# 使用 Visiual FoxPro 创建 Outlook 样式的桌面提示


## Kevin Ragsdale

## 翻译：xinjie　　QQ:411618689　　2019.04.24


我喜欢Outlook 2003中的桌面提示。它们提供有用的信息而不会干扰我的工作。我无法忍受在做一件事情时，突然从应用程序中出现了一个MESSAGEBOX，告诉我它代替我做了一些精彩的事情。桌面提醒只是在我的屏幕上“显示”几秒钟，让我知道我有新邮件。它们甚至提供一些细节（发件人姓名，主题）。如果我想立即阅读电子邮件，只需单击“提示”，Microsoft Outlook即可打开该邮件。否则，我可以忽略提示，它会逐渐消失。

我认为这样的东西对我的一些应用程序来说是一个很好的增强，所以我开始寻找如何模仿 Outlook 桌面提示的方法。

在本文档之后的部分，我将其称为 **DAS (Desktop Alerts System)**。

*请读者注意：当你就某个问题的最佳解决方案询问两个 VFP 开发人员时，你通常会得到五六种不同的结果。如果你有更好的方案实现 DAS，或者你发现了其中的错误或者遗漏，不要犹豫，立即告诉我。正如你在阅读完本白皮书和源代码之后的感觉那样：这并不怎么滴吗!*

**DAS的要求**

- 提示应该具有淡入淡出的效果，并且可以和用户进行交互
- 允许用户配置提示消息
- 提示信息应该是可关闭的，可移动的，可忽略的
- 提示应根据用户交互向客户端返回值
- 在运行时创建提示应该像 MESSAGEBOX 那样容易
- DAS 将基于COM（EXE），因此我可以在VFP 9中创建它并在早期版本的VFP（或其他语言）中使用它。 这也允许我将标准（默认）图形，声音文件等直接打包到其中。

我对 DAS 的最终要求是能够让它实例化一个“全局”提示管理器，并在必要时发出提示。这需要开发一个 AlertManager 类，我们将在后面讨论如何创建它。由于屏幕上可能会出现多个提示信息，因此我还希望每个提示信息都是唯一的。因此，我们还将创建一个 Alert 类。

我们最终得到的是一个可视类库（DESKALERT.VCX），它将包含 DAS UI 的所有元素；一个名为 AlertManager 的基于会话的类；一个名为 Alert 的基于会话的类；包含文件 VFPALERT.H，以及各种支持文件（图形，声音等）。

首先，我创建一个名为 VFPALERT 的文件夹。 在此文件夹中，我创建了一个名为 VFPALERT 的新项目。

接着，让我们开始建立可视类库（DESKALERT.VCX）。

**建立 DAS 用户界面**

由于我是从 Microsoft Outlook 获得灵感来创造 DAS 的，所以，DAS 的 UI 将与 Outlook 桌面提示非常相似。让我们看一下来自 Microsoft Outlook 的桌面提示（图1），然后选择我们需要的 DAS 元素（表1）：


图1：Microsoft Outlook 中的桌面提示

| 项目 | 图片 | 描述 |
|---|---|---|
| 标题栏 | | 这用于取代提示表单的标题栏 |
| 图标 | | 每个提示都有一个图标。如果我们不指定图标，将使用默认图标 |
| 标题 | VoicemailService@packet8.net | 每个提示都有一个标题。如果不指定，将使用默认标题 |
| 主题 | | 主题是可选的 |
| 描述 | Hello Charles K:<br>You have a new voicemail message from unknown number | 每个提示都有详细信息。可以是纯文本（仅文本），或显示为带有操作的超链接 |
| 设置按钮 | | 在提示显示时可隐藏，默认是显示的 |
| 关闭按钮 | | 关闭按钮永远都可见 |
| 任务1图标 | | 任务是可选的。如果我们包含了一个任务，那么我们可以选择一个图标，或者使用默认图标。 |
| 任务2图标 | | |

表1：桌面提示分解

**建立提示表单！**

我们首先创建一个表单类（frmAlert），它具有以下“标准”属性（表2）：

| 属性 | 属性值 | 描述 |
|---|---|---|
| Height | 69 | 只是因为我想模仿Outlook提示。 |
| Width | 321 | |
| ShowWindow | 2 (Top-Level Form) | 仅顶层表单可以实现透明效果。 |
| ShowInTaskBar | .F. | 任务栏无需显示提示。 |
| ShowTips | .T. | 对于按钮和任务，我们需要显示提示信息。 |
| BorderStyle | 3 (Sizable) | 此属性将在表单的 Init() 中予以更改。 |
| Caption | “DESKALERT” | 我们将其用于API函数寻找其他提示。 |
| ControlBox | .F. | 不需要这些，因为我们不使用TitleBar。 |
| MaxButton | .F. | |
| MinButton | .F. | |
| TitleBar | 0 (Off) | |
| AlwaysOnTop | .T. | 我们需要表单用于处于顶层。 |

表2：frmAlert 标准属性

我们需要在表单中添加一些自定义属性（表3），它们用于实现表单的淡入淡出效果。下面是实现“淡入淡出表单”的一组规则：

1 必须是顶层表单；

2 需要 Windows 2000 和更高版本 OS 的支持。

*信不信由你，因为我仍有很多的客户还在使用 Windows 2000 之前的 OS，我是深受其扰。*

我们将使用 Windows API 函数和 VFP Timer 控件来实现淡入淡出效果。后面会予以详述。

*特别感谢 Mike Lewis，他在 FoxPro Advisor 杂志上发表了一篇文章（2006年6月），其中的方法与我最初开发的方法相比，可以更好的实现“淡入淡出”效果。*

| 属性 | 属性值 | 描述 |
|---|---|---|
| lCanFade | .F. | 如果我们在 Windows 98 上使用，将不能实现淡入淡出效果，所以我们将它设置为.F。 除此之外，它应该设置为 .T.。我们将增加一个 Assign 方法来触发初始化的 API 函数（如果属性值为 .T.） |
| nTransparency | 0 | 表单当前的透明度。范围从0（完全透明）到255（完全不透明）。 由于我们要实现“淡入”，因此需要从0开始。 |
| nScreenHeight | 0 | 屏幕的物理高度，应该考虑任务栏的存在。 |
| nScreenWidth | 0 | 屏幕的物理宽度，应该考虑任务栏的存在。 |
| nStatus | -1 | 在任何给定的时间，表单将为0（消失）或1（出现）。 |
| nFadeFactor | 0 | 透明度。每次淡入淡出计时器被触发时都会增加或减少。 |

