Обратите внимание: этот компонент был разработан, используя Delphi 5. Для более новой версии требуются некоторые изменения, поскольку *Borland* изменил название некоторых модулей.

Этот компонент позволяет программисту устанавливать ключ системного реестра, где будут сохраняться параметры настройки. Имя раздела будет создано, используя составляющее имя, двоеточие и имя свойства. Значение сохраняется как строка.

Значения всех свойств преобразуется в строку. Следующие типы могут использоваться:

- tkInteger
- tkInt64
- tkFloat
- tkEnumeration
- tkSet
- tkChar
- tkString
- tkLString

Component **State Recorder** создает коллекцию **SavedComponents**, где все компоненты, чьи свойства должны быть сохранены в реестре, будут добавлены. Каждый элемент **TSavedComponent**
этой коллекции создает другую коллекцию. Все свойства
TSavedProperty
будут сохранены в пределах этой коллекции.

Исходный код Component **State Recorder**

unit ComponentStateRecovery; interface uses Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, TypInfo, DsgnIntf; type TSavedProperty = class; TSavedComponent = class; TComponentStateRecorder = class; // single property that will be saved TSavedProperty = class(TCollectionItem) private // name of property to be saved FPropertyName: String; // default value if property does not exist FDefaultValue: String; procedure SetPropertyName(const Value: String); procedure SetDefaultValue(const Value: String); function GetRegistryValue: String; procedure

```
SetRegistryValue(const Value: String); protected function GetDisplayName: string;
override; public constructor Create(aCollection: TCollection); override; procedure
Assign(Source: TPersistent); override; published property PropertyName: String read
FPropertyName write SetPropertyName; property RegistryValue: String read
GetRegistryValue write SetRegistryValue; property DefaultValue: String read
FDefaultValue write SetDefaultValue; end; TSavedProperties = class(TCollection)
 private // owner of this collection FSavedComponent: TSavedComponent; function
GetItem(Index: Integer): TSavedProperty; procedure SetItem(Index: Integer; const Value:
TSavedProperty); protected function GetOwner: TPersistent; override; procedure
Update(Item: TCollectionItem); override; public constructor Create(aSavedComponent:
TSavedComponent); function Add: TSavedProperty; property SavedComponent:
TSavedComponent read FSavedComponent; property Items[Index: Integer]:
TSavedProperty read GetItem write SetItem; published end; TSavedComponent
= class(TCollectionItem) private // name of component to be saved
FComponentName: String; // properties of 'this' component to be saved
FSavedProperties: TSavedProperties; procedure SetSavedProperties(const Value:
TSavedProperties); procedure SetComponentName(const Value: String); protected
function GetDisplayName: string; override; public constructor Create(aCollection:
TCollection); override; destructor Destroy; override; procedure Assign(Source:
TPersistent); override; published property SavedProperties: TSavedProperties read
FSavedProperties write SetSavedProperties; property ComponentName: String read
FComponentName write SetComponentName; end; TSavedComponents =
class(TCollection) private // owner of this collection FComponentStateRecorder:
TComponentStateRecorder; function GetItem(Index: Integer): TSavedComponent;
procedure SetItem(Index: Integer; const Value: TSavedComponent); protected function
GetOwner: TPersistent; override; procedure Update(Item: TCollectionItem); override;
public constructor Create(aComponentStateRecorder: TComponentStateRecorder);
function Add: TSavedComponent; property Items[Index: Integer]: TSavedComponent
read GetItem write SetItem; published end; // action of the record (save to registry -
// or - load from registry) TRecorderAction = (raSave, raLoad);
