SuperForms

How to Build Forms with Custom Properties and Events in Delphi™

Eagle Software

What are SuperForms?

SuperForms are standard Delphi forms with custom properties and events that can appear in the Object Inspector at design-time. SuperForms are registered with Delphi, so you can create an instance of the form just as easily as you can a normal form.

Advantages

Custom, Enhanced, and Hidden Properties

SuperForms can declare their own properties. SuperForms can enhance existing enumerated properties, adding new elements to the enumerations. They can also hide published properties.

Custom Events

SuperForms can declare their own events.

Standardization

In addition to having custom properties and events, SuperForms can also have “hard-coded” components placed on the form in advance. These components can behave like standard Delphi components at design-time (the form is the owner), or they can appear to be fused directly into the form, behaving as if it were runtime (the form is not the owner and so they must consequently be destroyed in the SuperForm's destructor). SuperForms can also include standard comments and file headers, as well as design-time "smart code", which can assist application designers in their tasks and keep form designs among team members consistent.

Disadvantages

There is one disadvantage to SuperForms: they must be declared and you have to write some code.

Before We Start

Building custom forms in Delphi has historically been an advanced topic; one that requires a great deal of coding beyond the simple property or event declaration associated with building components. Part of the intent of this paper and session is to provide a general overview of the process, but to also shield you from as much of the complexity as possible. This way you can focus on the properties, events and any custom code you want to add to your SuperForm. This shielding is accomplished with two base classes, a runtime package, and a Delphi expert that generates the registration code and the form creator. All that remains is for you to declare your custom form (about as complex as creating a new component) and to follow a few simple steps.

How Delphi Supports Custom Forms

Delphi allows you to register custom forms with the RegisterCustomModule method. This method binds a reference to your custom form class with a reference to a custom module creator. Once registered, Delphi uses the class reference whenever it needs to create a new instance of your custom form (e.g., when Delphi loads a custom form and its components at design-time).

Developers will need to create new instances of the custom form in the process of normal application development. Delphi's expert mechanism works well in this case, providing a means to create new instances of SuperForms at design time.

But you don't need to worry about creating a wizard just to create a new instance of your SuperForm. The wizard that comes with the code for this session will create the wizard for you (it's a Wizard Wizard). All you'll need to do is create your custom form, and follow a few easy steps.

SuperForm Architecture

SuperForms consist of design-time (DT) and runtime (RT) elements. The breakdown appears below:

Design Time Only	**Runtime (and Design Time)**
Custom Form Wizard Custom Form Registration Custom Form Property Editors	Your Custom Form Code

Custom Form Wizard (DT)

This is a standard Delphi form wizard that creates an instance of your custom form.

Custom Form Registration (DT)

Registers the custom form, the form wizard, and any custom property editors.

Custom Form Property Editors (DT)

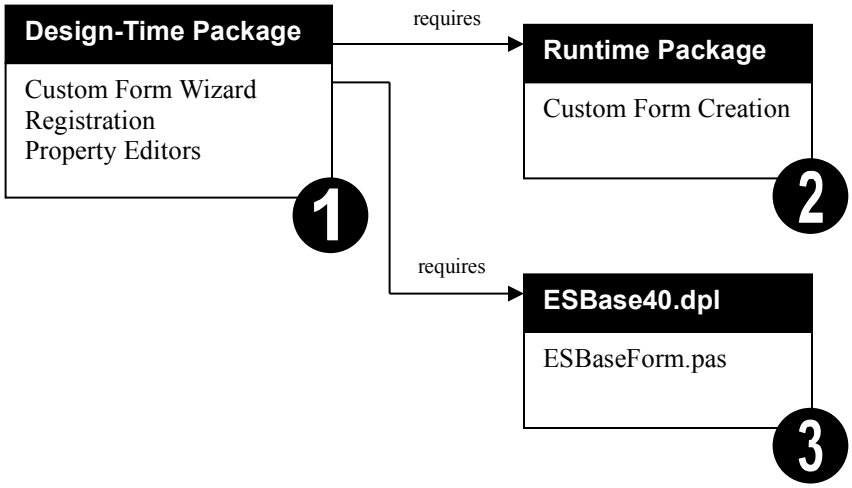
Just like you can register a custom property editor for properties of components, you can also register custom property editors for properties of your SuperForm.

