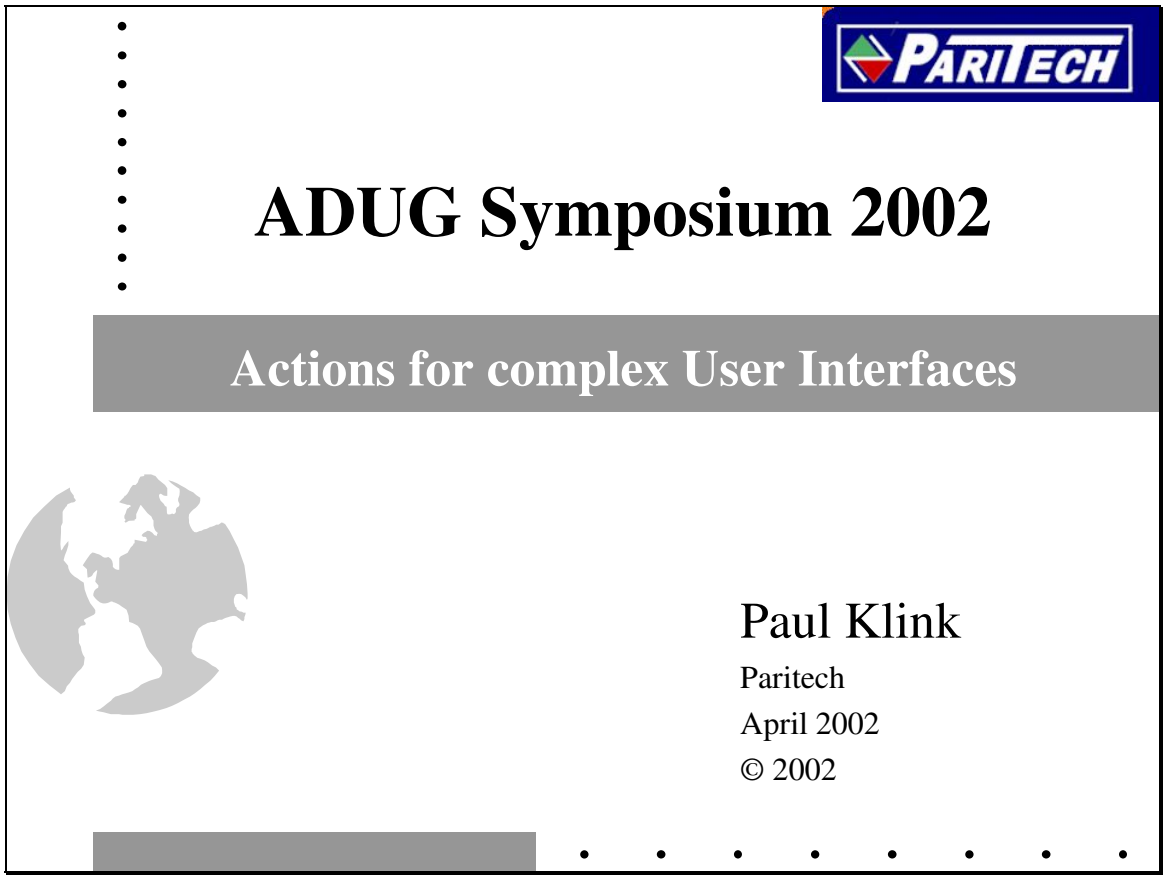