TComponentStateRecorder = class(TComponent) private // components of owner form to
be saved FSavedComponents: TSavedComponents; // registry key - where form
components will be saved FRegistryKey: String; procedure SetSavedComponents(const
Value: TSavedComponents); procedure SetRegistryKey(const Value: String); procedure
DoStates(Action: TRecorderAction); protected public constructor Create(aOwner:
TComponent); override; destructor Destroy; override; procedure SaveStates;
procedure LoadStates; published property SavedComponents: TSavedComponents
read FSavedComponents write SetSavedComponents; property RegistryKey: String
read FRegistryKey write SetRegistryKey; end; procedure Register;
implementation uses Registry; procedure Register; begin // register component
RegisterComponents('gate(n)etwork', [TComponentStateRecorder]); end; function
GetPropertyAsString( Component: TComponent; PropInfo: PPropInfo ): String; begin
with PropInfo^ do case PropType^.Kind of tkInteger: // get integer value Result :=
IntToStr(GetOrdProp(Component, PropInfo)); tkInt64: // get integer (64 bit) value
Result := IntToStr(GetOrdProp(Component, PropInfo)); tkFloat: // get float value
Result := FloatToStr(GetFloatProp(Component, PropInfo)); tkEnumeration: // get
```

```
enumeration value Result := GetEnumProp(Component, PropInfo); tkSet: // get set value Result := GetSetProp(Component, PropInfo); tkChar: // get single character value Result := Chr(GetOrdProp(Component, PropInfo)); tkString, tkLString: // get string value Result := GetStrProp(Component, PropInfo); else Result := ''; end; end; procedure SetPropertyFromString( Component: TComponent; PropInfo: PPropInfo; Value: String ); begin try with PropInfo^ do case PropType^.Kind of tkInteger: // set integer value SetOrdProp(Component, PropInfo, StrToIntDef( Value, GetOrdProp(Component, PropInfo) )); tkInt64: // set integer (64 bit) value SetInt64Prop(Component, PropInfo, StrToIntDef( Value, GetInt64Prop(Component, PropInfo) )); tkFloat: // set float value SetFloatProp(Component, PropInfo, StrToFloat(Value)); tkEnumeration: // set enumeration value SetEnumProp(Component, PropInfo, Value); tkSet: // set set value SetSetProp(Component, PropInfo, Value); tkChar: // set single character value SetOrdProp(Component, PropInfo, Ord(Value[1])); tkString, tkLString: // set string value SetStrProp(Component, PropInfo, Value); end; except end; end; { TSavedProperty } procedure TSavedProperty.Assign(Source: TPersistent); begin if Source is TSavedProperty then begin // assign all local values FPropertyName := TSavedProperty(Source).FPropertyName; FDefaultValue := TSavedProperty(Source).FDefaultValue; end else begin inherited Assign(Source); end; end; constructor TSavedProperty.Create(aCollection: TCollection); begin inherited Create(aCollection); // set default values FPropertyName := ''; FDefaultValue := ''; end; function TSavedProperty.GetDisplayName: string; begin // return property name or components name if FPropertyName '' then Result := FPropertyName else Result := inherited GetDisplayName; end; function TSavedProperty.GetRegistryValue: String; begin // the registry value is created by the // component and property names Result := TSavedProperties(Collection).FSavedComponent. ComponentName + ':' + FPropertyName; end; procedure TSavedProperty.SetDefaultValue(const Value: String); begin FDefaultValue := Value; end; procedure TSavedProperty.SetPropertyName(const Value: String); var PropInfo: PPropInfo; TmpComponent: TComponent; CSR: TComponentStateRecorder; begin // set property name FPropertyName := Value; // set default value on-demand if FDefaultValue = '' then begin // get state recorder CSR := TSavedComponents( TSavedProperties(Collection).FSavedComponent.Collection ).FComponentStateRecorder; // at