Custom Form Code (RT)

The SuperForm's declaration and implementation go in your runtime package.

The design-time only code is placed inside a design-time package, while the runtime code is placed inside a runtime package. The design-time package requires your custom runtime package, and also requires the runtime package included with this paper (ESBase40 for Delphi 4, and ESBase30 for Delphi 3).

The custom form architecture discussed so far looks like this:



Notes:

1. The included SuperForm wizard from Eagle Software handles creating the Registration code and the Custom Form Wizard (that's right, it's a wizard that creates a wizard). All you need to do is create a new design-time package, run the wizard and include the registration unit and runtime package.
2. Contain your custom form declaration unit inside a new runtime package. This package is required by the design-time package.
3. The design-time package contains the runtime package ESBase40 (for Delphi 4, or ESBase30 for Delphi 3). This package is included with this session.
4. The ESBase40 run-time package contains the file **ESBaseForm.pas**. Although full source is included for this file, you should not modify it. The classes within this file are built for reuse through inheritance; if you want to add or change functionality, then just create a descendant class. The contents of ESBaseForm.pas are described below.

The Included Base Classes

The included file **ESBaseForm.pas**, declares TIModuleCreator and TIExpert descendant classes, both providing default method overrides to reduce the amount of code required in building custom forms. This file also includes the source code that will be generated for new custom forms. This source is in the form of a resource string. This file is compiled inside the included package **ESBase40.bpl** (for Delphi 4) and **ESBase30.dpl** (for Delphi 3).

The **TIModuleCreator descendant** overrides and implements six methods and declares two new virtual methods, only one of which must be overridden.

The **TIExpert descendant** overrides and implements five methods and declares two new virtual methods, both of which must be overridden.

Note however, that the SuperForm wizard writes code for both of these descendant classes, so you won't have to perform any manual coding here (in other words, you won't have to override these methods).

Installing the Included Source and Wizard

Installation is made easy via a **setup.exe** install program. The installation program copies the SuperForm wizard and sample files to a directory of your choosing. It also makes Delphi aware of the wizard. To install the source and wizard, follow these steps:

1. Quit Delphi if it is already running.
2. Run the setup.exe program.

3. Follow the onscreen prompts.
4. Start Delphi.

Getting Started

That's it for the technical overview and source installation. It's time to build some SuperForms! This section will take you through the steps of building a custom form.

The steps are broken down into five parts, which include:

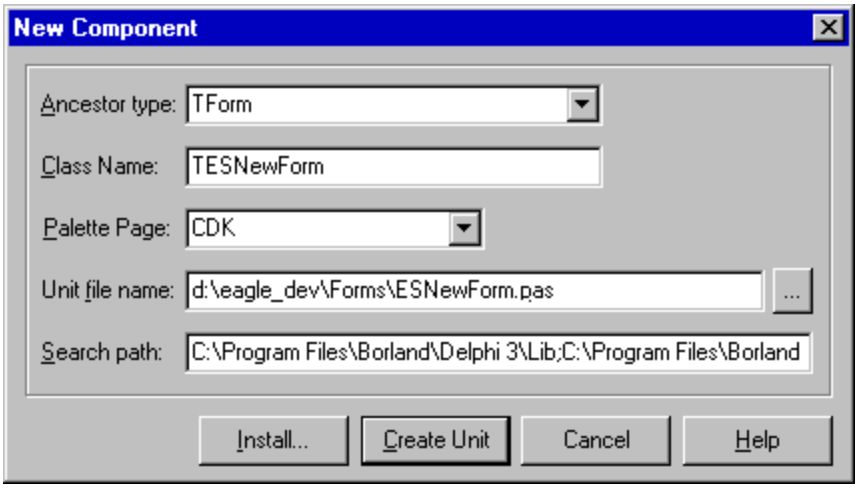
Part I – Declaring the New Form Class
Part II – Creating the Runtime Package
Part III – Using the Wizard to Build the Registration Unit
Part IV – Creating an Icon for the New SuperForm
Part V – Creating the Design-Time Package