表3：frmAlert自定义属性（与透明度相关）

我们还要添加一些和淡入淡出无关的自定义属性（表4）：

| 属性 | 属性值 | 描述 |
|---|---|---|
| oParent | NULL | 被创建的提示表单的对象引用。 |
| oParams | NULL | 通过 oParent 传入的参数对象。 |
| nResult | 0 | 表单的返回值。返回值可以是以下之一:<br>-1：警报超时<br>1：用户单击“关闭”按钮<br>2：用户单击了详细信息链接<br>3：用户选择任务1<br>4：用户选择任务2<br>我们将为此属性添加Assign方法。 |

表4：frmAlert 自定义属性（其他）

现在，让我们来看看前面提到的几个表单属性的 Assign（）方法（表5）：

| 属性的 Assign() 方法 | 描述 |
|---|---|
| lCanFade_Assign() | 如果我们运行Windows 2000或更高版本的OS，lCanFade设置为.T。，此assign方法将进行初始化的API函数调用。 |
| nResult_Assign() | 返回表单的 nResult 属性到 oParent（被创建的提示信息对象是这个表单类的一个实例）。oParent 处理对客户端的回调。 |

表5：frmAlert 属性的 Assign 方法

我们还需要以下一些自定义方法（表6）：

| 方法 | 描述 |
|---|---|
| SetupForm() | 从表单的 Init() 中调用。调用此方法计算 nScreenHeight 和 nScreenWidth，并计算表单的位置。 |
| RenderForm() | 基于通过 oParams 对象传递的参数“绘制”表单。 |
| HideForm() | 启动淡入淡出计时器，使表单淡出。 |
| GetScreenHeight() | 设置 nScreenHeight 属性 |
| GetScreenWidth() | 设置 nScreenWidth 属性 |

表6：frmAlert 的自定义方法

至此，我们已经完成了表单的属性和方法设置，接下来需要向其中添加一些控件。因为已经使用了可视类（DESKALERT.VCX）来创建表单，SO，我们可以在其中创建用于表单的控件类。

让我们先回顾一下表1的内容，首先，我们需要创建用于表单的按钮类。这里我并不打算使用 CommandButton ，而是使用包含 Image 的 Container 来创建。在 IDE 中，它看上去就像这样:

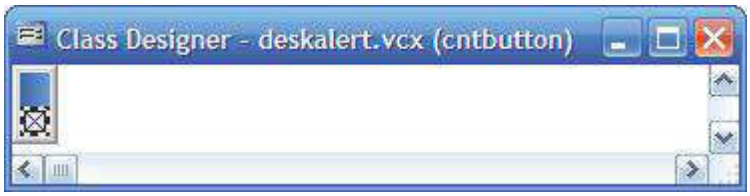



图2：cntButtom

以下是它的一些属性设置：

```
Width = 15
Height = 15
BackStyle = 0
BorderWidth = 0
MousePointer = 15
BackColor = RGB(192,192,192)
BorderColor=RGB(128,128,128)
Name = cntButton
```

对于其中的 Image ，其属性设置如下：

```
BackStyle = 0
Height = 11
Left = 2
MousePointer = 15
Top = 2
Width = 11
Name = "imgImage"
```

我们需要的效果是让按钮显示为一个图形（如图1中的关闭和设置按钮那样），当鼠标移过它时，它的背景色会改变并显示边框。因此，需要在 Container 的 MouseEnter() 和 MouseLeave() 方法中添加一些代码（以实现这个效果）。

在 MouseEnter() 中，我们将背景色更改为浅灰色，然后在使容器的边框呈现深灰色，并将 BackStyle 设置为不透明。

```
Procedure MouseEnter
    Lparameters nButton, nShift, nXCoord, nYCoord
        With This
            .BackColor = Rgb(192,192,192)
            .BorderWidth = 1
            .BackStyle = 1
        Endwith
Endproc
```

当 MouseLeave（）事件发生时，我们将BackColor设置为与表单相同，删除边框，并将 BackStyle 设置为透明：

```
Procedure MouseLeave
    Lparameters nButton, nShift, nXCoord, nYCoord
        With This
            .BackColor = Thisform.BackColor
            .BorderWidth = 0
            .BackStyle = 0
        Endwith
Endproc
```

Image 控件的 MouseEnter() 和MouseLeave() 中也包含有代码，这些代码基本上会“冒泡”到 Container 的 MouseEnter() 和 MouseLeave()：

```
Procedure imgImage.MouseEnter
    Lparameters nButton, nShift, nXCoord, nYCoord
        This.Parent.MouseEnter()
EndProc

Procedure imgImage.MouseLeave
    Lparameters nButton, nShift, nXCoord, nYCoord
        This.Parent.MouseLeave()
Endproc
```

我尽量不在“事件”中添加太多代码，因此我将向 Container 添加一个抽象的自定义方法并将其命名为 OnClick()。 然后我将以下代码添加到 Container 的 Click() 事件中：

```
Procedure Click
    This.OnClick()
Endproc
```

并使用以下代码填充到 Image 的 Click() 事件中，以“冒泡”到容器：

```
Procedure imgimage.Click
    This.Parent.OnClick()
Endproc
```

我将此类中的 OnClick() 方法置空，因为它是一个抽象方法。在我们将子类对象添加到表单后，我们将在表单上填写此方法。

由于我要做的是完整的“更改 BackColor 并添加边框”，因此我将以下代码添加到 Container 的 MouseDown() 和 MouseUp() 事件中。

