Control Array **Sample Help**

Sample Description: Control Array

Points of Interest

- Creating a Control Array
- Adding a Control Array to your Form
- Changing the Parent Object

Control

- Button

For Help on Help, Press F1

Control Array

A control array is a group of controls derived from a common "parent" object. The term "control array" is used by other applications on the market that define a control array as being a group of controls sharing the same name. These applications are not true object-oriented systems and do not support object inheritance. Therefore, they rely on the ability to share a control's name between controls and then "index" each instance by a number. This is their approach to reducing redundant code and creating copies of controls.

Since Envelop is a true "object-oriented" system. Object-Oriented Programming is the implementation of an Object-Oriented Design using a language that supports the object-oriented style. A true object-oriented programming language such as Envelop, exhibits three components:

1. Encapsulation of data and program into one object
2. Inheritance of characteristics from object or class to object
3. Polymorphism

Object-Oriented Programming uses objects, not algorithms, for constructing its basic building blocks. Every object is an instance of some class. Classes are related to each other via. inheritance relationships. Inheritance is the key concept by which a class can be defined as a special case of a more general class. The new class inherits all the original class members. The new class is a subclass that is derived from a base class. In addition to inheriting the members of the base class, the subclass can define its own members (i.e., methods, properties, events, etc.) or override the inherited characteristics of the original class.

The purpose of this sample, is to demonstrate the concepts behind inheritance and compare this concept to traditional notions of control arrays. In the sample, there are several buttons displayed on a form. Each button is an instance of a parent button. Clicking one of the checkboxes displayed at the bottom of the form will change one of the properties of the parent object. Since the buttons shown on the form were derived from the parent object, their properties are automatically updated when a parent's property, event, or method is changed.

Creating a Control Array

To create a Control Array in Envelop simply create a copy of an existing control, and then specialize it in some manner. In this example, the standard Envelop object **Button** is copied, and named **ArrayButton**. To specialize this button, the following Click event was added:

```
Sub Click()
   InfoBox.Msg(Caption)
End Sub
```

Each time you click on an ArrayButton, a message box will appear on the screen and display the button's Caption property. This is known as "specializing" the object.

After the copy called ArrayButton was created and specialized it was added to the form.

Adding a Control Array to your Form

Adding a control array to your form is simply a matter of dragging the parent object (ArrayButton) to the Control Palette, then clicking on it to create instances of the ArrayButton on the form. To do this, drag the ArrayButton object from the Object Browser to the Controls Palette and assign it to the "programmable control" button. This button is designed to be used as a location to which temporary controls can be assigned. The final step was to use the Left Mouse Button, click the programmable control and draw several ArrayButton control on the form. As each one is created, it is assigned a unique name by having it's number incremented. For example, ArrayButton1, ArrayButton2, etc.....

In an application development environment, it is very common to create several specialized controls and also create your own palettes to assign permanent control access. Since the Envelop user interface was created from its own objects and is entirely configurable, you can build new objects and palettes to contain them as part of your development environment.

Changing the Parent Object

Since each of the ArrayButton's on the form were derived from the parent ArrayButton object, they each have the specialized Click method. Therefore clicking on any of these buttons will display a message containing the button's Caption property.

This is a simple example of how each of the objects inherent their parent's methods. If you were to change the parent's Click method to the following:

```
Sub Click()
  InfoBox.Msg("I am button " & Caption)
End Sub
```

Each of the buttons on the test form, when clicked on, would display the new message. Another example of inheritance can be illustrated through one of the form's properties. In the sample, the form's Font property references a **font** object that was embedded as one of the form's objects. Each of the ArrayButtons inherits its font from the form unless it has a specific font reference.

At the bottom of the test form, there are four checkboxes. The checkboxes change various font properties of the **font** object on the form. When this changes, the ArrayButtons automatically inherit this change. If the parent object ArrayButton, were to include an embedded font object and then reference that object as its font source, then all instances of ArrayButton would automatically inherit that change.

It is possible for each instance of the ArrayButton to be further specialized to contain its own font reference. In this case, changes to the parent object would not be reflected in the specialized instances.