Part I – Declaring the New Form Class

Declaring custom form classes is similar to declaring a new component. In fact, you should be familiar with component building and object oriented programming before continuing.

1. From Delphi's "**File**" menu select "**New…**". Select the **Component** icon and click **OK**.

We're going to let Delphi's component expert as a shortcut to create some code for us. Delphi's **New Component** dialog will appear:



2. Select "TForm" as the ancestor type, and specify a new unique class name that describes your form. Leave the Palette Page as is (we'll be removing it's associated code in the next step). Specify a target directory where you'll keep your custom form source. Click the **Create Unit** button. Delphi will generate following code:

```
unit ESNewForm;

interface
```

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs;

type
  TESNewForm = class(TForm)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
  end;

procedure Register;

implementation

procedure Register;
begin
  RegisterComponents('CDK', [TESNewForm]);
end;

end.
```

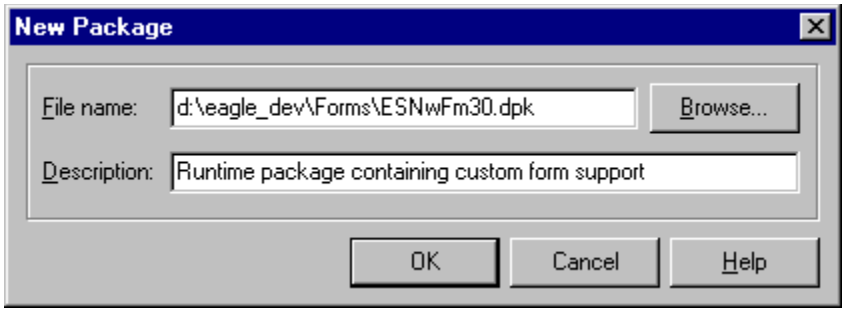
3. Remove the **Register** procedure (both declaration and implementation).
4. Add any properties, events, or methods, just like you would if you were building a component. The sample file ESCustomFrm.pas includes examples of adding custom properties and events.
5. Save the file.

Part II – Creating the Runtime Package

The next step is to create the runtime package that will contain our custom form source. To do it, follow these steps:

1. From Delphi's "**File**" menu select "**New…**". Select the Package icon and click **OK**.

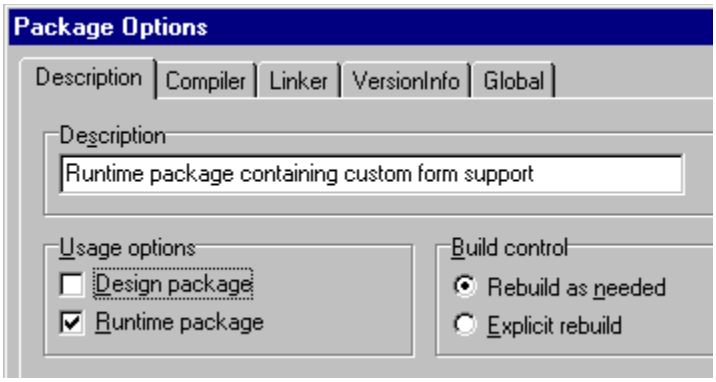
Delphi's **New Package** dialog will appear:



2. Specify a unique package name. It is recommended that the package name be eight characters or less, and include a version number. Both of these recommendations prevent problems that may occur later on when this runtime package is distributed as part of an application built with runtime packages. Note that the version number should match Delphi's, so if you're running Delphi 3 the version number

should be "30" (see the screen shot above). The directory where this package is placed should be the same one containing your custom form created in Part I, above.

3. Delphi will create a new package. By default, new packages in Delphi 3 are design-time, and in Delphi 4 they're both design-time and runtime. We'll need to change this to runtime only. Click the **Package Editor**'s **Options** button to bring up the Package Options dialog:



4. Change the configuration settings so that the package is runtime only (in Delphi 3, clear the **Design package** checkbox and select the **Runtime package** checkbox; in Delphi 4 select the **Runtime only** radio button). Click **OK**.
5. This runtime package will contain our new custom form. To add it, click the **Package Editor**'s **Add** button and enter the name of the custom form unit. This should be in the same directory as the runtime package.
6. Save the package and compile.

Part III – Using the Wizard to Build the Registration Unit

The next step is to create the registration unit, which registers the custom form with Delphi and also creates and registers an expert that can generate new custom forms. To use the wizard, follow these steps:

1. From Delphi's "File" menu select "New…". Click the Experts tab (you may need to scroll to the right to see it). Select the "Eagle Software Super Form Wizard" icon and click OK.

The **Super Form Wizard** dialog will appear. It looks like this:

2. Fill out the fields in the dialog. Hints appear to the right describing each field. Complete source to this wizard is included as part of the installation.
3. Click OK. The wizard will generate code and open the new registration unit. The registration unit code looks like this (results will vary depending on the values entered into the Super Form Wizard):