在 MouseDown() 中，我们将 BackColor 更改为“更暗”的灰色，将 BorderColor 更改为黑色，并使 BackStyle 变为不透明：

```
Procedure MouseDown
    Lparameters nButton, nShift, nXCoord, nYCoord
        With This
            .BackColor = Rgb(177,177,177)
            .BorderColor = Rgb(0,0,0)
            .BackStyle = 1
        Endwith
Endproc
```

然后在 MouseUp() 中，我们将恢复 BackColor 为浅灰色，BorderColor 为深灰色和并设置 BackStyle 透明：

```
Procedure MouseUp
    Lparameters nButton, nShift, nXCoord, nYCoord
        With This
            .BackColor = Rgb(192,192,192)
            .BorderColor = Rgb(128,128,128)
            .BackStyle = 0
        Endwith
Endproc
```

当然，我们同样要将代码添加到 Image 的 MouseDown() 和 MouseUp() 以实现“冒泡”到容器：

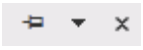
```
Procedure imgimage.MouseDown
    Lparameters nButton, nShift, nXCoord, nYCoord
        This.Parent.MouseDown()
Endproc

Procedure imgimage.MouseUp
    Lparameters nButton, nShift, nXCoord, nYCoord
        This.Parent.MouseUp()
Endproc
```

我还将以下代码添加到 Container 的 Init() 事件中：

```
Procedure Init
    With This
        .BackColor = Thisform.BackColor
        .SetAll("ToolTiptext",.ToolTipText)
    Endwith
Endproc
```

我们现在就拥有了一个用于表单的按钮基类，后面我们会用到它。我们要做的是基于我们刚刚创建的 cntButton 基类，创建一个设置按钮和一个关闭按钮。我们还将创建一个 Push-Pin 按钮，允许用户将提示“固定”到桌面。

要创建设置、关闭和 Push-Pin 按钮，我们需要做的就是为每个子类对象更改 Container 上 Image 对象的 Picture 属性，并添加 ToolTipText（对于关闭按钮，其设置为：“点击此处关闭提示”）。 这是他们在提示中的样子：



任务在提示表单上也表现为按钮，它也是从 cntButton 中派生出来的。和（其他按钮）唯一真正的区别是 Container （18x18）和 Image（16x16）的高度和宽度都更大一些。

与其他“按钮”的情况一样，我们将在设计将任务按钮拖放至提示表单，并在 OnClick() 方法中放置适当的代码。

现在，我们为标题、主题和描述标签创建适用于提示表单的类。

标题只是一个带有以下属性的 Label 控件：

```
AutoSize = .F.
FontBold = .T.
FontName = "Tahoma"
FontSize = 8
BackStyle = 0
Caption = "lblTitle"
Height = 15
Visible = .F.
Width = 250
Name = "lblTitle"
```

主题标签具有类似的属性：

```
FontName = "Tahoma"
FontSize = 8
BackStyle = 0
Caption = "lblSubject"
Height = 15
Visible = .F.
Width = 250
Name = "lblSubject"
```

而描述标签除了具有类似的属性之外，我们还为其添加了一个名为 lIsALink 的自定义属性一个 Assign() 方法：

```
FontName = "Tahoma"
FontSize = 8
WordWrap = .T.
BackStyle = 0
Caption = "lblDetails"
Height = 30
Visible = .F.
Width = 250
lIsALink = .F.
Name = "lbldetails"

Procedure lIsALink_Assign
    Lparameters vNewVal
    This.lIsALink = m.vNewVal

    With This
        If .lIsALink
            .ForeColor = Rgb(0,0,255)
            .MousePointer = 15
        Else
            .ForeColor = Rgb(0,0,0)
            .MousePointer = 0
        Endif
    Endwith
Endproc
```

如果为描述标签分配一个操作（这样将产生一个结果值），那么它将被视为'链接'——这将使它看起来好像是一个超链接。 否则，它仅仅是一个简单的文本。

现在，提示表单仅剩下两个 UI 部分：标题栏和图标。其实就是两个简单的 Image 控件，它们并没有任何的特殊功能。

现在 UI 部件已经就位，我们将在表单中添加两个 Timer 控件：tmrFader 和 tmrWait。 tmrFader 处理“淡入”和“淡出”，而 tmrWait 使提示表单在配置规定的时间内保持在屏幕上。 我们将在下一节中更详细地介绍这两个计时器。

我们现在已经构建并准备好了 UI 类。 以下是类浏览器中所展示的：

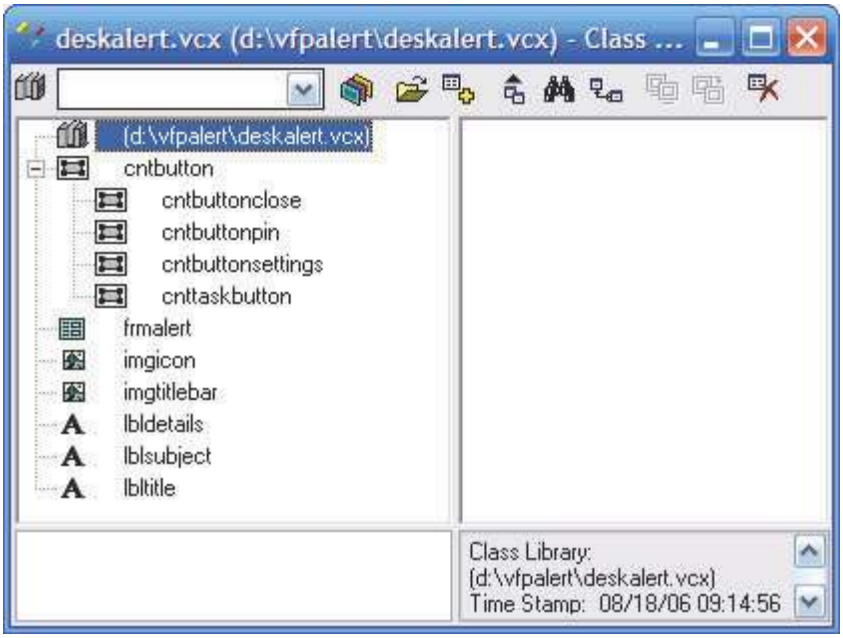



图3：类浏览器中的 DESKALERT.VCX

这是在 VFP IDE 中提示表单的设计状态截图：


图4：VFP IDE 中的 frmAlert

现在让我们仔细看看表单淡入和淡出。

在提示表单执行 Setup() （译者注： frmAlert 类中并不存在 Setup() 自定义方法，代码存在于 Init() ）时，我们检查是否运行于 Windows 2000 或更高版本的 OS ：

```
ThisForm.lCanFade = (VAL(OS(3))>=5)
```

