第二章 Intel Visual Fortran 编译器的使用

在窗口程序设计之前，让我们先了解一下 Intel Visual Fortran 编译器的基本使用方法和编译器中的项目类型。

2-1 Intel Visual Fortran 编译器简介

Fortran 是一种高级的计算机编程语言，所有 Fortran 程序源码必须经过编译器的编译、链接，才能被翻译成计算机所能识别的机器码，从而完成源程序设定的任务。

Fortran 语言的编译器较多，PC 中 Windows 平台上常见的有：

■ Fortran PowerStation 4.0 微软公司将 Fortran90 集成到 Developer Studio 开发环境中之后推出的 Fortran 编译器，真正实现了 Fortran 的可视化编程；

■ Digital Visual Fortran 微软公司和数据设备公司(DEC)联合开发的功能更强大的 Fortran 编译器；

■ Compaq Visual Fortran 数据设备公司(DEC)和康柏公司(Compaq)合并后推出的 Fortran 编译器以及康柏公司并入惠普后推出的最新版本；

■ Intel Visual Fortran 11.x 惠普公司将 Windows 下的编译器转售给 Intel 公司后，由 Intel 公司开发的 Fortran 编译器，目前是最新版。

■ Salford Fortran 95 由 silverfrost 公司开发的能用于基于 Win32 平台和.net 平台的应用程序开发，支持完整的 Fortran95 语法和部分 Fortran2003 语法。

其实，Fortran 编译器还有很多，如 Lahey Fortran、Absoft Fortran 和 OpenWatcom 等。从目前的使用情况来看，前四个编译器在 Windows 平台上最为常见。考虑到 Fortran 编译器目前的发展情况以及各种编译器版本使用的广泛性，本书所有的程序都基于 Intel Visual Fortran 11.x 编译器。

跟 C++一样，Fortran 语言本身没有提供图形界面输出方面的功能，所以为了用 Fortran 编写 Windows 图形接口程序，必须借助编译器提供的扩展功能。Intel Visual Fortran 编译器封装了几乎完整的 Win32 API 和 OpenGL 函数，提供了良好、稳定的编程接口，所以，借助 Intel Visual Fortran 编译器，Fortran 既能完成 UI 设计，也能胜任图形编程，从这个意义上讲，C 能做到的，Fortran 也能做到。

2-2 Intel Visual Fortran 编译器的使用

为了编写图形界面程序，首先应安装 Intel Visual Fortran 编译器。编译器的安装过程比较简单，这里不再赘述。不过有几点需要注意：

■ 先安装 Visual Studio 2008；

■ 然后安装 Intel Visual Fortran 编译器，并集成到 Visual Studio 2008；

■ 不建议用户安装 Visual Studio 2005，因为和 Intel Visual Fortran 的集成性不是很好，按照笔者的经验，曾经出现过不能正常使用控件箱的情况；

■ 建议完整安装 Intel Visual Fortran 编译器自带的帮助文件，尤其是 Windows 窗口程序设计的开发人员。

安装 Intel Visual Fortran 编译器后，就可打开 Visual Studio 2008 开始编译 Fortran 程序了。具体步骤如下：

■ 双击启动 Visual Studio 2008，选择“文件”菜单中的“新建”-“项目…”，弹出如图 2.1 所示的对话框，从中选择要建立的 Fortran 项目类型和模板，同时指定项目名称及其保存位置，单击“确定”按钮即可完成新项目的创建；

■ 按下组合键“Ctrl+F5”运行程序；

此处省略了程序的调试过程，对程序调试不熟练的读者请阅读相关资料或文献。

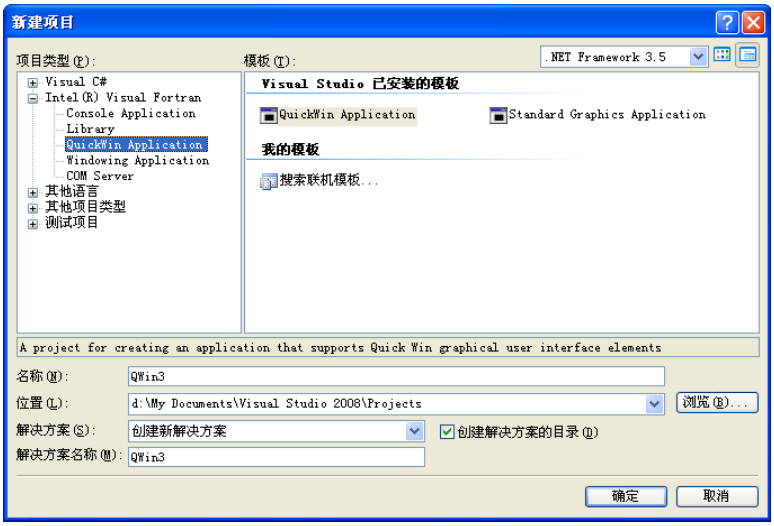


图 2.1 “新建项目”对话框

2-3 Intel Visual Fortran 中的项目类型

程序开发之前，我们应充分了解各种项目类型的特点。启动 Visual Studio 2008，按前面的步骤打开图 2.1 所示的对话框，在“项目类型”中列出了 Intel Visual Fortran 编译器可创建的几种项目类型。

■ Console Application 是基于字符模式的应用程序，这类项目适于不需要图形输出的程序设计，优点是执行速度快。图 2.2 所示为 Console Application 类工程的输出窗口。

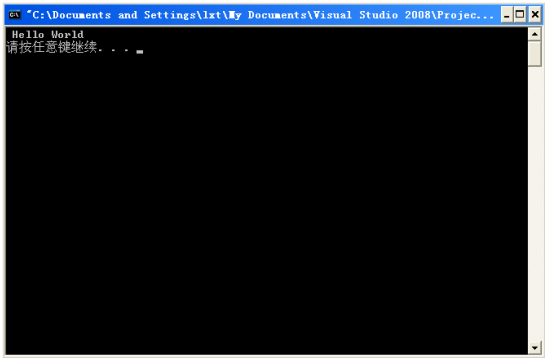


图 2.2 Console Application 工程输出窗口

■ Library 其中包括两种库文件工程：

● Static Libraries 静态库文件，是已编译的、独立于主程序的程序段，适合于大型项目中程序的结构组织和程序间函数或子程序的重复调用。但是，静态库文件会插入程序调用点，磁盘空间浪费较大；同时当静态库文件需要升级的时候，会带来工作效率的低下。为了解决这些问题，出现了动态链接库。

● Dynamic-link Library 动态链接库，很好地解决了静态库文件存在的问题，所以广泛使用在 Windows 程序设计中。

■ QuickWin Application 多文档窗口应用程序。编译器封装了部分 Win32 API 函数，使 Fortran 界面的开发变得更加容易。不管 QuickWin Application 应用程序有多简单，默认情况下界面中总会出现预设的系统菜单和状态栏。其运行界面如图 2.3 所示。

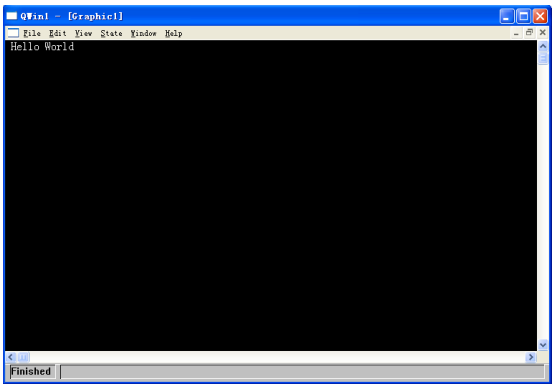


图 2.3 QuickWin Application 工程输出窗口

■ Standard Graphics Application 单文档窗口应用程序，通常以全屏的形式出现，如图 2.4 所示。全屏形式和窗口界面之间可通过组合键“Alt+Enter”互相切换。跟 QuickWin Application 不同，Standard Graphics Application 程序界面中并没有菜单栏和状态栏。



图 2.4 Standard Graphics Application 工程输出窗口

■ Windowing Application 窗口界面应用程序。经过编译器封装，Windowing Application 应用程序能够调用完整的 Win32 API 系统函数，所以 Windowing Application 应用程序的弹性更大、更加复杂。

第三章 QuickWin Application 基础

本章中，我们将利用 QuickWin 工程来创建简单的窗口程序。相比以前的命令行程序，图形界面窗口程序使用户在视觉和操作上更容易接受。

3-1 最简单的窗口程序

现在到该做些什么的时候了。如果您以前的程序都是基于 Console 工程类型，那么，从现在开始，让我们丢掉那个很不友好的“黑屏”，一起体验窗口程序设计的乐趣。

跟所有的语言一样，最简单的程序总是向世界问好的例子。由于第一次创建 Fortran 界面程序，接下来的步骤会较为详细，将逐步引导大家创建第一个窗口程序。具体过程如下：

■ 启动 Visual Studio 2008；

■ 选择“文件”菜单中的“新建”-“项目…”，弹出如图 2.1 所示对话框；

■ 在“项目类型”和“模板”中选择“QuickWin Application”，指定项目的保存位置和名称，单击“确定”按钮；

■ 编译器界面中出现了如图 3.1 所示的“解决方案资源管理器”；

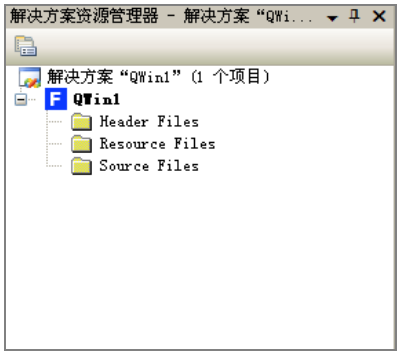


图 3.1 解决方案资源管理器

■ 移动鼠标至“Source Files”文件夹，单击 鼠标右键，选择“添加”-“新建项…”，弹出如图 3.2 所示的对话框；

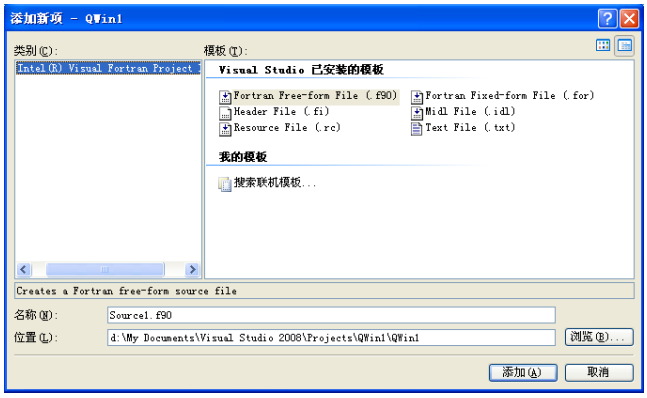


图 3.2 “添加新项”对话框

■ 选择“Fortran Free-form File(.f90)”模板，单击“添加”按钮；

■ 编译器界面中出现源代码编辑器，将下面的代码添加到“Source1.f90”；

```
#001 ! example 3-1
#002 program main
#003   implicit none
#004   write(*,*)"hello world!"
#005 end program main
```

■ 按下组合键“Ctrl+F5”运行程序，得到程序界面如图 3.3 所示。

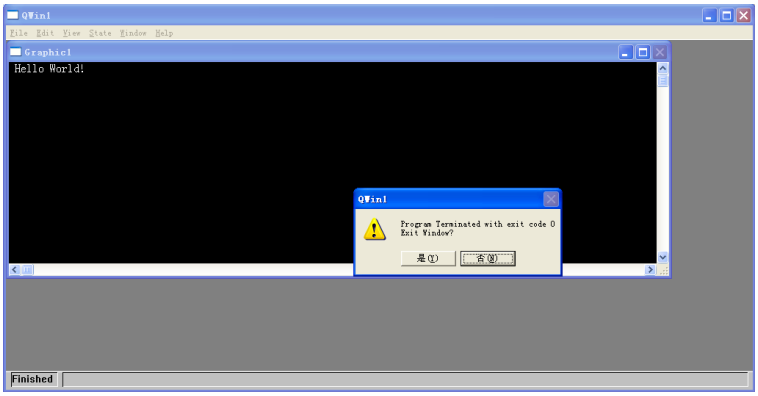


图 3.3 Example3-1 程序运行界面

这个程序中，简单的四个语句却实现了典型的 Windows 窗口界面，如菜单和状态栏等，所以，用 Fortran 编写窗口界面程序并没有想象中的困难，经过编译器对相关函数的有效封装后反而更加容易。

程序编译运行后，随着主窗口弹出了一个小提示窗口，同时，子窗口占了界面主框架的一部分，这些可能和读者想象的窗口程序有所区别。为了使这个简单的窗口程序更“像”，需要对程序做些修改。

3-2 “真正”的窗口程序

为了让窗口界面变得更容易理解，首先应该去掉程序运行后的对话框提示。方法有二：

■ 单击“否”，程序继续运行，但是每次执行程序时都有会出现该信息提示，使用不方便；

■ 借助函数 SETEXITQQ 让程序自动处理。SETEXITQQ 函数用来设置 QuickWin Application 程序退出时的提示行为，为正确使用该函数，必须在程序中包含语句“USE IFQWIN”。函数语法为：

USE IFQWIN

integer (kind=4) :: exitmode, result

result = SETEXITQQ (exitmode)

exitmode = QWIN$EXITPROMPT 为程序的默认情况；

或 = QWIN$EXITNOPERSIST 时，编译运行后窗口出现后马上消失；

或 = QWIN$EXITPERSIST 时，运行后窗口正常显示，没有任何提示信息。

将 SETEXITQQ 函数加入例 3-1，程序变为：

```
#001 ! example 3-2
#002 program main
#003 USE IFQWIN
#004   implicit none
#005   integer( kind=4 ) :: results
#006
#007   write(*,*)"hello world!"
#008   results = SETEXITQQ (QWIN$EXITPERSIST)
#009 end program main
```

运行程序，结果如图 3.4 所示。



图 3.4 Example3-2 程序运行界面

此时，信息提示框不再出现，但是子窗口和框架主界面并不统一。为了程序界面的整体感，程序主界面打开的同时，程序子窗口应最大化。为此，程序需要调用函数 SETWSIZEQQ 设置窗口的尺寸和位置，其语法为：

USE IFQWIN

integer (kind=4) :: result, unit

type (qwinfo) :: winfo

result = SETWSIZEQQ (unit, winfo)

其中，unit 为窗口代号，如果要最大化子窗口，则 unit 为子窗口代号，如果要最大化主窗口，其值应为已定义的符号常量 QWIN$FRAMEWINDOW。需要注意的是：如果程序中没有用 Open 语句显式打开子窗口，则默认子窗口的代号为 0、5 或 6。Winfo 是派生自用来存储窗口信息的 qwinfo 结构体变量，其结构如下：

TYPE QWINFO

INTEGER(2) TYPE　　！请求类型

INTEGER(2) X　　！窗口左上角的 X 坐标

INTEGER(2) Y　　！窗口左上角的 Y 坐标

INTEGER(2) H　　！窗口高度

INTEGER(2) W　　！　窗口宽度

END TYPE QWINF

其中 TYPE 可能的取值为：

Winfo% TYPE = QWIN$MIN 时，最小化窗口；

或 = QWIN$MAX 时，最大化窗口；

或 = QWIN$RESTORE 时，恢复最小化窗口到之前的大小；

或 = QWIN$SET 时，根据 Winfo 中的其它值设定窗口的位置和大小。

考虑到子窗口最大化，修改例 3-2，代码如下：

```
#001 ! example 3-3
#002 program main
#003   USE IFQWIN
#004   implicit none
#005   integer( kind=4 ) :: results
#006   type (qwinfo) :: winfo
#007
#008   winfo%type = QWIN$MAX
#009   ! 最大化主窗口
#010   results =    SETWSIZEQQ(QWIN$FRAMEWINDOW, winfo)
#011   ! 最大化子窗口
#012   results = SETWSIZEQQ (0, winfo)
#013   write(*,*)"Hello World!"
#014   ! 取消提示信息
#015   results = SETEXITQQ (QWIN$EXITPERSIST)
#016 end program main
```

编译、运行上述程序代码，结果如图 3.5 所示。显然主窗口最大化的同时，子窗口也最大化了。



图 3.5 Example3-3 程序运行界面

通过前面的代码，我们掌握了简单窗口程序的呈现方法。但是到目前为止，子窗口的黑色仍然让人难以忘却！为了改善视觉效果，我们必须更改窗口的背景色。

3-3 窗口和文本颜色

对于大多数搞工程的技术人员来说，数值计算本身难度较大，再加上只有黑白两种呈现色，那么程序开发的体验显然是很糟糕的。为了更好的呈现环境和视觉效果，程序开发中颜色的设置是必要的。Quickwin 工程中提供了多个颜色设置的函数，其中函数 SETBKCOLORRGB 用来更改窗口背景色。函数语法如下：

USE IFQWIN

integer (kind=4) :: color，result

result = SETBKCOLORRGB (color)

其中 color 是背景色，有两种方式来指定：其一是用十六进制数表达 RGB 分量，从左向右分别代表 B、G、R，如“#FF0000”代表蓝色，“#00FF00”代表绿色，“#0000FF”代表红色；其二是用函数 RGBTOINTEGER 将 RGB 分量值转换为代表相应颜色的整数，函数语法为：

USE IFQWIN

integer (kind=4) :: color

color = RGBTOINTEGER(255,0,0)

改变例 3-3 中程序窗口的背景色，程序变为：

```
#001 ! example 3-4
#002 program main
#003   USE IFQWIN
#004   implicit none
#005   integer( kind = 4 ) :: results, color
#006   type (qwinfo) :: winfo
#007
#008   color = RGBTOINTEGER(125,120,0)
#009  ! 目的是改变背景
#010   results = SETBKCOLORRGB (color)
#011  winfo%type = QWIN$MAX
#012   ! 最大化主窗口
#013   results = SETWSIZEQQ(QWIN$FRAMEWINDOW, winfo)
#014   ! 最大化子窗口
#015   results = SETWSIZEQQ (0, winfo)
#016   write(*,*)"hello world!"
#017   ! 取消提示信息
#018   results = SETEXITQQ (QWIN$EXITPERSIST)
#019 end program main
```

程序中第 8~10 行代码的目的是修改窗口背景色。但是程序运行后，子窗口的背景色并没有发生变化，仅是文本所在范围的背景色发生了改变！情况如图 3.6 所示。

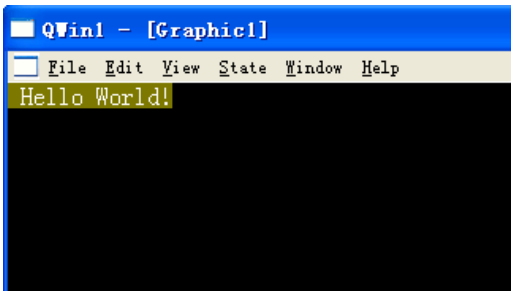


图 3.6 Example3-4 程序运行界面

为解决这个问题，需要重新研究窗口背景色设置函数 SETBKCOLORRGB。其实，该函数并不能自动改变窗口的背景色，只有当调用子程序 CLEARSCREEN 或者有文本读写发生时，背景才会改变。参考下面例 3-5 的程序代码：

```
#001 ! example 3-5
#002 program main
#003   USE IFQWIN
#004   implicit none
#005   integer( kind=4 ) :: results, color
#006   type (qwinfo) :: winfo
#007
#008   color = RGBTOINTEGER(125,120,0)
#009   results = SETBKCOLORRGB (color)
#010   ! 这样才能真正改变背景颜色
#011   call CLEARSCREEN($GCLEARSCREEN)
#012
#013   winfo%type = QWIN$MAX
```

```
#014   ! 最大化主窗口
#015   results = SETWSIZEQQ(QWIN$FRAMEWINDOW, winfo)
#016   ! 最大化子窗口
#017   results = SETWSIZEQQ (0, winfo)
#018   write(*,*)"Hello World!"
#019   ! 取消提示信息
#020   results = SETEXITQQ (QWIN$EXITPERSIST)
      end program main
```