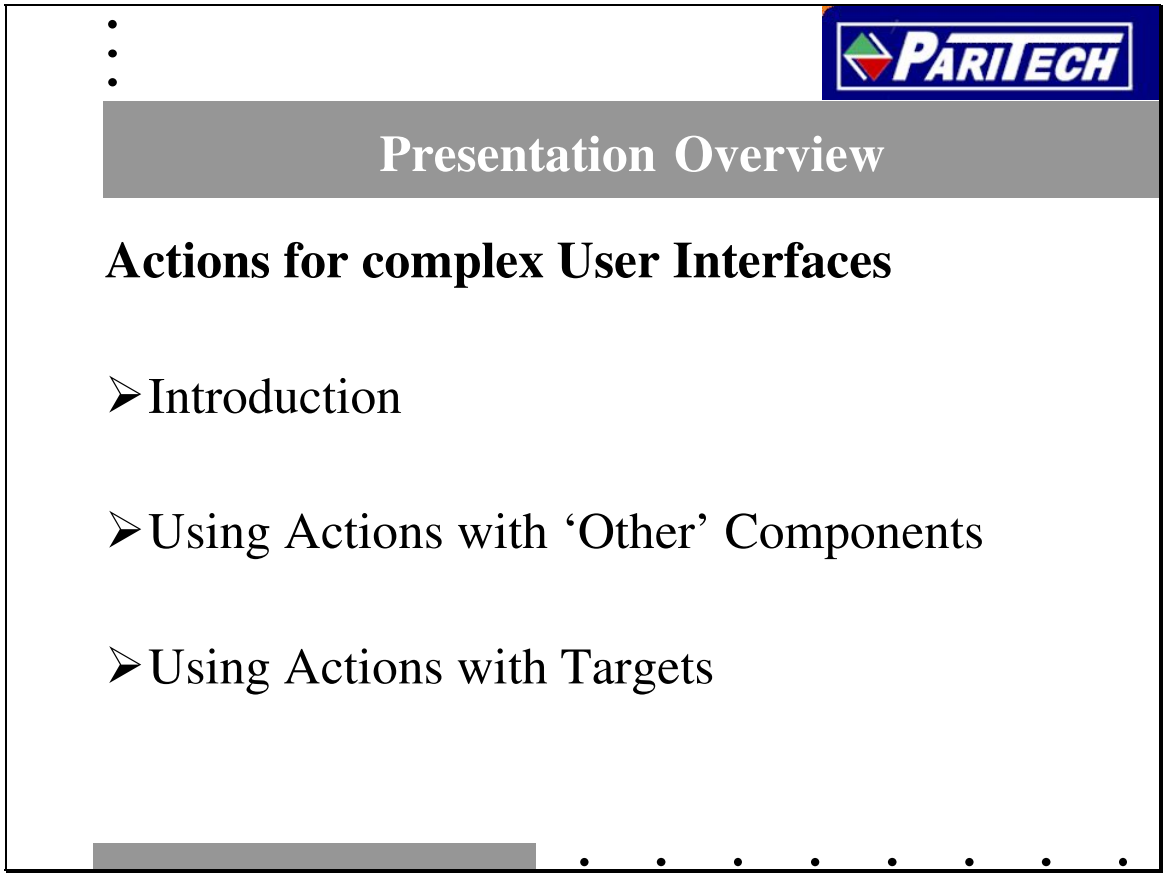
Slide 1