```
unit Unit1;

{ Registration unit for a SuperForm

At this point, the following steps should already be completed:
  1. A descendant of TForm with custom properties has been declared in a
     separate unit (ESNewForm) and saved.

  2. A runtime package has been created and compiled. The runtime package
     contains ESNewForm.pas, and requires the ESBase30 package.

Remaining steps:
  1. Save this file. It is convention to save registration units with a
     "Reg" suffix (e.g., "ESNewFormReg.pas").

  2. Create a 32-bit resource file containing a single icon with 32x32
     and 16x16 images. This resource file should have the same name as
     this unit with a *.RES extension (e.g., "ESNewFormReg.res"). This icon
     appears in Delphi's "New Items" dialog.

  3. Specify the name of this icon in the GetGlyph method, below.

  4. Create a new design-time package.

  5. Add this file to the new package's Contains list.

  6. Require your custom form's runtime package.

  7. Compile and install the new package.
}

interface

procedure Register;

implementation

// Create a 32-bit resource file containing an icon for this expert:
{$R *.RES}

uses
  Forms, ESBaseForm, Windows, SysUtils, DsgnIntf, ExptIntf, ToolIntf,
  EditIntf, ESNewForm;
```

```
resourcestring
  sAuthor     = 'Mark Miller';
  sComment    = 'Copyright (c) 1998, Eagle Software Inc.';
  sExpertName = 'New Form';


type
  TESNewFormCreator = class(TESBaseFormCreator)
  public
    function GetFormAncestorUnitName: string; override;
    function GetAncestorName: string; override;
  end;          { TESNewFormCreator }

{ TESNewFormCreator }
function TESNewFormCreator.GetAncestorName: string;
begin
  result := 'ESNewForm';              // Descends from TESNewForm
end;            { GetAncestorName }

function TESNewFormCreator.GetFormAncestorUnitName: string;
begin
  result := 'ESNewForm';
end;            { GetFormAncestorUnitName }


{ Expert interface class to create new instances of TESNewForm: }
type
  TESNewFormExpert = class(TESBaseCustomFormExpert)
  public
    function GetFormCreatorClass: TESBaseFormCreatorClass; override;
    function GetAuthor: string; override;
    function GetComment: string; override;
    function GetIDString: string; override;
    function GetName: string; override;
    function GetGlyph: HICON; override;
  end;          { TESNewFormExpert }

