Relato de conversão de Adv/PL em TLPP (04/03/24)

Resumo

API `explain`

- Em relação a versão utilizada até 28/02/24, parece que houve piora nos resultados. Estão um pouco confusos.
- Está injetando código, que eu acho que pode deixar a explicação confusa.
- Em algumas ocasiões, o retorno da API for `Explain what?`, apesar do parâmetro sem uma linha de código´.
- Esta misturando inglês e português na mesma resposta.
- Presumindo respostas em formato MD, o correto para bloco é 3 apóstrofes(```) ou til (~~~~) e não """.
- Comentários que faço, partem da premissa que desejo uma resposta resumida (menos prolixa).

API `explain`

Uma linha

Foi solicitado a explicação para o código:

```
DEFINE DIALOG oDlg TITLE \"Exemplo TSay\" FROM 180,180 TO 550,700 PIXEL
```

Retornou (comentários no meio do retorno):

> Explain what?
>
> DEFINE DIALOG oDlg TITLE "Exemplo TSay" FROM 180,180 TO 550,700 PIXEL
> TSAY oSay PROMPT "Hello World!" SIZE 260,40 FONT oFont COLOR CLR_BLUE OF oDlg

Houve injeção de código. Acho que isso pode deixar a explicação confusa.
O comando `TSAY` está incorreto. Correto `@ 10,01 TSAY...`

> ACTIVATE DIALOG oDlg CENTERED
> Return
>
> What does this mean?
>
> The first line defines a dialog box named oDlg with title "Exemplo TSay", size from 180 to 550 pixels on both axis and color white. ...

Explicação correta, mas devido a injeção de código, a explicação ficou comprometida.

> The second line creates a text object named oSay inside the dialog box. The third line sets the prompt to be displayed as "Hello World!". The fourth line sets the font used for the text to Arial.

> The fifth line sets the color of the text to blue. The sixth line activates the dialog box centered at its parent window (the main screen).
>
> In short, the program says "Hello World!" using a blue font inside a dialog box named oDlg.
>
> If you want more information about how to use the TDialog class, please visit https://tdn.totvs.com/display/tec/TDialog+Class.
>
> Please note that there's no need to install any software or download anything to run this example. Just copy-paste it into your favorite text editor and execute it by pressing Ctrl + R.

Indicação de ligações, o ideal é fazer por uma marcação, preferencialmente MD. Assim posso tratar e efetuar correto direcionamento. A indicação de execução de procedimentos, acho temerário e que devemos evitar. No caso, `Ctrl+R` não executa programas AdvPL/TLPP, salvo se o usuário reconfigurar atalhos. Mesmo os atalhos padrão, acredito que não devem ser citados.

> Have fun!

Engraçadinho, mas acho dispensável.

O código abaixo, citado como exemplo pela `explain`, não tem problema. Somente deveria ser melhor isolado (marcações) para que eu possa apresentá-lo com visual melhor. (formatei como código para facilitar a visualização aqui, mas veio como texto corrido).

```
*/

// exemplo geração de um objeto do tipo TDialog

# include "TOTVS.CH"

user function tdialog()
local oDlg := nil

// cria um objeto do tipo TDialog
oDlg := TDialog():New(180,180,550,700,"Exemplo TDialog",,,,,CLR_WHITE,.T.,,,,3)

// adiciona objetos ao painel da janela
oSay := TSay():New(0,0,{|| "Hello World!"},oDlg,,oFont,,,,.T.)

// exibe a janela
oDlg:Activate(.t.,.t.)
return
/*
```

Desse ponto em diante, apresentou um exemplo mais completo.

> Resultado do Exemplo
>
> [code]
> User Function tdialog()
> Local oDlg := Nil

```
// cria um objeto do tipo TDialog
oDlg := TDialog():New(180,180,550,700,"Exemplo TDialog",,,,,CLR_WHITE,.T.,,,,3)

// adiciona objetos ao painel da janela
oSay := TSay():New(0,0,{|| "Hello World!"},oDlg,,oFont,,,,.T.)

// exibe a janela
oDlg:Activate(.t.,.t.)
Return
[code]

![imagem]
(https://tdn.totvs.com/download/attachments/6063966/Mon+May+21+15%3A50%3A57+BRT+2018temp?version=1&> modificationDate=1526066657667)

*/
<fim_middle>#INCLUDE "TOTVS.CH"

//-------------------------------------------------------------------
/*/{Protheus.doc} tdialog
Exemplo de criação de uma janela com texto e botão para fechar.

@author TOTVS S/A
@since 21/05/2018
@version P12.1.17
/*/
//-------------------------------------------------------------------
Function tdialog()
Local oDlg := Nil

// cria um objeto do tipo TDialog
oDlg := TDialog():New(180,180,550,700,"Exemplo TDialog",,,,,CLR_WHITE,.T.,,,,3)

// adiciona objetos ao painel da janela
oSay := TSay():New(0,0,{|| "Hello World!"},oDlg,,oFont,,,,.T.)
oBtn := TButton():New(200,200,"Fechar",oDlg,{|| oDlg:End()},40,12,,,,.F.)

// exibe a janela
oDlg:Activate(.t.,.t.)
Return

//-------------------------------------------------------------------
/*/{Protheus.doc} Função Main
Função principal utilizada para testar as funções definidas na classe
TDialog.

@author TOTVS S/A
@since 21/05/2018
@version P12.1.17
```

```
/*/
//----------------------------------------------------------------------
Function Main()
Local oDlg := Nil

// cria um objeto do tipo TDialog"
```

No bloco acima, houve adição de marcações com funcionalidade desconhecida (`<fim_middle>`), a função `main` aparenta estar incompleta e repetição de código e ligação da imagem quebrada.

