
        Width = 27
        Height = 13
        Caption = 'Type:'
end
object clbFill: TColorBox
        Left = 90
        Top = 31
        Width = 227
        Height = 22
        DefaultColorColor =
clWhite
        NoneColorColor =
clWhite
        Selected = clWhite
        ItemHeight = 16
        TabOrder = 0
end
object btnFill: TButton
        Left = 323
        Top = 31
        Width = 18
        Height = 24
        Caption = '..'
        TabOrder = 1
end
object edtWorkspaceWidth:
TEdit
        Left = 89
        Top = 90
        Width = 227
        Height = 21
        TabOrder = 2
end
object edtWorkspaceHeight:
TEdit
        Left = 89
        Top = 116
        Width = 227
        Height = 21
        TabOrder = 3
end
object chlShowFieldType:
TCheckBox
        Left = 222
        Top = 168
        Width = 127
        Height = 18
        Caption = 'Show field
type'
        TabOrder = 4
end
object chlShowFieldIcon:
TCheckBox
        Left = 90
        Top = 167
        Width = 127
        Height = 18
        Caption = 'Show field
icon'
        TabOrder = 5
end
object combDbEngine:
TComboBox
        Left = 94
        Top = 208
        Width = 227
        Height = 22
        Style =
csOwnerDrawFixed
        ItemHeight = 16
        TabOrder = 6
        Items.Strings = (
'ORACLE'
'MYSQL'
'SQLSERVER'
'STANDARD')
end
end
object TabSheet1: TTabSheet
        Caption = '标签 2'
        ImageIndex = 1
end
end
end
```

保存到一个文件，假设是 D:\FormTest.dfm：



新建一个 PAS 脚本，内容如下：

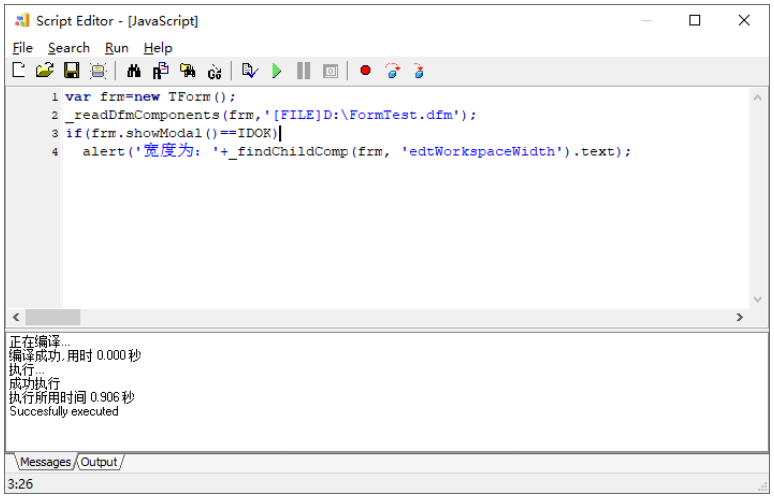
```
var
   frm: TForm;
begin
   frm:=TForm.Create(nil); //创建窗体
try
      ReadDfmComponents(frm,'[FILE]D:\FormTest.dfm'); //加载 DFM 文件
if frm.ShowModal=mrOk then//显示模态窗口
        alert('宽度为：'+TEdit(FindChildComp(frm, 'edtWorkspaceWidth')).Text);
finally
      frm.Free;
end;
end.
```

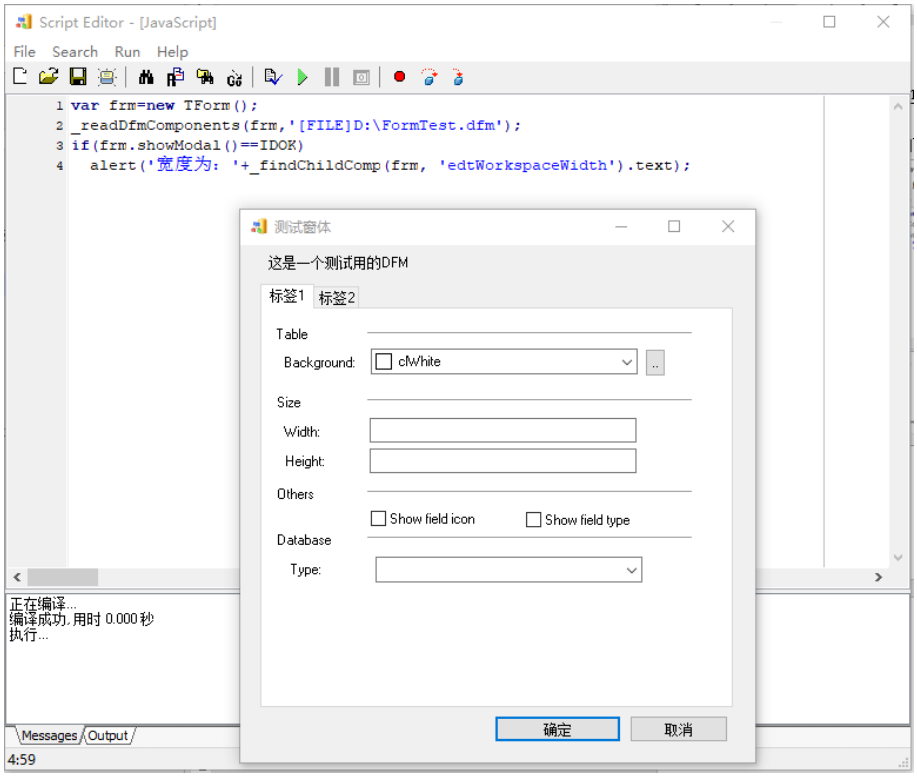
ReadDfmComponents 第二个参数是 DFM 内容，或者是文件名（需要以[FILE]开头）。

当然，JS 也是可以的：

```
varfrm=newTForm();
_readDfmComponents(frm,'[FILE]D:\FormTest.dfm');
if(frm.showModal()==IDOK)
alert('宽度为：'+_findChildComp(frm,'edtWorkspaceWidth').text);
```

运行结果如下：





相对简单对了吧。

这次先讲到这，下次再讲茴字的第 N+2 种写法。

4. 获取和使用程序主窗体

通过 application.mainForm 能够直接获取主窗体。获取主窗体有什么用呢？下面以 JS 为例作演示：

1) 首先，获取主窗体：var frm=application.mainForm;
2) 显示窗体标题：alert(frm.caption);
3) 修改标题：frm.caption="Hello EZDML";
4) 获取状态和大小：alert(frm.windowState+" "+frm.boundsRect);
5) 修改位置和大小：frm.left=20;frm.top=10;frm.width=800;frm.height=600;

或者这样也可以：frm.boundsRect="20,10,800,600";

6) 设置状态：frm.windowState='wsMaximized';

7) 将它隐藏，再显示：

```
frm.hide();
alert(frm.caption);
frm.show();
```

8) 查找组件：alert(_findChildComp(frm,'MainMenu1')._className);

执行以下脚本可以遍历所有组件：

```
var frm=application.mainForm;
var s="Main form componentCount: "+frm.componentCount;
for(var i=0;i<frm.componentCount;i++)
{
  var comp=frm.getComponent(i);
  if(comp.name)
  s=s+"\n"+comp.name+":"+comp._className;
}
alert(s);
```

9) 设置其它属性

比如要去掉主菜单：_setCompPropObject(frm,'Menu',null);

再加上：_setCompPropObject(frm,'Menu',_findChildComp(frm,'MainMenu1'));

10) 执行命令

主窗体目前有以下命令：

```
actOpenLastFile1
actGoTbFilter
actNewFile
actOpenFile
actSaveFile
actSaveFileAs
actShowFileInExplorer
actShowTmprFile
actExitWithoutSave
actExit
actNewTable
actNewModel
actImportDatabase
actGenerateDatabase
actGenerateCode
actTogglePhyView
actModelOptions
actExportModel
actExecScript
actFindObjects
actEditSettingFile
actEditMyDict
actEditGlobalScript
actBrowseScripts
actBackupDatabase
actRestoreDatabase
actSqlTool
actCharCodeTool
actBrowseCustomTools
```

```
actQuickStart
actEzdmlHomePage
actAboutEzdml
```

比如要执行 actAboutEzdml 打开“关于”：

```
_findChildComp(frm,'actAboutEzdml').execute();
```

11）关闭它（注意：程序会结束）：frm.close();

以上过程，其实对其它窗体（可用 screen.activeForm 对象获取当前窗体，或用 screen 遍历所有窗体，参见后面的说明）也适用，这里不展开了。

5. 运行其它程序

如果要运行一个 EXE，或打开一个文档，方法是调 ShellOpen 这个过程，它的声明如下：

```
procedure ShellOpen(FileName, Parameters, Directory: String); //执行 Shell 的 open 命令
```

使用示例如下（PAS）：

```
begin
  ShellOpen('notepad.exe','','');
end.
```

结果是打开了记事本，没什么好看的，这里就不截图了。

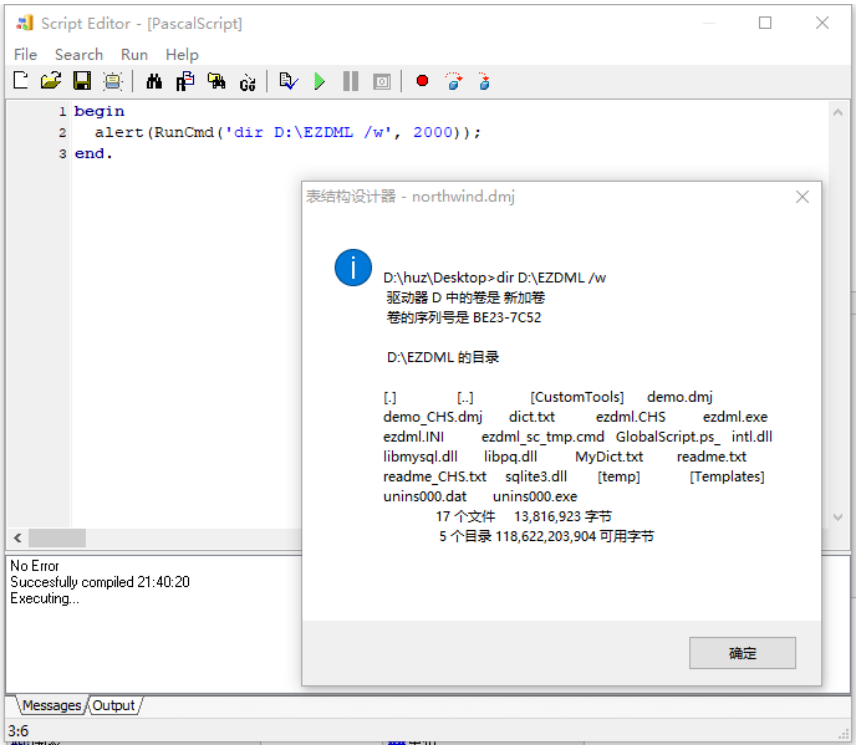
如果要运行一个批处理 DOS 命令，则需要用 RunCmd 这个函数，其声明如下：

```
function RunCmd(cmd: string; timeout: Integer): string; //运行批处理命令，并返回结果
```

示例如下（PAS）：

```
begin
  alert(RunCmd('dir D:\EZDML /w', 2000));
end.
```

效果截个图：

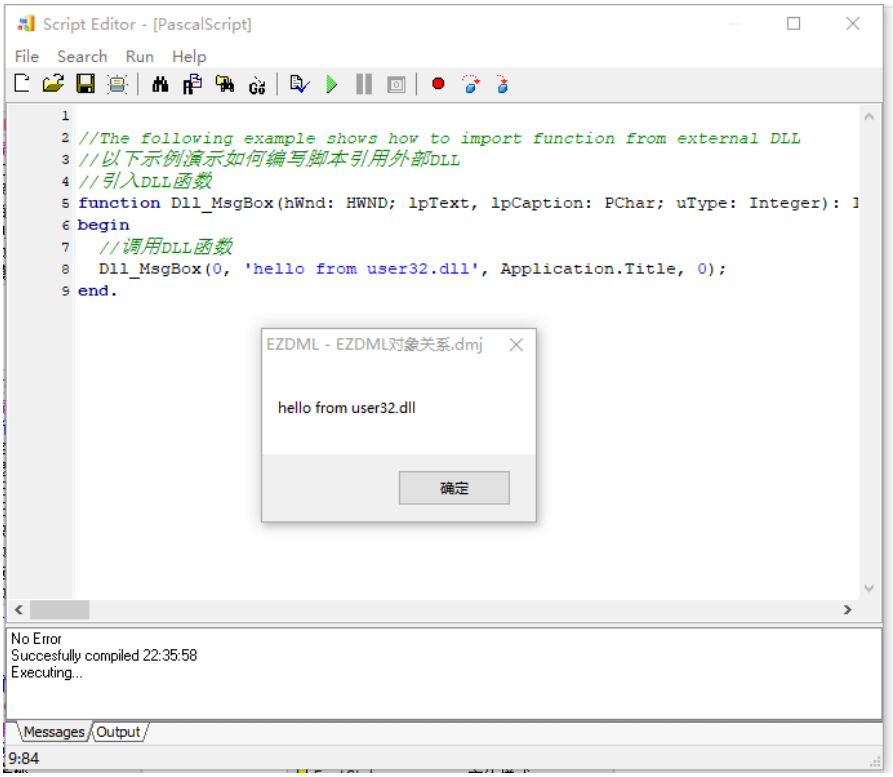


也许有人要问运行其它程序这功能有什么用，嗯，不知道，谁用谁知道。

6. 加载 DLL

PAS 支持加载 DLL，JS 不支持。PAS 加载 DLL 的示例如下：

```
//The following example shows how to import function from external DLL
//以下示例演示如何编写脚本引用外部 DLL
//引入 DLL 函数
function Dll_MsgBox(hWnd: HWND; lpText, lpCaption: PChar; uType: Integer): Integer;
external'MessageBoxA@user32.dll stdcall';
begin
//调用 DLL 函数
  Dll_MsgBox(0, 'hello from user32.dll', Application.Title, 0);
end.
```



JS 虽然不能直接支持 DLL，但 JS 可以混合调用 PAS 来间接调用 DLL，参见下节。

7. JS 与 PAS 混合

JS 可以混合调用 PAS，比如利用 PAS 来调用 DLL，再通过全局变量将结果传回来，主要是用这个方法：

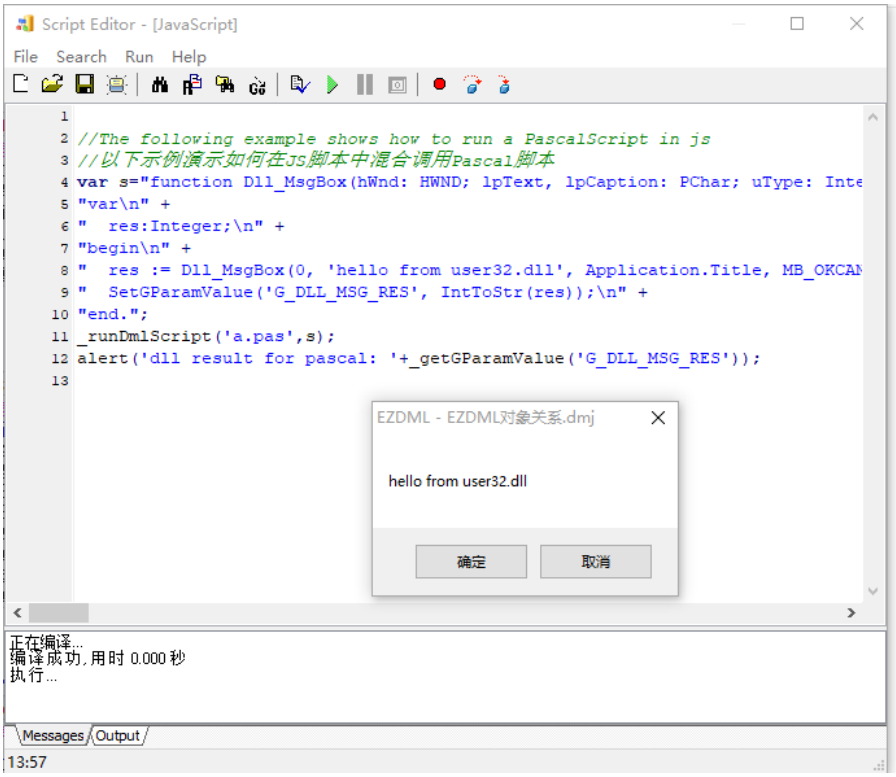
_runDmlScript(AFileName, AScript); *//运行脚本，文件后缀名决定脚本类型，AScript 为脚本内容，为空则从文件加载*

示例如下：