运行程序，结果如图 3.7 所示，说明子窗口的背景色设置成功。例 3-5 中，程序



图 3.7 Example3-5 程序运行界面

第 9 行设置了窗口的背景色，但是没有改变背景色，而第 11 行代码才真正修改了窗口的背景色。其中子程序 CLEARSCREEN 的语法如下：

USE IFQWIN

integer (kind=4) ::area

CALL CLEARSCREEN (area)

其中，area = $GCLEARSCREEN 时，用指定的颜色清除屏幕；

或 = $GVIEWPORT 时，用指定的颜色清除当前视口；

或 = $GWINDOW 时，用指定色清除当前用 SETTEXTWINDOW 指定的文本窗口；

需要特别注意的是：CLEARSCREEN 是子程序而非函数，所以在程序中调用时应注意和函数在调用形式上的不同。

除了窗口的背景色，还有文本及图形的颜色设置。QuickWin Application 中，文本的颜色通过函数 SETTEXTCOLORRGB 来改变。该函数的语法为：

USE IFQWIN

integer (kind=4) :: color，result

result = SETTEXTCOLORRGB (color)

其中参数的意义同 SETBKCOLORRGB 中。该函数只影响由文本输出函数 OUTTEXT, WRITE 和 PRINT 产生的输出。在例 3-5 中加入改变文本颜色的代码：

```
#001 ! example 3-6
#002 program main
#003   USE IFQWIN
#004   implicit none
#005   integer( kind=4 ) :: results, color
#006   type (qwinfo) :: winfo
#007
#008   color = RGBTOINTEGER(125,125,125)
#009   results = SETBKCOLORRGB (color)
#010   ! 这样才能真正改变背景颜色
#011   call CLEARSCREEN($GCLEARSCREEN)
#012
```

```
#013   winfo%type = QWIN$MAX
#014   ! 最大化主窗口
#015   results =  SETWSIZEQQ(QWIN$FRAMEWINDOW, winfo)
#016   ! 最大化子窗口
#017   results = SETWSIZEQQ (0, winfo)
#018
#019   ! 改变输出文本的颜色
#020   results = SETTEXTCOLORRGB (#00FF00)
#021   write(*,*)"Hello World!"
#022   ! 取消提示信息
#023   results = SETEXITQQ (QWIN$EXITPERSIST)
#024 end program main
```

程序运行结果如图 3.8 所示。

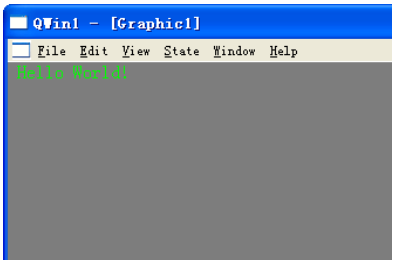


图 3.8 Example3-6 程序运行界面

到目前为止，窗口中仅输出了最简单的文本，这对程序开发者而言，还远不够！因为在程序中除了大量的数值计算外，还可能涉及到复杂的文本和图形输出。为了便于文本和图形输出，我们将窗口看成两种不同类型的窗口的叠加：

- 文本模式窗口 文本的输出以行和列为准；
- 图形模式窗口 文本和图形的输出以坐标为准。

默认情况下，这两种窗口是相互重叠的，窗口的裁剪区就是程序窗口中可输出部分的大小。有时为了输出需要，这两种窗口的大小也可以加以改变，具体方法留待后面讲述。关于如何利用这两种窗口，就是下面将要讨论的话题。

3-4 文本模式窗口

窗口程序设计中，程序设计者会关心窗口相关的属性，如窗口大小、字体样式、文字大小等。前面的程序中，如例 3-6，程序并没有赋予子窗口任何属性值，其中的子窗口为系统自动打开，此时，系统会赋予子窗口默认值。一般情况下，子窗口属性的默认值为：

- 子窗口大小 系统自动获取最大像素值赋予子窗口，典型值为 1024 x768；
- 文本行列值 默认情况下，子窗口中可输出 48 行 124 列文本；
- 字体类型 默认为“Courier New”；
- 字体大小 自动设为 8x16；
- 可用的颜色 程序中可用的颜色由 PC 显卡决定。

所以，在默认情况下，窗口中输出的文本字体为“Courier New”，字高 16，字宽为 8，每一行只能输出 80 个字符，第 80 个以后的自动换行输出。

为了验证窗口的默认属性值，可借助于函数 GETWINDOWCONFIG 获取子窗口的相关特征。函数的语法和后面要讲到的 SETWINDOWCONFIG 函数类似，请读者自己查阅帮助。下面是获取窗口属性的例子。

```
#001 ! example 3-7
#002 program main
#003   USE IFQWIN
#004   implicit none
#005   integer( kind=4 ) :: results, color
#006   type (qwinfo) :: winfo
#007   type (windowconfig) :: wc
#008   type (rccoord) :: curpos
#009   logical( kind=4 ):: status
#010
#011   ! 获取子窗口的属性
#012   status = GETWINDOWCONFIG(wc)
#013   ! 设置子窗口背景颜色
#014   color = RGBTOINTEGER(125,125,125)
#015   results = SETBKCOLORRGB (color)
#016   call CLEARSCREEN($GCLEARSCREEN)
#017   winfo%type = QWIN$MAX
#018   ! 最大化主窗口
#019   results = SETWSIZEQQ(QWIN$FRAMEWINDOW, winfo)
#020   ! 最大化子窗口
#021   results = SETWSIZEQQ (0, winfo)
#022   ! 设置文本颜色
#023   results = SETTEXTCOLORRGB (#00FF00)
#024   ! 输出窗口属性值
#025   write(*,*)"X 轴向像素为：", wc%numxpixels
#026   write(*,*)"Y 轴向像素为：", wc%numypixels
#027   write(*,*)"文本的列数为：", wc%numtextcols
#028   write(*,*)"文本的行数为：", wc%numtextrows
#029   write(*,*)"颜色索引数为：", wc%numcolors
#030   write(*,*)"字体的大小为：", wc%fontsize
#031   write(*,*)"窗口的标题为：", wc%title
#032   write(*,*)
#033   write(*,*)"单位像素位数：", wc%bitsperpixel
#034   write(*,*)"滚动条的模式：", wc%mode
#035   ! 取消提示信息
#036   results = SETEXITQQ (QWIN$EXITPERSIST)
#037 end program main
```

程序运行结果如图 3.9 所示。

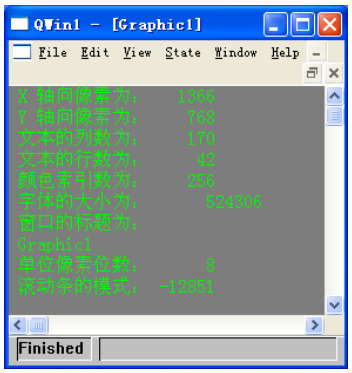


图 3.9 Example3-7 程序运行界面

从上图看出，程序的实际运行结果和前面的表述并不相同。其实这是由于计算机硬件的差别造成的，请读者自行测试。笔者使用的是 Compaq 宽屏笔记本，所以出现了上述结果。

为了灵活使用子窗口，在文本和图形输出前需要调整窗口的属性值，这时用到的函数为 SETWINDOWCONFIG，语法如下：

```
USE IFQWIN
type (windowconfig):: wc
logical( kind=4 ):: result
result = SETWINDOWCONFIG (wc)
```

其中 wc 是派生自 windowconfig 结构体的变量，保存了窗口的属性值。Windowconfig 结构体定义如下：

```
TYPE windowconfig
   INTEGER(2) numxpixels        ! X 方向像素值
   INTEGER(2) numypixels        ! Y 方向像素值
   INTEGER(2) numtextcols       ！ 文本行数
   INTEGER(2) numtextrows       ！ 文本列数
   INTEGER(2) numcolors         ！ 颜色索引数
   INTEGER(4) fontsize          ！ 字体大小，如果指定了扩展字体，fontsize 应该赋值为
                                QWIN$EXTENDFONT，此时由 extendfontsize 指定扩展字体的大小。
   CHARACTER(80) title          ！ 子窗口标题
   INTEGER(2) bitsperpixel      ！ 单个像素的位数
   INTEGER(2) mode              ！ 控制滚动条的滚动模式
   ！ 下面的三个参数用来定制扩展字体
   CHARACTER(32) extendfontname         ！ 字体名称
   INTEGER(4) extendfontsize        ！ 字体大小
   INTEGER(4) extendfontattributes     ！ 字体属性，如粗体、斜体等
END TYPE windowconfig
```

解决了文字的字体及大小问题后，还有文本输出的定位问题。系统提供了专门用于文本输出定位的子程序 SETTEXTPOSITION，它的作用是指定相对于文本窗口的文本输出位置，其语法为：

```
USE IFQWIN
integer ( kind=2 ) :: row, column
type(rccoord) :: t
CALL SETTEXTPOSITION (row,column,t)
```

其中 row 为文本的行定位数；column 为列定位数；t 是输出变量，派生自 rccoord 结构体，用来保存之前的光标位置。其结构定义为：

```
TYPE rccoord
    integer ( kind=2 ) :: row
```

integer (kind=2) :: col

END TYPE rccoord

下面是一个文本输出的例子，字体为隶书，大小为 16 x32，文本的输出位置在第 4 行第 4 列。代码如下：

```
#001   ! example 3-8
#002   program main
#003     USE IFQWIN
#004     implicit none
#005     integer( kind=4 ) :: results, color
#006     type (qwinfo) :: winfo
#007     type (windowconfig):: wc
#008     type (rccoord):: t
#009     logical( kind=4 ):: status
#010
#011     wc%numxpixels = -1
#012     wc%numypixels = -1
#013     wc%numtextcols = -1
#014     wc%numtextrows = -1
#015     wc%numcolors = -1
#016     wc%fontsize = QWIN$EXTENDFONT
#017     wc%title = 'QuickWin窗口'C
#018     wc%extendfontname = '隶书'C
#019     wc%extendfontsize = #000F001F
#020     wc%extendfontattributes  = ior(QWIN$EXTENDFONT_ITALIC,  &
#021                   QWIN$EXTENDFONT_BOLD )
#022     status = SETWINDOWCONFIG(wc)
#023     if (status == .false.) then
#024        status = MESSAGEBOXQQ('属性设置失败！系统调整中...'C, &
#025                      'Error'C,MB$ICONEXCLAMATION .OR.MB$OK )
#026        status = SETWINDOWCONFIG(wc)
#027     end if
#028
#029     ! 设置背景颜色
#030     color = RGBTOINTEGER(125,125,125)
#031     results = SETBKCOLORRGB (color)
#032     call CLEARSCREEN($GCLEARSCREEN)
#033
#034     winfo%type = QWIN$MAX
#035     ! 最大化主窗口
#036     results =    SETWSIZEQQ(QWIN$FRAMEWINDOW, winfo)
#037     ! 最大化子窗口
#038     results = SETWSIZEQQ (0, winfo)
#039     ! 文本颜色
#040     results = SETTEXTCOLORRGB (#00FF00)
#041     write(*,*)"中国，你好！"
#042     ! 调整输出位置到第行第列
#043     call SETTEXTPOSITION (4,4,t)
#044     call OUTTEXT("世界，你好！")
#045
#046     ! 重新设置文本颜色
#047     results = SETTEXTCOLORRGB (#FFFFFF)
#048     write(*,*)
#049     write(*,*)"输出位置调整之前的光标在："
#050     write(*,*)"第",t%row,"    行；"
#051     write(*,*)"第",t%col,"    列。"
```

```
#052     ! 取消提示信息
#053     results = SETEXITQQ (QWIN$EXITPERSIST)
#054   end program main
```

程序运行结果如图 3.10 所示。程序中出现了新函数和子程序，第一个为函数 MESSAGEBOXQQ，用来显示信息提示窗口；第二个是子程序 OUTTEXT，用于向文本或者图形窗口输出字符串。相关的语法和用法请读者自己查阅帮助文档。需要注意：子程序 OUTTEXT 在输出时不会产生换行动作，这点跟 WRITE 不同。

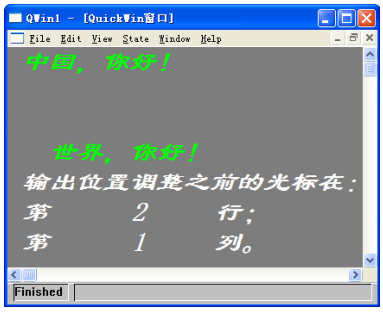


图 3.10 Example3-8 程序运行界面

3-5 图形模式窗口

子窗口既可看成文本模式窗口，此时文本的输出用行列定位，又能看成图形模式窗口，文本和图形的输出均以坐标定位。图形模式窗口中，为充分利用坐标定位带来的方便，文本的输出应调用子程序 OUTGTEXT，其语法如下：

USE IFQWIN

character(len=)::text*

CALL OUTGTEXT (text)

其中文本字符串要带双引号或者单引号。

字体格式的设置方法和例 3-8 相同。如果要改变字体，则首先调用无参函数 INITIALIZEFONTS( )初始化操作系统中的字体，函数语法为：

USE IFQWIN

integer (kind=2) :: result

result = INITIALIZEFONTS( )

其次，调用函数 SETFONT 指定字体及其大小：

USE IFQWIN

character (len=):: options*

integer (kind=2) :: result

result = SETFONT (options)

其中 options 为表述字体特征的字符串。具体请参阅帮助文档。常见的形式为 result = SETFONT ('t''Arial''h18w10')，'t'表明后面紧跟的是字体名称，'Arial'为采用的字体，'h18w10'表示字宽为 10，字高为 18。

由前面内容知，图形模式窗口中文本的输出用坐标定位，所以文本输出前，首先

应确定在哪个坐标位置输出。为了确定文本输出位置，需调用子程序 MOVETO 将画笔移至当前位置。如果您不理解该函数的用法，请把输出的文本理解为用画笔绘制的图形，输出文本就是绘图，所以首先应将画笔移动至目标位置，然后开始绘图，而子程序 MOVETO 就起到移动画笔的作用，其语法为：

USE IFQWIN

integer (kind=2) :: x, y

type (xycoord):: t

CALL MOVETO (x,y,t)

其中 x 和 y 是坐标，t 是派生自结构体 xycoord 的变量，其中保存了视口中前一位置的坐标。其结构定义如下：

TYPE xycoord

integer (kind=2) :: xcoord

integer (kind=2) :: ycoord

END TYPE xycoord

图形模式中，需要注意：

■ 无参函数 INITIALIZEFONTS( ) 和函数 SETFONT 只影响子程序 OUTGTEXT 输出的文本的字体格式；

■ OUTGTEXT 输出的文本颜色只能用函数 SETCOLORRGB 来改变。

下面是文本输出的例子。这个例子说明在子窗口中可用两种模式输出文本，其中图形模式下的输出方式具有较大的灵活性，同时说明了文本模式和图形模式之间的区别。源程序如下：

```
#001   ! example 3-9
#002   program main
#003     USE IFQWIN
#004     implicit none
#005     integer( kind = 4 ) :: results, color, i
#006     type (qwinfo) :: winfo
#007     type (windowconfig):: wc
#008     type (rccoord):: t
#009     type (xycoord):: wt
#010     logical( kind=4 ):: status
#011
#012     wc%numxpixels = -1
#013     wc%numypixels = -1
#014     wc%numtextcols = -1
#015     wc%numtextrows = -1
#016     wc%numcolors = -1
#017     wc%fontsize = QWIN$EXTENDFONT
#018     wc%title = 'QuickWin窗口'C
#019     wc%extendfontname = '隶书'C
#020     wc%extendfontsize = #OOOFOO1F
#021     wc%extendfontattributes  = ior(QWIN$EXTENDFONT_ITALIC,  &
#022                                QWIN$EXTENDFONT_BOLD )
#023     status = SETWINDOWCONFIG(wc)
#024     if (status == .false.) then
#025       status = MESSAGEBOXQQ('属性设置失败！系统调整中...'C, &
#026                   'Error'C,MB$ICONEXCLAMATION .OR.MB$OK )
```

```
#027      status = SETWINDOWCONFIG(wc)
#028    end if
#029
#030    ! 设置背景颜色
#031    color = RGBTOINTEGER(125,125,125)
#032    results = SETBKCOLORRGB (color)
#033    call CLEARSCREEN($GCLEARSCREEN)
#034    winfo%type = QWIN$MAX
#035    ! 最大化主窗口
#036    results =    SETWSIZEQQ(QWIN$FRAMEWINDOW, winfo)
#037    ! 最大化子窗口
#038    results = SETWSIZEQQ (0, winfo)
#039
#040    ! 文本为绿色
#041    results = SETTEXTCOLORRGB (#00FF00)
#042    do i = 4,7
#043      call SETTEXTPOSITION (i,10,t)
#044      write(*,*)"北京欢迎你！"
#045    end do
#046
#047    ! SETTEXTCOLORRGB没有起作用
#048    results = SETTEXTCOLORRGB (#FF0000)
#049    do i = 4,7
#050      call MOVETO(300,100+i*40,wt)
#051      call OUTGTEXT("北京欢迎你！")
#052    end do
#053
#054    ! SETCOLORRGB才能影响OUTGTEXT输出的文本
#055    results = SETCOLORRGB (#FF0000)
#056    results = INITIALIZEFONTS()
#057    results = SETFONT('t''幼圆''h18w10')
#058    do i = 4,7
#059      call MOVETO(480,280+i*40,wt)
#060      call OUTGTEXT("北京欢迎你！")
#061    end do
#062    ! 取消提示信息
#063    results = SETEXITQQ (QWIN$EXITPERSIST)
#064  end program main
```

程序运行结果如图 3.11 所示。



图 3.11 Example3-9 程序运行界面

图形模式下的文本输出具有更大的灵活性，读者可借助于系统提供的函数输出位置多变的文本。如果文本输出需要倾斜角度，可调用函数 SETGTEXTROTATION，其语法为：

USE IFQWIN

integer(kind=4) :: degree-tenths

CALL SETGTEXTROTATION (degree-tenths)

其中 degree-tenths 的取值为旋转角度乘 10 之后的数值，水平向右方向为角度 0 角度起始方向，逆时针旋转为角度正方向。

关于该函数的使用，请读者自行测试。

到目前为止，所有的输出都发生在同一个子窗口中。根据前面的内容，QuickWin Application 为多文档窗口工程，所以，QuickWin 程序具备打开多个子窗口的能力。多个子窗口的创建由函数 OPEN 完成。其简要的语法为(关于其详细的用法，请读者查阅帮助文档)：

OPEN (UNIT=子窗口代号, FILE= 'USER', TITLE= '子窗口标题', IOFOCUS='是否有焦点')