design-time only, load components current value as default if csDesigning in CSR.ComponentState then begin // load the named component (or form) if TSavedProperties(Collection).FSavedComponent.ComponentName = CSR.Owner.Name then TmpComponent := CSR.Owner else TmpComponent := CSR.Owner.FindComponent( TSavedProperties(Collection).FSavedComponent.ComponentName ); // check whether component was found if TmpComponent  nil then begin // get property information PropInfo := GetPropInfo(TmpComponent.ClassInfo, Value); // check whether property information where found if PropInfo  nil then // load current property value FDefaultValue := GetPropertyAsString( TmpComponent, PropInfo ); end; end; end; end; procedure TSavedProperty.SetRegistryValue(const Value: String); begin // ignore end; { TSavedProperties } function TSavedProperties.Add: TSavedProperty; begin Result :=
```

```
TSavedProperty(inherited Add); end; constructor
TSavedProperties.Create(aSavedComponent: TSavedComponent); begin inherited
Create(TSavedProperty); FSavedComponent := aSavedComponent; end; function
TSavedProperties.GetItem(Index: Integer): TSavedProperty; begin Result :=
TSavedProperty(inherited GetItem(Index)); end; function TSavedProperties.GetOwner:
TPersistent; begin Result := FSavedComponent; end; procedure
TSavedProperties.SetItem(Index: Integer; const Value: TSavedProperty); begin inherited
SetItem(Index, Value); end; procedure TSavedProperties.Update(Item: TCollectionItem);
begin inherited; // nothing to do end; { TSavedComponent } procedure
TSavedComponent.Assign(Source: TPersistent); begin if Source is TSavedComponent then
  begin // load values from source FComponentName :=
TSavedComponent(Source).FComponentName;
FSavedProperties.Assign(TSavedComponent(Source).SavedProperties); end else begin
inherited Assign(Source); end; end; constructor TSavedComponent.Create(aCollection:
TCollection); begin inherited Create(aCollection); FSavedProperties :=
TSavedProperties.Create(Self); end; destructor TSavedComponent.Destroy; begin if not
( csDesigning in
TSavedComponents(Collection).FComponentStateRecorder.ComponentState ) then begin
  // in desgin-time mode, the designer will free the objects for us FSavedProperties.Free;
  FSavedProperties := nil; end; inherited Destroy; end; function
TSavedComponent.GetDisplayName: string; begin if FComponentName '' then Result
:= FComponentName else Result := inherited GetDisplayName; end; procedure
TSavedComponent.SetComponentName(const Value: String); begin FComponentName :=
Value; end; procedure TSavedComponent.SetSavedProperties(const Value:
TSavedProperties); begin FSavedProperties.Assign(Value); end; { TSavedComponents
} function TSavedComponents.Add: TSavedComponent; begin Result :=
TSavedComponent(inherited Add); end; constructor TSavedComponents.Create(
aComponentStateRecorder: TComponentStateRecorder ); begin inherited
Create(TSavedComponent); FComponentStateRecorder := aComponentStateRecorder;
end; function TSavedComponents.GetItem(Index: Integer): TSavedComponent; begin
Result := TSavedComponent(inherited GetItem(Index)); end; function
TSavedComponents.GetOwner: TPersistent; begin Result := FComponentStateRecorder;
end; procedure TSavedComponents.SetItem( Index: Integer; const Value:
TSavedComponent ); begin inherited SetItem(Index, Value); end; procedure
TSavedComponents.Update(Item: TCollectionItem); begin inherited; // nothing to do
end; { TComponentStateRecorder } constructor
TComponentStateRecorder.Create(aOwner: TComponent); begin inherited
Create(aOwner); FSavedComponents := TSavedComponents.Create(Self); FRegistryKey
:= 'SoftwareYour SoftwareComponent State Recorder' + TForm(aOwner).Name; end;
 destructor TComponentStateRecorder.Destroy; begin FSavedComponents.Free;
FSavedComponents := nil; inherited Destroy; end; procedure