```
//The following example shows how to run a PascalScript in js
//以下示例演示如何在 JS 脚本中混合调用 Pascal 脚本
vars="function Dll_MsgBox(hWnd: HWND; lpText, lpCaption: PChar; uType: Integer): Integer; external
'MessageBoxA@user32.dll stdcall';\n"+
"var\n"+
"   res:Integer;\n"+
"begin\n"+
"   res := Dll_MsgBox(0, 'hello from user32.dll', Application.Title, MB_OKCANCEL);\n"+
"   SetGParamValue('G_DLL_MSG_RES', IntToStr(res));\n"+
"end.";
```

```
_runDmlScript('a.pas',s);
alert('dll result for pascal: '+_getGParamValue('G_DLL_MSG_RES'));
```

运行效果如下：

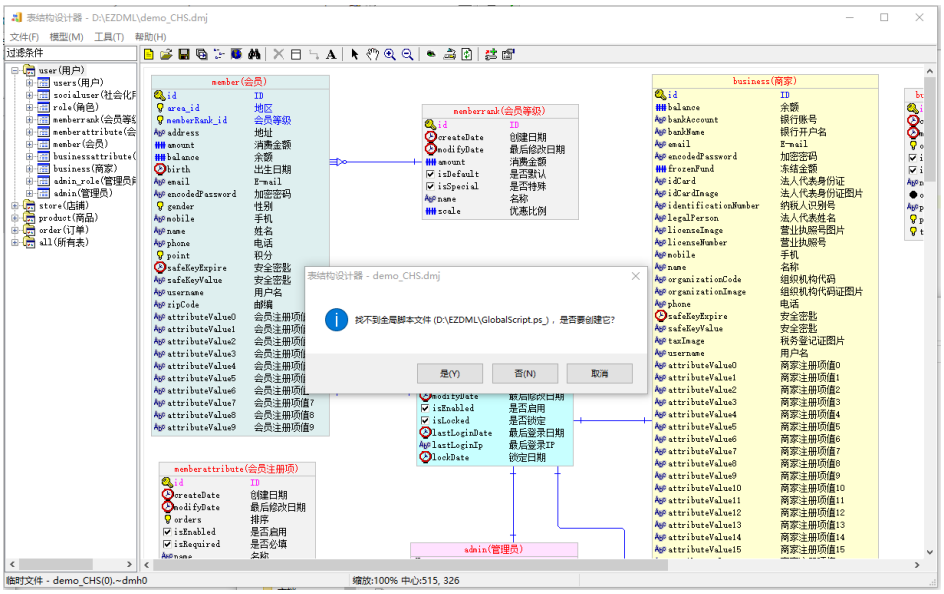


反过来 PAS 调 JS 也是可以的。理论上 JS 调 JS 和 PAS 调 PAS 也是可以的（吃饱了撑的吧），都是调这个方法，这里不展开了。

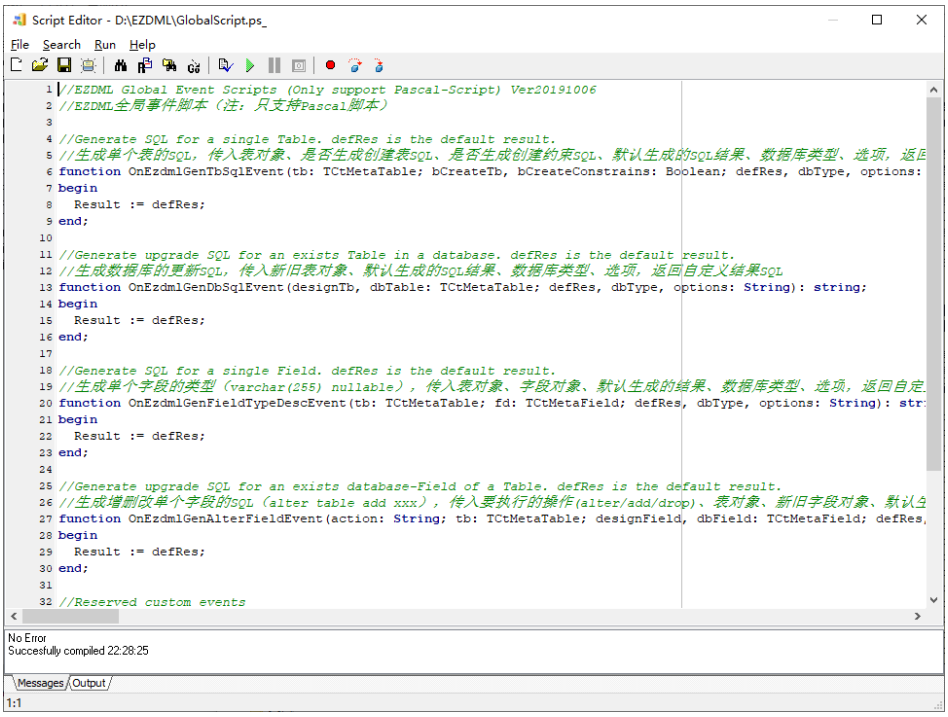
8. 全局事件脚本

全局事件脚本文件为 GlobalScript.ps_，位于主程序目录下。你可以在这接管生成 SQL、弹出窗口等事件。由于没找到 JS 拦截事件的方式，该脚本目前仅支持 PASCAL，不支持 JAVASCRIPT。

执行主窗口菜单命令“工具|编辑全局事件脚本”，第一次时程序会提示创建全局脚本文件：



点“是”创建并打开全局脚本进行编辑调试：




```
//EZDML Global Event Scripts (Only support Pascal-Script) Ver20191006
//EZDML 全局事件脚本（注：只支持 Pascal 脚本）

//Generate SQL for a single Table. defRes is the default result.
//生成单个表的 SQL，传入表对象、是否生成创建表 SQL、是否生成创建约束 SQL、默认生成的 SQL 结果、数据库类型、选项，返回自定义结果 SQL
//补充：bCreateTb, bCreateConstrains 均为 False 时表示生成的是 select 预览数据的 SQL 语句
function OnEzdmlGenTbSqlEvent(tb: TCtMetaTable; bCreateTb, bCreateConstrains: Boolean; defRes,
dbType, options: String): string;
begin
   Result := defRes;
end;
```

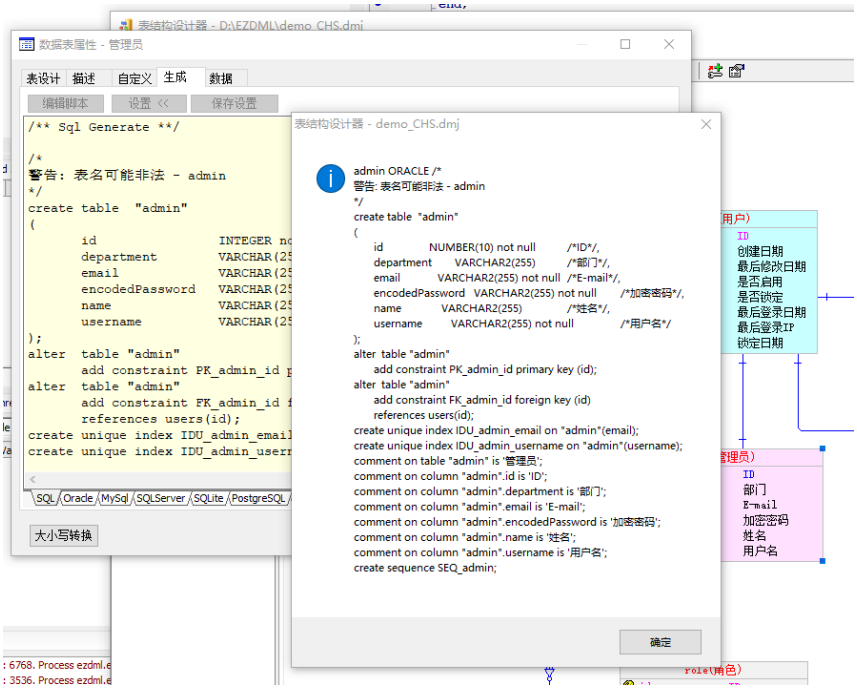
```
//Generate upgrade SQL for an exists Table in a database. defRes is the default result.
//生成数据库的更新SQL，传入新旧表对象、默认生成的SQL 结果、数据库类型、选项，返回自定义结果SQL
function OnEzdmlGenDbSqlEvent(designTb, dbTable: TCtMetaTable; defRes, dbType, options: String):
string;
begin
  Result := defRes;
end;

//Generate SQL for a single Field. defRes is the default result.
//生成单个字段的类型（varchar(255) nullable），传入表对象、字段对象、默认生成的结果、数据库类型、选项，返回自定义结果
function OnEzdmlGenFieldTypeDescEvent(tb: TCtMetaTable; fd: TCtMetaField; defRes, dbType,
options: String): string;
begin
  Result := defRes;
end;

//Generate upgrade SQL for an exists database-Field of a Table. defRes is the default result.
//生成增删改单个字段的SQL（alter table add xxx），传入要执行的操作(alter/add/drop)、表对象、新旧字段对象、默认生成的结果、数据库类型、选项，返回自定义结果
function OnEzdmlGenAlterFieldEvent(action: String; tb: TCtMetaTable; designField, dbField:
TCtMetaField; defRes, dbType, options: String): string;
begin
  Result := defRes;
end;

//Reserved custom events
//自定义命令事件
function OnEzdmlCmdEvent(cmd, param1, param2: String; parobj1, parobj2: TObject): string;
begin
  //WriteLog('OnEzdmlCmdEvent('+cmd+', '+param1+', '+param2+', obj1, obj2)');
  Result := '';
end;

begin
end.
```

全局事件脚本文件为 GlobalScript.ps_，位于主程序目录下。你可以在这接管生成 SQL、弹出窗口等事件。该脚本仅支持 PASCAL，不支持 JAVASCRIPT。

比如我们改一下第一个函数，加一行代码，把一些参数 SHOW 出来：

```
//Generate SQL for a single Table. defRes is the default result.
//生成单个表的SQL，传入表对象、是否生成创建表SQL、是否生成创建约束SQL、默认生成的SQL 结果、数据库类型、选项，返回自定义结果SQL
function OnEzdmlGenTbSqlEvent(tb: TCtMetaTable; bCreateTb, bCreateConstrains: Boolean; defRes,
dbType, options: String): string;
begin
  alert(tb.name+' '+dbType+' '+defRes);
  Result := defRes;
end;
```

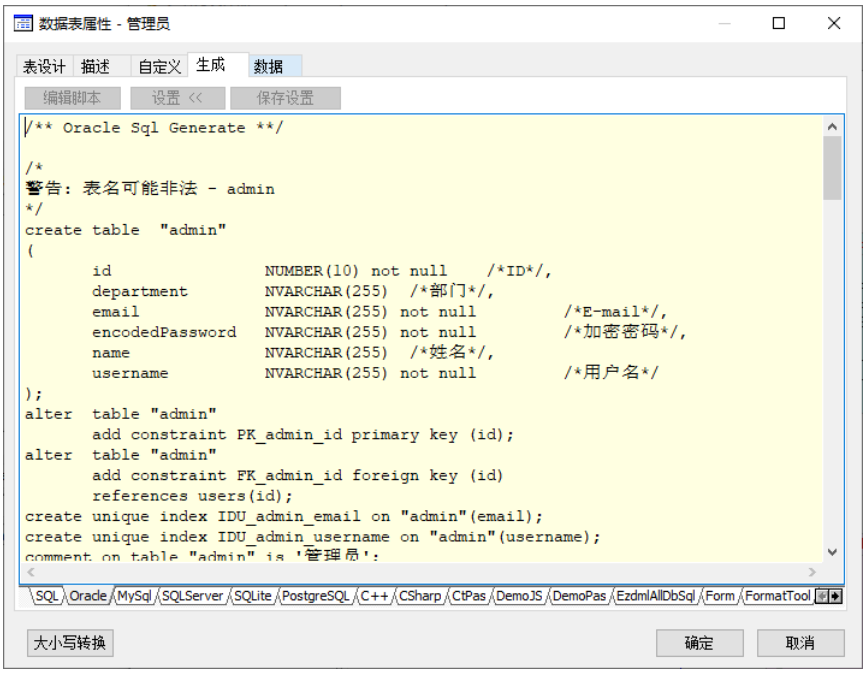
保存退出，打开 admin 表，并在生成切换几个数据库，这过程中你会不断地看到这行代码执行的结果：



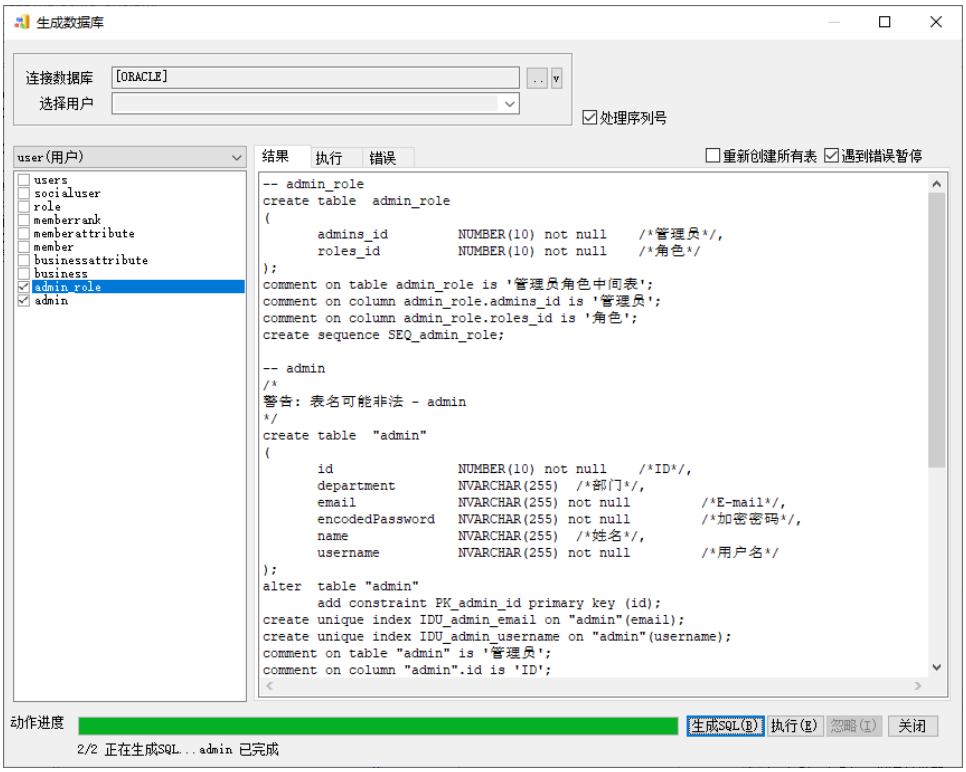
假设我们对生成的结果中的 VARCHAR2 不满意，要替换为 NVARCHAR，我们可以再次编辑全局事件脚本，这样改一下：