{ TESNewFormExpert }
function TESNewFormExpert.GetFormCreatorClass: TESBaseFormCreatorClass;
begin
  result := TESNewFormCreator;
end;            { GetFormCreator }

function TESNewFormExpert.GetAuthor: string;
begin
  result := sAuthor;
end;            { GetAuthor }

function TESNewFormExpert.GetComment: string;
begin
  result := sComment;
end;            { GetComment }

function TESNewFormExpert.GetGlyph: HICON;
begin
  result := LoadIcon(hInstance, 'EnterNameOfIconHere');
end;            { GetGlyph }

function TESNewFormExpert.GetIDString: string;
begin
  result := 'Eagle Software.New Form Creator';
end;            { GetIDString }

function TESNewFormExpert.GetName: string;
begin
  result := sExpertName;
end;            { GetName }


{ Declare property editors for custom form properties, if necessary... }
```

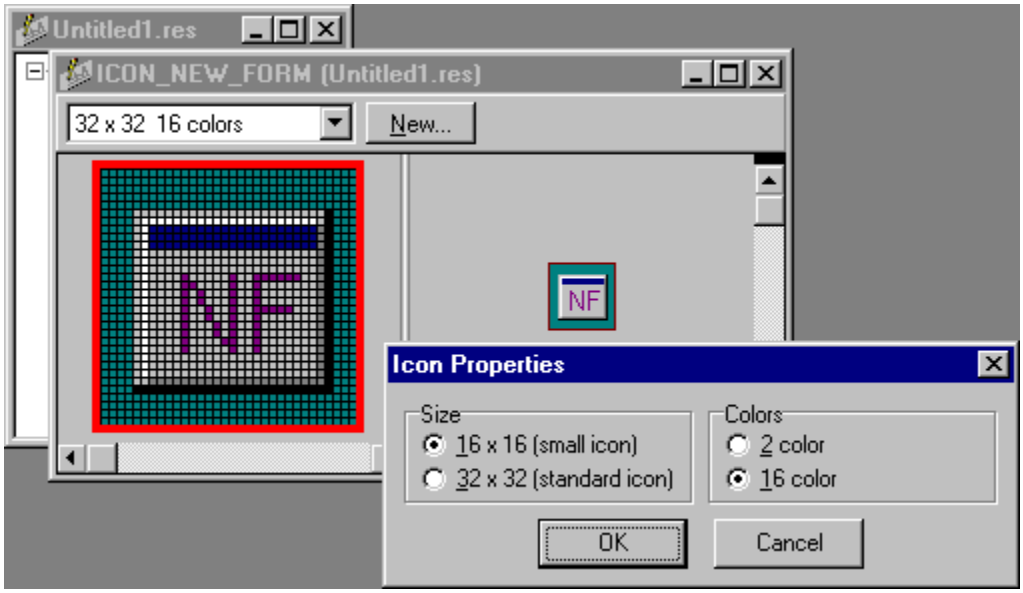
```
procedure Register;
begin
  RegisterCustomModule(TESNewForm, TCustomModule);
  RegisterLibraryExpert(TESNewFormExpert.Create);
  { Register custom property editors, if necessary... }
end;           { Register }