TComponentStateRecorder.DoStates(Action: TRecorderAction); var RegistryOpened:
Boolean; I, J: Integer; PropInfo: PPropInfo; TmpComponent: TComponent; SO:
TSavedComponent; SP: TSavedProperty; begin with TRegistry.Create do try //
generally save settings for the user! RootKey := HKEY_CURRENT_USER; // open the
registry key RegistryOpened := OpenKey(RegistryKey, True); try // iterate through
```

```
all Components to be saved for I := 0 to Pred(FSavedComponents.Count) do begin // get current component SO := FSavedComponents.Items[I]; // check, whether component name is set if SO.ComponentName = '' then Continue; // check, whether component is the owner form if SO.ComponentName = (Owner as TForm).Name then // use the owner forme TmpComponent := (Owner as TForm) else // find component on the owner form TmpComponent := (Owner as TForm).FindComponent( SO.ComponentName); // check component if TmpComponent = nil then // not found on form, check next in collection Continue; // iterate through all properties to be saved // (of current Component) for J := 0 to Pred(SO.SavedProperties.Count) do begin // get current property SP := SO.SavedProperties.Items[J]; // check, whether property name is set if SP.PropertyName = '' then Continue; // get property info pointer PropInfo := GetPropInfo( TmpComponent.ClassInfo, SP.PropertyName); // check for property if PropInfo = nil then // it does not exists, try next Continue; // registry access ? if RegistryOpened then // yes, save or load? if Action = raSave then // save WriteString( SP.RegistryValue, GetPropertyAsString( TmpComponent, PropInfo) ) else // load, does value exist? if ValueExists(SP.RegistryValue) then // yes, load SetPropertyFromString( TmpComponent, PropInfo, ReadString(SP.RegistryValue) ) else // no, get default SetPropertyFromString(TmpComponent, PropInfo, SP.DefaultValue) else // no registry access, in load mode? if Action = raLoad then // yes, load default SetPropertyFromString(TmpComponent, PropInfo, SP.DefaultValue); end; end; finally if RegistryOpened then CloseKey; end; finally Free; end; end; procedure TComponentStateRecorder.LoadStates; begin DoStates(raLoad); end; procedure TComponentStateRecorder.SaveStates; begin DoStates(raSave); end; procedure TComponentStateRecorder.SetRegistryKey(const Value: String); begin if Value = '' then FRegistryKey := 'SoftwareYour SoftwareComponent State Recorder' + TForm(Owner).Name else FRegistryKey := Value; end; procedure TComponentStateRecorder.SetSavedComponents( const Value: TSavedComponents ); begin FSavedComponents.Assign(Value); end; end.
```

Редактор свойств и компонентов

```
unit frmDesignTimeEditor; interface uses Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, ExtCtrls, Buttons, ComCtrls, ComponentStateRecovery, DsgnIntf, TypInfo; type // component editor for the TComponentStateRecorder class TCSRDesignEditor = class(TDefaultEditor) protected public function GetVerb(Index: Integer): String; override; function GetVerbCount: Integer; override; procedure ExecuteVerb(Index: Integer); override; end; // property editor that lists all properties // of a component at design-time TPropertyNameEditor = class(TStringProperty) public procedure GetValues(Proc: TGetStrProc); override; function GetAttributes: TPropertyAttributes; override; end; // property editor that lists all components at design-time TComponentNameEditor = class(TStringProperty) public procedure GetValues(Proc: TGetStrProc); override; function GetAttributes: TPropertyAttributes; override; end; TfrmCSRDesigner = class(TForm) Panel1:
```

```
TPanel; Label1: TLabel; edtRegKey: TEdit; Panel2: TPanel; btnOK: TBitBtn;
trvCollections: TTreeView; Panel3: TPanel; lblComponent: TLabel; cmbComponent:
TComboBox; grpProperty: TGroupBox; lblPropertyName: TLabel;
