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

Chapter 16: Getting help when there's no help

By David | March 21, 2016 0 Comment

I don't know whether its an issue with my Delphi XE7 (I'll find out when Seattle 10 arrives) but it doesn't seem to have any Win32/64 help in the system anymore. I'm assuming that this is related to the space in the help required by the new FireMonkey and cross platform support. Since most of my development is in the Win32/64 world I find this frustrating so I decided to do something about it on Saturday.

What trigger this was while finalizing the OTA Book I came across an interface `IOTAHelpServices` and wondered whether I could intercept calls to the help system and if they are not handled display some search information from say Google.

If you have followed my previous posts you will have created a wizard to create wizards. So rather than go through all the building of a new expert/wizard I've used my own tools to get most of the framework in place.

Eating your own dog food

…At least I think that the expression for using your own software.

If you haven't worked through the previous chapters then you can download the complete wizard from Chapter 15: IDE Main Menus or use the code with Chapter 4: Key Bindings and Debugging Tools as a template.

Once you have create the template and saved the files there are a few things you will need to do to get the project to compile properly.

Enable Packages

When I did this I selected a DLL project from the wizard so to get the project to compile (and find `ToolsAPI.pas`) you need to enabled packages and use at least the following packages:

- RTL;
- VCL;

- DesignIDE.

Your project should now compile BUT you will get some warnings.

Updating ConditionalDefinitions.inc

Depending upon your version of RAD Studio / Delphi the conditional compilation include file created by the wizard will not handle your version of the compiler (it only went up to XE2 not 11 – sorry couldn't resist). You need to extend the conditions far enough to cover your compiler (the pattern should be obvious and has been explain in Conditional Compilation of Open Tools API Experts).

The OTA Code

OTAHelpServices Interface

Below is the definition of the `IOTAHelpServices` from the XE7 `ToolsAPI.pas` file.

```
IOTAHelpServices = interface(IDispatch)
    ['{25F4CC12-EA93-4AEC-BC4A-DFDF427053B0}']
    procedure ShowKeywordHelp(const Keyword: WideString); safecall;
    function UnderstandsKeyword(const Keyword: WideString): WordBool; safecall;
    procedure ShowContextHelp(ContextID: Integer); safecall;
    procedure ShowTopicHelp(const Topic: WideString); safecall;
    function GetFileHelpTrait(const FileName: WideString): IOTAHelpTrait; safecall;
    function GetPersonalityHelpTrait(const Personality: WideString):
IOTAPersonalityHelpTrait; safecall;
  end;
```

Below are my understanding of their function as there are no comments in the file (note, I only use 1 of these methods for this expert):

ShowKeywordHelp

I believe that this method will display the appropriate help page for a given keyword.

UnderstandsKeyword

This function returns true if the given `Keyword` is understood by the IDE, i.e. there is a help topic for the Keyword else the function will return false. This method raises an exception if an empty string is passed as the Keyword. This is the function I use to determine whether the IDE can handle the Identifier under the cursor.

ShowContextHelp

This method will show the context help for the given context ID (integer). This is similar to WinHelp and HTMLHelp however I'm not sure where you would find the context numbers unless you can integrate your help with the IDE.

ShowTopicHelp

This method I believe allows you to open a help topic based on its description rather than just a help topic based on a Keyword.

GetFileHelpTrait

Not really sure about this one other than it would seem to return a `IOTAHelpTrait` for a given filename but there is no further references to what a `IOTAHelpTrait` is other than the definition below:

```
IOTAHelpTrait = interface(IDispatch)
    ['{DEE36173-1597-498A-A85A-C90BFCAE9B74}']
  end;
```

You could perhaps surmise that this will allow you to get a string which represents the personality the file belongs to in the IDE.