在以前的程序设计中，OPEN 用来打开数据文件，但是在 QuickWin 程序中，当 FILE= 'USER'时，打开的是子窗口。请看下面的例子：

```
#001   ! example 3-10
#002   program main
#003     USE IFQWIN
#004     implicit none
#005     integer( kind = 4 ) :: results
#006     open(11,file="user",title= 'no.1')
#007     write(11,*) "hello, no.1"
#008     open(12,file="user",title= 'no.2')
#009     write(12,*) "hello ,no.2"
#010     ! 取消提示信息
#011     results = SETEXITQQ (QWIN$EXITPERSIST)
#012   end program main
```

运行结果如图 3.12 所示。在这个例子中，程序打开了两个子窗口并产生了简单的输出。第 7 行和第 9 行的代码告诉我们，输出到子窗口和输出到数据文件时一样的，都是通过通道号来实现。

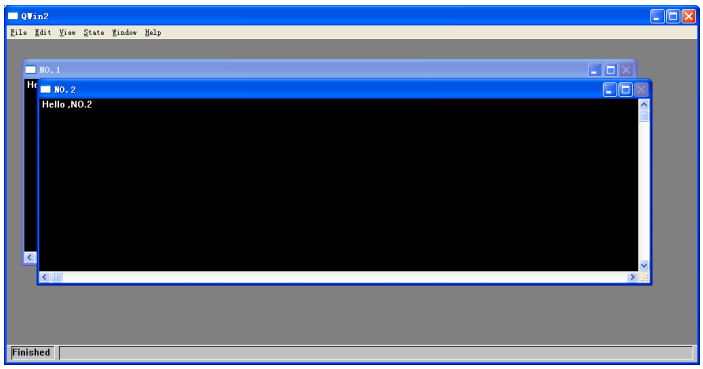


图 3.12 Example3-10 程序运行界面

多文档窗口在实际中非常有用，如多个绘图文件、多个文本文件以及多个窗口之间的交互等。例如，在工程中实践中，有很多非常优秀的数值计算程序，但是本身缺乏友好的界面和良好的交互能力。为了提高这类程序的用户体验，可用 QuickWin 为其搭建简单的交互平台。

下面的程序创建了两个子窗口，第一个为输入窗口，第二个为输出窗口，当用户

在第一个窗口内输入字符且回车后，输入在字符同时在第二个窗口内输出。程序实现的功能非常简单，但是提供了一种实用的思路。源程序如下：

```
#001   ! example 3-11
#002   program main
#003     USE IFQWIN
#004     implicit none
#005     integer( kind = 4 ) :: results,color
#006     type (windowconfig):: wc
#007     type (qwinfo) :: winfo
#008     character*(20)input
#009     ! 主框架窗口的大小
#010     winfo%type = QWIN$SET
#011     winfo%H =300
#012     winfo%W = 600
#013     resultS = SETWSIZEQQ (QWIN$FRAMEWINDOW , winfo)
#014
#015     ! 打开第一个输入窗口并设置窗口属性值
#016     open(unit=5,file='user',iofocus=.true.)
#017     wc.numxpixels = 300
#018     wc.numypixels = 300
#019     wc.numtextcols = -1
#020     wc.numtextrows = -1
#021     wc.numcolors = -1
#022     wc.title = "输入窗口"C
#023     wc.fontsize = -1
#024     results = SetWindowConfig(wc)
#025     results = DisplayCursor($GCURSORON) ! 显现出光
#026     ! 打开第二个输入窗口并设置窗口属性值
#027     open(unit=10,file='user')
#028     wc.numxpixels = 300
#029     wc.numypixels = 300
#030     wc.numtextcols = -1
#031     wc.numtextrows = -1
#032     wc.numcolors = -1
#033     wc.title = "输出窗口"C
#034     wc.fontsize = -1
#035     results = SetWindowConfig(wc)
#036     ! 相当于按下菜单中的Tile命令，使两个窗口之间不会互相重叠
#037     results = ClickMenuQQ(QWIN$TILE)
#038     do while (.true.)
#039       write(5,*)"请输入字符串进行测试："
#040       read(5,*)input
#041       write(10,*)input
#042     end do
#043     ! 取消提示信息
#044     results = SETEXITQQ (QWIN$EXITPERSIST)
#045   end program main
```

运行结果如图3.13所示。

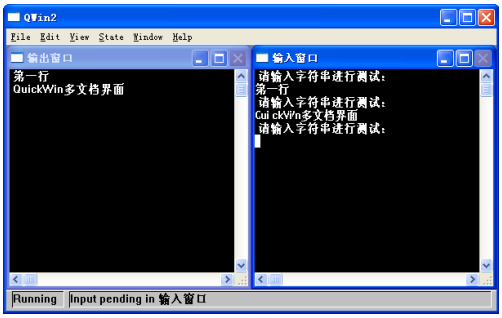


图 3.13 Example3-11 程序运行界面

程序中新出现了两个函数，第一个为DISPLAYCURSOR，作用是在子窗口中显示闪烁的光标，方便输入；第二个为CLICKMENUQQ，用来执行Quickwin预先封装的系统菜单。两个函数的用法都很简单，这里不再一一列举。

目前，我们已经掌握了创建Quickwin程序所需要的最基本的知识，但是距离用户交互还有距离。关于如何利用键盘和鼠标进行用户交互，这是下一章将要讨论的内容。

第四章 QuickWin用户交互

以前的程序都很简单，代码从头到尾按顺序执行，但是缺少了用户和程序之间的互动性，弊端显而易见。工程数值计算，经常会涉及到图形输出、线程调派等涉及和用户交互的动作，所以程序具有良好的交互性使非常重要的。本章中，我们将讨论在QuickWin 工程中如何利用键盘和鼠标进行人机交互。

4-1 键盘事件

键盘在计算机中扮演者很重要的角色。几乎所有的工作都能由键盘完成，常见的文档编辑整理、代码的输入以及游戏的操作等等。不管多复杂的程序，明白键盘事件及其输入将非常重要。下面结合例子说明 QuickWin 中如何读入和处理键盘事件。程序代码如下：

```
#001   ! example 4-1
#002   program main
#003     USE IFQWIN
#004     USE IFCORE
#005     implicit none
#006     integer( kind = 4 ) :: results
#007     type (qwinfo) :: winfo
#008     character*(1) ch
#009
#010     winfo%type = QWIN$MAX
#011     ! 最大化子窗口
#012     results = SETWSIZEQQ (0, winfo)
#013
#014     do while(.true.)
#015       ch = GETCHARQQ( )
#016       write(*,*)"已按下 ",ch,"  键；"
#017     end do
#018     ! 取消提示信息
#019     results = SETEXITQQ (QWIN$EXITPERSIST)
#020   end program main
```

程序运行结果如图 4.1 所示。

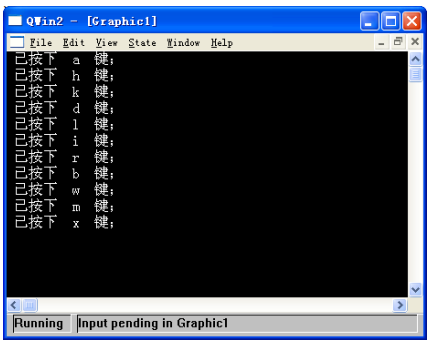


图 4.1 Example4-1 程序运行界面

其中无参函数 GETCHARQQ 用来捕获键盘事件，语法为：

USE IFCORE

character(1) result*

result = GETCHARQQ( )

该函数的返回值为单个字符，代表着当前键盘按键。经过测试例 4-1，发现 GETCHARQQ 函数只能监测字母和数字按键，对功能键和方向键就无能为力了。为准确监测这些特殊按键，须用到函数 PASSDIRKEYSQQ，当该函数的参数为真时，GETCHARQQ 函数会自动监测特殊按键；当函数参数为假时，GETCHARQQ 函数不能检测到方向键，此时，方向键的作用是移动滚动条的位置。下面的例子说明了这点。

```
#001   ! example 4-2
#002   program main
#003     USE IFQWIN
#004     USE IFCORE
#005     implicit none
#006     integer( kind=4 ) :: results
#007     logical( kind=4 ) :: res
#008     type (qwinfo) :: winfo
#009     character*(1) ch,ch1
#010
#011     winfo%type = QWIN$MAX
#012     ! 最大化子窗口
#013     results = SETWSIZEQQ (0, winfo)
#014     ! 允许函数getcharqq能检测到方向键
#015     res = PASSDIRKEYSQQ(.true.)
#016     do while(.true.)
#017       ch = getcharqq()
#018       if (ichar(ch) .eq. 0) then ! 功能键
#019         ch1 = getcharqq()
#020         write(*,*)"功能键= ",ichar(ch),ichar(ch1),"      ",ch1
#021       else if (ichar(ch) .eq. 224) then ! 方向键
#022         ch1 = getcharqq()
#023         write(*,*)"方向键= ",ichar(ch),ichar(ch1),"      ",ch1
#024       else ! 其它键
#025         write(*,*)"其它键= ",ichar(ch),"      ",ch
#026       endif
#027     end do
#028     ! 取消提示信息
#029     results = SETEXITQQ (QWIN$EXITPERSIST)
#030   end program main
```

程序运行结果如图4.2所示。

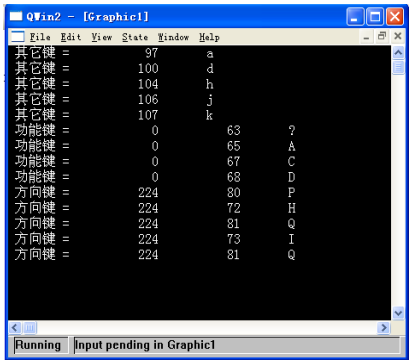


图 4.2 Example4-2 程序运行界面

例 4-2 中，程序首先用循环语句读入键盘事件，接着用函数 ICHAR 返回当前按键在整个字符集中的位置，以此判断按键种类。当函数 ICHAR 的返回值为 0 时，说明当前按下的是功能键；当返回值为 224 时，说明是方向键；如果是其它值，则是上述按键之外的其它键盘事件。

需要注意的是：函数 GETCHARQQ 的使用中，数字键和字母键按下后返回的是键本身所表达的字母或数字；而功能键返回的是十六进制数“00”；方向键返回的是十六进制数“E0”。这也就很好地解释了上面的程序中判断语句的依据。为了进一步判断功能键和方向键的名称，必须第二次调用函数 GETCHARQQ 获得相应的扩展码，详见程序代码。

4-2 鼠标事件

Window 应用程序是基于事件驱动的。操作系统时刻监控着键盘和鼠标的动作，如果有键盘的击打或者鼠标键的按下等动作，操作系统会在第一时间将这些事件通知用户的应用程序。为此，应用程序中必须有无限循环语句来捕捉操作系统通它的事件，一旦捕捉到用户关心的事件动作，马上会激或相应的动作程序。下面是鼠标事件的处理过程：

首先，为监测鼠标事件，应用程序必须包括无限循环；

```
do while(.true.)
  results = WaitOnMouseEvent( MOUSE$MOVE, state, x, y )
end do
```

其中函数 WaitOnMouseEvent 的作用是等待所关心的鼠标事件的发生。语法如下：

```
USE IFQWIN
integer( kind=4 ) :: result
integer( kind=4 ) :: mouseevents  ！传入值，用户关心的鼠标事件
integer( kind=4 ) :: keystate  ！返回值，鼠标事件发生时其他功能键(如Shift)的状态
integer( kind=4 ) :: x  ！返回值，鼠标的位置
integer( kind=4 ) :: y  ！返回值，鼠标的位置
result = WAITONMOUSEEVENT (mouseevents,keystate,x,y)
```

mouseevents 是用户所要监测的鼠标事件，如果需要检测多个鼠标事件，可用函数 IOR 加以组合。下面列出了常见的鼠标事件：

- MOUSE$LBUTTONDOWN　按下鼠标左键
- MOUSE$LBUTTONUP　释放鼠标左键
- MOUSE$LBUTTONDBLCLK　双击鼠标左键
- MOUSE$RBUTTONDOWN 按下鼠标右键
- MOUSE$RBUTTONUP　释放鼠标右键
- MOUSE$RBUTTONDBLCLK 双击鼠标右键
- MOUSE$MOVE　移动鼠标

Keystate 是鼠标事件中其他功能键的状态。适合于判断鼠标左键按下且移动等组

合事件。其值为：

- MOUSE$KS_LBUTTON　事件中按下鼠标左键
- MOUSE$KS_RBUTTON　事件中按下鼠标右键
- MOUSE$KS_SHIFT　事件中按下 SHIFT 键
- MOUSE$KS_CONTROL　事件中按下 CTRL 键

其次，应用程序中，不同的鼠标事件需要激活不同的任务，所以必须将鼠标事件和用户的目标任务结合起来，也就是鼠标事件的注册。函数为 REGISTERMOUSEEVENT，语法如下：

USE IFQWIN

integer(kind=4) :: result,

integer(kind=4) :: unit　！鼠标事件发生的窗口代码

integer(kind=4) :: mouseevents　！需要注册的鼠标事件

external :: ShowLocation　！回调子程序，必须申明为 EXTERNAL。

result = REGISTERMOUSEEVENT (unit,mouseevents,callbackroutine)

此时，代号为 unit 的窗口中，当鼠标事件 mouseevents 发生时，程序就会执行子程序 callbackroutine 设定的任务，从而完成了鼠标事件的注册。

这里，读者还需要了解回调子程序的接口形式：

INTERFACE

SUBROUTINE MouseCallBackRoutine (unit, mouseevent, keystate, MouseXpos, MouseYpos)

INTEGER unit　！事件发生的窗口代号

INTEGER mouseevent　！鼠标事件

INTEGER keystate　！功能键的状态

INTEGER MouseXpos　！鼠标位置坐标

INTEGER MouseYpos　！鼠标位置坐标

END SUBROUTINE

END INTERFACE

读者在程序中按照上述接口提供的形式编写程序即可。下面是鼠标移动事件的例子，示例中当鼠标移动时，输出鼠标当前位置的坐标。源程序为：

```
#001   ! example 4-3
#002   program main
#003     USE IFQWIN
#004     implicit none
#005     integer( kind=4 ) :: results,state,x,y,event
#006     type (qwinfo) :: winfo
#007     external:: showlocation
#008
#009     winfo%type = QWIN$MAX
#010     ! 最大化子窗口
#011     results = SETWSIZEQQ (0, winfo)
#012     ! 鼠标移动时，调用ShowLocation
#013     event = MOUSE$MOVE
#014     results = RegisterMouseEvent(0, event, ShowLocation)
#015     ! 等待鼠标输入
```

```
#016     do while(.true.)
#017       results = WaitOnMouseEvent( MOUSE$MOVE, state, x, y )
#018     end do
#019     ! 取消提示信息
#020     results = SETEXITQQ (QWIN$EXITPERSIST)
#021     end program main
#022
#023   subroutine showlocation(iunit, ievent, ikeystate, ixpos, iypos)
#024     USE IFQWIN
#025     implicit none
#026     integer( kind=4 ) :: iunit       ! 鼠标所在的窗口的代号
#027     integer( kind=4 ) :: ievent      ! 鼠标事件
#028     integer( kind=4 ) :: ikeystate    ! 其它控制键的状态
#029     integer( kind=4 ) :: ixpos,iypos  ! 鼠标位置
#030
#031     write(0,100)ixpos,iypos
#032     100 format("当前鼠标位置：(X=",I4," Y=",I4,")")
#033   end subroutine showlocation
```

程序运行如图4.3所示。

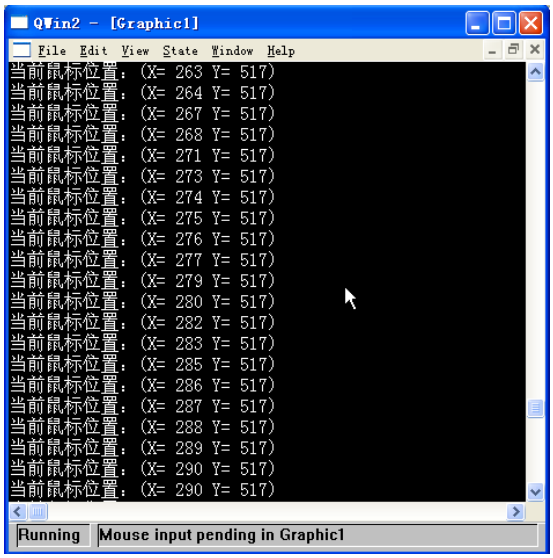


图 4.3 Example4-3 程序运行界面

关于组合键的判断，请看下面的例子：

```
#001   ! example 4-4
#002   program main
#003     USE IFQWIN
#004     implicit none
#005     integer( kind=4 ) :: results,state,x,y
#006     type (qwinfo) :: winfo
#007     external:: showlocation,leftbutton,rbutton
#008
#009     winfo%type = QWIN$MAX
#010     ! 最大化子窗口
#011     results = SETWSIZEQQ (0, winfo)
#012     ! 鼠标移动时，调用ShowLocation
#013     results = RegisterMouseEvent(0, MOUSE$MOVE, ShowLocation)
#014     results = RegisterMouseEvent(0, MOUSE$LBUTTONDOWN,Leftbutton)
#015     results = RegisterMouseEvent(0, MOUSE$RBUTTONDOWN,Rbutton)
#016     ! 等待鼠标输入
#017     do while(.true.)
```

```
#018      results = WaitOnMouseEvent( MOUSE$MOVE, state, x, y )
#019    end do
#020    ! 取消提示信息
#021    results = SETEXITQQ (QWIN$EXITPERSIST)
#022  end program main
#023
#024  subroutine showlocation(iunit, ievent, ikeystate, ixpos, iypos)
#025    USE IFQWIN
#026    implicit none
#027    integer( kind=4 ) :: iunit       ! 鼠标所在的窗口的代号
#028    integer( kind=4 ) :: ievent      ! 鼠标事件
#029    integer( kind=4 ) :: ikeystate    ! 其它控制键的状态
#030    integer( kind=4 ) :: ixpos,iypos  ! 鼠标位置
#031    if (ikeystate == MOUSE$KS_LBUTTON) then
#032      write(0,*)"鼠标左键按下且移动",ixpos,iypos
#033    elseif (ikeystate == MOUSE$KS_RBUTTON) then
#034      write(0,*)"鼠标右键按下且移动",ixpos,iypos
#035    elseif (ikeystate == MOUSE$KS_SHIFT) then
#036      write(0,*)"鼠标移动中按下了SHIFT键",ixpos,iypos
#037    elseif (ikeystate == MOUSE$KS_CONTROL) then
#038      write(0,*)"鼠标移动中按下了CONTROL键",ixpos,iypos
#039    elseif (ikeystate == 16) then
#040      write(0,*)"鼠标中键按下且移动",ixpos,iypos
#041    else
#042      write(0,*)"鼠标移动事件",ixpos,iypos
#043    endif
#044  end subroutine showlocation
#045
#046  subroutine leftbutton(iunit, ievent, ikeystate, ixpos, iypos)
#047    USE IFQWIN
#048    implicit none
#049    integer( kind=4 ) :: iunit       ! 鼠标所在的窗口的代号
#050    integer( kind=4 ) :: ievent      ! 鼠标事件
#051    integer( kind=4 ) :: ikeystate    ! 其它控制键的状态
#052    integer( kind=4 ) :: ixpos,iypos  ! 鼠标位置              、
#053    write(0,*)"鼠标左键单击事件",ixpos,iypos
#054  end subroutine leftbutton
#055
#056  subroutine rbutton(iunit, ievent, ikeystate, ixpos, iypos)
#057    USE IFQWIN
#058    implicit none
#059    integer( kind=4 ) :: iunit       ! 鼠标所在的窗口的代号
#060    integer( kind=4 ) :: ievent      ! 鼠标事件
#061    integer( kind=4 ) :: ikeystate    ! 其它控制键的状态
#062    integer( kind=4 ) :: ixpos,iypos  ! 鼠标位置
#063    write(0,*)"鼠标右键单击事件",ixpos,iypos
#064  end subroutine rbutton
```