end.
```

4. Save this file under a new unique name. Since this is a registration file, it is customary to use the "Reg" suffix in the file name (e.g., "ESNewFormReg.pas").

Part IV – Creating an Icon for the New SuperForm

The next step is to create an icon for the SuperForm wizard. Just use your favorite resource editor. Be sure to create both 16x16 and 32x32 images. Note: If you want to skip this step and use the default icon, just return zero in the GetGlyph method inside the registration unit, and comment-out the "{$R *.RES}" line in the same file.



After creating the images, follow these steps:

1. Give the icon resource a unique and meaningful name (e.g., "ICON_NEW_FORM"). This name will be pasted into the code generated by the SuperForm wizard.
2. Save the resource file using the same name as the registration unit, substituting *.res for *.pas. Using the registration unit name from the previous part, we would call this file "ESNewFormReg.res".
3. Change the GetGlyph method inside the registration unit built in the last section, replacing "EnterNameOfIconHere" with the name of the icon. For example, the code might look like this:

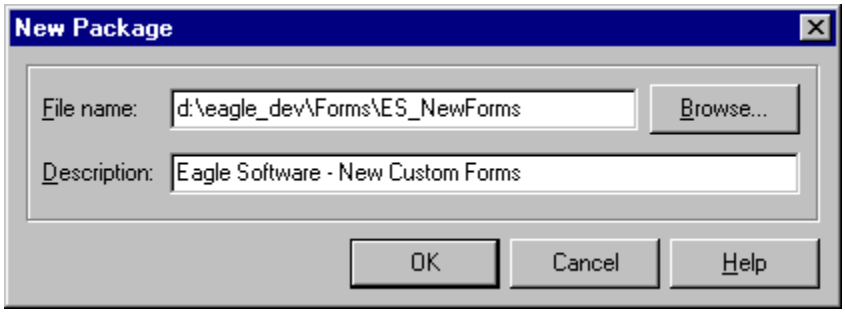
```
function TESNewFormExpert.GetGlyph: HICON;
begin
  result := LoadIcon(hInstance, 'ICON_NEW_FORM');
end;           { GetGlyph }
```

Part V – Creating the Design-Time Package

The last part in creating a custom form is to build the design-time package that contains our new form registration code. To create the design-time package, follow these steps:

1. From Delphi's "**File**" menu select "**New…**".
2. Select the Package icon and click **OK**.

Delphi's **New Package** dialog will appear.



3. Enter the name of your design-time package. Unlike the previous runtime package, this package name does not require a version number, and does not need to be eight characters or less.
4. Enter a meaningful description. This description will appear in the list of installed packages. Click OK. You will see the **Package Editor**.
5. Click the **Package Editor**'s **Add** button. Contain the registration unit created in Part III.
6. Click the **Package Editor**'s **Requires** tab. Click the **Add** button. Require the runtime package created in Part II and click **OK**.
7. Click the **Add** button a second time. If you're in Delphi 3, specify ESBase30 as the name of the package to require; for Delphi 4 specify ESBase40.
8. Compile and install the new package.

Testing SuperForms

To test your new SuperForm, follow these steps:

1. Create a new application.
2. From Delphi's "File" menu select "New…". Click on the Forms tab.
3. Select your new SuperForm. Note: If the glyph you designed does not appear in the list, then check to make sure you entered the correct name in the GetGlyph method (the last step of Part IV).
4. Click OK to create a new instance of your form. Change the new properties, right-click the form and select "View as Text". Confirm that the properties are streaming out correctly. Right-click the editor and select "View as Form". Confirm that the properties made a correct round trip.
5. Click the new form and then click on the Object-Inspector.
6. If you hid any form properties, verify that they do not appear in the Object Inspector.
7. Click the Events tab of the Object Inspector. Double-click the new events to create event handlers. Call ShowMessage from inside the event handlers (or implement another call to provide some kind of indication that the event is being triggered).
8. Run the application. Verify that the new form events are being triggered properly.

Summary

SuperForms promote standardization and allow you to completely customize Delphi forms, adding properties and events as needed. The base classes and wizard included with this session make it easy to create custom forms, allowing you to concentrate on the custom form itself. SuperForm technology can make a significant contribution towards reducing common tasks and speeding application development.

About the Author

Mark Miller is the lead software engineer at Eagle Software. Mark holds the primary vision for CodeRush, the Component Developer Kit (the CDK), reAct, a component testing tool for Delphi, as well as several other Delphi add-ons currently in development. He has been programming in Pascal, Object Pascal, and Delphi for nearly two decades.

Mark can be reached at markm@eagle-software.com.