How a non-windowed component can receive messages from Windows

Why do it?

Sometimes we need a non-windowed component (i.e. one that isn't derived from *TWinControl*) to receive Windows messages. To receive messages the component needs a window handle, but a non-windowed component hasn't got one! This article is about how to enable such a component to use a hidden window to receive messages.

How it's done

The Delphi library function *AllocateHWnd* is used to create a hidden window for us and the related *DeallocateHWnd* disposes of the window when we've finished with it.

The hidden window requires window procedure. *AllocateHWnd* enables us to use a method as a window procedure where Windows normally requires a *stdcall* function. We pass a reference to the required method to *AllocateHWnd* and it takes care of the problem of registering the method as a window procedure for us. Inside the registered method we handle the messages we are interested in and hand the rest off to Windows using the *DefWindowProc* API call.

Example

My *Clipboard Viewer Component* is a non-visual component that uses the hidden window techniques described here. The window receives Windows messages that provide information about changes to the clipboard.

Listing 2 below provides a skeleton of how to use *AllocateHWnd*. First though, *Listing 1* shows an outline definition for our component class:

```
type
  { Our class derived from TComponent
    or another ancestor class }
  TMyClass = class(TComponent)
  private
    fHWnd: HWND;
      { field to store the window handle }
    ...
  protected
    procedure WndMethod(var Msg: TMessage); virtual;
      { window proc - called by Windows to handle
        messages passed to our hidden window }
    ...
  public
     constructor Create(AOwner: TComponent); override;
       { create hidden window here: store handle in fHWnd}
     destructor Destroy; override;
       { free hidden window here }
     ...
  end;
```

Listing 1

And here are the implementation details:

```
constructor TMyClass.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  ...
  // Create hidden window using WndMethod as window proc
  fHWnd := AllocateHWnd(WndMethod);
  ...
end;

destructor TMyClass.Destroy;
begin
  ...
```

```
  // Destroy hidden window
  DeallocateHWnd(fHWnd);
  ...
  inherited Destroy;
end;

procedure TMyClass.WndMethod(var Msg : TMessage);
var
  Handled: Boolean;
begin
  // Assume we handle message
  Handled := True;
  case Msg.Msg of
    WM_SOMETHING: DoSomething;
      // Code to handle a message
    WM_SOMETHINGELSE: DoSomethingElse;
      // Code to handle another message
    // Handle other messages here
    else
      // We didn't handle message
      Handled := False;
  end;
  if Handled then
    // We handled message - record in message result
    Msg.Result := 0
  else
    // We didn't handle message
    // pass to DefWindowProc and record result
    Msg.Result := DefWindowProc(fHWnd, Msg.Msg,
      Msg.WParam, Msg.LParam);
end;
```

Listing 2

Of course, we could just use the Windows API to create a window the hard way and provide a windows procedure. But it is more difficult to use a method as a window procedure if we do it this way. The clever features about *AllocateHWnd* are that (a) it creates the hidden window for us and (b) it allows us to use a method, rather than a simple procedure, as the window procedure – and a method is more useful since it has access to the class's private data.