运行结果如图 4.4 所示，详细情况请读者自行测试。

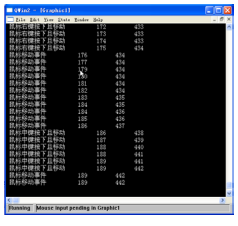


图 4.4 Example4-4 程序运行界面

4-3 窗口菜单

菜单功能是 Windows 窗口非常重要的特征。Quickwin 工程中，由于系统已经封装了部分菜单，而且对菜单的设置进行了相当的简化，所以菜单的设置显得直接而方便。

细心的读者已经发现，前面所有的程序中都有系统菜单，而且这些菜单中有的是比较常用的功能，如打印菜单项，目的是让用户的开发变得更简单。但是应用程序的开发本身具有多样性，统一的菜单功能并不能满足大多数开发者的需求，所以，如何定制应用程序需要的菜单就显得非常重要！为了使读者对菜单的定制有较全面的了解，首先列出菜单定制中常用的函数：

■ APPENDMENUQQ 添加菜单项到某菜单的最后且注册菜单项的回调函数。其语法如下：

```
USE IFQWIN
integer( kind=4 ) :: menuID
integer( kind=4 ) :: flags
character( Len=* ) :: text
external :: routine
logical( kind=4 ) :: result
result = APPENDMENUQQ (menuID, flags, text,routine)
```

其中，menuID 为菜单编号，从左向右数第一个菜单编号为 1，其它以此类推；flags 表明菜单的状态，如菜单项灰显等；text 为菜单项的名称，是 null-terminatedC 字符串，如'WORDS OF TEXT'C，如果要为菜单或菜单项设置快捷键，则需要在对应的字母前加上符号“&”，如“P&rint”，用户就能用组合键“CTRL+R”激活回调子程序，从而方便程序的操作；routine 为回调子程序，必须申明为 EXTERNAL 类，其中每个回调子程序都有一个逻辑参数，用来说明该菜单项是否被选中，同时，Routine 也可以是系统预先定义的子程序的名称，如 WINPRINT 为打印命令子程序，WINSAVE 为保存命令子程序等。

注意：用户并不需要给所要添加的菜单项提供标号，因为函数 APPENDMENUQQ 总是将菜单项添加到某菜单的尾部。如果要添加菜单项到一个并不存在的菜单，那么系统将该菜单项设为某个菜单的名称，当该菜单有其他菜单项时，系统自动忽略回调子程序，此时可将 *routine 的值设为 Nul*。

■ INSERTMENUQQ 将菜单项插入某菜单。语法如下：

```
USE IFQWIN
integer( kind=4 ) :: menuID, itemID
integer( kind=4 ) :: flags
character( Len=* ) :: text
external :: routine
logical( kind=4 ) :: result
result = INSERTMENUQQ (menuID,itemID,flag,text,routine)
```

其中 itemID 为菜单项的插入位置，顺序为从上到下。当 itemID=0 时，将插入的菜单项作

为第 menuID 个菜单出现，其它菜单顺次向右移位；当 itemID 为其它整数值时，将菜单项插入第 menuID 个菜单的指定位置。

■ DELETEMENUQQ 删除菜单项。语法为：

```
USE IFQWIN
integer( kind=4 ) :: menuID, itemID
logical( kind=4 ) :: result
result = DELETEMENUQQ (menuID, itemID)
```

其中 menuID 为菜单编号, itemID 为要删除的菜单项的编号，如果 itemID 为 0，则将删除第 menuID 个菜单。

■ MODIFYMENUFLAGSQQ 用来修改菜单项的状态。函数语法为：

```
USE IFQWIN
integer( kind=4 ) :: menuID, itemID
integer( kind=4 ) :: flags
logical( kind=4 ) :: result
result = MODIFYMENUFLAGSQQ (menuID,itemID,flag)
```

参数意义同上。

■ MODIFYMENUROUTINEQQ 用来改变菜单项的回调函数。函数语法为：

```
USE IFQWIN
integer( kind=4 ) :: menuID
integer( kind=4 ) :: itemID
external :: routine
logical( kind=4 ) :: result
result = MODIFYMENUROUTINEQQ (menuID,itemID,routine)
```

■ MODIFYMENUSTRINGQQ 用来改变菜单项名。语法为：

```
USE IFQWIN
integer( kind=4 ) :: menuID
integer( kind=4 ) :: itemID
character( Len=* ) :: text
logical( kind=4 ) :: result
result = MODIFYMENUSTRINGQQ (menuID,itemID,text)
```

■ INITIALSETTINGS( ) 用来设定 Quickwin 工程中主窗口的外观和菜单项。语法为：

```
USE IFQWIN
logical( kind=4 ) :: result
result = INITIALSETTINGS( )
```

注意：Quickwin 工程中主窗口的外观和菜单项是由系统预设的 INITIALSETTINGS 函数来实现的。如果程序开发者自己需要设定主窗口的外观和菜单项，则只要重新编写该函数即可。INITIALSETTINGS 函数通常是被系统自动调用，而不需要显式调用，

这点尤为重要。

下面是菜单定制的例子。例子将如图 4.5 所示的原始菜单中第三和第四项删除；在第二个菜单的最后追加了菜单分隔符和 Beep 菜单项；在第二个菜单的顶部插入了新菜单 New。源程序为：

```
#001   ! example 4-5
#002   program main
#003     USE IFQWIN
#004     implicit none
#005     integer( kind=4 ) :: results
#006     logical(kind=4)::res
#007     type (qwinfo) :: winfo
#008     external:: rbeep
#009
#010     winfo%type = QWIN$MAX
#011     ! 最大化子窗口
#012     results = SETWSIZEQQ (0, winfo)
#013     ! 添加菜单分隔符
#014     res = APPENDMENUQQ (2, $MENUSEPARATOR ,"sep"C,Nul)
#015     res = APPENDMENUQQ (2, $MENUENABLED,"&Beep\tCtrl+B"C,Rbeep)
#016     res = INSERTMENUQQ (2,1,$MENUENABLED,"&New"C,Nul)
#017     res = DELETEMENUQQ (3,0)
#018     res = DELETEMENUQQ (3,0)
#019     results = SETEXITQQ (QWIN$EXITPERSIST)
#020     ! 循环很重要，是菜单正确执行的前提
#021     do while (.true.)
#022     enddo
#023   end program main
#024
#025   subroutine rbeep (checked)
#026     use ifqwin
#027     implicit none
#028     logical(kind=4)::checked
#029
#030     call BEEPQQ(1200,800)
#031   end subroutine rbeep
```

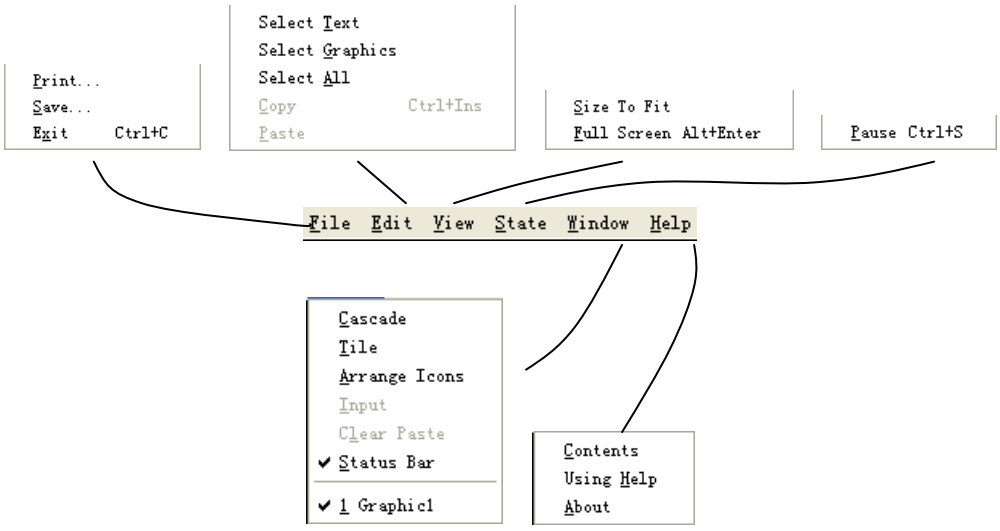


图 4.5 Quickwin 工程系统菜单

运行程序，菜单结构变为如图 4.6 所示的结构。

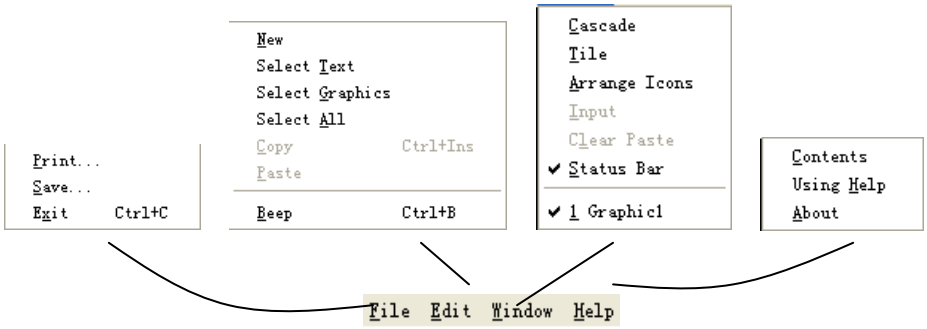


图 4.6 修改后的系统菜单

其实，系统提供的菜单对有的设计者来说或者没有任何用处，程序设计者需要完全由自己设计菜单，那么函数 INITIALSETTINGS 就必须重新定义。下面是利用 INITIALSETTINGS 函数初始化 Quickwin 主窗口和菜单的例子。跟前例不同，这里将涉及主窗口外观和菜单的程序代码放置在 INITIALSETTINGS 函数内，使代码的结构和可读性更好。

```
#001   ! example 4-6
#002   program main
#003   USE IFQWIN
#004   implicit none
#005   integer( kind=4 ) :: results
#006   logical(kind=4)::res
#007   type (qwinfo) :: winfo
#008   type (windowconfig) :: wc
#009
#010   winfo%type = QWIN$MAX
#011   res = GETWINDOWCONFIG(wc)
#012   ! 最大化子窗口
#013   results = SETWSIZEQQ (0, winfo)
#014   results = SETEXITQQ (QWIN$EXITPERSIST)
#015   ! 循环很重要，是菜单正确执行的前提
#016   results = ABOUTBOXQQ ('作者： \r     Version 1.0'C)
#017   do while (.true.)
#018   enddo
#019   end program main
#020
#021   LOGICAL(4) FUNCTION INITIALSETTINGS( )
#022     USE IFQWIN
#023     implicit none
#024     integer( kind=4 ) :: results
#025     logical(kind=4)::res
#026     type (qwinfo) :: winfo
#027     external :: Rbeep, Browse, Notes, Calculate, Instructions
#028
#029     winfo%w = 400
#030     winfo%h = 400
#031     winfo%type = QWIN$SET
#032     results = SetWSizeQQ( QWIN$FRAMEWINDOW, winfo )
#033
#034     res = appendmenuqq(1, $MENUENABLED, '文件'C, NUL)
#035     res = appendmenuqq(1, $MENUENABLED, '打印...'C,WINPRINT)
#036     res = appendmenuqq(1, $MENUENABLED, '另存为...'C, WINSAVE)
#037     res = appendmenuqq(1, $MENUSEPARATOR,'sep'C, NUL)
```

```
#038     res = appendmenuqq(1, $MENUENABLED, '主页'C, Browse)
#039     res = appendmenuqq(1, $MENUENABLED, '记事本'C,  Notes)
#040     res = appendmenuqq(1, $MENUENABLED, '计算器'C, calculate)
#041     res = appendmenuqq(1, $MENUSEPARATOR,'sep'C, NUL)
#042     res = appendmenuqq(1, $MENUENABLED, '退出'C, WINEXIT)
#043     res = appendmenuqq(2, $MENUENABLED, '帮助'C,  NUL)
#044     res = appendmenuqq(2, $MENUENABLED, '说明'C,Instructions)
#045     res = appendmenuqq(2, $MENUENABLED, '关于...'C, WINABOUT)
#046     INITIALSETTINGS= .true.
#047     return
#048   END FUNCTION INITIALSETTINGS
#049
#050   subroutine Rbeep (checked)
#051     USE IFQWIN
#052     implicit none
#053     logical(kind=4)::checked
#054
#055     call BEEPQQ(1200,800)
#056   end subroutine Rbeep
#057
#058   subroutine Browse
#059     USE IFQWIN
#060     USE IFPORT
#061     implicit none
#062     integer(kind=2) iret
#063     iret = RUNQQ('Explorer.exe','http://www.cnblogs.com'C)
#064   end subroutine Browse
#065
#066   subroutine Notes
#067     USE IFWIN
#068     USE IFQWIN
#069     implicit none
#070     integer(kind=2) iret
#071     iret = WinExec('NotePad.exe menu.f90',SW_SHOW)
#072   end subroutine Notes
#073
#074   subroutine Calculate
#075   USE IFWIN
#076   USE IFQWIN
#077   implicit none
#078   integer(kind=2) iret
#079   iret = WinExec('Calc.exe'C,SW_SHOW)
#080   end subroutine Calculate
#081
#082   subroutine Instructions
#083     USE IFWIN
#084     USE IFQWIN
#085     implicit none
#086     integer(kind=2) iret
#087     character(200) szAbout
#088     szAbout = ' 用户自定义菜单示例! \n 'C
#089     iret = MESSAGEBOXQQ(szAbout,'说明'C,
#090                         MB$ICONEXCLAMATION .OR. MB$OK )
#091   end subroutine Instructions
```

程序运行结果如图 4.7 所示。示例中出现了无限循环语句，这对菜单程序来说是必须的。其中函数 BEEPQQ 用来驱动扬声器发生；函数 RUNQQ 用来启动另一程序，这

跟函数 WinExec 的作用是相同的。但是代码中并没有采用同一个函数启动不同的应用程序，这是因为当用户用鼠标移动用 RUNQQ 启动的计算器界面窗口的时候，会出现界面刷新问题。

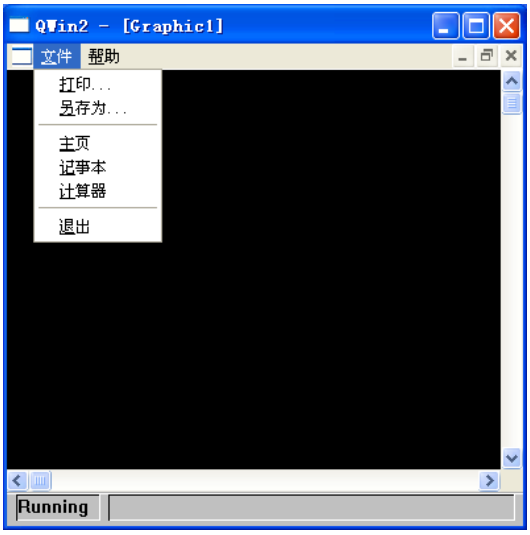


图 4.7 Example4-6 程序运行界面

通过键盘和鼠标事件，应用程序的交互性得到了很大的提高，通过程序菜单功能，程序具有了较好的直观性和易用性。但是，当应用程序中出现数据的输入或其他更加复杂的交互时，菜单就不能完全胜任，此时必须借助于对话框等资源文件。关于资源文件的用法，请阅读下一章要讲述的内容。

第五章 QuickWin 基本控件的使用

程序设计中，为了程序具有良好的交互能力，对话框控件必不可少。Windows 中的对话框控件功能强大、使用直观方便， 不但使程序的交互能力大大提高，而且使程序的应用直观方便了许多。本章主要介绍常见对话框控件的使用方法。

5-1 控件的使用方法

如果您在程序设计中还没有使用过对话框控件，那么您也不必担心，因为 Quickwin 中，对话框控件的使用很简单。只要掌握了一种控件的使用方法，则其它控件的使用大同小异。所以，让我们从最简单的控件开始来进入对话框程序的学习。

下面这个例子中，首先用函数 INITIALSETTINGS 设定初始界面和菜单，然后通过菜单打开一个对话框，向编辑框内输入任意字符串，最后取出编辑框内用户的输入并将其输出到屏幕上，以此来判断程序运行的正确性。因为是第一个对话框应用程序，所以下面将详细给出程序的编写过程。详细步骤如下：

■ 首先，按照以前的步骤建立 Quickwin 工程，将下面的源程序添加到工程中；

```
#001   example 5-1
#002   program main
#003     USE IFQWIN
#004     implicit none
#005     integer( kind=4 ) :: results
#006     logical(kind=4)::res
#007     type (qwinfo) :: winfo
#008     type (windowconfig) :: wc
#009
#010     winfo%type = QWIN$MAX
#011     res = GETWINDOWCONFIG(wc)
#012     ! 最大化子窗口
#013     results = SETWSIZEQQ (0, winfo)
#014     results = SETEXITQQ (QWIN$EXITPERSIST)
#015     do while (.true.)
#016     enddo
#017   end program main
#018
#019   LOGICAL(4) FUNCTION INITIALSETTINGS( )
#020     USE IFQWIN
#021     implicit none
#022     integer( kind=4 ) :: results
#023     logical(kind=4)::res
#024     type (qwinfo) :: winfo
#025     EXTERNAL:: Datainput
#026
#027     winfo%w = 800
#028     winfo%h = 600
#029     winfo%type = QWIN$SET
#030     results = SetWSizeQQ( QWIN$FRAMEWINDOW, winfo )
#031
#032     res = appendmenuqq(1, $MENUENABLED, '文件'C, NUL)
```

```
#033     res = appendmenuqq(1, $MENUENABLED, '数据…'C, Datainput)
#034     INITIALSETTINGS= .true.
#035     return
#036   END FUNCTION INITIALSETTINGS
#037
#038   subroutine Datainput
#039     USE IFWIN
#040     implicit none
#041
#042   end subroutine Datainput
```

这段程序中，函数 INITIALSETTINGS 将程序界面初始化为 800x600，同时创建了“文件”菜单，其中包括一个菜单项“数据...”。运行上面的程序，得到图 5.1 所示的程序界面。当单击菜单“数据...”时，程序没有任何响应，这是因为在菜单的回调子程序中目前还没有设置任何动作。至于响应什么动作，后面将逐一介绍。