cmbPropertyName: TComboBox; lblDefaultValue: TLabel; edtDefaultValue: TEdit;
btnAddComponent: TButton; btnRemove: TButton; btnAddProperty: TButton;
procedure btnOKClick(Sender: TObject); procedure trvCollectionsChange(Sender: TObject;
Node: TTreeNode); procedure btnAddComponentClick(Sender: TObject);
procedure cmbComponentChange(Sender: TObject); procedure
edtRegKeyChange(Sender: TObject); procedure cmbPropertyNameChange(Sender:
TObject); procedure edtDefaultValueChange(Sender: TObject); procedure
btnAddPropertyClick(Sender: TObject); procedure btnRemoveClick(Sender: TObject);
private FCSR: TComponentStateRecorder; FDesigner: IFormDesigner; procedure
SetCSR(const Value: TComponentStateRecorder); procedure ShowProperties(Name:
String); procedure UpdateForSelectedNode; procedure SetDesigner(const Value:
IFormDesigner); public property CSR: TComponentStateRecorder read FCSR write
SetCSR; property Designer: IFormDesigner read FDesigner write SetDesigner; end;
var frmCSRDesigner: TfrmCSRDesigner; procedure Register; implementation {$R
*.DFM} procedure Register; begin // register component
RegisterComponents('gate(n)etwork', [TComponentStateRecorder]); // register property
editors (they will provide // drop-down lists to the OI) RegisterPropertyEditor(
TypeInfo(String), TSavedComponent, 'ComponentName', TComponentNameEditor );
RegisterPropertyEditor( TypeInfo(String), TSavedProperty, 'PropertyName',
TPropertyNameEditor ); // register component editor
RegisterComponentEditor(TComponentStateRecorder, TCSRDesignEditor); end; {
TCSRDesignEditor } procedure TCSRDesignEditor.ExecuteVerb(Index: Integer); begin
with TfrmCSRDesigner.Create(Application) do try Designer := Self.Designer; CSR :=
TComponentStateRecorder(Component); ShowModal; finally Free; end; end;
function TCSRDesignEditor.GetVerb(Index: Integer): String; begin if Index = 0 then
Result := 'Edit all recorded Properties...' else Result := ''; end; function
TCSRDesignEditor.GetVerbCount: Integer; begin Result := 1; end; {
TPropertyNameEditor } function TPropertyNameEditor.GetAttributes: TPropertyAttributes;
begin // the property editor will provide a sorted // list of possible values Result :=
[paValueList, paSortList]; end; procedure TPropertyNameEditor.GetValues(Proc:
TGetStrProc); var I, Count: Integer; PropInfos: PPropList; TmpComponent:
TComponent; SC: TSavedComponent; begin // check property type if not
(GetComponent(0) is TSavedProperty) then Exit; // get TSavedComponent (parent
object) SC := TSavedProperties( TSavedProperty(GetComponent(0)).Collection
).SavedComponent; // find the corresponding component if SC.ComponentName =
Designer.Form.Name then TmpComponent := Designer.Form else TmpComponent :=
Designer.GetComponent(SC.ComponentName); // quit if component was not found if
TmpComponent = nil then Exit; // determine the property count Count :=
GetPropList(TmpComponent.ClassInfo, [ tkInteger, tkInt64, tkFloat, tkEnumeration, tkSet,
tkChar, tkString, tkLString ], nil); // reserve memory needed for property informations
GetMem(PropInfos, Count * SizeOf(PPropInfo)); try // load property list
GetPropList(TmpComponent.ClassInfo, [ tkInteger, tkInt64, tkFloat, tkEnumeration, tkSet,
tkChar, tkString, tkLString ], PropInfos); // save to object inspector list for I
```

```
:= 0 to Pred(Count) do Proc(PropInfos^[I]^.Name); finally // free resources
FreeMem(PropInfos); end; end; { TComponentNameEditor } function
TComponentNameEditor.GetAttributes: TPropertyAttributes; begin // the property editor
will provide a sorted // list of possible values Result := [paValueList, paSortList]; end;
procedure TComponentNameEditor.GetValues(Proc: TGetStrProc); var I: Integer; begin
// return name of form if Designer.Form.Name  '' then Proc(Designer.Form.Name); //