lCanFade_Assign() 方法执行以下操作：

```
Lparameters vNewVal
    This.lCanFade = m.vNewVal
    If This.lCanFade
        ** The SetWindowLong tells the system
        ** to make this form a "layered window".
        ** See the "Transparent Forms"
        ** solution sample in VFP for the 'code'.
        _SetWindowLong(This.HWnd, -20, 0x00080000)
```

```
            ** SetLayeredWindowAttributes
            _SetLayeredWindowAttributes(This.hWnd, 0, 0, 2)
        Endif
```

当然，此代码假定API声明已经完成。我们稍后会对 API 的声明予以介绍。Assign 方法告诉系统提示表单将是一个“分层”窗口，并设置属性以使表单完全透明。

在 SetupForm（）方法（译者注： frmAlert 类中并不存在 SetupForm() 自定义方法，代码存在于 Init() ）中，我们计算表单的'淡入淡出因子'，将表单的状态设置为'出现'，并启动 tmrFader 计时器（如果你查看类代码，你会发现下面的代码是包裹在一个WITH/ ENDWITH结构中）：

```
    ** Start the tmrFader, which causes the
    ** form to appear gradually with API calls.
    .nFadeFactor = 255/(1000/lnInterval)
    If .lCanFade
        .nStatus = 1
        .tmrFader.Enabled = .T.
    Endif
```

tmrFader 的 Timer() 事件每20毫秒（或一秒钟50次）触发一次。 每次触发时，都会降低表单的透明度，从而提供“淡入”效果。 当表单“淡出”时，tmrFader 会提高透明度：

```
    Do Case
        Case .nStatus = 0 && Disappearing
            .nTransparency = .nTransparency - .nFadeFactor
            _SetLayeredWindowAttributes(.HWnd, 0, .nTransparency, 2)
            If .nTransparency <= 1
                This.Enabled = .F.
                .Release()
            Endif

        Case .nStatus = 1 && Appearing
            .nTransparency = .nTransparency + .nFadeFactor
            _SetLayeredWindowAttributes(.HWnd, 0, .nTransparency, 2)
            If .nTransparency >= 255 * (.oParent.nPercent/100)
                This.Enabled = .F.
                .tmrWait.Enabled = .T.
            Endif
    EndCase
```

当表单“淡出”时，计时器将在达到小于或等于1的透明度级别时自行禁用，然后调用表单的 Release()。

当“淡入”时，一旦达到配置的透明度，计时器将自行禁用。 然后它将启用 tmrWait 计时器，这会使提示表单“停留”在屏幕上达到配置的秒数（等待用户交互）。

实质上，所有淡入和淡出活动都是由 tmrFader 驱动的。

**OnClick() 方法**

我们为拖放到提示表单上的大多数 UI 类创建了 OnClick() 方法。我们来看看它们的代码:

关闭按钮、任务1按钮、任务2按钮和描述标签都在其中用代码来设置提示表单的 nResult 属性，然后调用表单的 HideForm() 方法。

如果您还记得，提示表单将返回nResult的五个可能值之一:

-1：警报超时（用户未与表单交互）

1：用户单击“关闭”按钮

2：用户单击“描述”链接

3：用户选择任务1

4：用户选择任务2

所以，在关闭按钮的 OnClick() 方法中，我们只需设置 ThisForm.nResult = 1，然后调用 ThisForm.HideForm() 。

HideForm（）方法是如何执行的呢？它将表单的“状态”设置为0（消失）并启用 tmrFader 计时器，这会提高透明度，直到提示表单消失。

现在我们为提示表单留下了三个自定义方法：GetScreenHeight，GetScreenWidth 和 RenderForm。

GetScreenHeight（）和GetScreenWidth（）均使用 Windows 中的 API 函数 SystemParametersInfo 设置物理屏幕的高度和宽度，同时考虑到任务栏的存在。

在捕获屏幕高度和宽度值之后，但在为 lCanFade 属性分配值之前，从 SetupForm() 方法调用 RenderForm()。 RenderForm() 接受传递给表单的参数并设置各种属性（如标题，主题和描述的 Caption）。它还决定是否显示认为任务按钮和设置按钮，是否使描述标签显示为“链接”，以及用于表单的图标。

DAS的一个要求是允许警报可移动。但是，我们没有标题栏。如何在没有标题栏的情况下移动表单？ Windows API 函数！感谢VFP网站http://www.news2news.com/vfp 上Win32功能的优秀示例，我们可以通过在表单的 MouseDown() 事件中放置以下代码

来轻松移动表单:

*(译者注：此网站已关闭。其所有资料均迁移至：https://github.com/VFPX/Win32API，其中文版本请访问：https://github.com/vfp9/VFPX_CN)*

```
Procedure MouseDown
    Lparameters nButton, nShift, nXCoord, nYCoord
        If nButton = 1
            Local hWindow
            hWindow = GetFocus()
            = releaseCapture()
            = SendMessage(hWindow, WM_SYSCOMMAND, MOUSE_MOVE, WM_NULL)
            = SendMessage(hWindow, WM_LBUTTONUP, 0, 0)
        Endif
Endproc
```

和之前一样，这段代码假设 API 函数已经被声明，此后不再就此问题予以提示。

如果用户单击表单上的某个控件但想要移动表单怎么办？简单！对于没有定义 OnClick() 方法的表单上的每个对象，在 MouseDown() 事件中调用 ThisForm.MouseDown() 。这些对象包括 imgTitleBar，imgIcon，lblTitle，lblSubject 和 lblDetails（如果 lblDetails 不是'链接'的话）。由于显而易见的原因，任何具有 OnClick() 的对象都不应该将其 MouseDown() 冒泡到表单的 MouseDown() 。

我们还没有谈及的是 PushPin 按钮和设置按钮的 OnClick() 方法。PushPin按钮更改其 Picture 属性以使 Pin 显示为“push”，并且“冻结” tmrWait 计时器。这允许用户在屏幕上持续显示提示表单。

设置按钮将创建一个菜单，其中包含用于配置DAS设置的菜单栏，还将包括任务1和任务2的栏（如果适用）。**（译者注：事实上，作者在最后的成品中，仅仅是从设置按钮调用了一个配置DAS的表单）**

除了一些其他的“设置”功能，可以已经可以将提示表单放在屏幕上的特定位置，此时我们已经完成了提示表单!