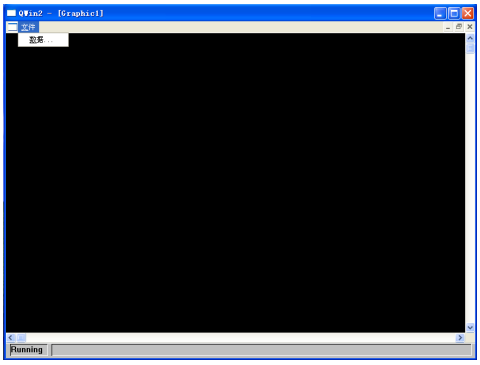


图 5.1 Example5-1 程序运行界面

■ 其次，为应用程序创建对话框；

按照最初的想法，当用户单击菜单“数据...”时，程序将弹出一个对话框，然后在编辑框内输入文本并获取，进而在子窗口中输出。为了创建对话框，应执行以下步骤：

● 在 VS2008 界面中，将鼠标移至“解决方案资源管理器”中的“Resource Files”文件夹，单击鼠标右键，从弹出快捷菜单中选择“添加”的下一级菜单“新建项…”，如图 5.2 所示；

● 在弹出的如图 5.3 所示的模板选择对话框中选中“Resource File(.rc)”，同时指定资源文件的保存路径，单击“添加”按钮完成资源文件的创建。此时资源文件变为当前可见，如图 5.4 所示；

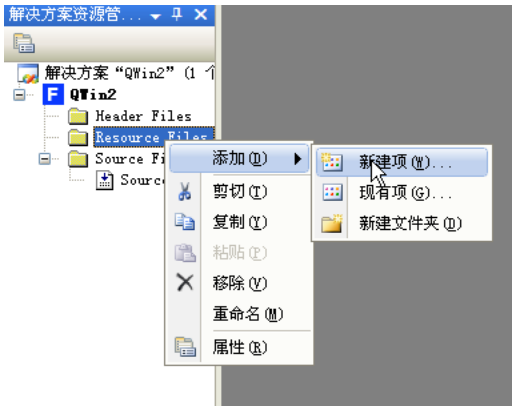


图 5.2 解决方案资源管理器

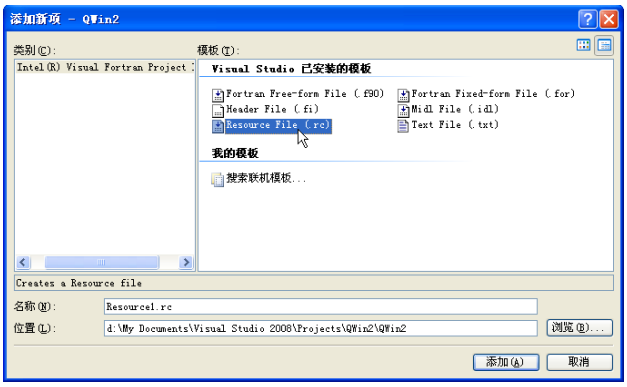


图 5.3 模板选择对话框

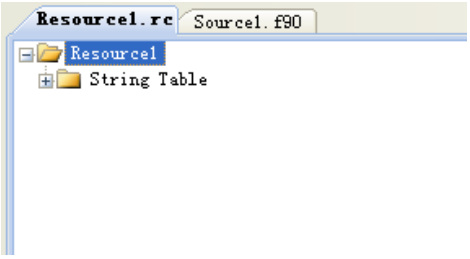


图 5.4 项目资源文件

● 将鼠标移至图 5.4 中的“Resource1”(资源文件名)文件夹，单击鼠标右键，在弹出的快捷菜单中选择“添加资源...”，如图 5.5 所示；

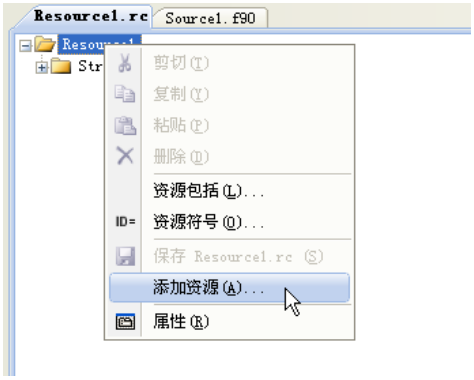


图 5.5 “资源添加”快捷菜单

● 接着弹出如图 5.6 所示的“添加资源”对话框，在“资源类型”列表中选择“Dialog”，单击“新建”按钮；

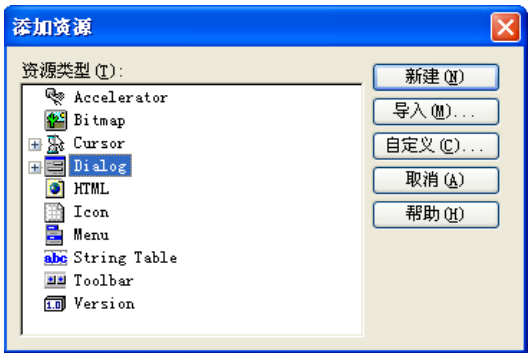


图 5.6 “添加资源”对话框

● 到此，对话框终于出现了，如图 5.7 所示。这是系统提供的默认对话框，对话框的右下角仅有两个常用按钮。

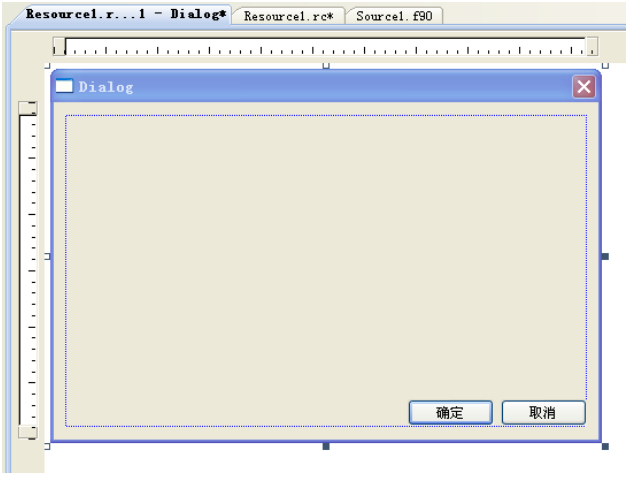


图 5.7 对话框控件

● 为了在对话框中加入其它控件，就必须找出控件工具箱。在 VS2008 界面左边边界处有一个动态可伸缩工具栏“工具箱”，将鼠标移至该工具栏图标处且稍作停留，系统会自动弹出控件工具箱如图 5.8 所示。

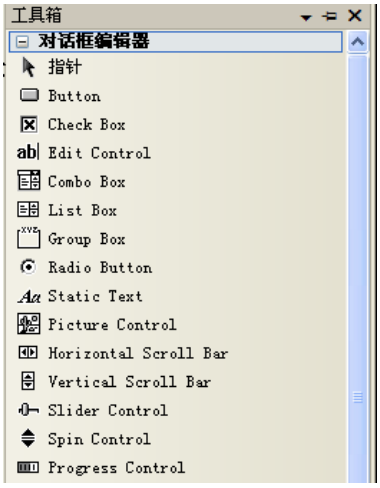


图 5.8 控件工具箱

● 为了在对话框中添加编辑框，首先用鼠标单击图 5.8 中的“Edit Control”，然后将鼠标移到对话框上，通过拖拉移动鼠标的方式完成编辑框的绘制；或者直接用鼠标双击“Edit Control”，此时在对话框上自动生成一个编辑框。

注意：如果编辑框的大小不符合要求，可再次用鼠标单击编辑框，利用周围的夹点进行多次编辑，直到符合要求为止；如果对话框中的控件较多，为了便于控件位置的统一安排，请借助如图 5.9 所示的“对话框编辑器”工具栏进行控件大小和位置的进一步控制。



图 5.9 对话框编辑器

● 通过控件的编辑，最终形成如图 5.10 所示的对话框布局。为了观察程序运行

中对话框的真实状态，可通过单击图 5.9 所示的“对话框编辑器”工具栏上的第一个按钮来查看。



图 5.10 对话框布局

此时，我们已经完成了对话框的制作，为了保证所作工作的可靠性，请单击工具栏上的“保存”按钮完成资源文件的存盘。如果现在打开项目所在的文件夹，我们发现在原文件数目的基础上多出了三个文件如图 5.11 所示。



图 5.11 资源文件

这三个文件中，第一个是我们最关心的。用记事本打开它，内容为：

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by Resource1.rc
//
#define IDD_DIALOG1                     101
#define IDC_EDIT1                       1000

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        102
#define _APS_NEXT_COMMAND_VALUE         40001
#define _APS_NEXT_CONTROL_VALUE         1001
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

乍一看，似乎根本看不懂。但是读者不要着急，我们首先来个大胆的猜想：对话框中出现了很多控件，程序又如何能识别呢？如果每个控件都有独一无二的名字，显然程序识别就没有困难。幸运的是，我们的想法非常真确！当我们在对话框中“构筑”一个个控件的同时，编译器已经为每个控件起好了名字，不仅如此，为了便于和程序代码的衔接，编译器同时给每个控件加以编号！

让我们先回到资源编辑器中。用鼠标左键单击编辑框控件，然后单击鼠标右键，在弹出的快捷菜单中选择“属性”，编译器界面中出现了如图 5.12 所示的控件属性对

话框。在对话框中，我们看到编辑框的 ID 属性值为 IDC_EDIT1，如图 5.12(左)；如果此时单击对话框本身，则对话框的 ID 为 IDD_DIALOG1，如图 5.12(右)。

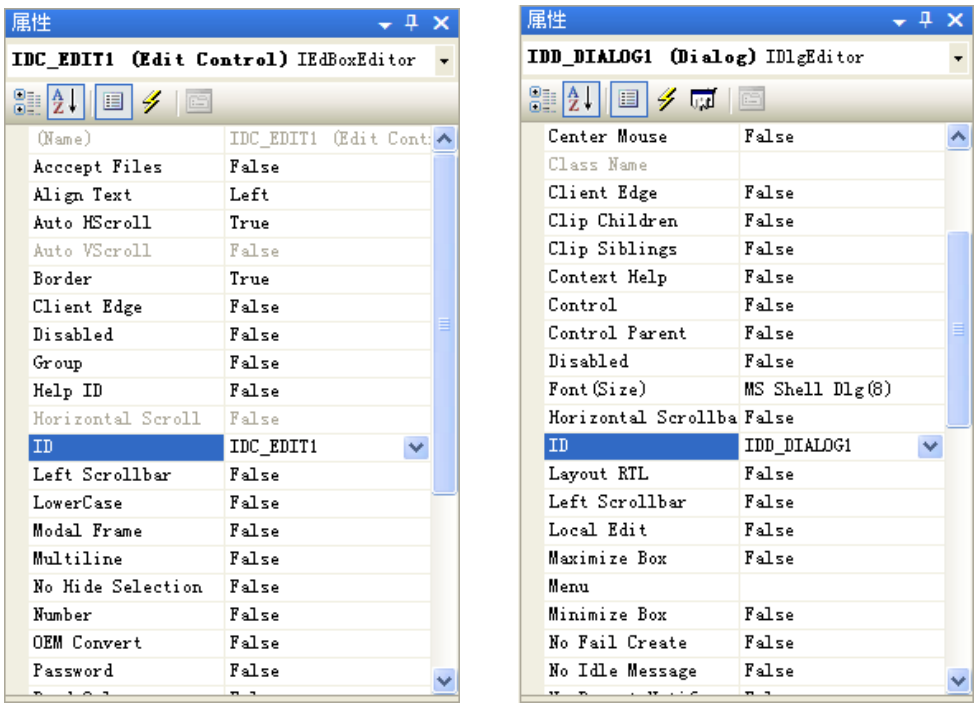


图 5.12 资源属性对话框

回到图 5.11 所示的第一个文件，原来文件中的第 5 行和第 6 行的目的是将控件和特定的编号一一对应。如此一来，程序设计就变得方便、简单。

有读者可能问：对话框中还有两个按钮，怎么没有出现它们的名称和代号呢？原因是这两个按钮控件为常用控件，程序中只需要用控件名称就足够了。

■ 最后，为对话框编写驱动程序。

为了在程序中控制对话框，首先在程序代码中包含语句“USE IFLOGM”，在这个模块中，编译器将对话框相关的派生变量和接口封装起来便于使用；其次，将已经创建好的对话框和派生自对话框类的变量联系起来，做法如下：

integer(kind=4) :: results

type(dialog)::datadlg

retlog = DLGINIT(IDD_DIALOG1, datadlg)

通过以上代码，对话框 IDD_DIALOG1 和程序联系起来了。在随后的代码中，如果需要，将对话框的名称 IDD_DIALOG1 和派生自对话框类的变量 datadlg 按参数传递就可以了；最后，为了节约空间，用函数 DLGUNINIT 释放对话框所占内存。下面是对话框 IDD_DIALOG1 完整的驱动程序：

```
#001   subroutine Datainput
#002   USE IFWIN
#003   USE IFLOGM
#004   USE CONTROLS
#005   implicit none
#006   integer( kind=4 ) :: results
#007   character( len=256 )::text
#008   logical( kind=4 )::retlog
```

```
#009   type( dialog )::datadlg
#010
#011   retlog = DLGINIT(IDD_DIALOG1, datadlg)
#012
#013   results = DLGMODAL (datadlg)
#014   if(results == IDOK) then
#015     retlog = DLGGET(datadlg, IDC_EDIT1, text)
#016     write(0,*) text
#017   endif
#018   call DLGUNINIT (datadlg)
#019   end subroutine Datainput
```

程序中，第 9 行定义了一个对话框类变量；第 11 行用来初始化对话框，也就是将变量和对话框联系起来；第 13 行将对话框显示出来；第 14~17 行进行判断，如果用户按下“确定”按钮，则用函数 DLGGET 获取编辑框内输入的文本并保存在字符串变量“text”中；第 18 行程序释放对话框所占内存空间。

到目前为止，一切似乎很顺利。将上面的程序段加入之前的主程序，编译运行，编译器却出现了提示错误！原因是编译器并不知道 IDD_DIALOG1 和 IDC_EDIT1 究竟是谁。再次回到图 5.11 所示资源文件中的第一个文件，原来这个文件是 Microsoft Visual C++自动产生，Fortran 编译器并不能自动识别，所以，我们必须将其中的控件名称和其对应的代号翻译过来，让 Fortran 编译器在程序运行的时候能够自动识别。翻译后的代码如下：

```
module CONTROLS
   integer, parameter :: IDD_DIALOG1    =   101
   integer, parameter :: IDC_EDIT1    =   1000
end module CONTROLS
```

将上述代码段加入主程序，继续编译运行，终于成功了。注意：第一，如果控件比较少，则将上述模块和主程序放在一起，方便修改编译，如果控件较多，此时应将该模块单独放在一个文件中，方便修改；第二，在翻译形成该模块的时候，请借助于有列编辑功能的编辑器，如 Uedit32 等，这样可提高工作效率。

至此，第一个对话框程序完成了。为了有更加清晰的思路，下面给出这个例子的完整代码。

```
#001   ! Example 5-2
#002   module CONTROLS
#003     integer, parameter :: IDD_DIALOG1  =  101
#004     integer, parameter :: IDC_EDIT1    =  1000
#005   end module CONTROLS
#006
#007   program main
#008     USE IFQWIN
#009     implicit none
#010     integer( kind=4 ) :: results
#011     logical(kind=4)::res
#012     type (qwinfo) :: winfo
#013     type (windowconfig) :: wc
#014
#015     winfo%type = QWIN$MAX
#016     res = GETWINDOWCONFIG(wc)
#017     ! 最大化子窗口
```

```
#018     results = SETWSIZEQQ (0, winfo)
#019     results = SETEXITQQ (QWIN$EXITPERSIST)
#020     do while (.true.)
#021     enddo
#022   end
#023
#024   LOGICAL(4) FUNCTION INITIALSETTINGS( )
#025     USE IFQWIN
#026     implicit none
#027     integer( kind=4 ) :: results
#028     logical(kind=4)::res
#029     type (qwinfo) :: winfo
#030     external:: Datainput
#031
#032     winfo%w = 400
#033     winfo%h = 400
#034     winfo%type = QWIN$SET
#035     results = SetWSizeQQ( QWIN$FRAMEWINDOW, winfo )
#036     res = appendmenuqq(1, $MENUENABLED, '文件'C, NUL)
#037     res = appendmenuqq(1, $MENUENABLED, '数据...'C,Datainput)
#038     INITIALSETTINGS= .true.
#039     return
#040   END FUNCTION INITIALSETTINGS
#041
#042   subroutine Datainput(checked)
#043     USE IFWIN
#044     USE IFLOGM
#045     USE CONTROLS
#046     implicit none
#047     integer( kind=4 ) :: results
#048     character( len=256 )::text
#049     logical( kind=4 )::retlog,checked
#050     type( DIALOG )::datadlg
#051
#052     retlog = DLGINIT(IDD_DIALOG1, datadlg)
#053     results = DLGMODAL (datadlg)
#054     if(results == IDOK) then
#055       retlog = DLGGET(datadlg, IDC_EDIT1, text)
#056       write(0,*) text
#057     endif
#058     call DLGUNINIT (datadlg)
#059   end subroutine Datainput
```

运行结果如图 5.13 所示。

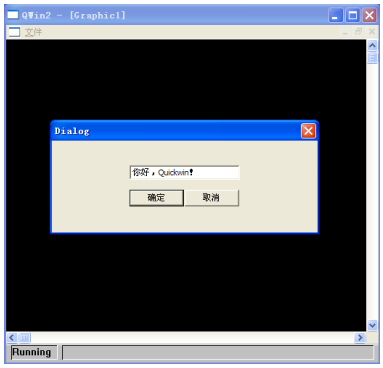


图 5.13 Example5-2 程序运行界面

前面这个例子中，对话框中仅有一个编辑框控件。其实对 Quickwin 工程来说，所有的对话框及其控件的使用方法都是一样的，不同的仅仅是控件的相关参数。至于 Intel 编译器支持的其他控件的用法，这将是下一节的内容。

5-2 各种控件的用法

Fortran 编译器支持的控件非常丰富，包括静态文本框，编辑框和按钮控件等。下面就不同控件的具体用法展开详细的讨论。

■ 编辑框 程序中使用编辑框可以使用户方便地输入相关数据。在文本编辑框中，不管是用户输入的内容，还是程序返回编辑框中的内容，一律被视为字符串。如果程序想通过编辑框输入数据或相反的操作，则必须进行数据类型的转换，具体方法是内部文件法。

文本编辑框有两个常见的动作，一是用函数 DLGDET 将用户输入的数据提取出来，二是用函数 DLGSET 将计算所得结果返回编辑框内。下面是详细步骤：

```
USE IFLOGM
type ( dialog )::dlg
logical( kind=4 ):: retlog
character( len=* ):: text
retlog = DLGSET ( dlg, IDC_EDITBOX1, text )
retlog = DLGGET ( dlg, IDC_EDITBOX1, text )
```

为从编辑框中读出输入的数据，需要进行数据类型转换。首先将编辑框内的数据按照字符串读入字符串变量 text，接着用内部文件将 text 内的字符串读入实型变量。代码如下：

```
real( kind=8 ) :: x
logical( kind=4 ) :: retlog
character( len=256 ) ::text
retlog = DLGGET ( dlg, IDC_EDITBOX1, text )
read (text, *) x
```

下面的代码是将整型数值返回文本编辑框的过程：

```
integer( kind=4 ) :: j
logical( kind=4 ) :: retlog
character( len=256 ):: text
write (text,'(i4)') j
retlog = DLGSET (dlg, IDC_EDITBOX1, text)
```