```
function OnEzdmlGenTbSqlEvent(tb: TCtMetaTable; bCreateTb, bCreateConstrains: Boolean; defRes, dbType, options: String): string;
begin
   Result := defRes;
Result := StringReplace(Result, 'VARCHAR2', 'NVARCHAR', [rfReplaceAll]);
end;
```

保存退出，再次打开 admin 表，切到 ORACLE 生成页，结果如下：



生成数据库的 SQL 也是一样发生了变化：



另外，OnEzdmlCmdEvent 事件可拦截以下事件(cmd，param1)：

- MAINFORM，CREATE：主窗口创建时触发
- MENU_ACTION，Tools_CustomMenu 等：点击主菜单自定义工具等操作时触发
- FIELD_PROP_DIALOG，SHOW/HIDE：字段属性窗口显示、隐藏时触发
- TABLE_PROP_DIALOG，SHOW/HIDE：表属性窗口显示、隐藏时触发
- NEW_MODEL：创建模型图时触发
- NEW_TABLE：创建表时触发
- MAINFORM，CLOSE：主窗口关闭时触发
- CHECK_CUSTOM_DICT_NAME：执行“大小写转换|应用 MyDict.txt 检查”时触发，可在这里做额外的处理工作
- EXTRACT_LOGIC_NAME_FROM_MEMO：执行“大小写转换|注释转为逻辑名”时，如果逻辑名为空，需要从注释中获取，此时将触发此事件。你可以接管事件，按自己的需要生成逻辑名

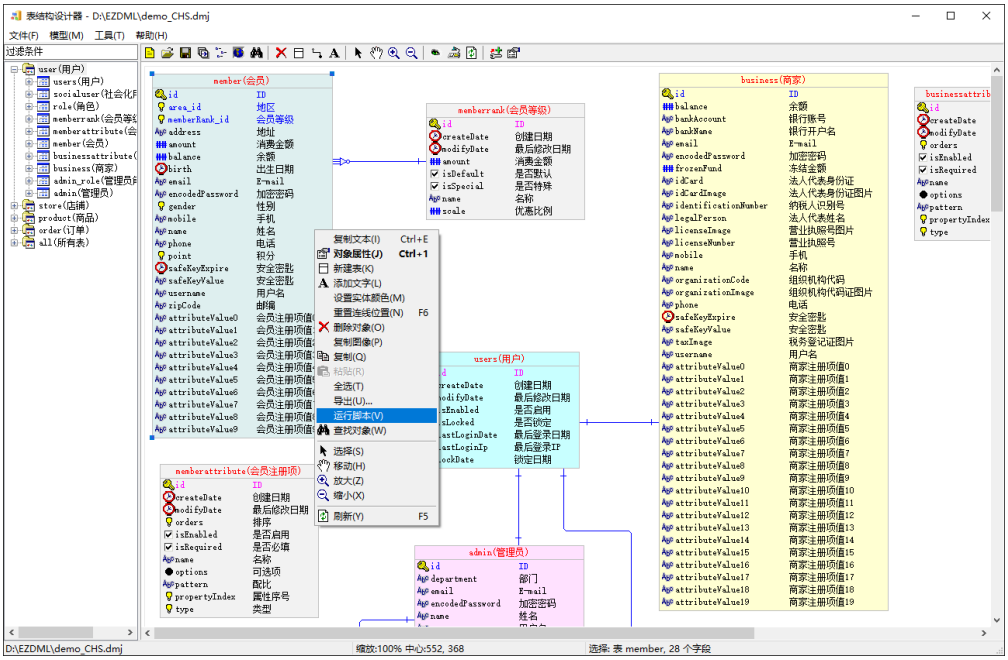
用处不大，就不一一说明了。

四、 应用场景

列一下脚本使用的几个场景。

1. 右键运行脚本

就是直接右键执行脚本了，用于临时想执行点脚本干点啥：



2. 带参数启动 EXE 运行脚本

EZDML 支持传一到两个文件名作为启动参数：

- 如果传一个参数，则参数可以是一个 DML 模型文件名（dmx、dmh 或 dmj）由程序直接打开，也可以是一个脚本文件名（js 或 pas）来加载并执行
- 如果传两个参数，则参数 1 应为一个 DML 模型文件名（dmx、dmh 或 dmj），参数二为脚本文件名（js 或 pas），程序将在打开参数一模型文件后再加载执行参数二的脚本
- 如果想执行完脚本就退出，不显示界面，可以这样：
  - 只传一个脚本文件名参数（如果需要加载文件，可以在脚本里执行 allModels.loadFromFile(xx)）
  - 在脚本里直接调_abortOut()中止
  - 这时主窗口不显示就会直接退出。

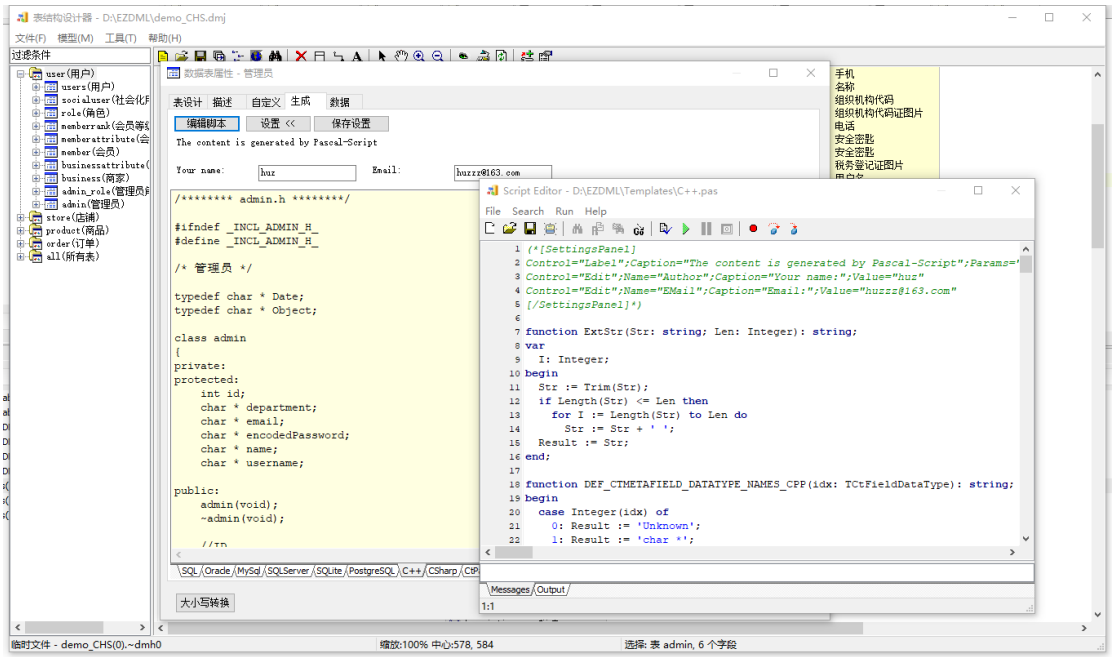
3. 表属性中的“生成”

表属性的生成页中，除了前面几个数据库 SQL 是原生代码生成的，后面的是由脚本模板文件生成，可自行编辑脚本生成自己想要的结果。

表属性中“生成”的脚本模板位于 Templates 目录下，此目录下的 JS 和 PAS（如果有同名的 JS 和 PAS，则 JS 优先）会直接在表属性的“生成”页中显示。

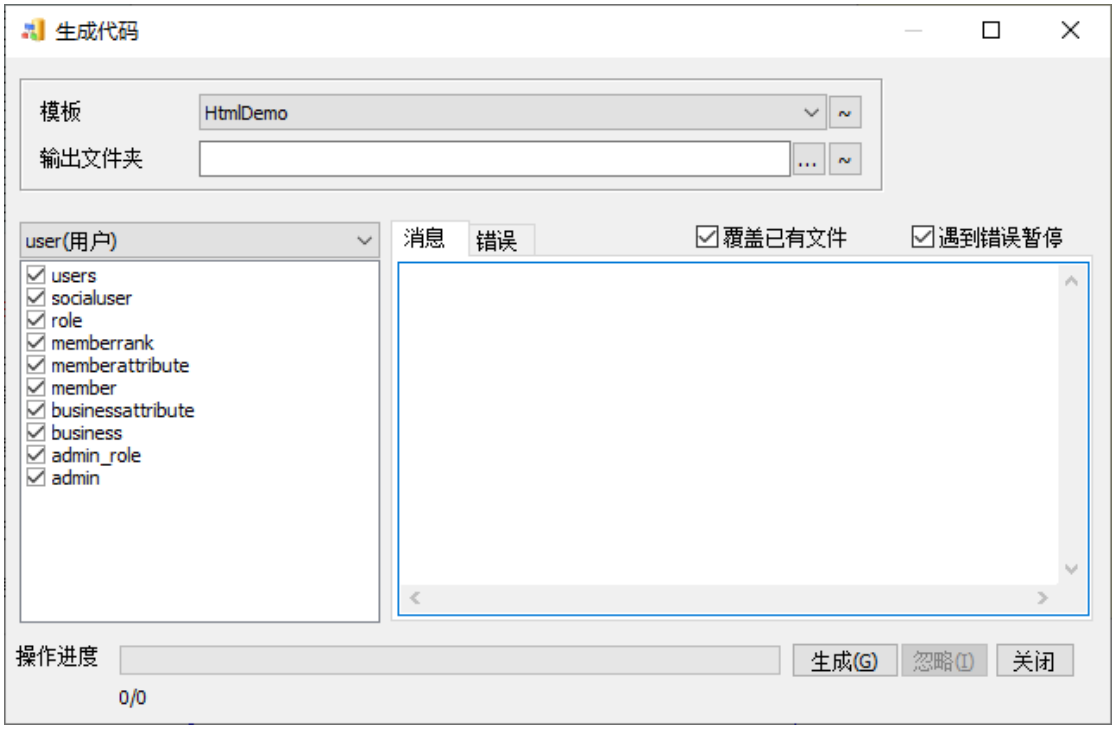
表属性中“生成”的脚本模板中可以加载参数面板，通过参数面板来获取用户输入的内

容。参见《参数面板》一节。



4. 生成代码

执行菜单命令“模型|生成代码”，会使用 Templates 目录下的文件作为模板批量生成代码文件。



其中输出文件夹如果不填则默认使用 EZDML\temp 目录。

如果有子目录，则子目录将作为一个项目来生成。对子目录生成代码时，程序会读取 _dml_config.INI 这个生成配置文件，此文件告诉程序要如何执行生成操作，示例如下：

[start.html]——对于 start.html 这个文件的配置

run_as_script=pas——该文件其实是一个 PAS 脚本文件，要作为 PAS 脚本加载执行，生成最终结果。可选值：pas js
encoding=ansi——该文件输出时转为 ANSI 编码，默认 utf8，可选值：空，utf8，ansi
20200310：注意仅控制输出文件编码，输入脚本文件必须为 utf8 编码

[index_top.html]——对于 index_top.html 这个文件的配置
rename=#curmodel_name#_index_top.html——该文件生成时要重命名，前面加当前模型名
run_as_script=js——该文件其实是一个 JavaScript 脚本文件,要作为 JS 脚本加载执行，生成最终结果
encoding=ansi——该文件输出时转为 ANSI 编码，默认 utf8，可选值：空，utf8，ansi

[table_ui] ——对于 table_ui 这个目录的配置