Slide 2



Slide 3

Standard usage of Actions

- Easy to use where supported by existing components
- Components can share the same Application Logic
- Similarities between Actions and Database TDataSource components

Slide 4

Sophisticated User Interfaces

- Actions are used in Paritech Charting App
- Needed to reduce the complexity of main Frame
- Significant amount of code applies to Application logic.
- This code mostly associated with components that don't support Actions
- Can Actions be used with these 'other' components?
- YES!

Slide 5

Actions & Component Development

- Using standard Actions with components that support Actions, requires little knowledge of component creation
- In order to make Custom Actions and Components to use these actions, some knowledge of Component Creation is required.
- We will approach this from the Application Developers point of view.
- Components we create will be application specific and not designed for reuse.

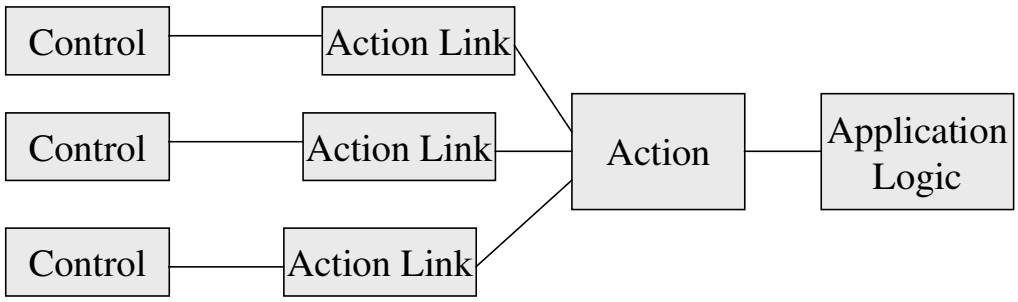
Slide 6

The ColorGradient demo project

- Custom Action/Component Demo; ‘evolves’ a project which displays a color gradient
- Focus is on demonstrating technique
- Last evolution may warrant Custom Actions