■ 静态文本框 一般用来显示描述性的文字或者输出结果。如果在程序中仅仅用来显示描述性文字，则静态文本框的用法非常简单，只需将静态文本框的“Caption”属性值改为需要的文字即可；如果需要在程序中用静态文本框显示计算结果，则资源文件中必须赋予静态文本框独立的名称和代号，然后在代码中用函数 DLGSET 更改静态文本框的内容，方法同编辑框。

■ 编组框 将对话框中的控件分类编组，使控件结构清晰，排列整齐。编组框的用法很简单，只需要程序设计者用鼠标拖拉的方式将成组的控件包围起来，必要时修改“Caption”属性即可。

■ 按钮 对话框程序设计中，按钮扮演着非常重要的角色。一般来说，当按钮按下时，程序会执行一定的动作，而这个动作是由按钮的回调函数完成。为了将按钮和特定的动作相关联，需用到函数 DlgSetSub，语法为：

```
USE IFLOGM
type ( dialog )::dlg
logical( kind=4 ) :: retlog
external :: DisplayTime
retlog = DlgSetSub( dlg, IDC_BUTTON_TIME, DisplayTime)
```

其中 DisplayTime 为按钮 IDC_BUTTON_TIME 的回调程序。也就是说，当用户按下该按钮时，名为 DisplayTime 的子程序同时被激活。为了正确设计回调子程序，请大家严格遵守下面的格式：

```
subroutine DisplayTime ( dlg, control_id, callbacktype)
  !DEC$ ATTRIBUTES DEFAULT :: DisplayTime
  USE IFLOGM
  type(dialog) :: dlg
  integer(kind=4):: control_id, callbacktype
end subroutine DisplayTime
```

当用户按下的是系统提供的“确定”或“取消”按钮，默认情况下对话框会消失，同时返回使对话框消失的按钮的 ID。

为了更好地理解前四种控件的用法，下面给出一个较为复杂的程序，程序的目的是求解一元二次方程。当用户在三个编辑框内输入一元二次方程的三个系数后，单击“求解”按钮，方程求解后，将方程的根显示出来。

首先设计对话框：先打开资源编辑器，根据图 5.14 所示的对话框设计图，在默认的对话框上分别绘制 ID 为 IDC_EDIT1、IDC_EDIT2 和 IDC_EDIT3 的三个编辑框，如图 5.14 中的 edit1、edit2 和 edit3；然后在编号为 1~4 的位置分别绘制静态文本框，将 ID 分别设为 IDC_STATIC1、IDC_STATIC2、IDC_STATIC3 和 IDC_STATIC4，同时将属性“Caption”的值设置为空；为了数据输入和结果显示的方便，用静态文本框做了一系列的说明性的文字，这些文本请读者自行添加；接着绘制编组框，将上部控件进行编组，更改编组框属性“Caption”的值为“方程求解器”；最后将系统提供的“确定”和“取消”按钮移位，同时改“确定”为“求解”，保存资源文件。最终的对话框测试图如图 5.15 所示。

其次，编写主程序和对话框驱动程序。在主程序中主要设置主窗口界面的尺寸大小及菜单结构，而在菜单的回调函数中初始化对话框，目的是当用户选择菜单时，界面中弹出对话框。为了单击“求解”按钮之后方程自动求解且返回方程根，程序中编写了“求解”按钮的回调子程序。详细代码如下：

```
#001   ! Example 5-3
#002   module CONTROLS
#003     integer, parameter :: IDD_DIALOG1  =  101
#004     integer, parameter :: IDC_EDIT1    =  1000 !a值
#005     integer, parameter :: IDC_EDIT2    =  1001 !b值
#006     integer, parameter :: IDC_EDIT3    =  1002 !c值
#007     integer, parameter :: IDC_STATIC1  =  1003 !>,=或<
#008     integer, parameter :: IDC_STATIC2  =  1004 !解的第一部分
#009     integer, parameter :: IDC_STATIC3  =  1005 !解的第二部分
#010     integer, parameter :: IDC_STATIC4  =  1006 !实根还是虚根
#011
#012     character( len=1 )::text1
#013     character( len=4 )::text2
#014   end module CONTROLS
#015
#016   ! 主程序
#017   program main
#018     USE IFQWIN
#019     implicit none
#020     integer( kind=4 ) :: results
#021     logical(kind=4)::res
#022     type (qwinfo) :: winfo
#023     type (windowconfig) :: wc
#024
#025     winfo%type = QWIN$MAX
#026     res = GETWINDOWCONFIG(wc)
#027     ! 最大化子窗口
#028     results = SETWSIZEQQ (0, winfo)
#029     results = SETEXITQQ (QWIN$EXITPERSIST)
#030     do while (.true.)
#031     enddo
#032   end program main
#033
#034   ! 初始化界面尺寸和菜单
#035   LOGICAL(4) FUNCTION INITIALSETTINGS( )
#036     USE IFQWIN
#037     implicit none
#038     integer( kind=4 ) :: results
#039     logical(kind=4)::res
#040     type (qwinfo) :: winfo
#041     external:: Dialoginput
#042
#043     winfo%w = 800
#044     winfo%h = 600
#045     winfo%type = QWIN$SET
#046     results = SetWSizeQQ( QWIN$FRAMEWINDOW, winfo )
#047     res = appendmenuqq(1, $MENUENABLED, '文件'C, NUL)
#048     res = appendmenuqq(1, $MENUENABLED, '解方程...'C,Dialoginput)
#049     INITIALSETTINGS= .true.
#050     return
#051   END FUNCTION INITIALSETTINGS
#052
#053   ! 菜单的回调子程序
#054   subroutine Dialoginput(checked)
#055     USE IFWIN
#056     USE IFLOGM
#057     USE CONTROLS
#058     implicit none
```

```
#059     integer( kind=4 )::results
#060     logical( kind=4 )::retlog,checked
#061     type( dialog )::datadlg
#062     external:: dialogbasedApply
#063
#064     ! 初始化对话框
#065     retlog = DLGINIT(IDD_DIALOG1, datadlg)
#066     ! 设置按钮“求解”的回调函数
#067     retlog = DlgSetSub(datadlg, IDOK, dialogbasedApply)
#068     ! 真正显示对话框
#069     results = DLGMODAL (datadlg)
#070     ! 释放占用资源
#071     call DLGUNINIT (datadlg)
#072   end subroutine Dialoginput
#073
#074   ! 按钮“求解”的回调子程序
#075   subroutine dialogbasedApply( dlg, id, callbacktype )
#076     !DEC$ ATTRIBUTES DEFAULT :: dialogbasedApply
#077     USE IFLOGM
#078     USE CONTROLS
#079     implicit none
#080     type(dialog) dlg
#081     logical*4 results
#082     integer(kind=4):: id, callbacktype
#083     real(kind=8)::a,b,c,x1,x2
#084     character(len=30)::str
#085
#086     ! 提取编辑框中的数据
#087     results = DlgGet(dlg,IDC_EDIT1,str)
#088     read(str,*)a
#089     results = DlgGet(dlg,IDC_EDIT2,str)
#090     read(str,*)b
#091     results = DlgGet(dlg,IDC_EDIT3,str)
#092     read(str,*)c
#093     ! 解方程
#094     call solve(a,b,c,x1,x2)
#095     ! 改变静态文本框的内容
#096     results = DlgSet(dlg,IDC_STATIC1,text1)
#097     results = DlgSet(dlg,IDC_STATIC4,text2)
#098     write(str,*)x1
#099     results = DlgSet(dlg,IDC_STATIC2,str)
#100     write(str,*)x2
#101     results = DlgSet(dlg,IDC_STATIC3,str)
#102   end subroutine dialogbasedApply
#103
#104   ! 方程求解
#105   subroutine solve(a,b,c,x1,x2)
#106     USE CONTROLS
#107     implicit none
#108     real(kind=8)::a,b,c,disc,x1,x2
#109     disc = (b*b - 4*a*c)
#110     if (disc .ge. 0) then
#111       x1 = (- b + sqrt( disc ) )/(2.0*a)
#112       x2 = (- b - sqrt( disc ) )/(2.0*a)
#113       text1 = ">="
#114       text2 = "实根"
#115     else
#116       x1 = -b/(2*a)
```

```
#117       x2 = sqrt(-disc)/(2*a)
#118       text1 = "<"
#119       text2 = "虚根"
#120     end if
#121   end subroutine solve
```

运行程序，在编辑框内输入数据，单击“求解”按钮，测试程序运行情况如图 5.16 所示。上述程序中并没有考虑用户输入的异常，如果希望有更加稳定、健壮的程序代码，程序设计中必须充分考虑到各种异常情况，从而避免程序运行中的不稳定或崩溃。

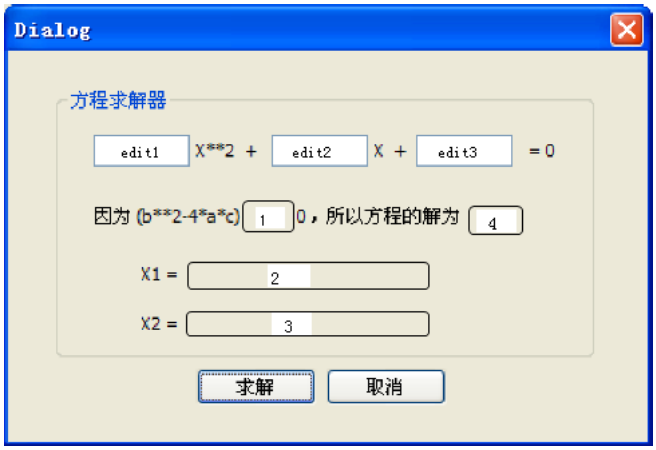


图 5.14 对话框设计图

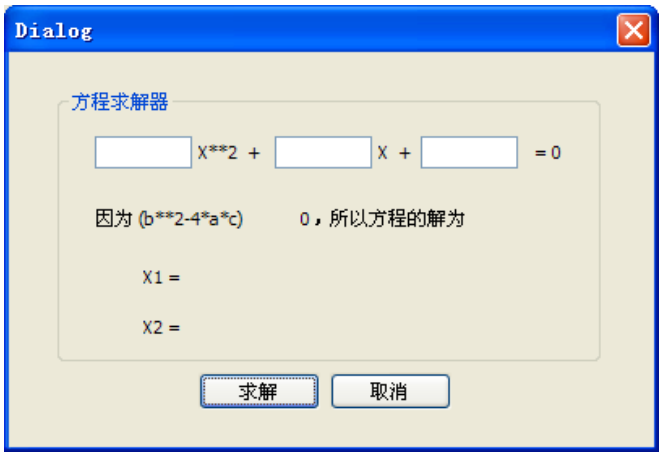


图 5.15 对话框测试图

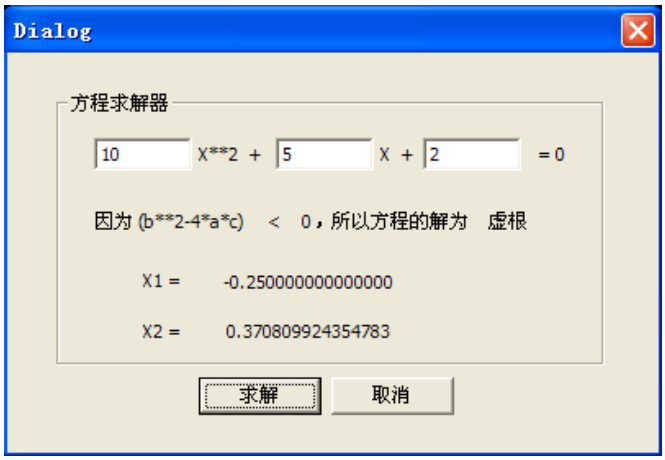


图 5.16 对话框运行图

■ 单选按钮 适用于应用程序中多选一的情况。为了方便选择，单选按钮经常和静态文本框配合使用。每个单选按钮的可能取值只有两种情况：当被选中时，其值为“.True.”；没有被选中时，值为“.False.”。为了获取和设置单选按钮的当前值，可用函数 DLGGET 和 DLGSET，用法为：

```
USE IFLOGM
type ( dialog ) :: dlg
logical( kind=4 ):: retlog,pushed_state
retlog = DLGGET (dlg, IDC_RADIOBUTTON1, pushed_state)
retlog = DLGSET (dlg, IDC_RADIOBUTTON1, .true.)
```

■ 下拉列表框 用于多选一，在这个意义上，下拉列表框和单选按钮的作用是相同的。但是下拉列表框中可以容纳很多不同的选项，而由于屏幕控件等的限制，单选按钮的数量总是很有限的。

下拉列表框是列表框和编辑框的组合体，所以在下拉列表中，用户可以自由选择其中的选项，也可以任意输入符合格式的项目。在这点上，比编辑框和列表框具有更大的灵活性。

为了在程序中使用下拉列表框，必须掌握下拉列表框的操作方法：

首先，初始化下拉列表框。在对话框显示之前，下拉列表框必须完成初始化。方法如下：

```
USE IFLOGM
logical( kind=4 )::retlog
type( DIALOG )::dlg
retlog = DlgSet ( dlg, IDC_COMBO1, 3, DLG_NUMIT )  ！共添加3 个选项
retlog = DlgSet ( dlg, IDC_COMBO1, "Moe", 1 )  ！第一个
retlog = DlgSet ( dlg, IDC_COMBO1, "Larry", 2 )  ！第二个
retlog = DlgSet ( dlg, IDC_COMBO1, "Curly", 3 )  ！第三个
```

其次，程序使用中需要获取当前的选择值：

```
USE IFLOGM
logical( kind=4 )::retlog
type( DIALOG )::dlg
character( len=* )::text
results = DLGGET (dlg, IDC_COMBO1, text)
```

最后，如果在程序的运行过程中需要动态增加或删除列表项，则方法为：

```
USE IFLOGM
logical( kind=4 )::retlog
type( DIALOG )::dlg
retlog = DlgSet ( dlg, IDC_COMBO1, 4 )  ！共4 个选项
retlog = DlgSet ( dlg, IDC_COMBO1, "John", 4 )  ！现在添加第4 个项目
```

此时在项目列表框中增加一项。如果需要删除，则只需重新确定项目个数，则总

个数之后的项目列表被自动删除。

下面是一个综合示例。用户填入基本资料后单击“确定”按钮，此时用户的资料被自动汇总在“个人简介”编组框中(程序中用更新静态文本框内容放入方法实现)。程序创建过程为：

首先创建资源文件。打开对话框编辑器，按照图 5.17 所示在对话框上创建各种控件。



图 5.16 对话框运行图

需要注意两点：一是将下拉列表框的有效列表范围绘制足够大；二是在图 5.17 中夹点的范围内需要创建静态文本框。

除了一些可以忽略的控件 ID(如两个编组框和显示描述性文字的静态文本框)之外，按照从左到右、从上到下的顺序，其它控件的 ID 分别为：IDC_EDIT1、IDC_RADIO1、IDC_RADIO2、IDC_COMBO1、IDC_COMBO2、IDC_COMBO3 和 IDC_STATIC1。至于各个控件的代号，请务必保证和资源编辑器生成的代号相同。

其次，编写程序代码驱动对话框及其控件。将下面的代码加入项目工程：

```
#001   ! Example 5-4
#002   module CONTROLS
#003     integer, parameter :: IDD_DIALOG1  =  101
#004     integer, parameter :: IDC_EDIT1    =  1000 !姓名
#005     integer, parameter :: IDC_RADIO1   =  1001 !男
#006     integer, parameter :: IDC_RADIO2   =  1002 !女
#007     integer, parameter :: IDC_COMBO1   =  1004 !年
#008     integer, parameter :: IDC_COMBO2   =  1005 !月
#009     integer, parameter :: IDC_COMBO3   =  1007 !日
#010     integer, parameter :: IDC_STATIC1  =  1009 !简介
#011   end module CONTROLS
#012
#013   program main
#014     USE IFQWIN
#015     implicit none
#016     integer( kind=4 ) :: results
#017     logical(kind=4)::res
#018     type (qwinfo) :: winfo
#019     type (windowconfig) :: wc
#020
```

```
#021     winfo%type = QWIN$MAX
#022     res = GETWINDOWCONFIG(wc)
#023     ! 最大化子窗口
#024     results = SETWSIZEQQ (0, winfo)
#025     results = SETEXITQQ (QWIN$EXITPERSIST)
#026     do while (.true.)
#027     enddo
#028   end program main
#029
#030   LOGICAL(4) FUNCTION INITIALSETTINGS( )
#031     USE IFQWIN
#032     implicit none
#033     integer( kind=4 ) :: results
#034     logical(kind=4)::res
#035     type (qwinfo) :: winfo
#036     external:: Dialoginput
#037
#038     winfo%w = 800
#039     winfo%h = 600
#040     winfo%type = QWIN$SET
#041     results = SetWSizeQQ( QWIN$FRAMEWINDOW, winfo )
#042     res = appendmenuqq(1, $MENUENABLED, '文件'C, NUL)
#043     res = appendmenuqq(1, $MENUENABLED, '个人资料...'C,Dialoginput)
#044     INITIALSETTINGS= .true.
#045     return
#046   END FUNCTION INITIALSETTINGS
#047
#048   subroutine Dialoginput(checked)
#049     USE IFWIN
#050     USE IFLOGM
#051     USE CONTROLS
#052     implicit none
#053     integer( kind=4 )::results
#054     logical( kind=4 )::retlog,checked
#055     type( dialog )::datadlg
#056     external:: dialogbasedApply
#057
#058     ! 初始化对话框
#059     retlog = DLGINIT(IDD_DIALOG1, datadlg)
#060     ! 初始化第一个下拉列表
#061     retlog = DlgSet ( datadlg, IDC_COMBO1, 3, DLG_NUMITEMS )
#062     retlog = DlgSet ( datadlg, IDC_COMBO1, "2009", 1 )
#063     retlog = DlgSet ( datadlg, IDC_COMBO1, "2010", 2 )
#064     retlog = DlgSet ( datadlg, IDC_COMBO1, "2011", 3 )
#065     ! 初始化第二个下拉列表
#066     retlog = DlgSet ( datadlg, IDC_COMBO2, 3, DLG_NUMITEMS )
#067     retlog = DlgSet ( datadlg, IDC_COMBO2, "9", 1 )
#068     retlog = DlgSet ( datadlg, IDC_COMBO2, "10", 2 )
#069     retlog = DlgSet ( datadlg, IDC_COMBO2, "11", 3 )
#070     ! 初始化第三个下拉列表
#071     retlog = DlgSet ( datadlg, IDC_COMBO3, 3, DLG_NUMITEMS )
#072     retlog = DlgSet ( datadlg, IDC_COMBO3, "1", 1 )
#073     retlog = DlgSet ( datadlg, IDC_COMBO3, "2", 2 )
#074     retlog = DlgSet ( datadlg, IDC_COMBO3, "3", 3 )
#075     ! 设置“确定”按钮的回调子程序
#076     retlog = DlgSetSub(datadlg, IDOK, dialogbasedApply)
#077     results = DLGMODAL (datadlg)
#078     call DLGUNINIT (datadlg)
```

```
#079   end subroutine Dialoginput
#080
#081   subroutine dialogbasedApply( dlg, id, callbacktype )
#082     !DEC$ ATTRIBUTES DEFAULT :: dialogbasedApply
#083     USE IFLOGM
#084     USE CONTROLS
#085     implicit none
#086     type(dialog) dlg
#087     logical*4 results,pushed_state
#088     integer(kind=4):: id, callbacktype
#089     character(len=10)::name,sex,year,month,day
#090     character(len=100)::text1
#091
#092     sex = "女"
#093     ! 取出姓名，提取性别单选钮
#094     results = DlgGet(dlg,IDC_EDIT1,name)
#095     results = DlgGet(dlg,IDC_RADIO1,pushed_state)
#096     if(pushed_state == .true.)then
#097       sex = "男"
#098     endif
#099     ! 读取下拉列表的当前值
#100     results = DLGGET (dlg, IDC_COMBO1, year)
#101     results = DLGGET (dlg, IDC_COMBO2, month)
#102     results = DLGGET (dlg, IDC_COMBO3, day)
#103     ! 将所有的资料汇总
#104     write(text1,*)"    我叫",trim(adjustl(name)),                &
#105                   ", ",trim(adjustl(sex)),", 出生于",             &
#106                   trim(adjustl(year)),"年",trim(adjustl(month)), &
#107                   "月",trim(adjustl(day)),"日。"
#108     ! 改写静态文本框的内容
#109     results = DlgSet(dlg,IDC_STATIC1,text1)
#110   end subroutine dialogbasedApply
```