rename=ui_#curtable_name#——该目录生成时要重命名
loop_each_table=1——该目录下的文件、脚本需要对每一个表都执行一遍生成过程，该子目录下应该也要有一个_dml_config.INI 来控制生成过程

[dml_settings]
auto_open_on_finished=start.html——生成结束时打开这个文件

每一次脚本执行前，系统会对以下全局变量赋值：

- GCODE_SRC_FILENAME——源文件名（如 D:\EZDML\templates\test\ index_top.html）
- GCODE_DST_FILETEMPLATE——目标文件名配置（如#curmodel_name#_index_top.html）
- GCODE_DST_FILENAME——目标文件名（如 D:\testout\users_index_top.html）
- GCODE_ENCODING——目标文件字符编码（默认 UTF-8）

脚本中可以用 GetGParamValue、SetGParamValue 方法来获取、修改这些变量。

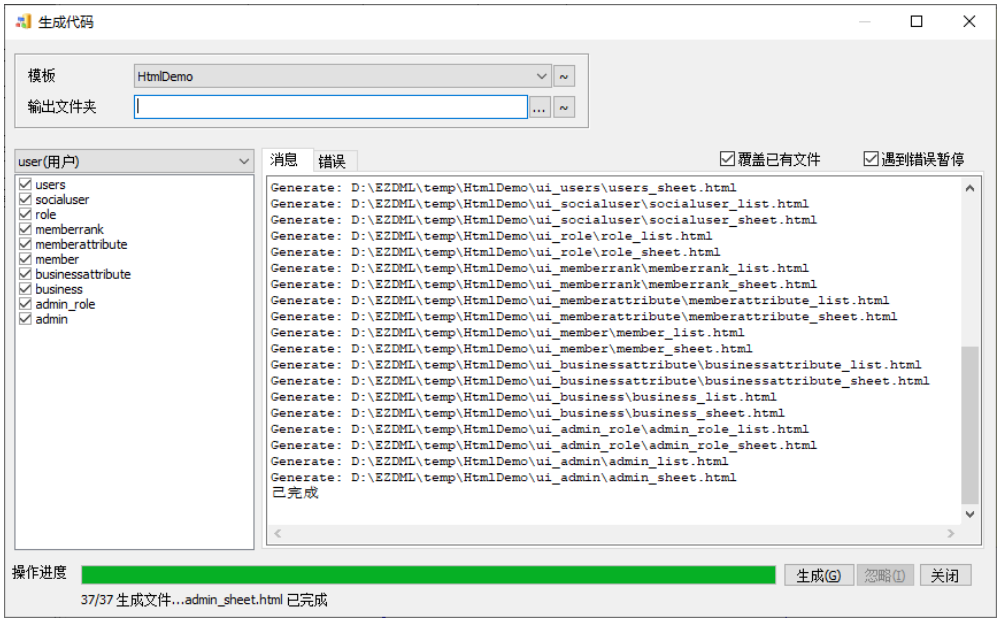
脚本执行后，默认会将 CurOut 内容以指定的编码保存到目标文件，你可以在脚本中修改全局变量 GCODE_ENCODING 和 GCODE_DST_FILENAME 来改变编码和目标文件名。

如果你自己用脚本保存了文件，或者不希望系统帮你保存文件，可以在脚本中用 SetGParamValue 将全局变量 GCODE_DST_FILENAME 的值清空，或者直接调用 AbortOut 中止脚本执行（注：AbortOut 仅中止当前脚本、当前表）。

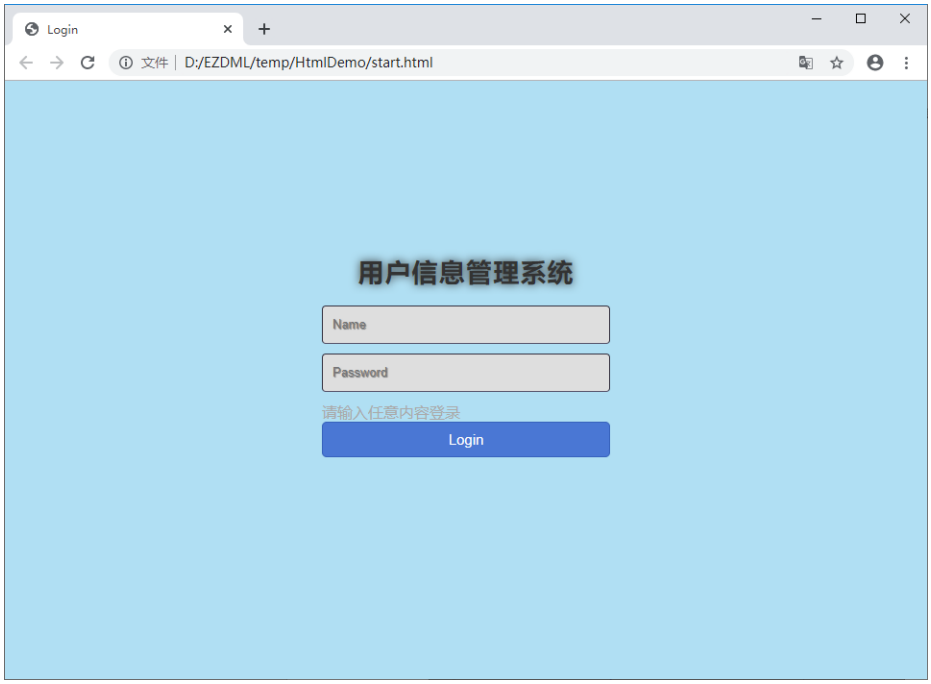
关于文件编码特别说明一下，如不指定 encoding，会认为文件编码是 UTF8，生成结果文件也是 UTF8。

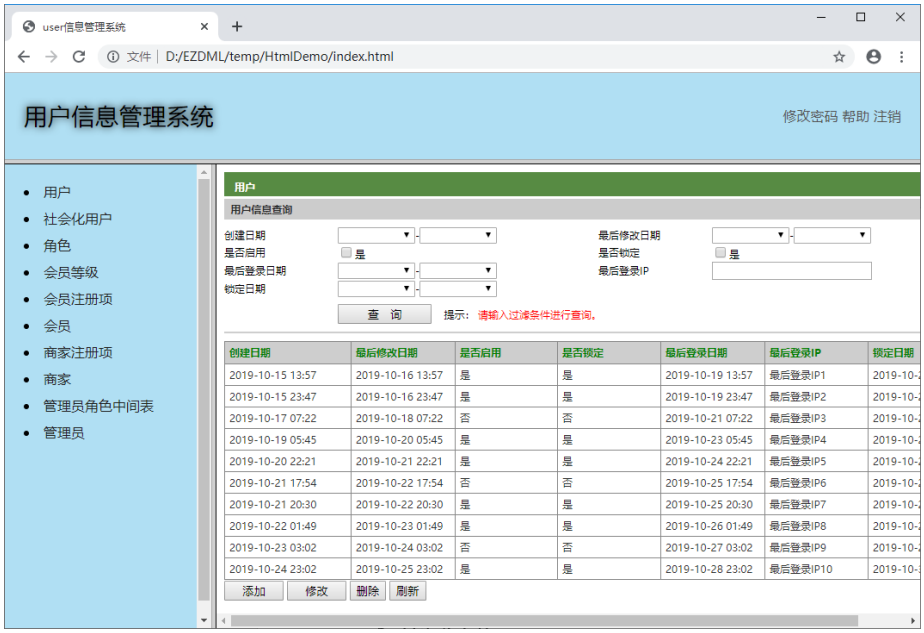
需要说明的是，32 位 EZDML 的 PAS 脚本引擎要求脚本内容是 ANSI 或 GBK 编码，JS 引擎要求是 UTF8 编码，因此，不管源文件是哪种编码，执行前都有可能会转换为需要的编码，执行后可能会再根据需要转换一次。

示例项目生成的结果：



生成完成自动按指示设置打开目标启动文件：

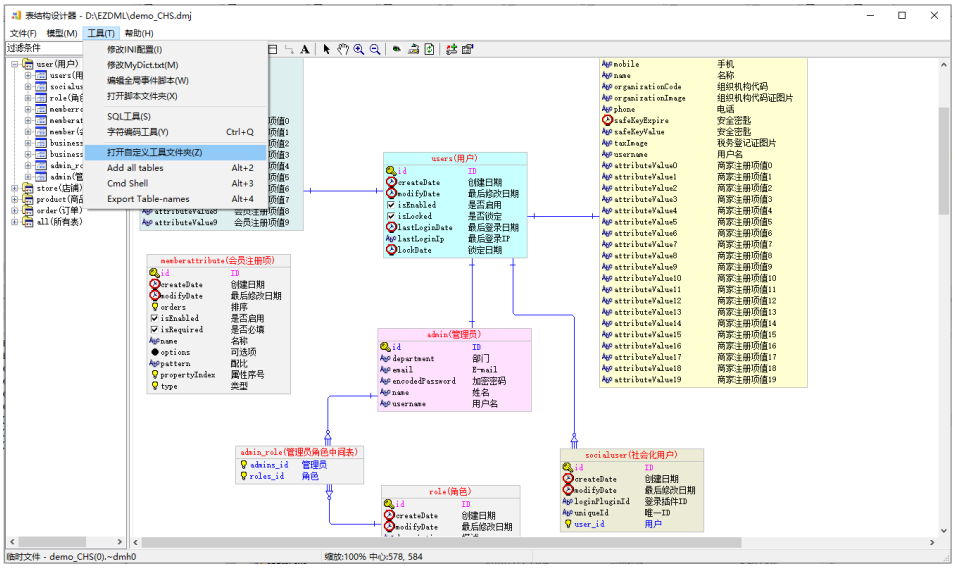


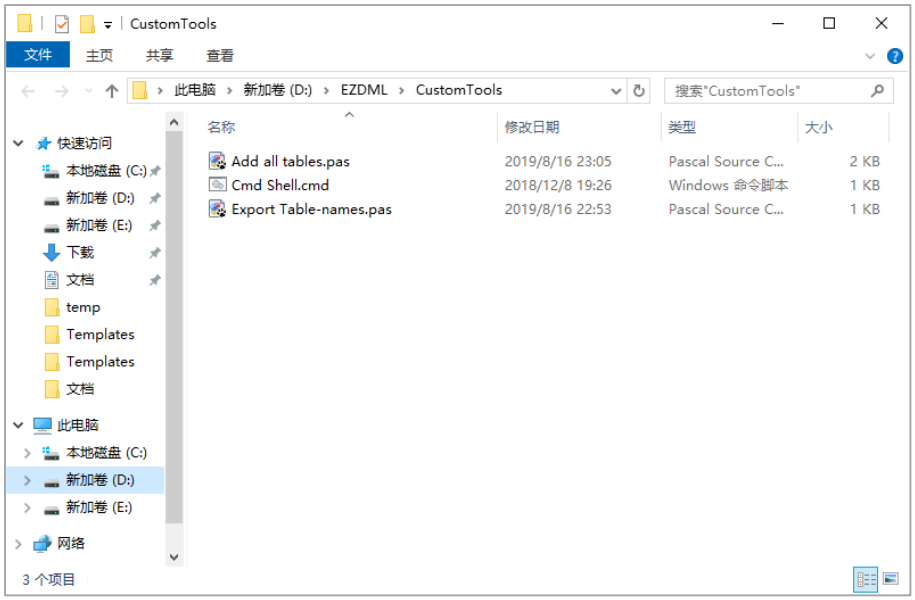


嗯，别怕，只是生成了个假的 DEMO。要是真能直接生成个系统就不好了，大家都要失业了。话说，现在 AI 这么发达，失业恐怕是迟早的事，唉！

5. 自定义工具

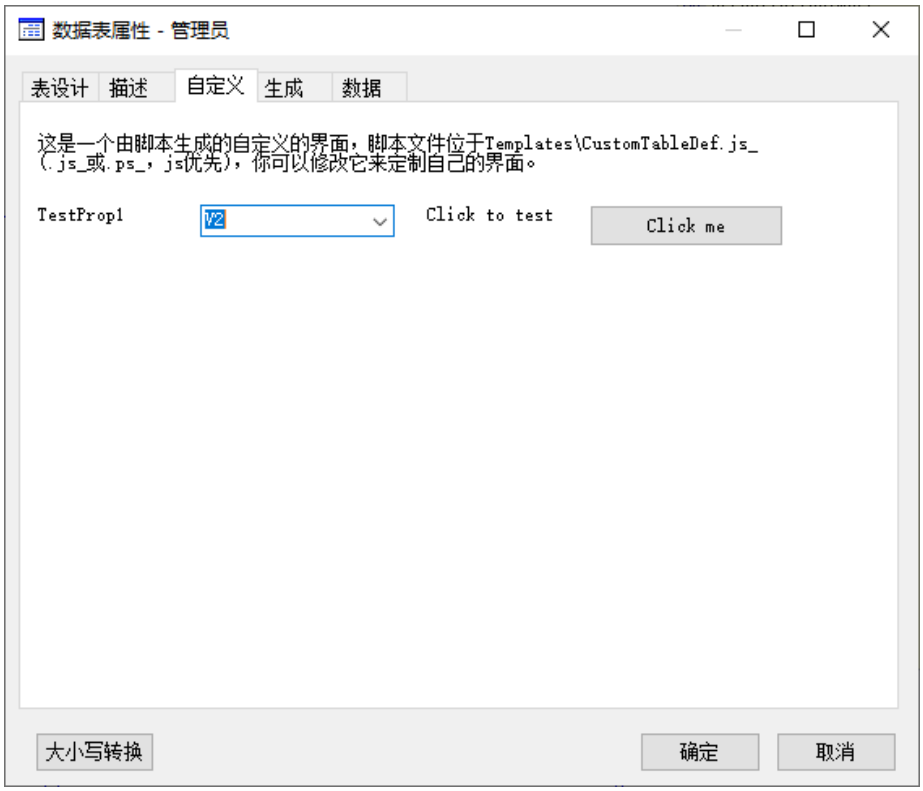
工具下面最后一组菜单，是根据 EZDML\CustomTools 目录下的文件自动生成的，此文件夹下的每个文件都会生成一个工具菜单。点击时会直接打开相应的文件；如果是 PAS 或 JS 文件，则尝试加载为脚本执行。



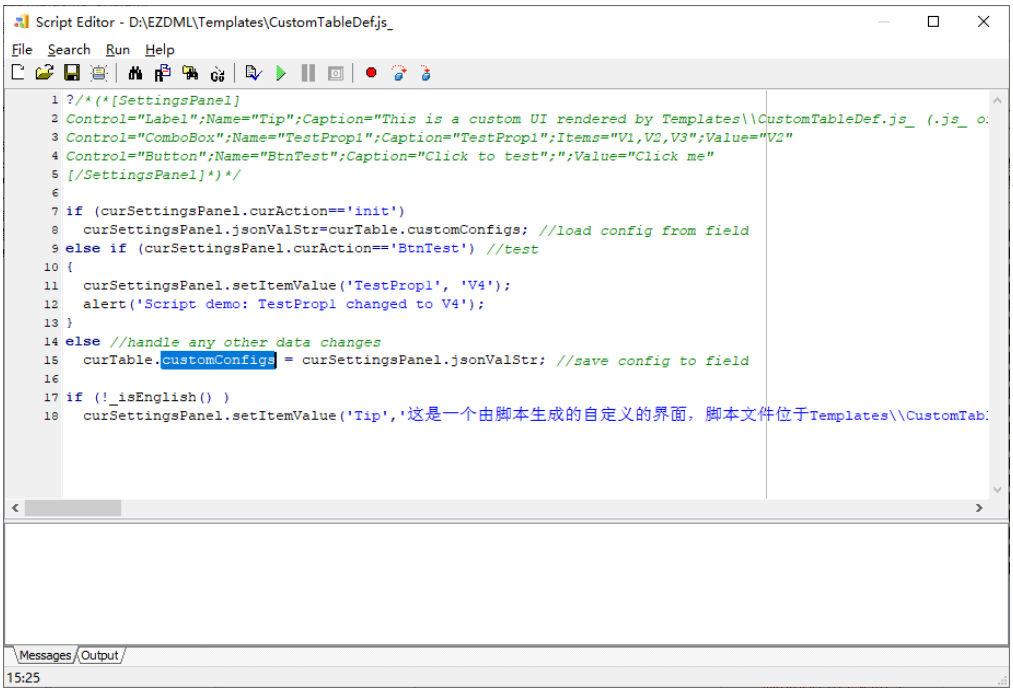


6. 自定义表属性 UI

自定义表属性 UI 由脚本文件 CustomTableDef.js_/.ps_控制，位于 Templates 目录下。在 INI 中设置 EnableCustomPropUI=1 后，表属性会多一页自定义，自定义的界面由此脚本生成。



将对应的脚本拖到调试编辑窗口里打开：

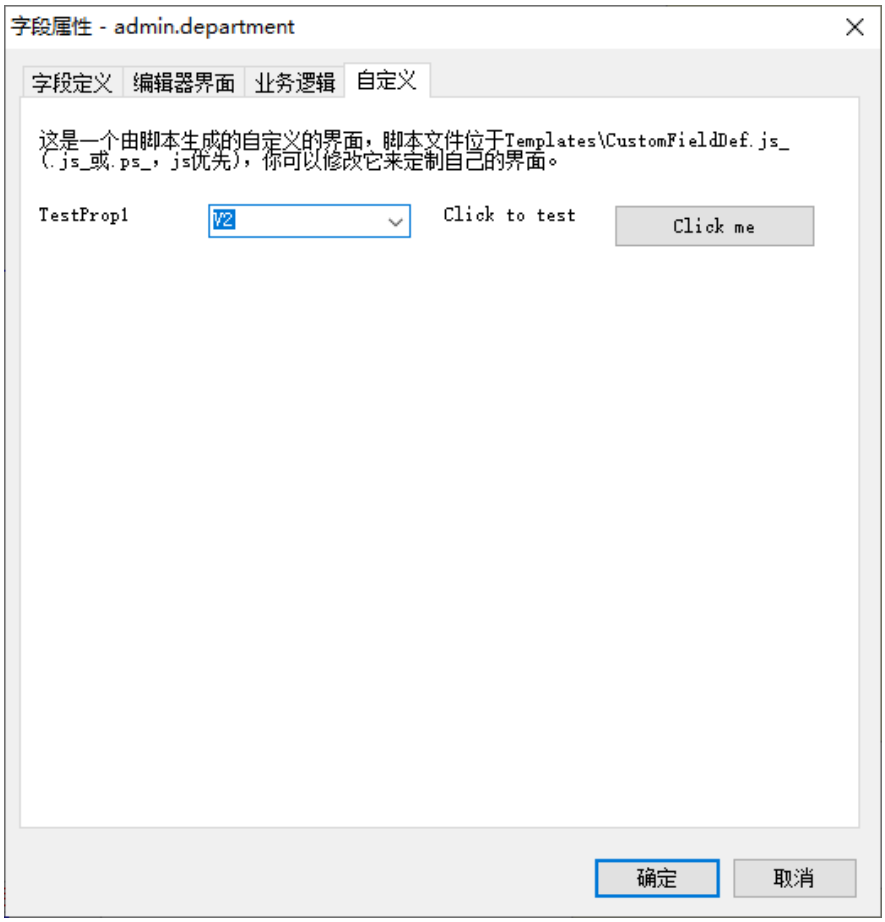


可以看到，这里是直接用 customConfigs 来加载和保存 JSON 结果了。

另外可注意到，第一次加载执行脚本时，参数面板的 CurAction 的值为 init，表示初始化。后续任何值被编辑修改发生改变（或点击按钮）时，参数面板的 CurAction 的值会重新写入，并重新执行整个脚本。

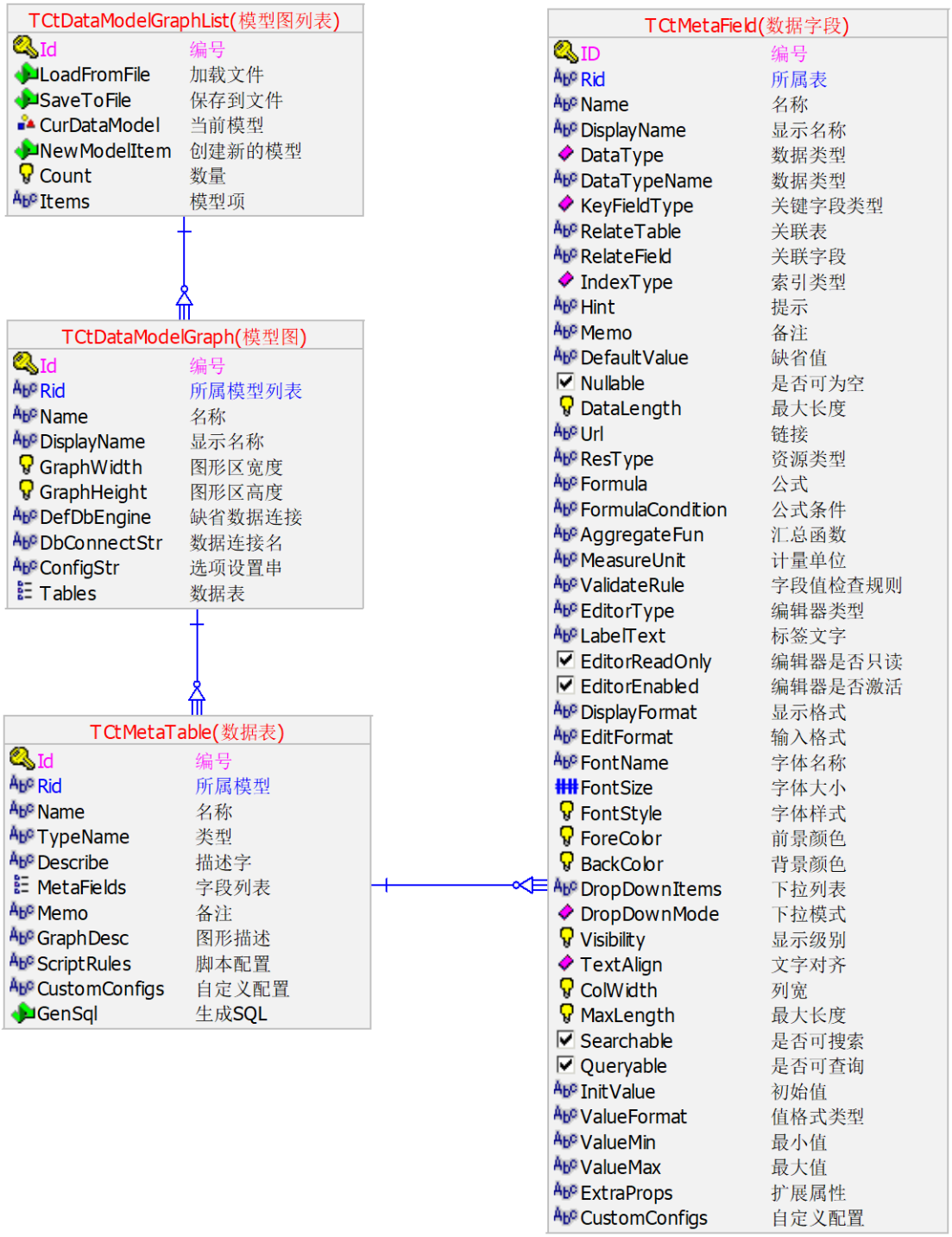
7. 自定义字段属性 UI

跟自定义表属性 UI 差不多是一样的，只是对应的文件名为 CustomFieldDef.js_/.ps_

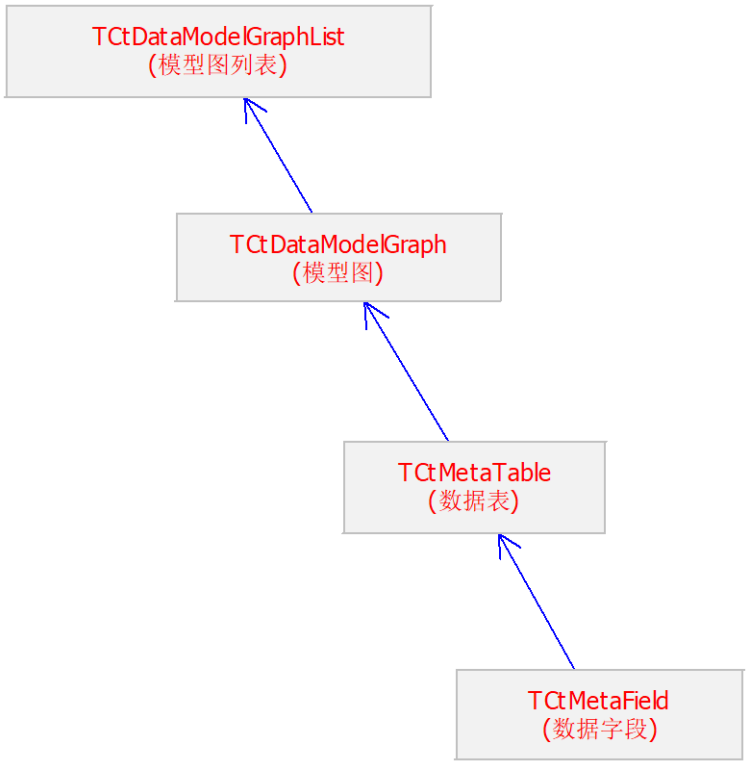


五、 对象关系图

先看看这张对象关系图（本来想用 visio 的，后来想想好像 EZDML 也能做）：



太大也不容易看，缩小下就清楚了：



类名都有 TCtMeta 前缀，因为：

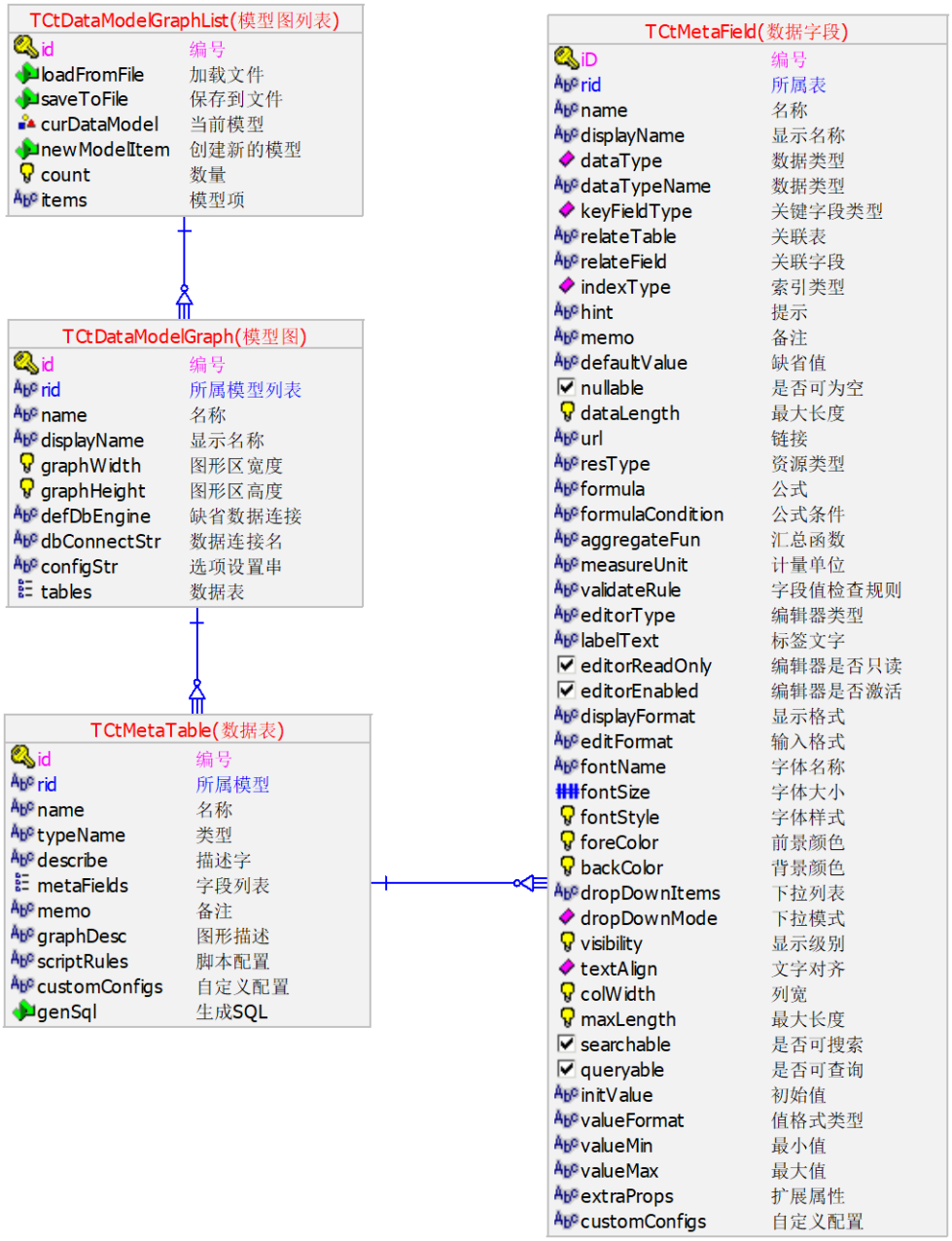
a) Dephi 的类默认都是以 T 开头，我猜是 Type 的意思？

b) CT 是我自己做的一套基类的简称

c) Meta 是元数据的意思

上面首字母是大写的，用 JS 的可能不习惯，可写段脚本改一下：