return names of all components for I := 0 to Pred(Designer.Form.ComponentCount) do if
Designer.Form.Components[I].Name  '' then Proc(Designer.Form.Components[I].Name);
end; { TfrmCSRDesigner } procedure TfrmCSRDesigner.btnAddComponentClick(Sender:
TObject); var Node: TTreeNode; SC: TSavedComponent; begin SC :=
CSR.SavedComponents.Add; Node := trvCollections.Items.AddChild(nil, SC.DisplayName);
trvCollections.Selected := Node; Node.Data := SC; UpdateForSelectedNode;
Designer.Modified; end; procedure TfrmCSRDesigner.btnAddPropertyClick(Sender:
TObject); var Node: TTreeNode; SP: TSavedProperty; begin if trvCollections.Selected
= nil then Exit; if trvCollections.Selected.Data = nil then Exit; if not
(TObject(trvCollections.Selected.Data) is TSavedComponent) then Exit; SP :=
TSavedComponent(trvCollections.Selected.Data). SavedProperties.Add; Node :=
trvCollections.Items.AddChild(trvCollections. Selected, SP.DisplayName); Node.Data :=
SP; trvCollections.Selected := Node; UpdateForSelectedNode; Designer.Modified; end;
procedure TfrmCSRDesigner.btnOKClick(Sender: TObject); begin ModalResult := mrOK;
end; procedure TfrmCSRDesigner.btnRemoveClick(Sender: TObject); begin if
trvCollections.Selected = nil then Exit; if trvCollections.Selected.Data = nil then Exit;
if (TObject(trvCollections.Selected.Data) is TSavedComponent) then begin
TSavedComponent(trvCollections.Selected.Data). Collection.Delete(
TSavedComponent(trvCollections.Selected.Data).Index );
trvCollections.Items.Delete(trvCollections.Selected); end; if
(TObject(trvCollections.Selected.Data) is TSavedProperty) then begin
TSavedProperty(trvCollections.Selected.Data).Collection.Delete(
TSavedProperty(trvCollections.Selected.Data).Index );
trvCollections.Items.Delete(trvCollections.Selected); end; Designer.Modified; end;
procedure TfrmCSRDesigner.cmbComponentChange(Sender: TObject); begin if
trvCollections.Selected = nil then Exit; if trvCollections.Selected.Data = nil then Exit;
if not (TObject(trvCollections.Selected.Data) is TSavedComponent) then Exit;
TSavedComponent(trvCollections.Selected.Data).ComponentName :=
cmbComponent.Text; trvCollections.Selected.Text :=
TSavedComponent(trvCollections.Selected.Data).DisplayName; Designer.Modified; end;
procedure TfrmCSRDesigner.cmbPropertyNameChange(Sender: TObject); begin if
trvCollections.Selected = nil then Exit; if trvCollections.Selected.Data = nil then Exit;
if not (TObject(trvCollections.Selected.Data) is TSavedProperty) then Exit;
TSavedProperty(trvCollections.Selected.Data).DefaultValue := '';
TSavedProperty(trvCollections.Selected.Data).PropertyName := cmbPropertyName.Text;
trvCollections.Selected.Text := TSavedProperty(trvCollections.Selected.Data).DisplayName;
edtDefaultValue.Text := TSavedProperty(trvCollections.Selected.Data).DefaultValue;
Designer.Modified; end; procedure TfrmCSRDesigner.edtDefaultValueChange(Sender:
TObject); begin if trvCollections.Selected = nil then Exit; if
trvCollections.Selected.Data = nil then Exit; if not (TObject(trvCollections.Selected.Data)
```

```
is TSavedProperty) then Exit;
TSavedProperty(trvCollections.Selected.Data).DefaultValue := edtDefaultValue.Text;
Designer.Modified; end; procedure TfrmCSRDesigner.edtRegKeyChange(Sender: TObject);
 begin FCSR.RegistryKey := edtRegKey.Text; Designer.Modified; end; procedure
TfrmCSRDesigner.SetCSR(const Value: TComponentStateRecorder); var I, J: Integer;
SC: TSavedComponent; SP: TSavedProperty; SCNode, SPNode: TTreeNode; begin
FCSR := Value; // load registry key edtRegKey.Text := FCSR.RegistryKey;
trvCollections.Items.Clear; // parse all selected components for I := 0 to
Pred(FCSR.SavedComponents.Count) do begin SC :=
FCSR.SavedComponents.Items[I]; SCNode := trvCollections.Items.AddChild(nil,
SC.DisplayName); SCNode.Data := SC; // parse all selected properties for J := 0 to