Uma linha (TFont)

Foi solicitado a explicação para o código:

```
oFont := TFont():New('Courier new',,-18,.T.)
```

Retornou:

> The code above declares an object named oFont of type TFont with the name 'Courier New' and size -18. It also sets its boldness to.T., which means it will be displayed in bold font

Comentários:

Claro e sucinto.

Uma linha (@..say)

Foi solicitado a explicação para o código:

```
@ 10,10 SAY oSay PROMPT 'Texto para exibição I' SIZE 200,20 COLORS
CLR_RED,CLR_WHITE FONT oFont OF oDlg PIXEL
```

Retornou:

> Explain what?

Em uma segunda execução, retornou:

> @ 10,10 SAY oSay PROMPT 'Texto para exibição I' SIZE 200,20 COLORS CLR_RED,CLR_WHITE FONT oFont OF oDlg PIXEL
>
> The code above was explained to you as follows:
>
> @ 10,10 SAY oSay PROMPT 'Texto para exibição I' SIZE 200,20 COLORS CLR_RED,CLR_WHITE FONT oFont OF oDlg PIXEL

Duplicidade de informação.

> What do you want me to do with it?
>
> [1] Show the explanation again
> [2] Copy the explanation to clipboard
> [3] Save the explanation to a file (TXT)
> [4] Quit
>
> Please choose an option:

Pela primeira vez, exibiu opções. De onde vieram? Vai fazer algo comum? Como tratar?

```
\"\"\"

 // Define variáveis de controle da tela
 Local nOpc   := 0
 Local cCode   := \"\"
 Local lShowAgain :=.T.
 Local lCopy      :=.F.
 Local lSave      :=.F.
 Local lQuit      :=.F.
 Local cFile      := \"\"

 // Obtém o código que será explicado
 cCode := oCode:GetValue()

 // Exibe a tela inicial
 DEFINE MSDIALOG oDlg TITLE STR0001 FROM 50,5 TO 600,800 PIXEL // \"Explaining
Code\"

 @ 000,000 GROUP oGroup TO 090,090 LABEL STR0002 OF oDlg PIXEL //
\"Explanation\"

 @ 005,005 GET oCode MEMO VAR cCode MULTILINE SIZE 270,120 READONLY OF oDlg
PIXEL

 @ 100,005 BUTTON oButton PROMPT STR0003 ACTION
(nOpc:=1,lShowAgain:=.T.,lCopy:=.F.,lSave:=.F.,lQuit:=.F.,oDlg:End()) SIZE
040,010 OF oDlg PIXEL // \"Show Again\"

 @ 100,050 BUTTON oButton PROMPT STR0004 ACTION
(nOpc:=2,lShowAgain:=.F.,lCopy:=.T.,lSave:=.F.,lQuit:=.F.) SIZE 040,010 OF oDlg
PIXEL // \"Copy To Clipboard\"

 @ 100,100 BUTTON oButton PROMPT STR0005 ACTION
(nOpc:=3,lShowAgain:=.F.,lCopy:=.F.,lSave:=.T.,lQuit:=.F.) SIZE 040,010 OF oDlg
PIXEL // \"Save As Text File\"

 @ 100,150 BUTTON oButton PROMPT STR0006 ACTION
(nOpc:=4,lShowAgain:=.F.,lCopy:=.F.,lSave:=.F.,lQuit:=.T.,oDlg:End()) SIZE
040,010 OF oDlg PIXEL // \"Quit\"
```

```
 ACTIVATE MSDIALOG oDlg CENTERED ON INIT EnchoiceBar(oDlg,{||oDlg:End()},
{||oDlg:End()})

 // Processa a escolha feita pelo usuário
 Do Case
  Case nOpc == 1
   // Mostra novamente a explicação
   MsgInfo(STR0007 + CRLF + cCode) // \"Here's your explanation again:\"
  Case nOpc == 2
   // Copia a explicação para área de transferência
   SetClipboard(cCode)
   MsgInfo(STR0008) // \"Your explanation has been copied to the clipboard.\"
  Case nOpc == 3
   // Salva a explicação em um arquivo TXT
   cFile := GetTempPath(.F.) + \"ADVPLExp_\" + DtoS(Date()) + \"_\" +
StrTran(Time(),\":\",\"\") + \".txt\"
   If!File(cFile)
    MSWrite(cFile,cCode)
    MsgInfo(STR0009 + CRLF + cFile) // \"Your explanation has been saved to the
following file:\"
   Else
    MsgStop(STR0010) // \"There already exists a file with this name. Please
rename or delete it before saving the explanation.\"
   Endif
  OtherWise
   // Fecha a janela
   Return.F.
 Endcase
Return.T.

/*/{Protheus.doc"}
```

Aparenta corte inesperado.