运行程序，对话框界面如图 5.17 所示。

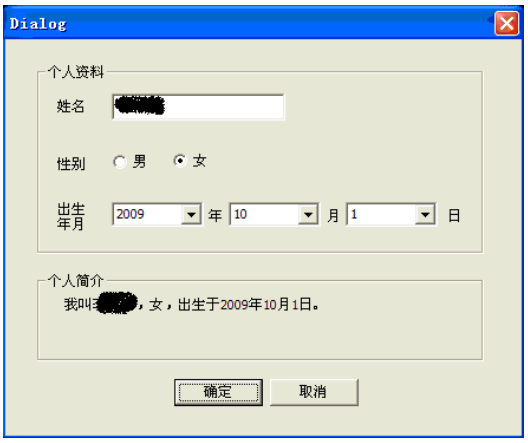


图 5.17 Example 5-4 运行结果

■ 滚动条 为了在有限的范围内容纳更多的内容，经常用到滚动条。滚动条通过中间的滑块来确定位置，所以滑块有一个可滑动的范围，这个范围的确定方法为：

USE IFLOGM

type (dialog) :: dlg

logical(kind=4):: retlog

retlog = DLGSET (dlg, IDC_SCROLLBAR1, 0, DLG_RANGEMIN)

retlog = DLGSET (dlg, IDC_SCROLLBAR1, 200, DLG_RANGEMAX)

滚动条使用中，用户或者用鼠标单击两端的箭头实现中间滑块的滑动，这种情况下，滑块的步长为 1；或者用鼠标单击滑块和箭头之间的空白区域实现滑块的滑动，这时可用下面的方法设置滑块的步长：

```
USE IFLOGM
type ( dialog ) :: dlg
logical( kind=4 ):: retlog
retlog = DLGSET (dlg, IDC_SCROLLBAR1, 20, DLG_BIGSTEP)
```

如果需要设置或者获取滚动条的当前位置，则应借助于函数 DLGSET 和 DLGGET 来实现：

```
retlog = DLGSET (dlg, IDC_SCROLLBAR1, 50, DLG_POSITION)
retlog = DLGGET (dlg, IDC_SCROLLBAR1, slide_position, DLG_POSITION)
```

■ 进度条 如果程序中有较长时间的等待，建议使用进度条。进度条仅仅是输出控件，所以本身不支持任何的回调函数。为了使用进度条，程序设计者只需指定进度条的有效范围和位置。具体方法为：

```
USE IFLOGM
type ( dialog ) :: dlg
logical( kind=4 ):: retlog
retlog = DLGSET (dlg, IDC_PROGRESS1, 0, DLG_RANGEMIN)   ！最小范围
retlog = DLGSET (dlg, IDC_PROGRESS1, 200, DLG_RANGEMAX)   ！最大范围
retlog = DLGSET (dlg, IDC_PROGRESS1, 50, DLG_POSITION)   ！设定位置
```

■ 滑动条 可通过滑动改变或者调节数值的大小。跟前面的两个控件很像，程序设计者首先应指定滑动条的范围和当前位置：

```
USE IFLOGM
type ( dialog ) :: dlg
logical( kind=4 ):: retlog
retlog = DLGSET (dlg, IDC_SLIDER1, 0, DLG_RANGEMIN)   ！最小范围
retlog = DLGSET (dlg, IDC_SLIDER1, 200, DLG_RANGEMAX)   ！最大范围
retlog = DLGSET (dlg, IDC_SLIDER1, 50, DLG_POSITION)   ！设定位置
retlog = DLGGET (dlg, IDC_SLIDER1, slide_position, DLG_POSITION)   ！读取位置
```

同时，当键盘方向键按下时，这样设置滑块的步长：

```
retlog = DLGSET (dlg, IDC_SLIDER1, 10, DLG_SMALLSTEP)
```

或者响应“PAGE UP”、“PAGE DOWN”以及当鼠标单击滑道时的步长：

```
retlog = DLGSET (dlg, IDC_SLIDER1, 10, DLG_BIGSTEP)
```

或设置滑动条上的刻度间距：

```
retlog = DLGSET (dlg, IDC_SLIDER1, 20, DLG_TICKFREQ)
```

下面的示例给出了这三种控件的用法。程序中，当用户移动滚动条或滑动条其中之一时，其它两个控件的位置随之改变，达到同步互动的效果。项目创建过程为：

首先创建对话框。打开资源编辑器，按照图 5.18 所示创建对话框且绘制控件。创建完成后单击“保存”按钮完成该资源文件的保存。



图 5.18 对话框资源文件

其次，编写程序及对话框驱动程序。将下面的代码添加到工程：

```
#001   ! Example 5-5
#002   module CONTROLS
#003     integer, parameter :: IDD_DIALOG1     =  101
#004     integer, parameter :: IDC_SCROLLBAR1  =  1000
#005     integer, parameter :: IDC_SLIDER1      =  1001
#006     integer, parameter :: IDC_PROGRESS1   =  1002
#007   end module CONTROLS
#008
#009   program main
#010     USE IFQWIN
#011     implicit none
#012     integer( kind=4 ) :: results
#013     logical(kind=4)::res
#014     type (qwinfo) :: winfo
#015     type (windowconfig) :: wc
#016
#017     winfo%type = QWIN$MAX
#018     res = GETWINDOWCONFIG(wc)
#019     ! 最大化子窗口
#020     results = SETWSIZEQQ (0, winfo)
#021     results = SETEXITQQ (QWIN$EXITPERSIST)
#022     do while (.true.)
#023     enddo
#024   end program main
#025
#026   LOGICAL(4) FUNCTION INITIALSETTINGS( )
#027     USE IFQWIN
#028     implicit none
#029     integer( kind=4 ) :: results
#030     logical(kind=4)::res
#031     type (qwinfo) :: winfo
#032     external:: Dialoginput
#033
#034     winfo%w = 800
#035     winfo%h = 600
#036     winfo%type = QWIN$SET
#037     results = SetWSizeQQ( QWIN$FRAMEWINDOW, winfo )
#038
```

```
#039     res = appendmenuqq(1, $MENUENABLED, '文件'C, NUL)
#040     res = appendmenuqq(1, $MENUENABLED, '控件条...'C,Dialoginput)
#041     INITIALSETTINGS= .true.
#042     return
#043   END FUNCTION INITIALSETTINGS
#044
#045   subroutine Dialoginput(checked)
#046     USE IFWIN
#047     USE IFLOGM
#048     USE CONTROLS
#049     implicit none
#050     integer( kind=4 )::results
#051     logical( kind=4 )::retlog,checked
#052     type( dialog )::datadlg
#053     external:: BARCALLBACK
#054
#055     retlog = DLGINIT(IDD_DIALOG1, datadlg)
#056     ! 初始化滚动条
#057     retlog = DLGSET (datadlg, IDC_SCROLLBAR1, 1, DLG_RANGEMIN)
#058     retlog = DLGSET (datadlg, IDC_SCROLLBAR1, 200, DLG_RANGEMAX)
#059     ! 初始化进度条
#060     retlog = DLGSET (datadlg, IDC_PROGRESS1, 1, DLG_RANGEMIN)
#061     retlog = DLGSET (datadlg, IDC_PROGRESS1, 200, DLG_RANGEMAX)
#062     ! 初始化滑动条
#063     retlog = DLGSET (datadlg, IDC_SLIDER1, 0, DLG_RANGEMIN)
#064     retlog = DLGSET (datadlg, IDC_SLIDER1, 200, DLG_RANGEMAX)
#065     ! 回调函数
#066     retlog = DlgSetSub(datadlg, IDC_SCROLLBAR1, BARCALLBACK)
#067     retlog = DlgSetSub(datadlg, IDC_SLIDER1,  BARCALLBACK)
#068     ! 显示对话框
#069     results = DLGMODAL (datadlg)
#070     call DLGUNINIT (datadlg)
#071   end subroutine Dialoginput
#072
#073   subroutine BARCALLBACK( dlg, id, callbacktype )
#074     !DEC$ ATTRIBUTES DEFAULT :: dialogbasedApply
#075     USE IFLOGM
#076     USE CONTROLS
#077     implicit none
#078     type(dialog) dlg
#079     logical(kind=4)::results
#080     integer(kind=4):: id, callbacktype ,bar_position
#081
#082     select case (id)
#083     case (IDC_SCROLLBAR1)
#084       results = DLGGET (dlg, IDC_SCROLLBAR1, bar_position, DLG_POSITION)
#085       results = DLGSET (dlg, IDC_SLIDER1, bar_position, DLG_POSITION)
#086       results = DLGSET (dlg, IDC_PROGRESS1, bar_position, DLG_POSITION)
#087     case (IDC_SLIDER1)
#088       results = DLGGET (dlg, IDC_SLIDER1, bar_position, DLG_POSITION)
#089       results = DLGSET (dlg, IDC_SCROLLBAR1, bar_position, DLG_POSITION)
#090       results = DLGSET (dlg, IDC_PROGRESS1, bar_position, DLG_POSITION)
#091     end select
#092   end subroutine BARCALLBACK
```

运行程序，对话框中控件的工作状态如图 5.19 所示。显然，对话框编辑时和运行时的外观是不同的。

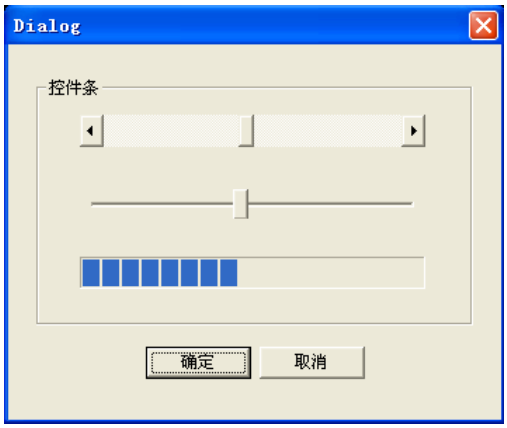


图 5.19 Example 5-5 运行结果

■ 复选按钮 可以多选，其操作方法和单选按钮相同，这里不再赘述。

■ 列表框 适合多选一或者多选多。首先在程序中初始化列表框，也就是向列表框添加一些项目：

```
USE IFLOGM
logical( kind=4 )::retlog
type( DIALOG )::dlg
retlog = DlgSet ( dlg, IDC_LISTBOX1, 3, DLG_NUMIT EMS )  ！共添加3 个选项
retlog = DlgSet ( dlg, IDC_LISTBOX1, "Moe", 1 )  ！第一个
retlog = DlgSet ( dlg, IDC_LISTBOX1, "Larry", 2 )  ！第二个
retlog = DlgSet ( dlg, IDC_LISTBOX1, "Curly", 3 )  ！第三个
```

当然也可以在程序的运行中动态添加项目，方法为：

```
retlog = DLGSET ( dlg, IDC_LISTBOX1, 4)  ！一共4 项
retlog = DLGSET ( dlg, IDC_LISTBOX1, "Shemp", 4)  ！添加第4 项
```

为了在程序中判断用户选择的项目，首先要了解列表框对选择动作的响应方式，以上面的列表框为例，列表框中一共有三个项目，分别为“*Moe*”、“*Larry*”和“*Curly*”，假如用户共选了两项，第一次选中了“*Moe*”，第二次选中了“*Curly*”。此时列表框会自动记录如表 5.1 所示的选择情况：

表 5.1

用户选择次数	所选项目的索引号
1	1
2	3
3	0

其中的前两项容易理解。最后一项是列表框自动加上的记录，意味着选择的结束。了解列表框的响应后，借助程序代码，就能获取所有被选中的项目的索引号，方法为：

```
integer(kind=4):: j, num, test
integer(kind=4), allocatable :: values(:)
logical(kind=4):: retlog
！获取列表框中的项目个数
```

```
retlog = DLGGET (dlg, IDC_LISTBOX1, num, DLG_NUMITEMS)
allocate (values(num))
j = 1
test = -1
do while (test .NE. 0)
   retlog = DLGGET (dlg, IDC_LISTBOX1, values(j), j)   !取出第j 次选中项目的索引号
   test = values(j)
   j = j + 1
end do
```

通过以上步骤，我们获取了项目索引号，但是最终希望得到的是项目本身。所以在此基础上，用函数 DLGGET 进行第二次提取：

```
retlog = DLGGET (dlg, IDC_LISTBOX1, str, values(j))
```

下面是列表框和复选框的例子。程序的目的是按照用户的选择输出即可。首先，创建如图 5.20 所示对话框及其控件。如果需要可多选的列表框，应将列表框属性“Selection”的值改为“Multiple”。



图 5.20 对话框界面

然后将下面的程序添加到工程，编译运行。

```
#001   ! Example 5-6
#002   module CONTROLS
#003     integer, parameter :: IDD_DIALOG1  =  101
#004     integer, parameter :: IDC_LIST1    =  1000
#005     integer, parameter :: IDC_CHECK1   =  1001
#006     integer, parameter :: IDC_CHECK2   =  1002
#007     integer, parameter :: IDC_CHECK3   =  1003
#008   end module CONTROLS
#009
#010   program main
#011     USE IFQWIN
#012     implicit none
#013     integer( kind=4 ) :: results
#014     logical(kind=4)::res
#015     type (qwinfo) :: winfo
#016     type (windowconfig) :: wc
```

```
#017
#018    winfo%type = QWIN$MAX
#019    res = GETWINDOWCONFIG(wc)
#020    ! 最大化子窗口
#021    results = SETWSIZEQQ (0, winfo)
#022    results = SETEXITQQ (QWIN$EXITPERSIST)
#023    do while (.true.)
#024    enddo
#025  end program main
#026
#027  LOGICAL(4) FUNCTION INITIALSETTINGS( )
#028    USE IFQWIN
#029    implicit none
#030    integer( kind=4 ) :: results
#031    logical(kind=4)::res
#032    type (qwinfo) :: winfo
#033    external:: Dialoginput
#034
#035    winfo%w = 800
#036    winfo%h = 600
#037    winfo%type = QWIN$SET
#038    results = SetWSizeQQ( QWIN$FRAMEWINDOW, winfo )
#039
#040    res = appendmenuqq(1, $MENUENABLED, '文件'C, NUL)
#041    res = appendmenuqq(1, $MENUENABLED, '列表框...'C,Dialoginput)
#042    INITIALSETTINGS= .true.
#043    return
#044  END FUNCTION INITIALSETTINGS
#045
#046  subroutine Dialoginput(checked)
#047    USE IFWIN
#048    USE IFLOGM
#049    USE CONTROLS
#050    implicit none
#051    integer( kind=4 )::results
#052    logical( kind=4 )::retlog,checked,checkboxinput
#053    type( DIALOG )::datadlg
#054    integer(kind=4)::i, j, num, test
#055    integer(kind=4), allocatable :: values(:)
#056    character(len=10)::str
#057    ! 初始化对话框
#058    retlog = DLGINIT(IDD_DIALOG1, datadlg)
#059    ! 共添加3个选项
#060    retlog = DlgSet ( datadlg, IDC_LIST1, 3, DLG_NUMITEMS )
#061    retlog = DlgSet ( datadlg, IDC_LIST1, "菠菜", 1 )  ! 第一个
#062    retlog = DlgSet ( datadlg, IDC_LIST1, "油菜", 2 )  ! 第二个
#063    retlog = DlgSet ( datadlg, IDC_LIST1, "空心菜", 3 )  ! 第三个
#064    ! 显示对话框
#065    results = DLGMODAL (datadlg)
#066
#067    ! 如果用户按下“确定”按钮，则将用户的选择结果输出
#068    if(results == IDOK)then
#069      ! 获取列表框中的项目个数
#070      retlog = DLGGET (datadlg, IDC_LIST1, num, DLG_NUMITEMS)
#071      allocate (values(num))
#072      j = 1
#073      test = -1
#074      do while (test .NE. 0)
```

```
#075         !取出第j次选中项目的索引号
#076         retlog = DLGGET (datadlg, IDC_LIST1, values(j), j)
#077         test = values(j)
#078         j = j + 1
#079       end do
#080
#081       ! 输出列表框中的选择项
#082       i = 1
#083       do while (i .LE. j-2)
#084         retlog = DLGGET (datadlg, IDC_LIST1, str, values(i))
#085         i = i + 1
#086         write(0,*)str
#087       end do
#088
#089       ! 处理复选框
#090       retlog = DLGGET (datadlg, IDC_CHECK1, checkboxinput)
#091       if (checkboxinput == .true.) write(0,*)"香蕉"
#092       retlog = DLGGET (datadlg, IDC_CHECK2, checkboxinput)
#093       if (checkboxinput == .true.) write(0,*)"石榴"
#094       retlog = DLGGET (datadlg, IDC_CHECK3, checkboxinput)
#095       if (checkboxinput == .true.) write(0,*)"菠萝"
#096     endif
#097     call DLGUNINIT (datadlg)
#098   end subroutine Dialoginput
```

运行结果如图 5.21 所示。



图 5.21 Example 5-6 程序运行界面

■ 滚动控件 和以前的控件有所不同，滚动控件一般和文本编辑框或者静态文本框配合使用，文本编辑框或者静态文本框时滚动控件的伙伴窗口。也就是说，程序运行时，这两种控件完全合二为一，共同协同工作。下面举例详细说明文本编辑框和滚动控件是如何协同工作的。

首先，按照图 5.22 所示的情况绘制控件；

其次，设置滚动控件的属性值。其中包括：Alignment = Right Align；Auto Buddy = True；Set Buddy Integer = True。

然后，单击对话框测试按钮，对话框的外观如图 5.23 所示。显然，两种控件已经完全达到了统一。



图 5.22 对话框界面

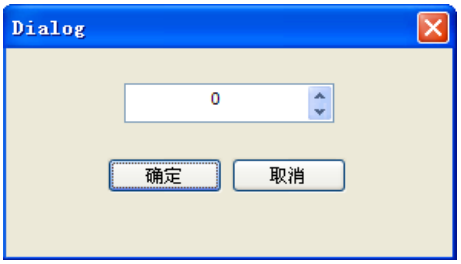


图 5.23 对话框测试界面

最后，编写控件驱动程序。对滚动控件来说，一般有三个状态参数需要设置：最大范围、最小范围和当前值，下面是设置方法：

USE IFLOGM

type (dialog) :: dlg

logical(kind=4):: retlog

retlog = DLGSET (dlg, IDC_SPIN1, 0, DLG_RANGEMIN) ！最小范围

retlog = DLGSET (dlg, IDC_ SPIN1, 200, DLG_RANGEMAX) ！最大范围

retlog = DLGSET (dlg, IDC_ SPIN1, 10, DLG_POSITION) ！设定位置

为了获取当前位置，则：

retlog = DLGGET (datadlg, IDC_EDIT1, str)