Pred(SC.SavedProperties.Count) do begin SP := SC.SavedProperties.Items[J];
SPNode := trvCollections.Items.AddChild(SCNode, SP.DisplayName); SPNode.Data :=
SP; end; end; // select the first item in the list if trvCollections.Items.Count > 0 then
 trvCollections.Selected := trvCollections.Items.Item[0]; if Designer  nil then begin //
return name of form if Designer.Form.Name  '' then
cmbComponent.Items.Add(Designer.Form.Name); // return names of all components for
I := 0 to Pred(Designer.Form.ComponentCount) do if Designer.Form.Components[I].Name
'' then cmbComponent.Items.Add(Designer.Form.Components[I].Name); end; //
show state of selection UpdateForSelectedNode; end; type TEnableStates =
(esComponent, esProperty); TEnableStateSet = set of TEnableStates; procedure
TfrmCSRDesigner.SetDesigner(const Value: IFormDesigner); begin FDesigner := Value;
end; procedure TfrmCSRDesigner.ShowProperties(Name: String); var I, Count: Integer;
 PropInfos: PPropList; TmpComponent: TComponent; begin // clear list
cmbPropertyName.Clear; // stop if no component name is provided if Name = '' then
Exit; // get component if CSR.Owner.Name = Name then TmpComponent :=
CSR.Owner else TmpComponent := CSR.Owner.FindComponent(Name); // stop if
component was not found if TmpComponent = nil then Exit; // determine the property
count Count := GetPropList(TmpComponent.ClassInfo, [ tkInteger, tkInt64, tkFloat,
tkEnumeration, tkSet, tkChar, tkString, tkLString ], nil); // reserve memory needed
for property informations GetMem(PropInfos, Count * SizeOf(PPropInfo)); try // load
property list GetPropList(TmpComponent.ClassInfo, [ tkInteger, tkInt64, tkFloat,
tkEnumeration, tkSet, tkChar, tkString, tkLString ], PropInfos); // save to
object inspector list for I := 0 to Pred(Count) do
cmbPropertyName.Items.Add(PropInfos^[I]^.Name); finally // free resources
FreeMem(PropInfos); end; end; procedure
TfrmCSRDesigner.trvCollectionsChange(Sender: TObject; Node: TTreeNode); begin
UpdateForSelectedNode; end; procedure TfrmCSRDesigner.UpdateForSelectedNode;
var CompName, PropertyName: String; EnableStates: TEnableStateSet; begin
EnableStates := []; Name := ''; if trvCollections.Selected  nil then if
trvCollections.Selected.Data  nil then begin if TObject(trvCollections.Selected.Data) is
 TSavedComponent then begin cmbComponent.Text :=
TSavedComponent(trvCollections.Selected.Data).ComponentName; EnableStates :=
EnableStates + [esComponent]; cmbPropertyName.Text := ''; edtDefaultValue.Text
:= ''; trvCollections.Selected.Text :=
TSavedComponent(trvCollections.Selected.Data).DisplayName; CompName := '';
```

```
PropertyName := '';  end;  if TObject(trvCollections.Selected.Data) is
TSavedProperty then  begin  EnableStates := EnableStates + [esProperty];
CompName :=  TSavedProperties(TSavedProperty(
trvCollections.Selected.Data  ).Collection).SavedComponent.ComponentName;
cmbComponent.Text := CompName;  PropertyName :=
TSavedProperty(trvCollections.Selected.Data).PropertyName;  cmbPropertyName.Text :=
Name;  edtDefaultValue.Text :=
TSavedProperty(trvCollections.Selected.Data).DefaultValue;  trvCollections.Selected.Text
:=  TSavedProperty(trvCollections.Selected.Data).DisplayName;  end;  end;
cmbComponent.Enabled := esComponent in EnableStates;  lblComponent.Enabled :=
esComponent in EnableStates;  btnAddProperty.Enabled := esComponent in EnableStates;
cmbPropertyName.Enabled := esProperty in EnableStates;  lblPropertyName.Enabled :=
esProperty in EnableStates;  edtDefaultValue.Enabled := esProperty in EnableStates;
lblDefaultValue.Enabled := esProperty in EnableStates;  grpProperty.Enabled := esProperty in
EnableStates;  btnRemove.Enabled := EnableStates  [];  ShowProperties(CompName);
cmbPropertyName.Text := PropertyName;  trvCollections.Update;  end;  end.
```

Daniel Wischnewski, Content Ace