```
for(varj=0;j<curModel.tables.count;j++)
{
  vartb=curModel.tables.getItem(j);

  for(vark=0;k<tb.metaFields.count;k++)
  {
    varfd=tb.metaFields.getItem(k);
    vars=fd.name;
    fd.name=s.substring(0,1).toLowerCase()+s.substr(1);//首字母小写
  }
}
```



模型图、表、字段是 EZDML 的核心对象，简单说明如下：

- 一个 DML 文件就对应了一个模型图列表（TCtDataModelGraphList）对象，通过 allModels（这是 JS 语法，JS 里需要首字母小写，PAS 则无所谓）这个全局变量获取，可以直接调它的 loadFromFile(fileName)和 saveToFile(fileName)方法存取文件，例如这样：allModels.saveToFile('d:\\ab.dmj');。
- 通过模型图列表对象（TCtDataModelGraphList）的 getItem(i)（PAS 里是 Items[i]），可以遍历获取模型图对象（TCtDataModelGraph）；也可以直接用 curModel 这个全局变量获取当前模型图。
- 模型图对象（TCtDataModelGraph）的 tables 属性是一个列表对象，可通过它的 getItem(i)（PAS 里是 Items[i]）遍历获取数据表对象（TCtMetaTable）；也可以直接用 curTable 这个全局变量获取当前表（注意：当前表可能为空）。
- 表对象（TCtMetaTable）的 metaFields 属性是一个列表对象，可通过它的 getItem(i)

（PAS 里是 Items[i]）遍历获取数据字段对象（TCtMetaField）；也可以直接用 curField 这个全局变量获取当前字段（注意：当前字段可能为空）。

六、 核心对象

EZDML 的核心对象就是三个：

a) 模型图（TCtDataModelGraph）
b) 表（TCtMetaTable）
c) 字段（TCtMetaField）。

嗯，刚才说三个可能错了，应该是六个，因为它们有对应的集合列表对象：

a) 模型图列表（TCtDataModelGraphList）
b) 表集合（叫表列表似乎有点怪，就叫表集合吧）（TCtMetaTableList）
c) 字段列表（TCtMetaFieldList）

它们之间有上下层级包含关系：

- allModels：模型图列表（TCtDataModelGraphList）
- TCtDataModelGraphList.getItem(i)：模型图（TCtDataModelGraph）
  - TCtDataModelGraph. Tables：表集合（TCtMetaTableList）
  - TCtMetaTableList.getItem(i)：表（TCtMetaTable）
    - ✓ TCtMetaTable. MetaFields：字段列表（TCtMetaFieldList）
    - ✓ TCtMetaFieldList.getItem(i)：字段（TCtMetaField）

1. 公共属性

EZDML 的三个核心对象——模型图（TCtDataModelGraph）、表（TCtMetaTable）、字段（TCtMetaField）——它们都是从一个公共对象 TCtMetaObject 继承，有公共的属性方法，TCtMetaObject 定义如下（PASCAL 的，懒得翻译了，用 JS 的朋友记得首字母小写就好）：

```
//CTMeta 对象基类，包含基本属性和串行化接口
TCtMetaObject= class(TObject)
public
procedure DoCustomEvent(EvtParamA, EvtParamB: string); virtual;

//排序
procedure SortByOrderNo; virtual;
procedure SortByName; virtual;

//状态保存恢复接口
procedure Reset; virtual;
procedure AssignFrom(ACtObj: TCtObject); virtual;
procedure LoadFromFile(fn: string); virtual;
procedure SaveToFile(fn: string); virtual;
```

```
//基本属性：ID PID RID  名称别名类型注释创建时间和创建人最后修改时间和修改人数据级别
property ID: Integer read FID write FID;
property CtGuid: stringread FCtGuid write FCtGuid;
property PID: Integer read FPID write FPID;
property RID: Integer read FRID write FRID;
property Name: stringread FName write SetName;
property Caption: stringread FCaption write SetCaption;
property TypeName: stringread FTypeName write FTypeName;
property Memo: stringread FMemo write FMemo;
property DisplayText: stringread GetDisplayText;

property CreateDate: TDateTime read FCreateDate write FCreateDate;
property Creator: Integer read FCreator write FCreator;
property CreatorName: stringread FCreatorName write FCreatorName;
property ModifyDate: TDateTime read FModifyDate write FModifyDate;
property Modifier: Integer read FModifier write FModifier;
property ModifierName: stringread FModifierName write FModifierName;

//历史编号不同的人修改或长时间修改（如两周）后之后自动创建版本历史
property HistoryID: Integer read FHistoryID write FHistoryID;
//版本号每次修改自动加一，便于同步
property VersionNo: Integer read FVersionNo write FVersionNo;
//锁定信息当前锁定信息，包含锁定人、锁定时间、超时限制
property LockStamp: stringread FLockStamp write FLockStamp;

property DataLevel: _ENUM_TctDataLevel read FDataLevel write FDataLevel;
//排序号
property OrderNo: Double read FOrderNo write FOrderNo;

property SubItems: TCtObjectList read GetSubItem write SetSubItem;
property HasSubItems: Boolean read GetHasSubItems;
//父列表
property ParentList: TCtObjectList read FParentList write FParentList;
//所有对象全局列表
property GlobeList: TCtGlobList read FGlobeList write SetGlobeList;
//用户自定义对象
property UserObjectList: TStrings read GetUserObjectList;
property UserObjectCount: Integer read GetUserObjectCount;
property UserObject[Index: string]: TObject read GetUserObject write SetUserObject;
//参数转为TSTRINGS
property ParamList: TStrings read GetParamList;
property Param[Name: string]: stringread GetParam write SetParam;
//TCtMetaObject
property MetaModified: Boolean read GetMetaModified write SetMetaModified;
property IsSelected: Boolean read FIsSelected write FIsSelected; //是否被选中
end;
```

是的，每一个表、字段对象都有这么多属性、方法，具体每个属性方法的作用嘛，懒得写了，估计大家这么聪明的差不多的都能猜得到。

2. 列表

EZDML 的三个核心对象对应的列表对象——模型图列表（TCtDataModelGraphList）、表集合（TCtMetaTableList）、字段列表（TCtMetaFieldList）——它们也是从一个公共对象

对象继承而来，这个公共对象就是 TCtObjectList 了，其定义如下：

```
TCtObjectList = class(TList)
public
procedure CheckMaxSeq; virtual;

//状态保存恢复接口
procedure AssignFrom(ACtObj: TCtObjectList); virtual;

///新建对象 ××××××××××××××注意：子类必须仍然保留使用此方法创建对象××××××××××××
function NewObj: TCtObject; virtual;
//子项释放通知
procedure NotifyChildFree(AItem: TCtObject); virtual;
//创建全局列表，这样一个列表中就记录了同一棵树下的所有节点
function CreateGlobeList: TCtGlobList; virtual;
//获取子项
function ItemByID(AID: Integer): TCtObject; virtual;
function ItemByName(AName: string; bCaseSensive: Boolean = False): TCtObject; virtual;
function NameOfID(AID: Integer): string; virtual;

//删除无效节点
procedure Pack; virtual;
//仅删除（不FREE）
procedure MereRemove(AItem: TCtObject); virtual;
//排序
procedure SortByOrderNo; virtual;
procedure SortByName; virtual;
//保存当前顺序
procedure SaveCurrentOrder; virtual;

property Items[Index: Integer]: TCtObject read GetCtItem write SetCtItem; default;
//所有对象全局列表
//这样一个全局列表中记录了同一棵树下的所有节点
property GlobeList: TList read FGlobeList write FGlobeList;
end;
```

这个 TCtObjectList 又是从 TList 列表继承的，TList 定义如下：

```
TList = class(TObject)
public
function Add(Item: TObject): Integer;
procedure Clear; virtual;
procedure Delete(Index: Integer);
procedure Exchange(Index1, Index2: Integer);
function IndexOf(Item: TObject): Integer;
procedure Insert(Index: Integer; Item: TObject);
procedure Move(CurIndex, NewIndex: Integer);
function Remove(Item: TObject): Integer;
procedure Pack;
property Capacity: Integer read FCapacity write SetCapacity;
property Count: Integer read FCount write SetCount;
property Items[Index: Integer]: TObject read GetItem write SetItem; default;
end;
```

3. 模型图

定义说明都是很枯燥的，能坚持看到这里不容易。

模型图本质上就是一个容器，把表装在里面。可通过 allModels 这个列表遍历所有模型图，也可以直接用 curModel 这个全局变量获取当前模型图。

模型图对象定义如下（嗯，前面公共属性方法都列得差不多了，这里就没多少内容了）：

```
TCtDataModelGraph = class(TCtMetaObject)
public
//显示名称
property DisplayName: stringread GetDisplayName;
//图形区宽度
property GraphWidth: Integer read FGraphWidth write FGraphWidth;
//图形区高度
property GraphHeight: Integer read FGraphHeight write FGraphHeight;
//缺省数据连接
property DefDbEngine: stringread FDefDbEngine write FDefDbEngine;
//数据连接名
property DbConnectStr: stringread FDbConnectStr write FDbConnectStr;
//选项设置串
property ConfigStr: stringread FConfigStr write FConfigStr;
//数据表
property Tables: TCtMetaTableList read FTables write FTables;

property OwnerList: TCtDataModelGraphList read FOwnerList;
end;
```

对应的模型图列表对象定义如下：

```
TCtDataModelGraphList = class(TCtObjectList)
public
procedure LoadFromFile(fn: string);
procedure SaveToFile(fn: string);
function TableCount: Integer;
function NewModelItem: TCtDataModelGraph;
property CurDataModel: TCtDataModelGraph read GetCurDataModel write FCurDataModel;
end;
```

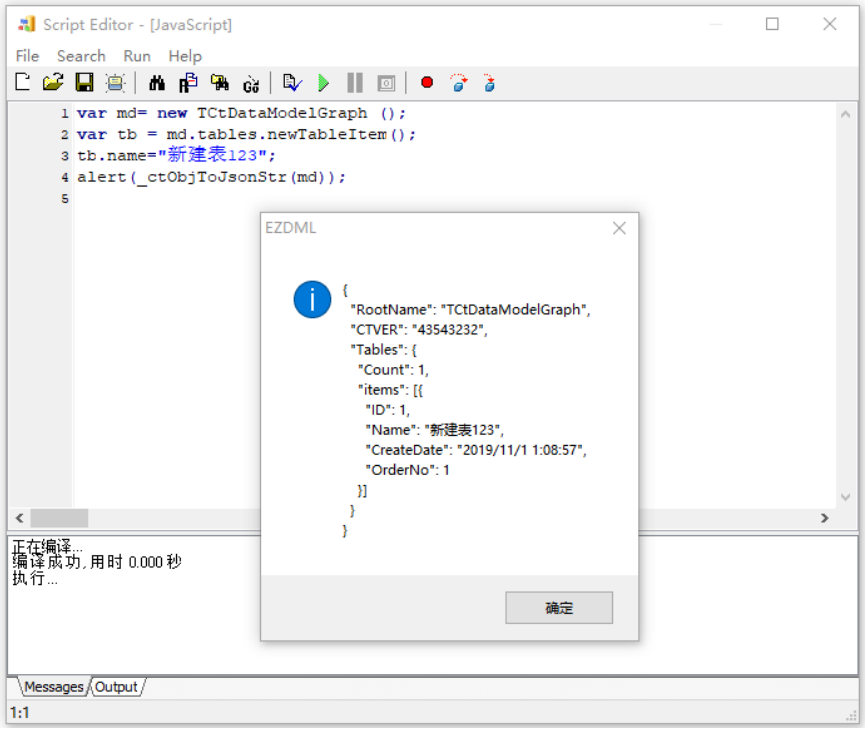
如果想在当前模型中新建一个表，可以这样写（JS）：