GetPersonalityHelpTrait

This method allows you to get access to personality specific help by specifying the IDE personality (one of the predefined strings in `ToolsAPI.pas`) which then provides access to personality specific implementations of `ShowKeywordHelp` and `UnderstandsKeyword` as described above (see definition of the `IOTAPersonalityHelpTrait` below).

```
IOTAPersonalityHelpTrait = interface(IDispatch)
    ['{914E82DB-4123-4AA8-91D9-DB105E1FEC64}']
    procedure ShowKeywordHelp(const Keyword: WideString); safecall;
    function UnderstandsKeyword(const Keyword: WideString): WordBool; safecall;
  end;
```

Getting the Identifier at the Cursor

In order to be able to first ask the IDE if it understand a Keyword/Identifier and then secondly search for that on the internet I need to get the word underneath the cursor in the IDE's editor. To do this (see

below code) I use one of the utility functions (see Chapter 5: Useful Open Tools Utility Functions) `EditorAsString` which returns all the text of the current editor as a string. I then put that into a string list to get it into individual lines and then see if there is a word under the column position of the cursor. If so I trace the start and end of the word and then return that word from this function. If there is no word under the cursor I return a null string.

```
Function TKeybindingTemplate.GetWordAtCursor: String;

Const
  strIdentChars = ['a'..'z', 'A'..'Z', '_', '0'..'9'];

Var
  SE: IOTASourceEditor;
  EP: TOTAEditPos;
  iPosition: Integer;
  sl: TStringList;

Begin
  Result := '';
  SE := ActiveSourceEditor;
  EP := SE.EditViews[0].CursorPos;
  sl := TStringList.Create;
  Try
    sl.Text := EditorAsString(SE);
    Result := sl[Pred(EP.Line)];
    iPosition := EP.Col;
    If (iPosition > 0) And (Length(Result) >= iPosition) And
      CharInSet(Result[iPosition], strIdentChars) Then
      Begin
        While (iPosition > 1) And (CharInSet(Result[Pred(iPosition)], strIdentChars))
Do
          Dec(iPosition);
        Delete(Result, 1, Pred(iPosition));
        iPosition := 1;
        While CharInSet(Result[iPosition], strIdentChars) Do
          Inc(iPosition);
        Delete(Result, iPosition, Length(Result) - iPosition + 1);
        If CharInSet(Result[1], ['0'..'9']) Then
          Result := '';
      End Else
        Result := '';
  Finally
    sl.Free;
  End;
End;
```

Determining if the IDE's Help can handle the Identifier

When I first implemented this expert I passed the search URL to `ShellExecute` to bring the search up in the default browser but like most thing I do it grew legs and now implements a dockable form with a

TWebBrowser embedded in it. I'm not going to go through this implementation especially the web browser portion as this is all about OTA but I'm not going to go through the dockable form either. Why? Well that's in the book… coming to a web site near you soon.

In essence the code gets the Keyword at the cursor as described above and if its a valid word asks the IDE if it has any help for the word. If not then the word is embedded in a search URL and passed to the dockable browser (or you could simply use `ShellExecute`).

```
Procedure TKeybindingTemplate.ProcessKeyBinding(Const Context: IOTAKeyContext;
  KeyCode: TShortcut; Var BindingResult: TKeyBindingResult);

Const
  strMsg = 'Your search URLs are misconfigured. Ensure there is a Search URL and that
' +
    'it is checked in the list in the configuration dialogue.';

Var
  strWordAtCursor : String;
  boolHandled: Boolean;

Begin
  strWordAtCursor := GetWordAtCursor;
  If strWordAtCursor <> '' Then
    Begin
      boolHandled :=
        (BorlandIDEServices As IOTAHelpServices).UnderstandsKeyword(strWordAtCursor);
      If boolHandled Then
        BindingResult := krUnhandled
      Else
        Begin
          BindingResult := krHandled;
          If (AppOptions.SearchURLIndex <= AppOptions.SearchURLs.Count - 1) And
            (AppOptions.SearchURLIndex >= 0) Then
            TfrmDockableBrowser.Execute(
              Format(AppOptions.SearchURLs[AppOptions.SearchURLIndex],
[strWordAtCursor]))
          Else
            MessageDlg(strMsg, mtError, [mbOK], 0);
        End;
    End Else
      BindingResult := krUnhandled;
End;
```

Code

The RAD Studio XE7 code for this article can be downloaded from the page IDE Help Helper or directly from this link.

Hope this proves helpful to all.

regards
Dave.

Category: Open Tools API Tags: Borland, BorlandIDEServices, CodeGear, Delphi, Embarcadero, Experts, IOTAHelpServices, OTA, RAD Studio