Slide 7

Actions and ActionLinks and Controls/Clients



- Actions link to Controls via Action Links
- Action interacts with Application Logic
- ActionLinks broadcast changes from Action to Controls

Slide 8

Adding a custom Action

- ColorGradient1 project
- Easiest to derive from TAction. Could use TCustomAction if you wanted to remove unwanted published items

Slide 9

```
TColorGradientAction = class(TAction)
private
  FImage: TImage;
  FSelectedPoint: TPoint;
  FSelectedColor: TColor;
  FOnSelectPoint: TNotifyEvent;

  function ColorFromPoint(const Point: TPoint): TColor;
  function PointFromColor(Color: TColor): TPoint;

  procedure SetSelectedColor(const Value: TColor);
  procedure SetSelectedPoint(const Value: TPoint);

  procedure UpdateClients;

public
  constructor Create(AOwner: TComponent); override;
  property SelectedPoint: TPoint read FSelectedPoint
                                 write SetSelectedPoint;
  property SelectedColor: TColor read FSelectedColor
                                 write SetSelectedColor;

published
  property Image: TImage read FImage write FImage;
  property OnSelectPoint: TNotifyEvent read FOnSelectPoint
                                       write FOnSelectPoint;
  property OnExecute stored False;
end;
```

Slide 10

PARITECH

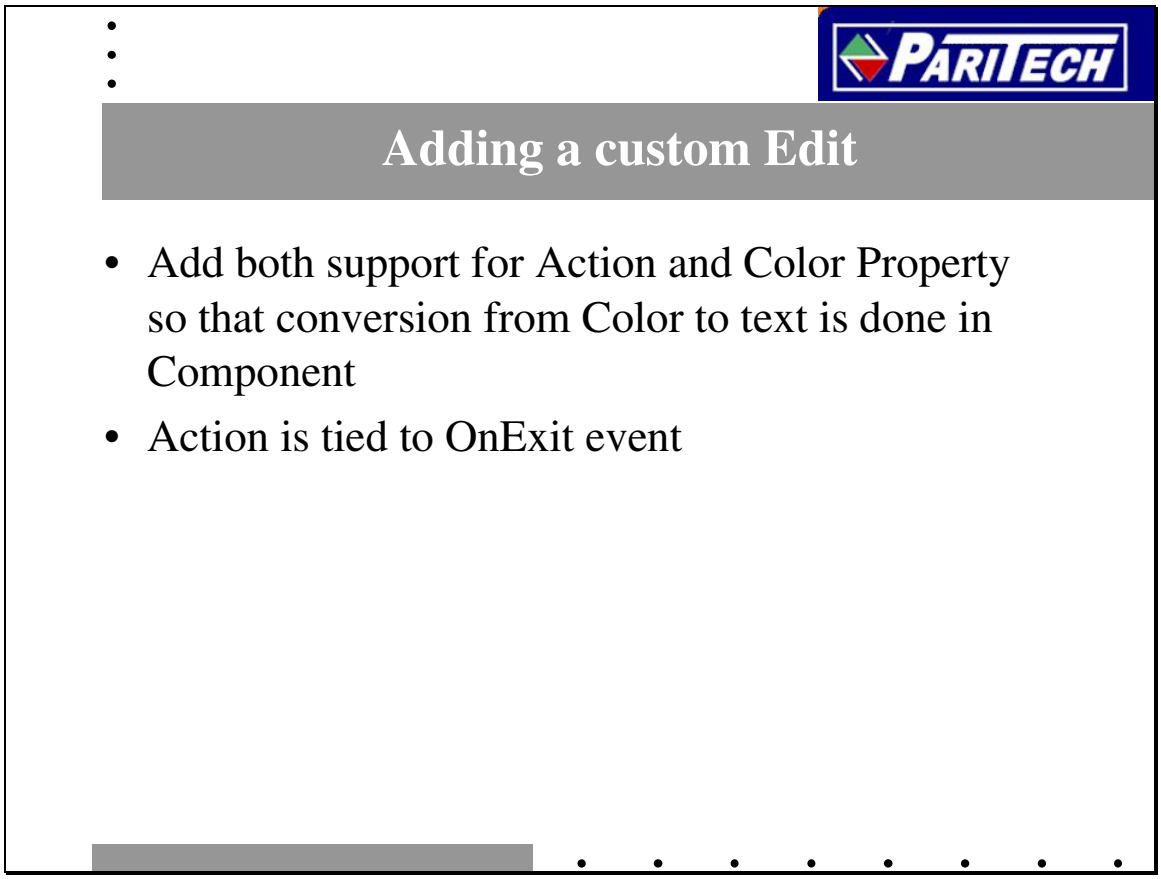
Adding a custom Action Link

- Class hierarchy of ActionLink components, tends to reflect hierarchy of controls

```
TGradientColorEditActionLink = class(TControlActionLink)
protected
  function IsOnExecuteLinked: Boolean; override;
  procedure SetColor(Value: TColor);
end;
```

- IsOn functions can be used to determine what ancestor properties are broadcast to controls

Slide 11



Slide 12

```
const
  DefaultColorEditRGBSelect = rgbRed;
  DefaultColorEditColor = 0;

type
  TColorEdit = class(TEdit)
  private
    FColor: TColor;
    FRGBSelect: TRGBSelect;
    procedure SetRGBSelect(const Value: TRGBSelect);

  protected
    procedure DoExit; override;
    procedure Loaded; override;

    procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); override;

  public
    constructor Create(AOwner: TComponent); override;
    function GetActionLinkClass: TControlActionLinkClass; override;
    procedure SetColor(const Value: TColor);

  published
    property Action;

    property RGBSelect: TRGBSelect read FRGBSelect write SetRGBSelect
                                  default DefaultColorEditRGBSelect;
    property Color: TColor read FColor write SetColor default DefaultColorEditColor;

    property Text stored False;
  end;
```

Slide 13

Registering Custom Action and Custom Edit

```
procedure Register;
begin
  RegisterComponents('ColorGrad', [TColorEdit]);
  RegisterActions('ColorGrad',
                  [TColorGradientAction], nil);
end;
```