```
var tb = curModel.tables.newTableItem();
tb.name="新建表 123";
```

当然，你也可以凭空在内存中创建一个临时的模型图：

```
var md= new TCtDataModelGraph ();
var tb = md.tables.newTableItem();
tb.name="新建表 123";
alert(_ctObjToJsonStr(md));
```

没办法看到图，不过看看 JSON 结果还是可以的（请注意 CtObjToJsonStr 是 PAS 实现的，出来的结果首字母是大写的）：



4. 表

表也是一个容器，把字段装在里面了。

在选择了表的时候，可以直接用 curTable（PAS 里其实是 CurTable，不过 PAS 里大小写不敏感）这个全局变量获取当前表（如果没有选中有的话取出来是空的）。

表的类定义如下：

```
TCtMetaTable = class(TCtMetaObject)
public
//状态保存恢复接口
procedure SyncPropFrom(ACtTable: TCtMetaTable); virtual;

function GetTableComments: string; virtual;
function GetPrimaryKeyNames(AQuotDbType: string = ''): string;
function IsText: Boolean;
function IsTable: Boolean;
function IsSqlText: Boolean;

function GenSqlEx(bCreatTb: Boolean; bFK: Boolean; dbType: string = '';
    dbEngine: TCtMetaDatabase = nil): string; virtual;
function GenSql(dbType: string = ''): string; virtual;
function GenSqlWithoutFK: string; virtual;
function GenFKSql: string; virtual;
function GenDqlDmlSql(dbType: string = ''): string; virtual;
function GenSelectSql(maxRowCount: Integer; dbType: string = ''): string; virtual;

function GetCustomConfigValue(AName: string): string;
procedure SetCustomConfigValue(AName, AValue: string);

property OwnerList: TCtMetaTableList read FOwnerList;
```

```
//编号
//property ID;（祖先类已定义）
//名称
//property Name;
//类型
//TYPENAME 为空或TABLE 时是表，为TEXT 时是纯文字
//property TypeName;
//备注
//property Memo;
//图形描述
property GraphDesc: stringread FGraphDesc write FGraphDesc;

//元字段列表
property MetaFields: TCTMetaFieldList read GetMetaFields;
//主键字段名
property KeyFieldName: stringread GetKeyFieldName;

//描述字
property Describe: stringread GetDescribe write SetDescribe;
//自定义配置
property CustomConfigs: stringread FCustomConfigs write FCustomConfigs;
//脚本配置
property ScriptRules: stringread FScriptRules write FScriptRules;
end;
```

对应的表集合（列表）对象定义如下：

```
TCtMetaTableList = class(TCtMetaObjectList)
public
function NewTableItem: TCtMetaTable;
property OwnerModel: TCtDataModelGraph read FOwnerModel;
end;
```

属性很少是不是，没办法，就一个列表来的，祖先传下来的东西已经够用了。

5. 字段

在选择了字段的时候，可以直接用 curField 这个全局变量获取当前表（如果没有选中有的话取出来是空的）。

字段的类定义如下：

```
TCtMetaField = class(TCtMetaObject)
public
//状态保存恢复接口
function GetLogicDataTypeName: string;
function GetFieldTypeDesc(bPhy: Boolean = False; dbType: string = ''): string;
function GetFieldComments: string;
function GetFieldDefaultValDesc(dbType: string = ''): string;

function GetConstraintStr: string;
procedure SetConstraintStr(Value: string);
```

```
function GetConstraintStrEx(bWithKeys, bWithRelate: Boolean): string;
procedure SetConstraintStrEx(Value: string; bForce: Boolean);

function GetCustomConfigValue(AName: string): string;
procedure SetCustomConfigValue(AName, AValue: string);

property OwnerList: TCtMetaFieldList read FOwnerList;
property OwnerTable: TCtMetaTable read GetOwnerTable;

//编号
//property ID;
//表编号
//property RID;
//名称
//property Name;
//显示名称
property DisplayName: stringread FDisplayName write FDisplayName;
//数据类型
property DataType: _ENUM_TCtFieldDataType read FDataType write FDataType;
//数据类型
property DataTypeName: stringread FDataTypeName write FDataTypeName;
//关键字段类型_IdPidRid...
property KeyFieldType: _ENUM_TCtKeyFieldType read FKeyFieldType write FKeyFieldType;
//关联表
property RelateTable: stringread FRelateTable write FRelateTable;
//关联字段
property RelateField: stringread FRelateField write FRelateField;
//索引类型_0 无 1 唯一 2 普通
property IndexType: _ENUM_TCtFieldIndexType read FIndexType write FIndexType;
//提示
property Hint: stringread FHint write FHint;
//备注
//property Memo;
//缺省值
property DefaultValue: stringread FDefaultValue write SetDefaultValue;
//是否可为空
property Nullable: Boolean read GetNullable write FNullable;
//最大长度
property DataLength: Integer read FDataLength write FDataLength;
//精度
property DataScale: Integer read FDataScale write FDataScale;


//注：后面的属性是跟界面逻辑相关，对大部分人都是没有用的
//链接
property Url: stringread FUrl write FUrl;
//资源类型
property ResType: stringread FResType write FResType;
//公式
property Formula: stringread FFormula write FFormula;
//公式条件
property FormulaCondition: stringread FFormulaCondition write FFormulaCondition;
//汇总函数
property AggregateFun: stringread FAggregateFun write FAggregateFun;
//计量单位
property MeasureUnit: stringread FMeasureUnit write FMeasureUnit;
//字段值检查规则
```

```
property ValidateRule: stringread FValidateRule write FValidateRule;
//编辑器类型
property EditorType: stringread FEditorType write FEditorType;
//标签文字
property LabelText: stringread FLabelText write FLabelText;
//编辑器是否只读
property EditorReadOnly: Boolean read FEditorReadOnly write FEditorReadOnly;
//编辑器是否激活
property EditorEnabled: Boolean read FEditorEnabled write FEditorEnabled;
//显示格式
property DisplayFormat: stringread FDisplayFormat write FDisplayFormat;
//输入格式
property EditFormat: stringread FEditFormat write FEditFormat;
//字体名称
property FontName: stringread FFontName write FFontName;
//字体大小
property FontSize: Double read FFontSize write FFontSize;
//字体样式
property FontStyle: Integer read FFontStyle write FFontStyle;
//前景颜色
property ForeColor: Integer read FForeColor write FForeColor;
//背景颜色
property BackColor: Integer read FBackColor write FBackColor;
//下拉列表
property DropDownItems: stringread FDropDownItems write FDropDownItems;
//下拉模式_0 无 1 选择 2 编辑 3 添加
property DropDownMode: _ENUM_TCtFieldDropDownMode read FDropDownMode write
FDropDownMode;
//DML 图形描述
property GraphDesc: stringread FGraphDesc write FGraphDesc;

//显示级别
property Visibility: Integer read FVisibility write FVisibility;
//文字对齐
property TextAlign: _ENUM_TAlignment read FTextAlign write FTextAlign;
//列宽
property ColWidth: Integer read FColWidth write FColWidth;
//最大长度
property MaxLength: Integer read FMaxLength write FMaxLength;
//是否可搜索
property Searchable: Boolean read FSearchable write FSearchable;
//是否可查询
property Queryable: Boolean read FQueryable write FQueryable;

//初始值
property InitValue: stringread FInitValue write FInitValue;
//值格式类型
property ValueFormat: stringread FValueFormat write FValueFormat;
//最小值
property ValueMin: stringread FValueMin write FValueMin;
//最大值
property ValueMax: stringread FValueMax write FValueMax;
//扩展属性
property ExtraProps: stringread FExtraProps write FExtraProps;
//自定义配置
property CustomConfigs: stringread FCustomConfigs write FCustomConfigs;
end;
```

字段列表定义：

```
TCtMetaFieldList = class(TCtObjectList)
private
protected
public
function NewMetaField: TCtMetaField;

function FieldByName(AName: string): TCtMetaField;

property OwnerTable: TCtMetaTable read FOwnerTable;
end;
```

字段相关的枚举类型，参见后面的《枚举类型》一节。

七、 扩展对象

1. 输出

就是那个全局变量 curOut 了，它其实是一个 TStringList 对象，参见 TStringList 的定义。

在不同时候，curOut 的指向不一样，一般是指向一个内存对象，但调试时可能指向调试窗口的输出框，表属性生成时指向生成结果框，或者说，这些情况下执行完成后会把 CurOut 的结果拷贝到结果框中。而其它情况下，它可能不重要，没地方显示，执行完成就扔了，你看不到它的内容。

2. 参数面板

这个是我的“小发明”，有时需要输入点参数，老是用 InputBox 输入框太单调，写代码做窗体对话框又太烦，因此我搞了个折中的方案：某些情况下（目前就是表或字段的属性页里）在脚本的最前面插一段特别的注释，就可以生成一个参数面板，显示在界面上，每当你修改了值或点了按钮，参数就会记下来，并重新执行你的脚本，你就可以在脚本里存取这些参数了。

参数设置面板配置格式如下，包括(*和*)（Javascript 下还需要在前后加/**/注释），属性间不能有空格，放在 PAS 文件的最前面：

```
(*[SettingsPanel]
Control="Label";Caption="Please choose the parameters";Params="[FULLWIDTH][HEIGHT=40]"
Control="Edit";Name="Author";Caption="Your name";";Value="huz";Params="[FULLWIDTH]"
Control="Edit";Name="IsTest";Caption="Test flag";";Value="1";Params="[HIDDEN]"
Control="Memo";Name="Introduce";Caption="Introduce";";Value="Your info..#13#10 to be
continued..";Params="[FULLWIDTH][HEIGHT=80]"
```

```
Control="ComboBox";Name="GenType";Caption="Generate
type";Items="Type1,Type2,Type3";Value="Type1"
Control="RadioBox";Name="Param1";Caption="Param1";Items="V1,V2,V3";Value="V1"
Control="CheckBox";Name="Param2";Caption="Param2";Items="V1";Value="V2"
Control="Button";Name="Help";Caption="Click here for help";";Value="Help..."
[/SettingsPanel]*)
```

在脚本中可以使用 curSettingsPanel 这个全局变量来获取参数面板对象，参数面板对应的类是 TDmlScriptControlList，它是一个列表（TList），每个子项的对象类为 TDmlScriptControl，它们的定义如下：

```
{ DML 脚本控件 }
TDmlScriptControl = class(TObject)
private
public
function HasValue: Boolean;

//编号
property Id: Integer read FId write FId;
//控件类型
property ControlType: stringread FControlType write FControlType;
//名称
property Name: stringread FName write FName;
//标题
property Caption: stringread FCaption write FCaption;
//子项
property Items: stringread FItems write FItems;
//值
property Value: stringread FValue write SetValue;
//其它参数
property Params: stringread FParams write FParams;

//描述字
property TextDesc: stringread GetTextDesc write SetTextDesc;
//关联控件
property Control: TControl read FControl write FControl;
end;

TDmlScriptControlList = class(TList)
public
function NewItem: TDmlScriptControl; virtual;
function GetItemValue(AName: string): string;
procedure SetItemValue(AName: string; AValue: string);

procedure RegenControls;
procedure SaveSettings;

property Items[Index: Integer]: TDmlScriptControl read GetItem write PutItem; default;
property CurFileName: stringread FCurFileName write FCurFileName;
property CurAction: stringread FCurAction write FCurAction;
//描述字
property TextDesc: stringread GetTextDesc write SetTextDesc;
//JSON 值
property JsonValStr: stringread GetJsonValStr write SetJsonValStr;
property ParentWnd: TWinControl read FParentWnd write FParentWnd;
end;
```

再强调下，不是所有地方都会加载这个参数面板配置。

3. 全局变量

以下为主要的全局变量：

- AllModels: TCtDataModelGraphList;//所有模型列表。
- CurModel: TCtDataModelGraph; //当前模型（注：可能为空）
- CurTable: TCtMetaTable;//当前表（注：可能为空）
- CurField: TCtMetaField;//当前字段（注：可能为空）
- CurOut: TStringList;//当前输出设备
- CurSettingsPanel: TDmlScriptControlList;//当前参数面板（注：可能为空）
- Application: TApplication; //程序应用对象，可获取 MainForm、ExeName
- Screen: TScreen; //屏幕对象，可用于遍历窗体和获取当前活动的窗体、控件

4. 环境参数

这里列举一些常用的环境参数获取方法（PAS）：

- 主程序 EXE 文件名：Application.ExeName
- 主程序 EXE 文件所在目录：ExtractFilePath(Application.ExeName)
- 主窗口：Application.MainForm
- 当前焦点所在窗口：Screen.ActiveForm
- 当前焦点控件：Screen.ActiveControl
- 当前（或最近一次）连接的数据库类型：CurDbType()
- 当前 WINDOWS 用户名：GetEnv('WINUSER')
- 当前 WINDOWS 计算机名：GetEnv('COMPUTER')
- 当前机器 IP：GetEnv('IP')
- 当前 DML 文件全名：GetEnv('DMLFILEPATH')
- 当前程序命令行：GetEnv('CMDLINE')
- 当前选中的对象：GetSelectedCtMetaObj()

5. 全局方法

以下函数是 Pascal 的实现，在 JavaScript 中使用时，除了首字母小写，还要加下划线前缀，如：_sleep(500);

```
function ExecAppCmd(Cmd, param1, param2: String): String;
//获取环境参数 AName= WINUSER,COMPUTER,IP
function GetEnv(const AName: string): string;
//当前选中的对象（可能是表、字段、模型图）
```

```
function GetSelectedCtMetaObj: TCtMetaObject;
//把对象（可能是表、字段、模型图）输出为 JSON 字符串
function CtObjToJsonStr(ACtObj: TCtMetaObject): String;
//从 JSON 字符串中读取对象信息
function ReadCtObjFromJsonStr(ACtObj: TCtMetaObject; AJsonStr: String): Boolean;

procedure Sleep(milliseconds: Cardinal); //延时

function EncodeDate(Year, Month, Day: Word): TDateTime; //生成日期
function EncodeTime(Hour, Min, Sec, MSec: Word): TDateTime;//生成时间
function DayOfWeek(const DateTime: TDateTime): Word;//获取星期几
function Date: TDateTime;//当前日期
function Time: TDateTime;//当前时间
function Now: TDateTime;//当前日期和时间
function CurrentTimeMillis: Int64;//当前毫秒数
function DateTimeToJSTime(D: TDateTime): Int64;     //Delphi 时间与 JavaScript 的毫秒数互转
function JSTimeToDateTime(tm: Int64): TDateTime;

function DateToStr(D: TDateTime): string;//日期转字符串
function StrToDate(const s: string): TDateTime;//字符串转日期
function FormatDateTime(const fmt: string; D: TDateTime): string;
function TimeToStr(const DateTime: TDateTime): string;
function DateTimeToStr(const DateTime: TDateTime): string;
function StrToTime(const S: string): TDateTime;
function StrToTimeDef(const S: string; constDefault: TDateTime): TDateTime;
function StrToDateTime(const S: string): TDateTime;
function StrToDateTimeDef(const S: string; constDefault: TDateTime): TDateTime;

function FileAge(const FileName: string): Integer;//文件时间（微软格式）
function FileExists(const FileName: string): Boolean;
function DirectoryExists(const Directory: string): Boolean;
function ForceDirectories(Dir: string): Boolean;

function FileGetDate(Handle: Integer): Integer;//文件时间（微软格式）
function FileSetDate(Handle: Integer; Age: Integer): Integer;
function FileGetAttr(const FileName: string): Integer;
function FileSetAttr(const FileName: string; Attr: Integer): Integer;
function FileIsReadOnly(const FileName: string): Boolean;
function FileSetReadOnly(const FileName: string; ReadOnly: Boolean): Boolean;
function DeleteFile(const FileName: string): Boolean;
function RenameFile(const OldName, NewName: string): Boolean;
function ChangeFileExt(const FileName, Extension: string): string;
function ExtractFilePath(const FileName: string): string;
function ExtractFileDir(const FileName: string): string;
function ExtractFileDrive(const FileName: string): string;
function ExtractFileName(const FileName: string): string;
function ExtractFileExt(const FileName: string): string;
function ExpandFileName(const FileName: string): string;
function ExtractRelativePath(const BaseName, DestName: string): string;
function ExtractShortPathName(const FileName: string): string;
function FileSearch(const Name, DirList: string): string;
function DiskFree(Drive: Byte): Int64;
function DiskSize(Drive: Byte): Int64;
function FileDateToDateTime(FileDate: Integer): TDateTime;
function DateTimeToFileDate(DateTime: TDateTime): Integer;
function GetCurrentDir: string;
function SetCurrentDir(const Dir: string): Boolean;
```

