Dave's Development Blog
Software Development using Borland / Codegear / Embarcadero RAD Studio

Chapter 4: Key Bindings and Debugging Tools

By David | February 10, 2010 1 Comment

In this chapter I'll solve a problem I've found with all the Borland/CodeGear IDE's I've used. I like to use just the keyboard where possible rather than grab the mouse. With the keyboard (and I use the IDE Classic keyboard binding) you can create a breakpoint with Ctrl+F8 but you can not edit its properties. You also can't disable the breakpoint with the keyboard either.

So I wrote the following code. First were have the declaration of the wizard as follows:

```
unit DebuggingToolsWizard;

interface

Uses
  ToolsAPI, Classes;

Type
  TDebuggingWizard = Class(TNotifierObject, IOTAWizard)
  Private
  Protected
    { IOTAWizard }
    Function GetIDString: string;
    Function GetName: string;
    Function GetState: TWizardState;
    Procedure Execute;
    { IOTAMenuWizard }
    {$HINTS OFF}
    Function GetMenuText: string;
    {$HINTS ON}
  Public
    Constructor Create;
    Destructor Destroy; Override;
  End;

  TKeyboardBinding = Class(TNotifierObject, IOTAKeyboardBinding)
  Private
    Procedure AddBreakpoint(const Context: IOTAKeyContext;
```

```
      KeyCode: TShortcut; var BindingResult: TKeyBindingResult);
  Protected
    function GetBindingType: TBindingType;
    function GetDisplayName: string;
    function GetName: string;
    procedure BindKeyboard(const BindingServices: IOTAKeyBindingServices);
  Protected
  Public
  End;

  Procedure Register;
  Function InitWizard(Const BorlandIDEServices : IBorlandIDEServices;
    RegisterProc : TWizardRegisterProc;
    var Terminate: TWizardTerminateProc) : Boolean; StdCall;

Exports
  InitWizard Name WizardEntryPoint;

implementation
```

Here we create a wizard that does nothing (but is needed by the RegisterProc() method for DLLs and a keyboard binding class to handle the keyboard input.

Next we need to register both classes with the system so we need to update the InitialiseWizard() function as follows:

```
Var
  iWizardIndex : Integer = 0;
  iKeyBindingIndex : Integer = 0;

Function InitialiseWizard(BIDES : IBorlandIDEServices) : TDebuggingWizard;

Begin
  Result := TDebuggingWizard.Create;
  Application.Handle := (BIDES As IOTAServices).GetParentHandle;
  iKeyBindingIndex := (BorlandIDEServices As IOTAKeyboardServices).AddKeyboardBinding(
    TKeyboardBinding.Create)
End;
```

… and we need to update the Initialization / Finalization section as follows:

```
Initialization
Finalization
  If iKeyBindingIndex > 0 Then
    (BorlandIDEServices As IOTAKeyboardServices).RemoveKeyboardBinding(iKeyBindingIndex);
  If iWizardIndex > 0 Then
    (BorlandIDEServices As IOTAWizardServices).RemoveWizard(iWizardIndex);
```

So next we'll implement the property getters for the IOTAKeyboardBinding interface as follows:

```
function TKeyboardBinding.GetBindingType: TBindingType;
begin
  Result := btPartial;
end;
```

The above indicates its a partial binding, i.e. adds to the main binding and not an entirely new keyboard binding set like IDE Classic.

```
function TKeyboardBinding.GetDisplayName: string;
begin
  Result := 'Debugging Tools Bindings';
end;
```

This above is the display name that appears under the Key Mappings, Enhancement Modules section.

```
function TKeyboardBinding.GetName: string;
begin
  Result := 'DebuggingToolsBindings';
end;
```

And finally this is the unique name for the keyboard binding.

Next we'll tackle the keyboard bindings with the following code:

```
procedure TKeyboardBinding.BindKeyboard(
  const BindingServices: IOTAKeyBindingServices);
begin
  BindingServices.AddKeyBinding([TextToShortcut('Ctrl+Shift+F8')], AddBreakpoint, Nil);
  BindingServices.AddKeyBinding([TextToShortcut('Ctrl+Alt+F8')], AddBreakpoint, Nil);
end;
```

Here Ctrl+Alt+F8 creates / toggles the breakpoint between enabled and disabled and Ctrl+Shift+F8 creates / edits the breakpoint's properties.

Finally the code that does the actual work as follows:

```
procedure TKeyboardBinding.AddBreakpoint(const Context: IOTAKeyContext;
  KeyCode: TShortcut; var BindingResult: TKeyBindingResult);

var
  i: Integer;
  DS : IOTADebuggerServices;
  MS : IOTAModuleServices;
  strFileName : String;
  Source : IOTASourceEditor;
  CP: TOTAEditPos;
  BP: IOTABreakpoint;

begin
  MS := BorlandIDEServices As IOTAModuleServices;
  Source := SourceEditor(MS.CurrentModule);
  strFileName := Source.FileName;
  CP := Source.EditViews[0].CursorPos;
  DS := BorlandIDEServices As IOTADebuggerServices;
  BP := Nil;
  For i := 0 To DS.SourceBkptCount - 1 Do
    If (DS.SourceBkpts[i].LineNumber = CP.Line) And
      (AnsiCompareFileName(DS.SourceBkpts[i].FileName, strFileName) = 0) Then
      BP := DS.SourceBkpts[i];;
  If BP <> Nil Then
    Begin
      If KeyCode = TextToShortCut('Ctrl+Shift+F8') Then
        BP.Edit(True)
      Else
        BP.Enabled := Not BP.Enabled;
    End Else
    Begin
      BP := DS.NewSourceBreakpoint(strFileName, CP.Line, Nil);
      If KeyCode = TextToShortCut('Ctrl+Alt+F8') Then
        BP.Enabled := False;
      BP.Edit(True);
    End;
  BindingResult := krHandled;
end;
```

This code first searches for a break point at the current line and creates one IF it doesn't exist and displays the Source Breakpoint Properties dialogue for editing the breakpoint's attributes.

I've not described the TDebuggingWizard getter methods as these have been described in earlier chapters.

Its not perfect but I think you get the idea and can work out the bugs for yourselves (hint: look what happens when you press Ctrl+Alt+F8 without an existing breakpoint).

I hope this proves to be useful.

regards
Dave 🙂

Category: Open Tools API Tags: Borland, Breakpoints, CodeGear, Delphi, Experts, IOTADebuggerServices, IOTAKeyBindingServices, IOTAKeyboardBinding, IOTAWizard

One thought on “Chapter 4: Key Bindings and Debugging Tools”

Toyota Tundra
July 23, 2010

i just wanna thank you for sharing your this info on your blog

Comments are closed.