**AlertManager 类**

之前，我曾说过，我对 DAS 的最终目标是能够让我的应用实例化一个“全局”提示管理器，并在必要时发出提示。为了实现这个目标，我们需要创建一个 lertManager 类。

创建 AlertManager 类后，我可以向我的应用程序对象添加一个新的“提示管理器”属性（或者，您可以创建一个全局变量）：

```
oAlertMgr = .NULL.
```

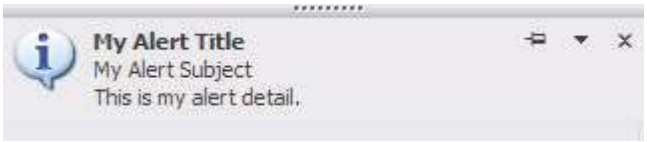
在应用程序启动时，我将赋予它一个实例：

```
MyApp.oAlertMgr = CREATEOBJECT("vfpalert.AlertManager")
```

当我需要显示提示信息时，我只需要像下面这样的代码：

```
loAlert = MyApp.oAlertMgr.NewAlert()
loAlert.Alert("This is my alert detail.",64,"My Alert Title", ;
         "My Alert Subject")
```

GOOD!提示表单开始淡入到桌面：




基于 Session 的 AlertManager 类将是 OLEPUBLIC 。 这是DAS的主要编程接口。

AlerManager 类拥有以下自定义属性：

| 属性 | 描述 |
|---|---|
| Alerts | 一个集合对象，它引用系统中当前的每个警报 |
| nWait | 在淡出之前提示应出现在屏幕上的时间（默认为10秒） |
| nPercent | 提示的“不透明”百分比（默认值为90） |
| lSound | 如果我们想在提示出现时发出声音，设置为 .T。（默认）；否则，.F。 |
| cSound | 我们要播放的声音的文件名（默认为ALERT.WAV） |
| cSettingsPath | 存储设置的“可写”文件夹的完整路径 |
| lCleanMemory | .T。 减少内存消耗(默认为.F。)**（译者注：此属性在成品中被删除？）** |

AlerManager 类拥有以下自定义方法：

| 方法 | 描述 |
|---|---|
| Setup() | 从 Init() 中调用，其调用下面两个方法并创建 Alerts 集合 |
| APIDeclare() | 因为我们需要各种API声明，所以在此处统一声明 |
| ReadSettings() | 从配置文件 DACONFIG.XML 读取设置（如果存在）；此外，载入默认设置 |
| NewAlert() | 创建一个新的提示，将其添加到集合中，并将警报的对象引用返回给调用程序 |
| WriteSettings() | 获取用户提供的配置信息（通过“设置”表单）并将其写入DACONFIG.XML文件 |
| SetSettingsPath() | 以完整路径发送到“可写”文件夹，其中应存储CONFIG.XML文件 |

当实例化 AlertManager 类时，Init() 调用 Setup() 方法，该方法又调用

APIDeclare() 和 ReadSettings() 方法。同时，还会创建一个提示集合。

我们在 APIDelare() 方法中执行所有API声明，即使是我们已经创建的提示表单所需的所有声明也是在此处声明。这就是为什么提示表单中所有与 API 相关的代码都假定已经进行了声明的原因。

此时，AlertManager 正在等待我们创建单独的提示（使用 NewAlert() 方法）。

**Alert 类**

Alert（Session）类的作用是真正完成创建提示信息的大部分工作。 它包含以下自定义属性:

| 属性 | 描述 |
|---|---|
| oMgr | 对实例化此提示的 AlertManager 的对象引用 |
| oCallback | 引用被回调的对象（我们将提示的结果返回给调用者） |
| oAlertForm | 显示在屏幕上的实际提示表单 |
| nResult | 其值返回给 oCallback |
| nWait | 与 AlertManager.nWait 相同 |
| nPercent | 与 AlertManager.nPercent 相同 |
| lSound | 与 AlertManager.lSound 相同 |
| cSound | 与 AlertManager.cSound 相同 |

Alert 类也具有以下自定义方法:

| 方法 | 描述 |
|---|---|
| Setup() | 从 oMgr 读取配置信息，并将值存储到相应的属性中 |
| SetCallback() | 处理设置 oCallback 的引用 |
| Alert() | 分析参数并创建 oAlertForm 的主要方法 |
| nResult_Assign | 触发 oCallback 事件处理程序 |

当 AlertManager 实例化新提示时，Alert 类实例会设置 oMgr 属性并调用 Setup() 方法。Setup() 方法检索配置设置，并使用参数 NULL 调用 SetCallback() 。

此时，Alert 类实例正在等待客户端应用程序调用 SetCallback() 方法，因此提示可以在具有结果后触发 oCallback 事件处理程序。设置 oCallback 后，Alert 类实例会等待调用 Alert() 方法。