```
function CreateDir(const Dir: string): Boolean;
function RemoveDir(const Dir: string): Boolean;
function StrToFloatDef(const S: string; constDefault: Extended): Extended;

procedure Beep;

function StringReplace(const S, OldPattern, NewPattern: string; Flags: TReplaceFlags): string;
function Format(const Format: string; const Args: arrayof Variant): string;
procedure Randomize;
function RandomI(Range: Integer): Integer;
function RandomD: Double;
function CtGenGUID: string;

procedure Writeln(s: string);
function Readln(question: string): string;
procedure WriteLog(s: string);   //输出一行消息到日志文件（ezdml.log），会自动加时间
function InputBox(const ACaption, APrompt, ADefault: string): string;
function InputMemoQuery(const ACaption, APrompt: string; var Value: string; bReadOnly: Boolean):
Boolean; Sender.AddRegisteredVariable('Application', 'TApplication

procedure RaiseErr(msg: string); //脚本报错，将中止脚本执行
procedure Abort;   //中止脚本执行
procedure RaiseErrOut(msg: string); //脚本报错并中止所有操作
procedure AbortOut; //中止所有操作
function CtCopyStream(src,dst: TStream; Count: LongInt): LongInt;
procedure ShowMessage(const Msg: string);
procedure MsgBox(const Msg: Variant);
procedure Alert(const Msg: Variant);
function EscapeXml(const XML: Variant): string;
procedure ShellOpen(FileName, Parameters, Directory: String); //执行 Shell 的 open 命令
function OpenFileDialog(const ATitle, AFilter, AFileName: string; bMulti: Boolean): string;
function OpenPicDialog(const ATitle, AFilter, AFileName: string; bMulti: Boolean): string;
function ExtractCompStr(const Strsrc: string; const sCompS, sCompE: string; bCaseInsensitive:
Boolean; const Def: string): string;
function AddOrModifyCompStr(const Strsrc, SubVal: string; const sCompS, sCompE: string;
bCaseInsensitive: Boolean): string;
function RunCmd(cmd: string; timeout: Integer): string; //运行批处理命令，并返回结果
function StringExtToWideCode(Str: string): string; //字符串转成 HEX 格式
function WideCodeNarrow(Str: string): string;   //HEX 格式转字符串

function GetClassName(v: TObject):string; //仅限 Pascal 脚本，JS 直接调 obj._className
functionCreateComponent (AOwner: TComponent; const AClassName: string): TComponent;
procedureReadDfmComponents(ARoot: TComponent; constDfm: string);
function FindChildComp(AOwner: TComponent; const AName: string): TComponent;
procedure SetCompPropValue(AComponent: TComponent; AKey, sValue: string);
function GetCompPropValue(AComponent: TComponent; AKey: string): string;
function GetCompPropObject(AComponent: TComponent; AKey: string): TObject;
procedure SetCompPropObject(AComponent: TComponent; AKey: string; sValue: TObject);

function Utf8Encode(const WS: WideString): String;
function Utf8Decode(const S: String): WideString;

function URLEncode(const msg: String): String;
function URLDecode(const url: string): string;
function Utf8URLEncode(const msg: String): String;

function GetPointerByObj(v: TObject): Integer;
```

```
function GetObjByPointer( p: Integer):TObject;
function GetPointerByStr(var v: string): Integer;
function GetStrByPointer( p: Integer):string;
function GetPointerByWStr(var v: WideString): Integer;
function GetWStrByPointer( p: Integer):WideString;
function Send_Msg(AHandle: Integer; Msg: Integer; wParam: Integer; lParam: Integer): Integer;
function Send_StrMsg(AHandle: Integer; Msg: Integer; wParam: WideString; lParam: WideString):
WideString;

function TextToShortCut(Text: string): TShortCut;

function ChnToPY(Value: string): string;

function GetGParamValue(AName: string): string; //设置全局参数字符值
function GetGParamObject(AName: string): TObject; //设置全局参数对象值
procedure SetGParamValue(AName, AValue: string); //获取全局参数字符值
procedure SetGParamObject(AName: string; AObject: TObject); //获取全局参数对象值
procedure SetGParamValueEx(AName, AValue: string; AObject: TObject); //设置全局参数的字符和对
象值

function IsReservedKeyworkd(str: string): Boolean;

function GetDbQuotName(AName, dbType: string): string;
function GetIdxName(ATbn, AFieldName: string): string;

function GetAppDefTempPath: string;   //获取临时文件路径
function GetUnusedFileName(AFileName: string): string; //获取可用的文件名，如果文件已经存在，会
自动加数字到文件名上

function Clipboard: TClipboard;   //获取剪贴板对象

function IsEnglish: Boolean; //当前语言是否为英文
function CurDbType: String;   //当前数据库连接类型
function ExecCtDbLogon(DbType, database, username, password: string; bSavePwd, bAutoLogin,
bShowDialog: Boolean; opt: string): string; //执行数据库连接，DbType 不为空时可指定数据服务名、用
户名和密码
function ExecSql(sql, opts: string): TDataSet; //尝试执行 SQL，结果如果非空，需要自行 Free

//从参数面板取值
function CurAction: string;//当前按下的按钮，已过时，请用 CurSettingsPanel.CurAction 代替
function GetParamValue(AName: string): string;//获取某个面板项的值，已过时，请用
CurSettingsPanel.GetItemValue 代替
function GetParamValueDef(AName,ADef: string): string;

procedure SyncTableProps(Tb: TCtMetaTable); //修改了表属性后，同步属性到模型中其它所有同名的表

procedure RunDmlScript(AFileName, AScript: string); //运行脚本，文件后缀名决定脚本类型，AScript
为脚本内容，为空则从文件加载
procedure PrintVar(const AVal: Variant); //往 CurOut 中输出内容（不换行）
```

6. 全局参数

所谓全局参数，就是只要程序不关闭就一直存在的一个参数列表，方便你在多次执行脚

本时传递一些内容。

全局参数的读写就是靠下面几个方法：

```
function GetGParamValue(AName: string): string; //设置全局参数字符值
function GetGParamObject(AName: string): TObject; //设置全局参数对象值
procedure SetGParamValue(AName, AValue: string); //获取全局参数字符值
procedure SetGParamObject(AName: string; AObject: TObject); //获取全局参数对象值
procedure SetGParamValueEx(AName, AValue: string; AObject: TObject); //设置全局参数的字符和对象值
```

7. 常量

下面这些常量可以直接在脚本里用（PAS 和 JS 里都可以）：

- MB_OK
- MB_OKCANCEL
- MB_ABORTRETRYIGNORE
- MB_YESNOCANCEL
- MB_YESNO
- MB_RETRYCANCEL
- MB_ICONQUESTION
- MB_ICONWARNING
- MB_ICONERROR
- MB_ICONINFORMATION
- MB_ICONSTOP
- MB_DEFBUTTON1
- MB_DEFBUTTON2
- MB_DEFBUTTON3
- MB_DEFBUTTON4
- IDOK
- IDCANCEL
- IDABORT
- IDRETRY
- IDIGNORE
- IDYES
- IDNO
- IDCLOSE
- IDCONTINUE

8. 枚举类型

对于枚举类型，如整数字段数据类型 cfdtInteger，PAS 里可直接用 cfdtInteger，JS 脚本里是用字符串'cfdtInteger'来转换的。

表字段相关的枚举类型主要有：

TctDataLevel = (ctdlUnknown, ctdlNormal, ctdlNew, ctdlModify, ctdlDeleted, ctdlDraft, ctdlDebug);

```
TCtFieldDataType = (
    cfdtUnknow,
    cfdtString,
    cfdtInteger,
    cfdtFloat,
    cfdtDate,
    cfdtBool,
    cfdtEnum,
    cfdtBlob,
    cfdtObject,
    cfdtList,
    cfdtFunction,
    cfdtEvent,
    cfdtOther);

TCtKeyFieldType =
    (
    cfktNormal,
    cfktId,
    cfktPid,
    cfktRid,
    cfktName,
    cfktCaption,
    cfktComment,
    cfktTypeName,
    cfktOrgId,
    cfktPeriod,
    cfktCreatorId,
    cfktCreatorName,
    cfktCreateDate,
    cfktModifierId,
    cfktModifierName,
    cfktModifyDate,
    cfktVersionNo,
    cfktHistoryId,
    cfktLockStamp,
    cfktInstNo,
    cfktProcID,
```

```
    cfktURL,
    cfktStatus,
    cfktOrderNo,
    cfktOthers
    );

  TCtFieldIndexType =
    (
    cfitNone,
    cfitUnique,
    cfitNormal
    );
```

9. 其它

接下来的内容又长又没营养，建议跳过。会 DELPHI 的也可以忽略本节内容。

这里只列 EZDML 脚本支持的内容，各类属性方法的详细说明就随意了，有需要的请参考 DELPHI 的文档。

JS 里不支持数组下标方式取对象属性，所以 PAS 里类似 Items[I]、Components[I]、Controls[I]、Forms[I]之类的属性，在 JS 里要写成 getItem(i)、getComponent(i)、getControl(i)、getForm(i)等。

接下来开启疯狂省略和拷贝粘贴模式。

i. TComponent

DELPHI 的基础对象之一，接下来的 Control、WinControl、Form、Screen、Application 都是它的子类，定义如下：

```
  TComponent = class(TPersistent)
private
public
function FindComponent(const AName: string): TComponent;
property Components[Index: Integer]: TComponent read GetComponent;
property ComponentCount: Integer read GetComponentCount;
property ComponentIndex: Integer read GetComponentIndex write SetComponentIndex;
property ComponentState: _ENUM_TComponentState read FComponentState;
property ComponentStyle: _ENUM_TComponentStyle read FComponentStyle;
property Owner: TComponent read FOwner;

property Name: Stringread FName write SetName stored False;
property Tag: Longint read FTag write FTag default0;
end;
```

至于它的父类，不重要，不必在意是它是谁。

ii. TControl

控件的祖先类，继承自 TComponent，后面的 WinControl、Form 是它的子类，你在 EZDML 上看到的所有界面，基本上都是它的子类。

```
  TControl = class(TComponent)
private
public
procedure BringToFront;
function ClientToScreen(const px, py: Integer): String;
function Dragging: Boolean;
property Enabled: Boolean read GetEnabled write SetEnabled stored IsEnabledStored default True;
function HasParent: Boolean; override;
procedure Hide;
procedure Invalidate; virtual;
procedure Refresh;
procedure Repaint; virtual;
function ScreenToClient(const px, py: Integer): String;
procedure SendToBack;
procedure SetBounds(ALeft, ATop, AWidth, AHeight: Integer); virtual;
procedure Show;
procedure Update; virtual;
property Action: TBasicAction read GetAction write SetAction;
property Align: _ENUM_TAlign read FAlign write SetAlign default alNone;
property Anchors: Stringread FAnchors write SetAnchors stored IsAnchorsStored default [akLeft,
akTop];
property BoundsRect: Stringread GetBoundsRect write SetBoundsRect;
property ClientHeight: Integer read GetClientHeight write SetClientHeight stored False;
property ClientRect: Stringread GetClientRect;
property ClientWidth: Integer read GetClientWidth write SetClientWidth stored False;
property Parent: TWinControl read FParent write SetParent;
property ShowHint: Boolean read FShowHint write SetShowHint stored IsShowHintStored;
property Visible: Boolean read FVisible write SetVisible stored IsVisibleStored default True;

property Left: Integer read FLeft write SetLeft;
property Top: Integer read FTop write SetTop;
property Width: Integer read FWidth write SetWidth;
property Height: Integer read FHeight write SetHeight;
property Cursor: Integer read FCursor write SetCursor default crDefault;
property Hint: stringread FHint write FHint stored IsHintStored;

property Caption: Stringread GetCaption write SetCaption;
property Text: Stringread GetText write SetText;
property Color: Integer read FColor write SetColor stored IsColorStored default clWindow;
property FontName: Stringread GetFontName write SetFontName;
property FontSize: Integer read GetFontSize write SetFontSize;
property FontColor: Integer read GetFontColor write SetFontColor;
property ParentColor: Boolean read FParentColor write SetParentColor default True;
property ParentFont: Boolean read FParentFont write SetParentFont default True;
property ParentShowHint: Boolean read FParentShowHint write SetParentShowHint default True;
end;
```

iii. TWinControl

这是 Windows 自带的组件的封装，最大的特点是它有窗口句柄 Handle，就是 hWnd。定义如下：

```
  TWinControl = class(TControl)
public
constructor Create(AOwner: TComponent); override;
destructor Destroy; override;
function CanFocus: Boolean; dynamic;
function ContainsControl(Control: TControl): Boolean;
function ControlAtPos(const Pos_X, PosY: Integer; AllowDisabled: Boolean;
        AllowWinControls: Boolean = False; AllLevels: Boolean = False): TControl;
procedure DisableAlign;
property DoubleBuffered: Boolean read FDoubleBuffered write FDoubleBuffered;
procedure EnableAlign;
function FindChildControl(const ControlName: string): TControl;
function Focused: Boolean; dynamic;
function HandleAllocated: Boolean;
procedure HandleNeeded;
procedure InsertControl(AControl: TControl);
procedure RemoveControl(AControl: TControl);
procedure Realign;
procedure ScaleBy(M, D: Integer);
procedure ScrollBy(DeltaX, DeltaY: Integer);
procedure SetFocus; virtual;
procedure UpdateControlState;
property AlignDisabled: Boolean read GetAlignDisabled;
property MouseInClient: Boolean read FMouseInClient;
property Controls[Index: Integer]: TControl read GetControl;
property ControlCount: Integer read GetControlCount;
property Handle: HWnd read GetHandle;
property ParentWindow: HWnd read FParentWindow write SetParentWindow;
property Showing: Boolean read FShowing;
property TabOrder: Integer read GetTabOrder write SetTabOrder default -1;
property TabStop: Boolean read FTabStop write SetTabStop default False;
end;
```

iv. TForm

TForm 为窗体对象，定义如下：

```
  TForm = class(T..WinControl)
public
constructor Create(AOwner: TComponent); override;
destructor Destroy; override;
procedure Close;
function CloseQuery: Boolean; virtual;
procedure DefocusControl(Control: TWinControl; Removing: Boolean);
procedure FocusControl(Control: TWinControl);
procedure Hide;
procedure Print;
```

```
procedure Release;
function SetFocusedControl(Control: TWinControl): Boolean; virtual;
procedure Show;
function ShowModal: Integer; virtual;
property Active: Boolean read FActive;
property ActiveControl: TWinControl read FActiveControl write SetActiveControl
stored IsForm;
property BorderStyle: _ENUM_TFormBorderStyle read FBorderStyle write SetBorderStyle
stored IsForm default bsSizeable;
property ModalResult: Integer read FModalResult write FModalResult;
property WindowState: _ENUM_TWindowState read FWindowState write SetWindowState
stored IsForm default wsNormal;
end;
```

程序主窗体可以通过 application.mainForm 获得，参见 Application 对象的说明。

程序的所有窗体可以通过 screen. getForm(i)（JS）或 Screen.Forms[I]（PAS）遍历，参见 TScreen 对象的说明。

v. TScreen

DELPHI 的屏幕对象，定义如下：

```
  TScreen = class(TComponent)
private
public
constructor Create(AOwner: TComponent); override;
destructor Destroy; override;
function MonitorFromPoint(const Point_X, Point_Y: Integer;
        MonitorDefault: _ENUM_TMonitorDefaultTo = mdNearest): TMonitor;
function MonitorFromRect(const Rect_Left, Rect_Top, Rect_Right, Rect_Bottom: Integer;
        MonitorDefault: _ENUM_TMonitorDefaultTo = mdNearest): TMonitor;
function MonitorFromWindow(const Handle: THandle;
        MonitorDefault: _ENUM_TMonitorDefaultTo = mdNearest): TMonitor;
procedure DisableAlign;
procedure EnableAlign;
procedure Realign;
procedure ResetFonts;
property ActiveControl: TWinControl read FActiveControl;
property ActiveCustomForm: TCustomForm read FActiveCustomForm;
property ActiveForm: TForm read FActiveForm;
property CustomFormCount: Integer read GetCustomFormCount;
property CustomForms[Index: Integer]: TCustomForm read GetCustomForms;
property CursorCount: Integer read FCursorCount;
property Cursor: Integer read FCursor write SetCursor;
property Cursors[Index: Integer]: THandle read GetCursors write SetCursors;
property DataModules[Index: Integer]: TDataModule read GetDataModule;
property DataModuleCount: Integer read GetDataModuleCount;
property FocusedForm: TCustomForm read FFocusedForm write FFocusedForm;
property SaveFocusedList: TList read FSaveFocusedList;
property MonitorCount: Integer read GetMonitorCount;
property Monitors[Index: Integer]: TMonitor read GetMonitor;
property DesktopRect: TRect read GetDesktopRect;
property DesktopHeight: Integer read GetDesktopHeight;
```