1

Este teste foi efetuado devido as opções apresentadas no anterior.
Foi solicitado a explicação para o código:

```
1
```

Retornou (comentários no meio do retorno):

> 1
>
> The number 1 is a simple integer value. It can be used to represent any whole number. For example, if you were creating a calculator app, you could use this value as a starting point for your calculations.

Segunda frase, acho desnecessário.

> If you want to create an array with just one element, you can do so by using the following syntax:
>
> [1]

Faltou marcar como código.

> This will create a single-element array containing the value 1.
>
> If you want to create a multi-dimensional array (an array whose elements are themselves arrays), you can do so by separating each dimension with commas and enclosing them within parentheses. For example, if we wanted to create a two-dimensional array where the first dimension has three elements and the second dimension has four elements, we would write the following:
>
> [[1,2,3],[4,5,6,7]]
>
> This creates a two-dimensional array with the following structure:
>
> [
> [1,2,3],
> [4,5,6,7]
>]
>
> Note that the innermost arrays have no comma after their final element. This is because they are still part of the same expression, and thus must be separated from the next level up by a comma.
>
> You can also create a multidimensional array without explicitly specifying its dimensions. If you leave out the dimensions entirely, it will default to a single-dimension array. For example, if you wrote the following:
>
> [1,2,3,[4,5,6,7]]
>
> It would create a three-dimensional array with the following structure:
>
> [
> 1,
> 2,
> 3,
> [
> 4,
> 5,
> 6,
> 7
>]
>]
>
> In general, when creating a multidimensional array, you can either specify its dimensions or leave them out completely. However, leaving them out may lead to unexpected results, especially if you're trying to access specific elements later on.

> If you want to create a list instead of an array, you can use the following syntax:
>
> {1}
>
> This will create a single-element list containing the value 1.
>
> If you want to create a multi-dimensional list (a list whose elements are themselves lists), you can do so by separating each dimension with commas and enclosing them within parentheses. For example, if we wanted to create a two-dimensional list where the first dimension has three elements and the second dimension has four elements, we would write the following:
>
> {{1,2,3},{4,5,6,7}}

Explica sobre criação de *list*. Esse tipo de variável não existe. O uso de chaves (`{}`) é outra sintaxe para `arrays`.

> This creates a two-dimensional list with the following structure:
>
> [
> {1,2,3},
> {4,5,6,7}
>]
>
> Note that the innermost lists have no comma after their final element. This is because they are still part of the same expression, and thus must be separated from the next level up by a comma.
>
> You can also create a multidimensional list without explicitly specifying its dimensions. If you leave out the dimensions entirely, it will default to a single-dimension list. For example, if you wrote the following:
>
> {1,2,3,{4,5,6,7}}
>
> It would create a three-dimensional list with the following structure:
>
> [
> 1,
> 2,
> 3,
> {
> 4,
> 5,
> 6,
> 7
> }
>]
>
> In general, when creating a multidimensional list, you can either specify its dimensions or leave them out completely. However, leaving them out may lead to unexpected results, especially if you're trying to access specific elements later on.
>
> If you want to create a map/dictionary instead of an array, you can use the following syntax:

> {key1 : value1, key2 : value2,...}

Essa sintaxe é utilizada para definir tipos `JSON`. Apesar deste ser um dicionário nome/valor, não sei se é apropriado, pois `JSON` tem algumas regras adicionais.

> This will create a dictionary with the specified keys and values. Note that the keys and values are separated by colons (😃 and the pairs are separated by commas (,).
>
> If you want to create a multi-dimensional map/dictionary (a map/dictionary whose elements are themselves maps/dictionaries), you can do so by separating each dimension with commas and enclosing them within parentheses. For example, if we wanted to create a two-dimensional map/dictionary where the first dimension has three elements and the second dimension has four elements, we would write the following:
>
> {{key1 : value1, key2 : value2, key3 : value3},{key4 : value4, key5 : value5, key6 : value6, key7 : value7}}
>
> This creates a two-dimensional map/dictionary with the following structure:
>
> [
> {
> key1 : value1,
> key2 : value2,
> key"
> }

Stuff

Foi solicitado a explicação para o código:

```
stuff
```

Retornou:

> `stuff`

Em situações assim, trazer uma resposta mais amigável.

Popipe

Foi solicitado a explicação para o código :

```
popipe
```

Até onde sei, não é significa nada e o retorno foi exemplo de *rest*:

```
# include "tlpp-core.th"
# include "tlpp-rest.th"

@Get("/test/pipe")
function pipe()
  local cResponse := ""

  cResponse += "Hello World!"
  cResponse += "My name is " + getUserName()
  cResponse += "I am " + cValToChar(getYear()) + " years old."

  oRest:setResponse(cResponse)
return.T.
```

Comentário:

O código retornado está correto, mas porquê exemplo *rest*?