当我创建一个新警报并调用 Alert() 方法时，我将传递一些参数，以便Alert对象可以动态创建警报表单。

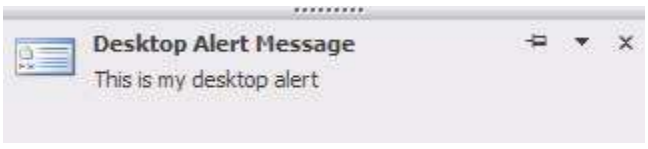
这是我提出的 Alert() 方法参数:

| 参数 | 描述 |
| --- | --- |
| cDetails | 提示的“描述”部分的实际文本 |
| nAlertType | 要创建的提示类型。普通提示将包含描述信息，但描述信息不是超链接。当鼠标在其上移动时，具有链接的提示将使**描述**看起来像超链接。<br>`#DEFINE DA_TYPEPLAIN 0 && 无链接，无任务`<br>`#DEFINE DA_TYPELINK 1 && 链接，无任务`<br>`#DEFINE DA_TYPETASK 2 && 链接，一个任务`<br>`#DEFINE DA_TYPEMULTI 4 && 链接，两个任务` |
| cAlertTitle | 标题文字。 默认为“桌面提示消息” |
| cAlertSubject | 主题的文本。 默认为空字符串 |
| cIconFile | 要使用的图标的文件名（如果 nAlertType = 128）。默认为以下图标（包含在项目中，停止，问题，感叹号和信息图标）：<br>  <br>如果 nAlertType = 128（自定义图形），则此参数应为要使用的图标的路径和文件名。<br>我们还可以使用其他“内置”图标，这些图标也在 nAlertType 参数中确定（就如MESSAGEBOX一样）：<br>`** 提示图标`<br>`#DEFINE DA_ICONDEFAULT 8 && 默认图标`<br>`#DEFINE DA_ICONSTOP 16 && 关键消息`<br>`#DEFINE DA_ICONQUESTION 32 && 问号`<br>`#DEFINE DA_ICONEXCLAMATION 48 && 警告`<br>`#DEFINE DA_ICONINFORMATION 64 && 信息`<br>`#DEFINE DA_ICONCUSTOM 128 && 用户自定义` |
| cTask1 | 文本，显示为任务一的 ToolTip |
| cTask1Icon | 任务一的图标 |
| cTask2 | 文本，显示为任务二的 ToolTip |
| cTask2Icon | 任务二的图标 |

cDetails 是唯一必需的参数。使用下面的命令并与之前的示例予以比较：

```
loAlert.Alert("This is my desktop alert.")
```

此命令结果：




这将显示默认标题和默认图标，描述信息行为纯文本（无超链接）。有时我们可能想要隐藏“设置”按钮，因此我们在 VFPALERT.H 中定义了一个常量：

```
#DEFINE DA_NOSETTINGS 4096 && 不显示设置按钮
```

在 nAlertType 参数中使用 4096 将导致设置按钮对用户隐藏。这种类型的参数

系统（与已知的#DEFINEd元素结合使用）可以在解析参数值时以极低的复杂性成本实现更好的灵活性。

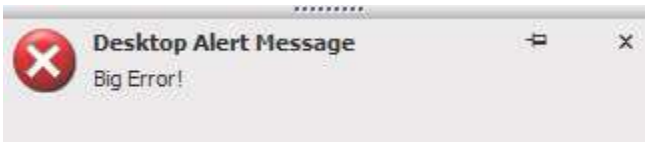
以下是 nAlertType 参数的完整列表：

```
** VFP 桌面提示参数
** 提示信息类型
#DEFINE DA_TYPEPLAIN    0 && 无链接，无任务
#DEFINE DA_TYPELINK     1 && 链接，无任务
#DEFINE DA_TYPETASK     2 && 链接，一个任务
#DEFINE DA_TYPEMULTI    4 && 链接，两个任务

** 提示图标
#DEFINE DA_ICONDEFAULT      8 && 默认图标
#DEFINE DA_ICONSTOP         16 && 关键图标
#DEFINE DA_ICONQUESTION     32 && 问号
#DEFINE DA_ICONEXCLAMATION 48 && 警告
#DEFINE DA_ICONINFORMATION 64 && 信息
#DEFINE DA_ICONCUSTOM       128 && 自定义

** 这里预留一部分位置以便“内置”一些其他的“默认”图标。

#DEFINE DA_TASKICONDEFAULT 2048 && 默认任务图标

** 提示选项
#DEFINE DA_NOSETTINGS  4096 && 不显示设置按钮
```

我们可以像在 MESSAGEBOX 中那样使用 nAlertType ：

```
loAlert.Alert("Big Error!",DA_TYPEPLAIN + DA_ICONSTOP + DA_NOSETTINGS)
```

结果就是：




由于每个DEFINEd值都对应一个BIT值，因此Alert（）方法执行一系列BITTEST来确定我们想要的值。

由于每个预定义值都对应一个 BIT 值，因此 Alert() 方法执行一系列BITTEST确定我们想要的值。

```
** Do we want to hide the settings button?
llHideSettings = Bittest(nAlertType,12)

** What kind of ICON do we want?
For i = 7 To 3 Step -1
    If Bittest(nAlertType,i)
        If i = 5
            ** Bit 5 is 32 (ICONQUESTION). We need to
            ** check to see if the next bit is "on".
            ** This will mean 48 was passed in, which
            ** is the value of ICONEXCLAMTION.
            If Bittest(nAlertType,i-1)
                ** Yes, bit 4 = 16. Add it to the 32.
                lnIcon = (2^i) + (2^(i-1))
                Exit
            Else
                ** Nope, a 32 was passed in.
                lnIcon = 2^i
                Exit
            Endif
        Else
            lnIcon = 2^i
            Exit
        Endif
    Endif
Endfor

If lnIcon = 0
    lnIcon = DA_ICONDEFAULT
Endif
```

一旦我们有图标可“设置”，我们就会确定提示类型：

```
** Now, check the Alert Type, which
** will be in either BIT 2 or 1.
Do Case
    Case Bittest(nAlertType,2)
        ** MultiTask (4)
        lnType = DA_TYPEMULTI
    Case Bittest(nAlertType,1)
        ** Task (2)
        lnType = DA_TYPETASK
    Case Bittest(nAlertType,0)
        ** Link (1)
        lnType = DA_TYPELINK
    Otherwise
```

```
        ** Plain (0)
        lnType = DA_TYPEPLAIN
Endcase
```