- Can now Drop Action and Edit on project using IDE

Slide 14

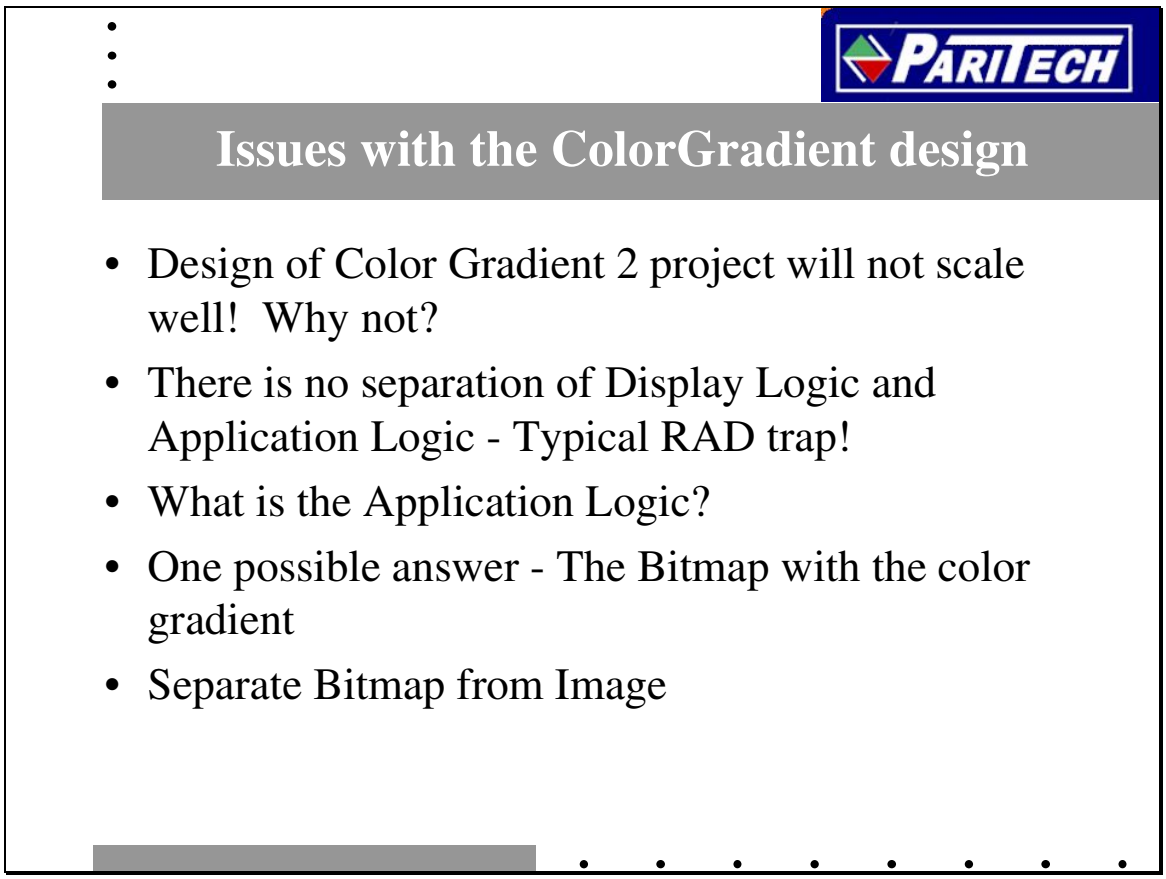
Adding a Scroll Bar

- ColorGradient2 project
- Action is tied to Scroll event
- Added properties to CustomAction to support Custom ScrollBar (MaxRedColor, MaxBlueColor)
- Modified UpdateClients to support new Client

Slide 15

```
TGradientColorScrollBarActionLink = class(TControlActionLink)
protected
  function IsOnExecuteLinked: Boolean; override;
  procedure SetColor(Value: TColor);
end;

TColorScrollBar = class(TScrollBar)
private
  FColor: TColor;
  FRGBSelect: TRGBSelect;
  procedure SetRGBSelect(const Value: TRGBSelect);

protected
  procedure Scroll(ScrollCode: TScrollCode; var ScrollPos: Integer); override;
  procedure Loaded; override;

  procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); override;

public
  constructor Create(AOwner: TComponent); override;
  function GetActionLinkClass: TControlActionLinkClass; override;
  procedure SetColor(const Value: TColor);

published
  property Action;

  property RGBSelect: TRGBSelect read FRGBSelect write SetRGBSelect
                                 default DefaultColorEditRGBSelect;
  property Color: TColor read FColor write SetColor default DefaultColorEditColor;

  property Position stored False;
  property Max stored False;
  property Min stored False;
  property LargeChange stored False;
  property SmallChange stored False;
  property PageSize stored False;
end;
```

Slide 16



Slide 17

Properly separating Application from Display Logic

- ColorGradient3 project
- Action has now been modified to refer to Color Gradient Bitmap instead of Color Gradient Image
- Action has been modified to return Bitmap (with current position marked)
- Added new Custom Image component (with support for Action) to project
- Action is tied to MouseDown event

Slide 18

```
TGradientColorImageActionLink = class(TControlActionLink)
protected
  function IsOnExecuteLinked: Boolean; override;
  procedure SetBitmap(Bitmap: TBitmap);
end;

TColorGradientImage = class(TImage)
private
  procedure SetBitmap(const Value: TBitmap);

protected
  procedure MouseDown(Button: TMouseButton;
                      Shift: TShiftState; X, Y: Integer); override;

  procedure ActionChange(Sender: TObject; CheckDefaults: Boolean); override;

public
  function GetActionLinkClass: TControlActionLinkClass; override;
  property Bitmap: TBitmap write SetBitmap;

published
  property Action;

  property Picture stored False;
end;
```

Slide 19

Review of Design

- TColorGradientBitmap class manages bitmap. Our 'Application Logic'
- TColorGradientAction provides interface for controls to Application Logic.
- Broadcasts are done via ActionLinks.
- Controls now encapsulate all the code required to display the information
- MainForm now focuses only on Component Placement

Slide 20

Software Complexity and scaling

- A significant part of Software Engineering involves managing Complexity
- One of the most important techniques is to develop classes which strongly reflect appropriate real world entities
- Code should be then suitably assigned to these classes
- Try and keep out of Form!!

Slide 21

Adding dynamics to ColorGradient Demo

- ColorGradient4 project
- Application Logic now includes dynamic objects
- Main change is to add the procedure HandleBitmapRedOffsetChange procedure so that controls are updated when the ControlGradientBitmap changes
- No Change to MainForm!