```
property DesktopLeft: Integer read GetDesktopLeft;
property DesktopTop: Integer read GetDesktopTop;
property DesktopWidth: Integer read GetDesktopWidth;
property WorkAreaRect: TRect read GetWorkAreaRect;
property WorkAreaHeight: Integer read GetWorkAreaHeight;
property WorkAreaLeft: Integer read GetWorkAreaLeft;
property WorkAreaTop: Integer read GetWorkAreaTop;
property WorkAreaWidth: Integer read GetWorkAreaWidth;
property Fonts: TStrings read GetFonts;
property FormCount: Integer read GetFormCount;
property Forms[Index: Integer]: TForm read GetForm;
property Imes: TStrings read GetImes;
property DefaultIme: stringread GetDefaultIme;
property Height: Integer read GetHeight;
property PixelsPerInch: Integer read FPixelsPerInch;
property PrimaryMonitor: TMonitor read GetPrimaryMonitor;
property Width: Integer read GetWidth;
end;
```

vi. TApplication

应用程序生命周期管理对象，只有一个的，通过全局变量 application 访问，类定义如下：

```
  TApplication = class(TComponent)
public
function MessageBox(const Text, Caption: PChar; Flags: Longint = MB_OK): Integer;
procedure Minimize;
procedure Restore;
procedure ProcessMessages;
procedure Terminate;
property Active: Boolean read FActive;
property ActiveFormHandle: Cardinal read GetActiveFormHandle;
property ExeName: stringread GetExeName;
property Handle: Cardinal read FHandle write SetHandle;
property MainForm: TForm read FMainForm;
property MainFormHandle: Cardinal read GetMainFormHandle;
property Terminated: Boolean read FTerminate;
property Title: stringread GetTitle write SetTitle;
end;
```

vii. TList

Delphi 的列表对象：

```
  TList = class(TObject)
public
function Add(Item: TObject): Integer;
procedure Clear; virtual;
```

```
procedure Delete(Index: Integer);
procedure Exchange(Index1, Index2: Integer);
function IndexOf(Item: TObject): Integer;
procedure Insert(Index: Integer; Item: TObject);
procedure Move(CurIndex, NewIndex: Integer);
function Remove(Item: TObject): Integer;
procedure Pack;
property Capacity: Integer read FCapacity write SetCapacity;
property Count: Integer read FCount write SetCount;
property Items[Index: Integer]: TObject read GetItem write SetItem; default;
end;
```

viii. TStringList

字符串列表：

```
  TStrings = class(TPersistent)
public
function Add(const S: string): Integer; virtual;
function AddObject(const S: string; AObject: TObject): Integer; virtual;
procedure Append(const S: string);
procedure AddStrings(Strings: TStrings); virtual;
procedure Assign(Source: TPersistent); override;
procedure BeginUpdate;
procedure Clear; virtual; abstract;
procedure Delete(Index: Integer); virtual; abstract;
procedure EndUpdate;
function Equals(Strings: TStrings): Boolean;
procedure Exchange(Index1, Index2: Integer); virtual;
function GetEnumerator: TStringsEnumerator;
function GetText: PChar; virtual;
function IndexOf(const S: string): Integer; virtual;
function IndexOfName(const Name: string): Integer; virtual;
function IndexOfObject(AObject: TObject): Integer; virtual;
procedure Insert(Index: Integer; const S: string); virtual; abstract;
procedure InsertObject(Index: Integer; const S: string;
        AObject: TObject); virtual;
procedure LoadFromFile(const FileName: string); virtual;
procedure LoadFromStream(Stream: TStream); virtual;
procedure Move(CurIndex, NewIndex: Integer); virtual;
procedure SaveToFile(const FileName: string); virtual;
procedure SaveToStream(Stream: TStream); virtual;
procedure SetText(Text: PChar); virtual;
function Clone: TStringList;

property Capacity: Integer read GetCapacity write SetCapacity;
property CommaText: stringread GetCommaText write SetCommaText;
property Count: Integer read GetCount;
property Delimiter: Char read GetDelimiter write SetDelimiter;
property DelimitedText: stringread GetDelimitedText write SetDelimitedText;
property LineBreak: stringread GetLineBreak write SetLineBreak;
property Names[Index: Integer]: stringread GetName;
property Objects[Index: Integer]: TObject read GetObject write SetObject;
property QuoteChar: Char read GetQuoteChar write SetQuoteChar;
```

```
property Values[const Name: string]: stringread GetValue write SetValue;
property ValueFromIndex[Index: Integer]: stringread GetValueFromIndex write SetValueFromIndex;
property NameValueSeparator: Char read GetNameValueSeparator write SetNameValueSeparator;
property StrictDelimiter: Boolean read GetStrictDelimiter write SetStrictDelimiter;
property Strings[Index: Integer]: stringread GetString write SetString; default;
property Text: stringread GetTextStr write SetTextStr;
end;
```

ix. TFileStream

注意：后面这几个类都是属于试水性质的，我自己都没用过的，具有不可预料的 BUG 出没的特点。

文件流对象（这是 JS 的，PAS 的实现可能稍有不同）：

```
  TFileStream = class(TObject)
public
function Read(Count: Longint): String;
procedure Write(S: String); virtual; abstract;
function Seek(Offset: Longint; Origin: Word): Longint; overload; virtual;
function CopyFrom(Source: TStream; Count: Int64): Int64;
procedure Flush;
procedure Close;
property Position: Int64 read GetPosition write SetPosition;
property Size: Int64 read GetSize write SetSize64;
property FileName: stringread FFileName;
property Eof: Boolean read GetEof;
end;
```

x. TClipboard

显然是剪贴板了：

```
TClipboard = class(TPersistent)
private
public
  destructor Destroy; override;
  procedure Assign(Source: TPersistent); override;
  procedure Clear; virtual;
  procedure Close; virtual;
  function GetComponent(Owner, Parent: TComponent): TComponent;
  function GetAsHandle(Format: Word): THandle;
  function GetTextBuf(Buffer: PChar; BufSize: Integer): Integer;
```

```
  function HasFormat(Format: Word): Boolean;
  procedure Open; virtual;
  procedure SetComponent(Component: TComponent);
  procedure SetAsHandle(Format: Word; Value: THandle);
  procedure SetTextBuf(Buffer: PChar);
  property AsText: string read GetAsText write SetAsText;
  property FormatCount: Integer read GetFormatCount;
  property Formats[Index: Integer]: Word read GetFormats;
end;
```

获取全局剪贴板对象（注意是个全局方法而不是全局变量）：

```
function Clipboard: TClipboard;   //获取剪贴板对象
```

xi. TIniFile

读写 INI 文件用的：

```
  TIniFile = class(TObject)
private
    FFileName: string;
public
constructor Create(const FileName: string);
procedure Clear;
procedure Rename(const FileName: string; Reload: Boolean);
procedure GetStrings(List: TStrings);
procedure SetStrings(List: TStrings);
function SectionExists(const Section: string): Boolean;
function ReadString(const Section, Ident, Default: string): string; virtual; abstract;
procedure WriteString(const Section, Ident, Value: String); virtual; abstract;
function ReadInteger(const Section, Ident: string; Default: Longint): Longint; virtual;
procedure WriteInteger(const Section, Ident: string; Value: Longint); virtual;
function ReadBool(const Section, Ident: string; Default: Boolean): Boolean; virtual;
procedure WriteBool(const Section, Ident: string; Value: Boolean); virtual;
function ReadBinaryStream(const Section, Name: string; Value: TStream): Integer; virtual;
function ReadDate(const Section, Name: string; Default: TDateTime): TDateTime; virtual;
function ReadDateTime(const Section, Name: string; Default: TDateTime): TDateTime; virtual;
function ReadFloat(const Section, Name: string; Default: Double): Double; virtual;
function ReadTime(const Section, Name: string; Default: TDateTime): TDateTime; virtual;
procedure WriteBinaryStream(const Section, Name: string; Value: TStream); virtual;
procedure WriteDate(const Section, Name: string; Value: TDateTime); virtual;
procedure WriteDateTime(const Section, Name: string; Value: TDateTime); virtual;
procedure WriteFloat(const Section, Name: string; Value: Double); virtual;
procedure WriteTime(const Section, Name: string; Value: TDateTime); virtual;
procedure ReadSection(const Section: string; Strings: TStrings); virtual; abstract;
procedure ReadSections(Strings: TStrings); overload; virtual; abstract;
procedure ReadSections(const Section: string; Strings: TStrings); overload; virtual;
procedure ReadSectionValues(const Section: string; Strings: TStrings); virtual; abstract;
procedure EraseSection(const Section: string); virtual; abstract;
procedure DeleteKey(const Section, Ident: String); virtual; abstract;
procedure UpdateFile; virtual; abstract;
```

```
function ValueExists(const Section, Ident: string): Boolean; virtual;
property FileName: stringread FFileName;
property CaseSensitive: Boolean read GetCaseSensitive write SetCaseSensitive;
end;
```

xii. JSON

JS 天生支持 JSON，但 PAS 没有，所以专门为 PAS 导入了几个类：

```
    TZAbstractObject = class
      AutoFreeChild: Boolean;
      UserObj1: TObject;
      UserObj2: TObject;
function equals(const Value: TZAbstractObject): Boolean; virtual;
function hash: LongInt;
function Clone: TZAbstractObject; virtual;
function toString: string; virtual;
end;

{ @abstract(Classe que representa um objeto JSON) }
  TJSONObject = class (TZAbstractObject)
private
public
constructor create;        virtual;

constructor createWithStr (s : string);

procedure clean;
function clone : TZAbstractObject; override;
function accumulate (key : string; value : TZAbstractObject): TJSONObject;
function get (key : string) : TZAbstractObject;
function getBoolean (key : string): boolean;
function getDouble (key : string): double;
function getInt (key : string): integer;
function getList (key : string) :TJSONArray;
function getMap (key : string) : TJSONObject;
function getString (key : string): string;
function has (key : string) : boolean;
function isNull (key : string) : boolean;
function keys : TStringList ;
function length : integer;
function Count : integer;
function names : TJSONArray;
function opt (key : string) : TZAbstractObject;
function optBoolean (key : string): boolean;
function optBooleanDef (key : string; defaultValue : boolean): boolean;
function optDouble (key : string): double;
function optDoubleDef (key : string; defaultValue : double): double;
function optInt (key : string): integer;
function optIntDef (key : string; defaultValue : integer): integer;
function optString (key : string): string;
function optStringDef (key : string; defaultValue : string): string;

function optJSONArray (key : string): TJSONArray;
```

```
function optJSONObject (key : string): TJSONObject;

function putBoolean (key : string; value : boolean): TJSONObject;
function putDouble (key : string; value : double): TJSONObject;
function putInt (key : string; value : integer): TJSONObject;
function putString (key : string; value : string): TJSONObject;

function put (key : string; value : TZAbstractObject): TJSONObject;
function putOpt (key : string; value : TZAbstractObject): TJSONObject;
classfunction quote (s : string): string;
function remove (key : string): TZAbstractObject;
procedure assignTo(json: TJSONObject);

function toJSONArray (names : TJSONArray) : TJSONArray;
function toString (): string ;     override;
function toStringIF (indentFactor : integer): string;
function toStringEx (indentFactor, indent : integer): string;

destructor destroy;override;
classfunction NULL : NULL;
end;

{ @abstract(Trata um array JSON = [...])}
   TJSONArray = class (TZAbstractObject)
protected
public
destructor destroy ; override;
constructor create ;
constructor createWithStr (s : string);
function get (index : integer) : TZAbstractObject;
function getBoolean (index : integer) : boolean;
function getDouble (index : integer) : double;
function getInt (index : integer): integer;
{
        Get the TJSONArray associated with an index.
        @param(index The index must be between 0 and length() - 1.)
        @return(A TJSONArray value.)
        @raises(NoSuchElementException if the index is not found or if the
        value is not a TJSONArray)        }
function getList (index : integer) : TJSONArray;
function getMap (index : integer) : TJSONObject;
function getString (index : integer) : string;
function isNull (index : integer): boolean;
function join (separator : string) : string;
function Count: Integer;
function opt (index : integer) : TZAbstractObject;
function optBoolean ( index : integer) : boolean;
function optBooleanDef ( index : integer; defaultValue : boolean) : boolean;
function optDouble (index : integer) : double;
function optDoubleDef (index : integer; defaultValue :double ) : double ;
function optInt (index : integer) : integer;
function optIntDef (index : integer; defaultValue : integer) : integer;
function optJSONArray (index : integer) : TJSONArray ;
function optJSONObject (index : integer) : TJSONObject ;
function optString (index : integer) : string;
function optStringDef (index : integer; defaultValue : string) : string;
function putBoolean ( value : boolean) : TJSONArray;
```

```
function putDouble ( value : double ) : TJSONArray;
function putInt ( value : integer) : TJSONArray;
function putZ ( value : TZAbstractObject) : TJSONArray;
function putString ( value: string): TJSONArray;
function putBooleanAt ( index : integer ; value : boolean): TJSONArray;
function putDoubleAt ( index : integer ; value : double) : TJSONArray;
function putIntAt ( index : integer ; value : integer) : TJSONArray;
function putZAt ( index : integer ; value : TZAbstractObject) : TJSONArray;
function putStringAt ( index: integer; value: string): TJSONArray;
procedure Insert(Index: Integer; value: TZAbstractObject);
procedure Exchange(Index1, Index2: Integer);
procedure Move(CurIndex, NewIndex: Integer);
function toString : string;   override;
function toStringIF (indentFactor : integer) : string;
function toStringEx (indentFactor, indent : integer) : string;
function toList () : TList;
end;
```

八、 结束

终于搞完，闲扯几句。

一下复制粘贴了那么多内容，简直有点丧心病狂了。EZDML 以简单为卖点，照理是不应该弄脚本这种复杂的东西的。但对程序员来说，脚本是家常便饭，写脚本也可能是实现目标的简单办法。我最开始也没想到会搞脚本这东西，但搞着搞着就搞上瘾了，可能程序员就是喜欢把简单的问题搞复杂，脚本这东西更多是我搞出来自娱自乐玩耍的。

脚本能够访问模型中的所有对象及其属性，能够批量进行各种增删改查，能生成你想要的各种格式。但有得必有失，脚本这东东有那么一点点门槛，可能需要点学习成本，其实对程序员来说还是不难的。那一堆复杂的姿势用法一般也是没有必要学的。

EZDML 本身是 DELPHIPASCAL 写的，对 PASCAL 脚本支持非常好，做界面接管事件（注：x64 版目前不支持 PASCAL 脚本拦截事件）调 DLL 都不在话下，用的当然是 RemObjects 的 PASCAL 引擎了，因为仅此一家，别无分店。

JS 是最近才加的，BUGs 虫出没是难免的了。这年头会 PASCAL 的人太少，会 JAVASCRIPT 的到处都是，不会 JS 都不好意思跟人打招呼了，恰好接触了一下 BESEN 这个 JS 脚本引擎，觉得还行，就把它加进去了。为什么选它而不选高大上的微软谷歌火狐的 JS 引擎？就因为它也是用 PASCAL 实现的，可以较好地融入 EZDML 里。

但我还没找到在 JS 里接管事件的方法，所以 JS 脚本目前是无法直接接管 EZDML 里的事件的（无法直接，但可以间接，比如自定义 UI，你点一个按钮后传递新参数重新执行所有脚本，这是可以的）。另外 JS 里无法直接加载调用 DLL，占用内存大，执行效率相对低。当然 JS 也有自己的优点，比如书写简单，学习曲线平稳，天生就支持 JSON、正则表达式等。

脚本可以说是 EZDML 很重要的一个特色功能，但我觉得大多数情况下是照着示例简单遍历一下表字段就够了。一直有点犹豫要不要把脚本功能文档写全，因为里面有些东西我感觉

对大部分人完全没有用，更多是我自己拿来玩耍显摆的，写出来有人可能会说我光会炫技做了一堆无用的功能。我平时确实经常干这种作无用功的事，以后也可能还会时不时给 EZDML 加一些看似有点意思实则无用的功能。

不过，我后来想还是写一下，也许个别人用得上呢，虽然用的人少，但我也应该为他们指明“正确”的姿势，免得走弯路。不需要的人可以忽略。