此时，Alert() 方法将创建一个参数对象并传递给我们的提示表单。我使用 Empty 类充当参数对象，并在其中添加我需要的属性。

我们需要遍历所有剩余参数，并在必要时填写默认值：

```
** We have the TYPE, and the ICON. Create a
** parameter Object to pass to the alert form.
** We'll use the EMPTY class for the parameter
** object, using ADDPROPERTY() to, you know,
** add the properties.

Local loParams As Object
loParams = Createobject("Empty")

AddProperty(loParams,"Name",lcName)
AddProperty(loParams,"Type",lnType)
AddProperty(loParams,"Icon",lnIcon)
AddProperty(loParams,"IconFile","")
AddProperty(loParams,"Title","")
AddProperty(loParams,"Subject","")
AddProperty(loParams,"AlertText",cAlertText)
AddProperty(loParams,"HideSettings",llHideSettings)
AddProperty(loParams,"HideClose",llHideClose)
AddProperty(loParams,"HidePin",llHidePin)

** If the 'type' includes tasks, add the
** Task properties.
If lnType > DA_TYPELINK
    AddProperty(loParams,"Task1","")
    AddProperty(loParams,"Task1Icon","")
    If lnType > DA_TYPETASK
        AddProperty(loParams,"Task2","")
        AddProperty(loParams,"Task2Icon","")
    Endif
Endif

** We need to "walk" through the rest of the parameters
If Vartype(cAlertTitle) # "C"
    If Vartype(cAlertTitle) = "L"
        loParams.Title = DA_DEFAULTTITLE
    Else
        loParams.Title = Substr(Transform(cAlertTitle),1,30)
```

```
    Endif
Else
    loParams.Title = Substr(cAlertTitle,1,30)
Endif

If Vartype(cAlertSubject) # "C"
    If Vartype(cAlertSubject) = "L"
        loParams.Subject = ""
    Else
        loParams.Subject = Substr(Transform(cAlertSubject),1,50)
    Endif
Else
    loParams.Subject = Substr(cAlertSubject,1,50)
Endif

If Vartype(cIconFile) # "C"
    loParams.IconFile = DA_DEFAULTICONFILE
Else
    If File(cIconFile)
        loParams.IconFile = cIconFile
    Else
        loParams.IconFile = DA_DEFAULTICONFILE
    Endif
Endif

** Check for Tasks and Task Icons
If lnType > DA_TYPELINK
    ** The next param should be the Task1
    If Vartype(cTask1) # "C"
        If Vartype(cTask1) = "L"
            loParams.Task1 = ""
        Else
            loParams.Task1 = Transform(cTask1)
        Endif
    Else
        loParams.Task1 = Alltrim(cTask1)
    Endif

    ** The next param should be the Task1Icon
    If !Empty(Alltrim(loParams.Task1))
        If Vartype(cTask1Icon) # "C"
            loParams.Task1Icon = DA_DEFAULTTASKFILE
        Else
            If File(cTask1Icon)
```

```
                loParams.Task1Icon = cTask1Icon
            Else
                loParams.Task1Icon = DA_DEFAULTTASKFILE
            Endif
        Endif
    Endif

    ** If a Multi Task, the next param
    ** should be Task2
    If lnType >= DA_TYPEMULTI
        If Vartype(cTask2) # "C"
            If Vartype(cTask2) = "L"
                loParams.Task2 = ""
            Else
                loParams.Task2 = Transform(cTask2)
            Endif
        Else
            loParams.Task2 = Alltrim(cTask2)
        Endif

        If !Empty(Alltrim(loParams.Task2))
            If Vartype(cTask2Icon) # "C"
                loParams.Task2Icon = DA_DEFAULTTASKFILE
            Else
                If File(cTask2Icon)
                    loParams.Task2Icon = cTask2Icon
                Else
                    loParams.Task2Icon = DA_DEFAULTTASKFILE
                Endif
            Endif
        Endif
    Endif
Endif
```

***这是有很多代码。 正如我之前所说，这看上去也不怎么滴吗。这里的目标是在创建提示表单之前计算出所有参数。*

我们已经解析了参数并将它们放入参数对象中。现在，让我们创建提示表单：

```
** Create an instance of the alert form, passing in the
** loParams object and a reference to THIS alert.
This.oAlertForm = Createobject("frmAlert",loParams,This)
```

此时，Alert 对象已完成其工作。它现在只需等待来自提示表单的结果，然后将结果传递回oCallback，并自行释放（也就是说，它将其从 AlertManager 的 Alerts 集合中删除）。

**将提示结果返回给调用者**

将提示结果返回到调用者最棘手的部分是 Alert (基于 Session) 类中的 SetCallback () 方法。

SetCallback () 将尝试通过接口定义将 DAS 与调用应用程序 "链接"。接口由一个名为 AlertEvents 的类处理:

```
Define Class AlertEvents As Session OlePublic
    Procedure AlertResult (tnResult As Integer) As Integer
    Endproc
Enddefine
```

这是发生的事情的顺序:

- AlertManager 创建一个 Alert 实例。
- 调用者（或对象）调用 Alert.SetCallback()，传入对自身的对象引用。
- 提示表单是根据调用者传入的参数“构建”的，这些参数由调用者解析 Alert() 。
- 提示表单将提示结果返回到 Alert 类
- Alert 类使用 nResult_Assign（）方法“触发”调用对象的 AlertResult() 方法

这是来自 Alert.nResult_Assign() 的代码:

```
Procedure nResult_Assign
    Lparameters vNewValue
    This.nResult_Assign = m.vNewValue

    Local loException As Exception
    loException = .Null.

    If This.nResult_Assign # 0
        Try
            This.oCallBack.AlertResult(This.nResult_Assign)
        Catch To loException
        Endtry
    Endif
Endproc
```

在 Alert 的 SetCallback() 方法中设置对 oCallback 的引用。 我们将把调用包装到 TRY / CATCH / ENDTRY 中的 oCallback.AlertResult()，以防我们遇到某种错误。