Slide 22

Disadvantages of using Custom Actions and Custom Components

- More work required (obvious)
- Component Creation knowledge required (could make maintenance harder)
- Need to keep components and application synchronized

Slide 23

Part 2

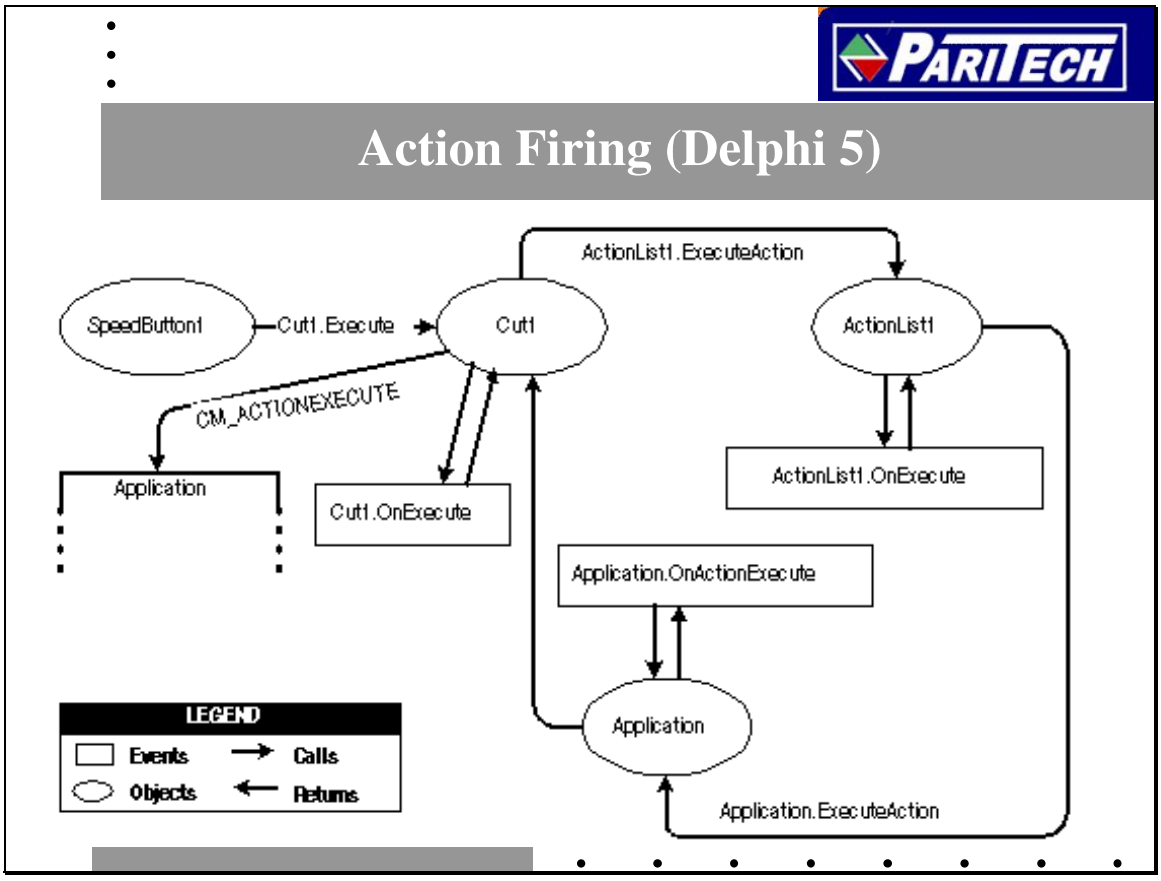
Using Actions with Targets

Slide 24

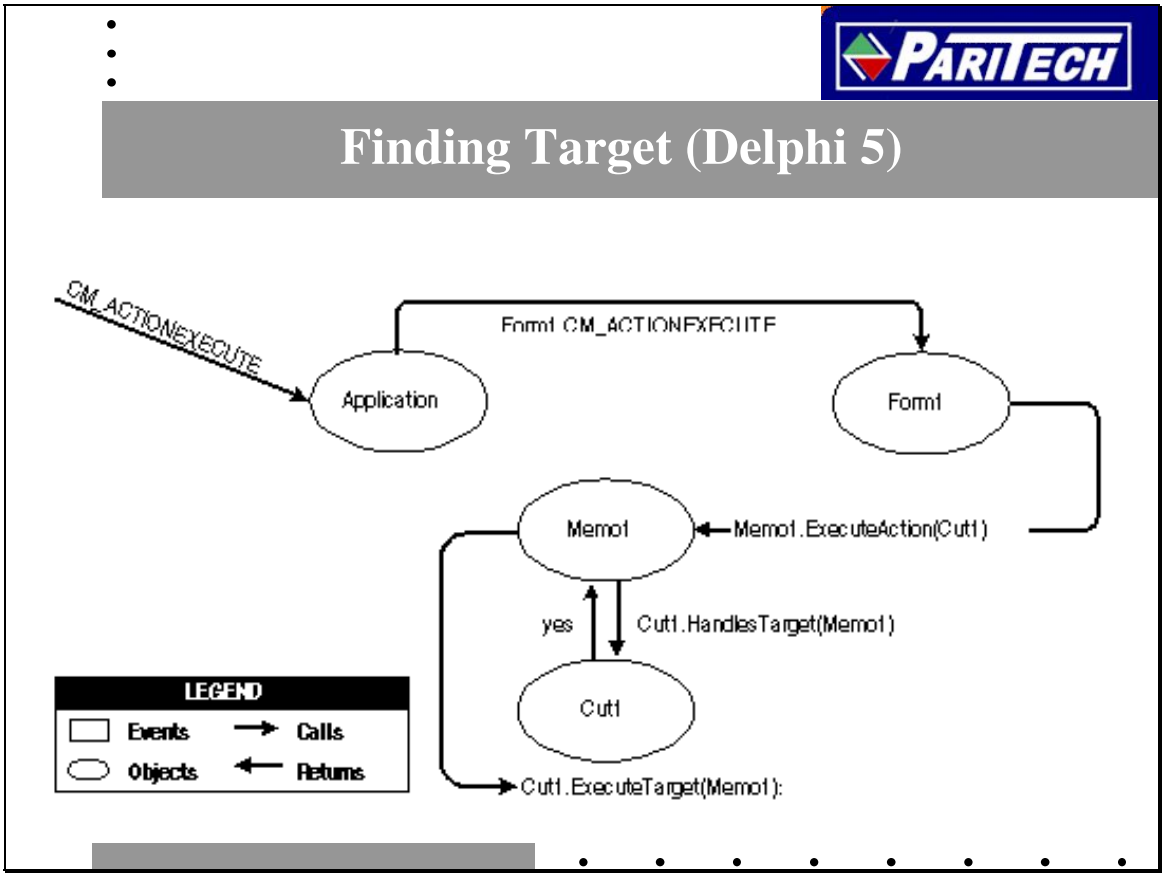
What happens when an Action Fires

- When an action fires it will look for an event handler.
  1. Action List
  2. Application
  3. Action
- Handled needs to be set to stop the search for the event handler
- If it does not find a handler, it looks for a target

Slide 25



Slide 26



Slide 27

Finding targets

- Memo is the target for the Action
- The application looks for the target using the following sequence:
  - Active Control
  - Active Form
  - Controls on the Form

Slide 28

Sub Window View Tool Bar Action

```
TViewToolBarAction = class(TAction)
public
  constructor Create(AOwner: TComponent); override;

  function HandlesTarget(Target: TObject): Boolean; override;

  procedure ExecuteTarget(Target: TObject); override;
  procedure UpdateTarget(Target: TObject); override;
end;
```

Slide 29

Testing Target

- Test whether Action handles (potential) target

```
function TViewToolBarAction.HandlesTarget(Target: TObject):
                                                      Boolean;
begin
  Result := Target is TMainForm;
end;
```

Slide 30

Using Target

- Execute Action using Target

```
procedure TViewToolBarAction.ExecuteTarget(Target: TObject);
begin
  (Target as TMainForm).ViewToolBarActionExecute(Self);
end;
```

- Targets can also be used with Action Update

```
procedure TViewToolBarAction.UpdateTarget(Target: TObject);
begin
  (Target as TMainForm).ViewToolBarActionUpdate(Self);
end;
```

Slide 31

Action Update tips

- **Do not carry out intensive tasks in the OnUpdate Handler**
- **Action Update will not occur in an ActiveX component or an ActiveForm (or DLL)**

Slide 32

Conclusion

- **You can use Actions with ‘other’ components**
- **Use Actions to separate Application Logic from Display Logic**
- **Actions can search for Targets.**