这是 SetCallback() 方法的代码:

```
  Procedure SetCallback (toCallBack As Variant) As VOID ;
          HELPSTRING "Gets/Sets a reference to the client event handler"

      Local loException As Exception
      loException = .Null.

      If Isnull(toCallBack)
          ** Dummy instance that does nothing: virtual function
          This.oCallback = Createobject("AlertEvents")
      Else
          If Vartype(toCallBack) # "O"
              Comreturnerror("Function SetCallback()","Callback object
must be an Object")
          Endif

          Try
              This.oCallback = ;
          GETINTERFACE(toCallBack,"Ialertevents","vfpalert.AlertManager")
          Catch To loException
          Endtry

          If !Isnull(loException)
              ** An exception was created on the GETINTERFACE line.
              ** Reference the object that came in, and hope for the best.
              This.oCallback = toCallBack
          Endif
      Endif
  Endproc
```

还记得 AlertEvents 类有一个名为 AlertResult 的方法吗？ 这就是回调的问题。

由于我们发布了一个名为 iAlertEvents 的接口（来自此OLEPUBLIC类），因此任何COM客户端都可以实现接口，并自动响应 AlertResult！

如果我们得到某种错误（或者如果我们使用的 DAS 没有能够实现接口），我们只需将 oCallback 设置为传入的对象。***这假设 oCallback 将有一个名为 AlertResult 的方法！！！***

对不起，这是我为达到目的所想到的最好的实现方法了。我说过我写的代码并不怎么滴。

娱乐一下，让我们创建一个小程序，看看我们可以使用的不同结果/选项。 程序运行后，我们会在屏幕上看到以下三个警报（从桌面的右下角开始）：


图5：三个提示的演示

请记住，提示表单在 SetupForm() 方法中计算其位置，但我们没有在本文档中介绍它。（**译者注：在成品中也没有哦**😉）

以下是启动这三个提示所需的代码：

```
Local oMgr As vfpalert.AlertManager, ;
o As vfpalert.AlertManager, ;
o2 As vfpalert.AlertManager, ;
o3 As vfpalert.AlertManager

** Create an event handler for the first alert (o)
x=Createobject("myregularclass")

** Create an event handler for the second alert (o2)
Y=Newobject("myclass")

** We won't do one for the third alert (o3)

** Create the AlertManager
oMgr = Newobject("vfpalert.AlertManager")

** First alert
o2 = oMgr.NewAlert()

** Second alert
o2 = oMgr.NewAlert()

** Third alert
o3 = oMgr.NewAlert()
```

```
    ** SetCallback() for the first two alerts
    o.SetCallback(x)
    o2.SetCallback(Y)

    ** Launch the first alert form
    o2.AlertManager("This is a test of the alert system.",64,"First Alert")
    Inkey(.5,"hc")

    ** Second alert form
    o2.AlertManager("This is a test of the alert system.",65,"Second
Alert")
    Inkey(.5,"hc")

    ** Third alert form
    o3.AlertManager("This is a test of the alert system.",129,"Third
Alert","This one has no callback","F:\test.ico")

    ** Just for demonstration purposes, 'hang' the system
    ** long enough to see the results.
    WAIT WINDOW "" TIMEOUT 20

    Define Class myregularclass As Session
        Procedure AlertResult(tnResult As Number) As Number
            Messagebox("You selected: " + Transform(tnResult) + " using
    AlertResult() In AlertManager Normal Class",64,"MyRegularClass")
        Endproc
    Enddefine

    Define Class myclass As Session OlePublic
        Implements Ialertevents In "vfpalert.AlertEvents"
        Procedure Ialertevents_AlertResult(tnResult As Number) As Number
            Messagebox("You selected: " + Transform(tnResult) + " in the
  Ialertevents.",64,"iAlertEvents_AlertResult")
        Endproc
     Enddefine
```

对于提示，我执行了以下操作:

提示1（o）：我什么都不做。 我会在默认的十秒后让提示“超时”

警报2（o2）：我点击详细信息'链接'

警报3（o3）：我点击关闭按钮。

以下是我对警报的操作结果:

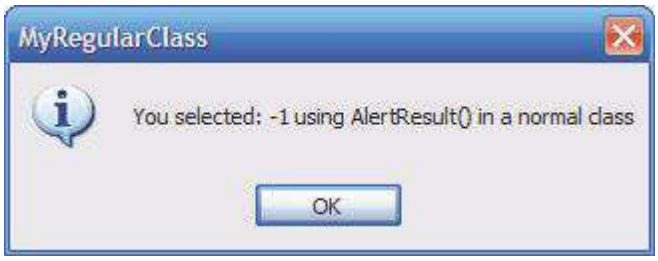



图6：提示1的结果


图7：提示2的结果

如您所见，提示3(o3)没有结果。为什么？因为它没有事件处理程序。 我们从未做过 SetCallback()，因为 Alert 初始化时将回调对象设置为.NULL。它创建一个 AlertEvents 对象，实际上什么都不做。提示出现，用户进行交互，但结果是没有任何反应。

**结论**

虽然 DAS 非常有用，但重要的是要谨慎使用。当我需要向用户提供信息而**不需要**响应时，我发现最好使用它们。实际使，这种情况下没有必要使用模态对话框。

使用VFP 7或更高版本，您可以直接从对象浏览器拖放 iAlertEvents 界面，并准备好在应用程序中接收提示结果。对于VFP 6，您需要使用 AlertResult() 方法创建事件处理程序对象（您甚至可以将该方法添加到现有对象，并将该对象传递给 Alert 的 SetCallback() 方法）。

最后，我们将使用最少量的代码直接从PowerPoint演示文稿创建提示：

```
Dim oAlertMgr As New vfpalert.AlertManager
Dim oAlert As Object
Dim Alert As Variant
Dim oCB As Variant

Public Sub AlertResult(ByVal nResult As Integer)
   Me.txtResult.Text = "The result was " &
Trim(Str(nResult))
```

```
End Sub

Private Sub cmdGo_Click()
   Set oCB = Me

   Set oAlert = oAlertMgr.NewAlert()
   oAlert.SetCallback (oCB)
   Alert = oAlert.Alert(Me.txtDetails.Text,
Val(Me.txtStyle.Text), Me.txtTitle.Text,
Me.txtSubject.Text, Me.txtIcon.Text)
 End Sub
```

通过利用 Visual FoxPro 的惊人力量，我们可以做任何事情!