最后形成完整的对话框驱动程序：

```
#001   ! Example 5-7
#002   module CONTROLS
#003     integer, parameter :: IDD_DIALOG1  =  101
#004     integer, parameter :: IDC_EDIT1    =  1000
#005     integer, parameter :: IDC_SPIN1    =  1001
#006   end module CONTROLS
#007
#008   program main
#009     USE IFQWIN
#010     implicit none
#011     integer( kind=4 ) :: results
#012     logical(kind=4)::res
#013     type (qwinfo) :: winfo
#014     type (windowconfig) :: wc
#015
#016     winfo%type = QWIN$MAX
#017     res = GETWINDOWCONFIG(wc)
#018     ! 最大化子窗口
#019     results = SETWSIZEQQ (0, winfo)
#020     ! 取消提示信息
```

```
#021    results = SETEXITQQ (QWIN$EXITPERSIST)
#022    do while (.true.)
#023    enddo
#024  end program main
#025
#026  LOGICAL(4) FUNCTION INITIALSETTINGS( )
#027    USE IFQWIN
#028    implicit none
#029    integer( kind=4 ) :: results
#030    logical(kind=4)::res
#031    type (qwinfo) :: winfo
#032    external:: Dialoginput
#033
#034    winfo%w = 800
#035    winfo%h = 600
#036    winfo%type = QWIN$SET
#037    results = SetWSizeQQ( QWIN$FRAMEWINDOW, winfo )
#038    res = appendmenuqq(1, $MENUENABLED, '文件'C, NUL)
#039    res = appendmenuqq(1, $MENUENABLED, '混合控件...'C,Dialoginput)
#040    INITIALSETTINGS= .true.
#041    return
#042  END FUNCTION INITIALSETTINGS
#043
#044  subroutine Dialoginput(checked)
#045    USE IFWIN
#046    USE IFLOGM
#047    USE CONTROLS
#048    implicit none
#049    integer( kind=4 )::results
#050    logical( kind=4 )::retlog,checked
#051    type( dialog )::datadlg
#052    character(len=4)::str
#053
#054    retlog = DLGINIT(IDD_DIALOG1, datadlg)
#055    retlog = DLGSET (datadlg, IDC_SPIN1, 200, DLG_RANGEMAX)
#056    retlog = DLGSET (datadlg, IDC_SPIN1, 0, DLG_RANGEMIN)
#057    retlog = DLGSET (datadlg, IDC_SPIN1, 10, DLG_POSITION)
#058    results = DLGMODAL (datadlg)
#059    if(results == IDOK)then
#060      retlog = DLGGET (datadlg, IDC_EDIT1, str)
#061      write(0,*)str
#062    end if
#063    call DLGUNINIT (datadlg)
#064  end subroutine Dialoginput
```

程序运行结果如图 5.24 所示。



图 5.24 Example 5-7 程序运行界面

■ 图片控件 通常作为图标、图片等的载体，仅是输出窗口。为了说明图片控件的用法，下面给出一个例子。

首先，用绘图软件绘制如图 5.25 的图形并保存为.bmp 格式；

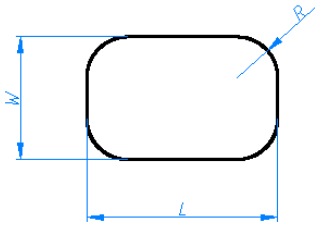


图 5.25 长方形及其尺寸标注

其次，添加如图 5.26 所示的对话框，将图片控件的属性“Type”改为“Bitmap”；

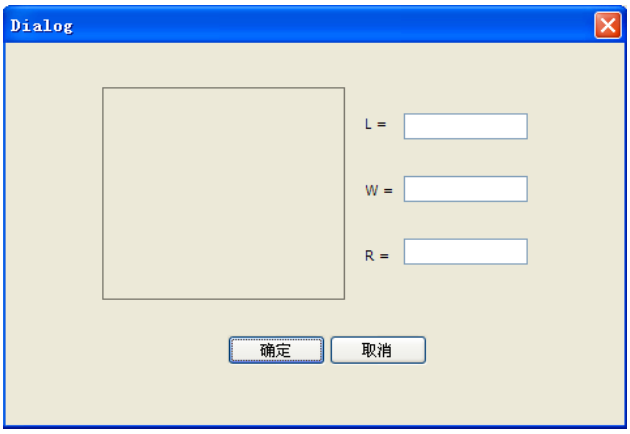


图 5.26 对话框及其控件

然后，和添加对话框一样添加“Bitmap”资源，同时选择“导入…”，将第一步所得的图片导入资源编辑器；

接着，在图片控件的属性“Image”中选择上一步导入的图片，此时在图片控件中出现了该图片。调整对话框中控件之间的相对位置，最终对话框如图 5.27 所示。

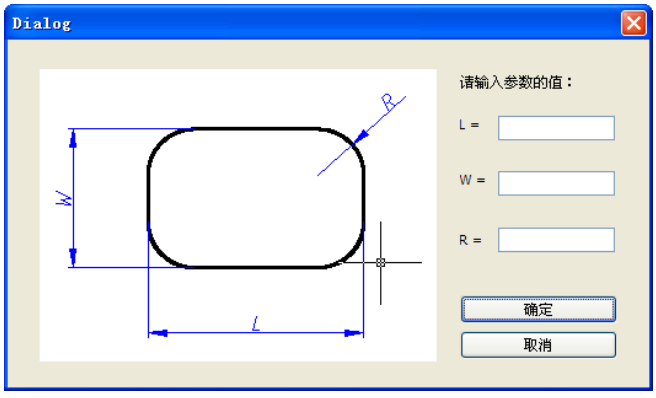


图 5.27 调整后的对话框及其控件

最后，将下面的代码加入工程，编译运行。

```
#001   ! Example 5-8
#002   module CONTROLS
#003     integer, parameter :: IDD_DIALOG1  =  101
#004     integer, parameter :: IDB_BITMAP1  =  102
#005     integer, parameter :: IDC_EDIT1    =  1000
#006     integer, parameter :: IDC_EDIT2    =  1001
```

```
#007     integer, parameter :: IDC_EDIT3   =  1002
#008   end module CONTROLS
#009
#010   program main
#011     USE IFQWIN
#012     implicit none
#013     integer( kind=4 ) :: results
#014     logical(kind=4)::res
#015     type (qwinfo) :: winfo
#016     type (windowconfig) :: wc
#017
#018     winfo%type = QWIN$MAX
#019     res = GETWINDOWCONFIG(wc)
#020     ! 最大化子窗口
#021     results = SETWSIZEQQ (0, winfo)
#022     results = SETEXITQQ (QWIN$EXITPERSIST)
#023     do while (.true.)
#024     enddo
#025   end program main
#026
#027   LOGICAL(4) FUNCTION INITIALSETTINGS( )
#028     USE IFQWIN
#029     implicit none
#030     integer( kind=4 ) :: results
#031     logical(kind=4)::res
#032     type (qwinfo) :: winfo
#033     external:: Dialoginput
#034
#035     winfo%w = 800
#036     winfo%h = 600
#037     winfo%type = QWIN$SET
#038     results = SetWSizeQQ( QWIN$FRAMEWINDOW, winfo )
#039
#040     res = appendmenuqq(1, $MENUENABLED, '文件'C, NUL)
#041     res = appendmenuqq(1, $MENUENABLED, '图片控件...'C,Dialoginput)
#042     INITIALSETTINGS= .true.
#043     return
#044   END FUNCTION INITIALSETTINGS
#045
#046   subroutine Dialoginput(checked)
#047     USE IFWIN
#048     USE IFLOGM
#049     USE CONTROLS
#050     implicit none
#051     integer( kind=4 )::results
#052     logical( kind=4 )::retlog,checked,checkboxinput
#053     type( dialog )::datadlg
#054     character(len=10)::str1,str2,str3
#055
#056     retlog = DLGINIT(IDD_DIALOG1, datadlg)
#057     results = DLGMODAL (datadlg)
#058     if(results == IDOK)then
#059       retlog = DLGGET (datadlg, IDC_EDIT1, str1)
#060       write(0,*)str1
#061       retlog = DLGGET (datadlg, IDC_EDIT2, str2)
#062       write(0,*)str2
#063       retlog = DLGGET (datadlg, IDC_EDIT3, str3)
#064       write(0,*)str3
```

```
#065     endif
#066     call DLGUNINIT (datadlg)
#067   end subroutine Dialoginput
```

■ 选项卡控件 这是 Quickwin 中功能最强大的控件，也是最复杂的控件。利用选项卡控件，能将其他控件有效组织和安排，方便程序设计。为了讲述选项卡控件的用法，我们从示例入手。

首先，打开资源编辑器，绘制如图 5.28 所示的对话框及选项卡控件。默认情况下，选项卡控件在外观上提供五个选项卡，而在这个例子中，我们在选项卡控件中设置三个选项卡。

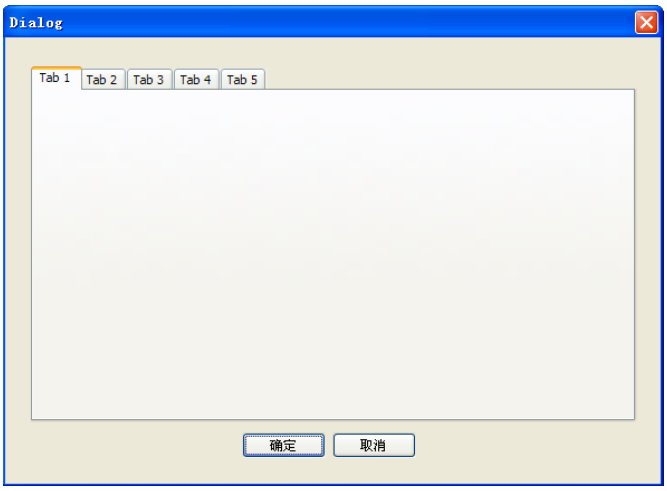


图 5.28 对话框及选项卡控件

到目前为止，每个选项卡都是空白的，而选项卡存在的理由是安排其他控件。为了达到这个目的，我们必须为每一个空白的选项卡建立相联系的对话框及控件。所以，利用编辑对话框资源的方法再添加三个独立的对话框，同时将每个对话框的"Style"属性值改为"Child"，将属性值"Border"改为"None"，属性值"Title Bar"改为"False"，外观如图 5.29 所示；



图 5.29 和选项卡对应的对话框

其次，编写对话框驱动程序。

```
#001 ! Example 5-9
#002 module CONTROLS
#003   USE IFLOGM
#004   USE IFWIN
#005   integer, parameter :: IDD_DIALOG1  =  101
#006   integer, parameter :: IDD_TAB_DIALOG1  =  102
#007   integer, parameter :: IDD_TAB_DIALOG2  =  103
#008   integer, parameter :: IDD_TAB_DIALOG3  =  104
#009   integer, parameter :: IDC_TAB1    =  1000
#010   integer, parameter :: IDC_EDIT1    =  1001
```

```
#011   integer, parameter :: IDC_EDIT2    =  1002
#012
#013   type( DIALOG )::datadlg
#014   type( DIALOG )::data_tabdlg1
#015   type( DIALOG )::data_tabdlg2
#016   type( DIALOG )::data_tabdlg3
#017 end module CONTROLS
#018
#019 program main
#020   USE IFQWIN
#021   implicit none
#022   integer( kind=4 ) :: results
#023   logical(kind=4)::res
#024   type (qwinfo) :: winfo
#025   type (windowconfig) :: wc
#026
#027   winfo%type = QWIN$MAX
#028   res = GETWINDOWCONFIG(wc)
#029   ! 最大化子窗口
#030   results = SETWSIZEQQ (0, winfo)
#031   ! 取消提示信息
#032   results = SETEXITQQ (QWIN$EXITPERSIST)
#033   do while (.true.)
#034   enddo
#035 end program main
#036
#037 LOGICAL(4) FUNCTION INITIALSETTINGS( )
#038   USE IFQWIN
#039   implicit none
#040   integer( kind=4 ) :: results
#041   logical(kind=4)::res
#042   type (qwinfo) :: winfo
#043   external:: Dialoginput
#044
#045   winfo%w = 800
#046   winfo%h = 600
#047   winfo%type = QWIN$SET
#048   results = SetWSizeQQ( QWIN$FRAMEWINDOW, winfo )
#049
#050   res = appendmenuqq(1, $MENUENABLED, '文件'C, NUL)
#051   res = appendmenuqq(1, $MENUENABLED, '选项卡...'C,Dialoginput)
#052   INITIALSETTINGS= .true.
#053   return
#054 END FUNCTION INITIALSETTINGS
#055
#056 subroutine Dialoginput(checked)
#057   USE IFWIN
#058   USE IFLOGM
#059   USE CONTROLS
#060   implicit none
#061   integer( kind=4 )::results
#062   logical( kind=4 )::retlog,checked
#063
#064   external::DLG_TABINIT
#065   external::DIALOG1_INIT,DIALOG2_INIT
#066   ! 初始化对话框
#067   retlog = DLGINIT(IDD_DIALOG1, datadlg)
#068   ! 初始化选项卡控件
```

```
#069   ! 共有三个选项卡
#070   retlog = DlgSet (datadlg, IDC_TAB1, 3, DLG_NUMITEMS )
#071   retlog = DlgSet (datadlg, IDC_TAB1, "姓名", 1) ! 第一个
#072   retlog = DlgSet (datadlg, IDC_TAB1, "班级", 2) ! 第二个
#073   retlog = DlgSet (datadlg, IDC_TAB1, "学号", 3) ! 第三个
#074   ! 将选项卡和对话框对应
#075   retlog = DlgSet (datadlg, IDC_TAB1, IDD_TAB_DIALOG1, 1)
#076   retlog = DlgSet (datadlg, IDC_TAB1, IDD_TAB_DIALOG2, 2)
#077   retlog = DlgSet (datadlg, IDC_TAB1, IDD_TAB_DIALOG3, 3)
#078
#079   ! 初始化对话框中和tab联系的对话框且设置回调子程序
#080   ! 初始化和第一个选项卡对应的对话框
#081   retlog = DLGINIT(IDD_TAB_DIALOG1, data_tabdlg1)
#082   ! 设置第一个对话框的回调子程序
#083   retlog=DLGSETSUB (data_tabdlg1,IDD_TAB_DIALOG1,DIALOG1_INIT)
#084   ! 初始化和第二个选项卡对应的对话框
#085   retlog = DLGINIT(IDD_TAB_DIALOG2, data_tabdlg2)
#086   ! 设置第二个对话框的回调子程序
#087   retlog=DLGSETSUB (data_tabdlg2,IDD_TAB_DIALOG2,DIALOG2_INIT)
#088   ! 初始化和第三个选项卡对应的对话框
#089   retlog = DLGINIT(IDD_TAB_DIALOG3, data_tabdlg3)
#090   ! 对话框回调子程序，用来初始化TAB控件
#091   retlog = DLGSETSUB (datadlg,IDD_DIALOG1,DLG_TABINIT )
#092   ! 显示对话框
#093   results = DLGMODAL (datadlg)
#094   ! 释放对话框占用的资源
#095   call DLGUNINIT (datadlg)
#096   call DLGUNINIT (data_tabdlg1)
#097   call DLGUNINIT (data_tabdlg2)
#098   call DLGUNINIT (data_tabdlg3)
#099 end subroutine Dialoginput
#100
#101 ! 主对话框回调子程序
#102 subroutine DLG_TABINIT ( dlg, id, callbacktype )
#103   !DEC$ ATTRIBUTES DEFAULT :: DLG_TABINIT
#104   USE IFWIN
#105   USE IFLOGM
#106   USE CONTROLS
#107   implicit none
#108   type(dialog) dlg
#109   logical(kind=4)::retlog
#110   integer(kind=4):: id, callbacktype ,hwnd
#111
#112   if (callbacktype == dlg_init) then
#113     hwnd = GetDlgItem(datadlg%hwnd, IDC_TAB1)
#114     retlog = DlgModeless(data_tabdlg1, SW_HIDE, hwnd)
#115     retlog = DlgModeless(data_tabdlg2, SW_HIDE, hwnd)
#116     retlog = DlgModeless(data_tabdlg3, SW_HIDE, hwnd)
#117     retlog = DlgSet(datadlg, IDC_TAB1, 1, dlg_state)
#118   else
#119     ! 对话框退出时执行的代码
#120   endif
#121 end subroutine DLG_TABINIT
#122
#123 subroutine DIALOG1_INIT ( dlg, id, callbacktype )
#124   !DEC$ ATTRIBUTES DEFAULT :: DIALOG1_INIT
#125   USE IFWIN
#126   USE IFLOGM
```

```
#127   USE CONTROLS
#128   implicit none
#129   type(dialog) dlg
#130   logical(kind=4)::retlog
#131   integer(kind=4):: id, callbacktype
#132   character(len=8)::str
#133
#134   if (callbacktype == dlg_init) then
#135     ! 对话框显示时执行
#136   else
#137     ! 对话框退出时执行
#138     retlog = DlgGet (dlg, IDC_EDIT1, str)
#139     write(0,*)str
#140   endif
#141 end subroutine DIALOG1_INIT
#142
#143 subroutine DIALOG2_INIT ( dlg, id, callbacktype )
#144   !DEC$ ATTRIBUTES DEFAULT :: DIALOG2_INIT
#145   Use IFWIN
#146   USE IFLOGM
#147   USE CONTROLS
#148   implicit none
#149   type(dialog) dlg
#150   logical(kind=4)::retlog
#151   integer(kind=4):: id, callbacktype
#152   character(len=8)::str
#153
#154   if (callbacktype == dlg_init) then
#155     ! 对话框显示时执行
#156   else
#157     ! 对话框退出时执行
#158     retlog = DlgGet (dlg, IDC_EDIT2, str)
#159     write(0,*)str
#160   endif
#161 end subroutine DIALOG2_INIT
```

运行程序，结果如图 5.30 所示。

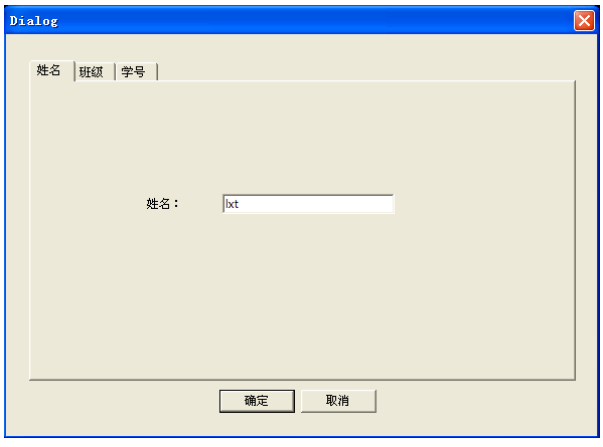


图 5.30 Example 5-9 程序运行结果

需要特别强调的是：程序运行中，对话框的回调函数会自动执行两次，第一次是对话框显示时，第二次为对话框退出的时候。所以在上面的例子中对话框的回调函数中做出了相关的判断。

5-3 创建工具栏

在这所有的控件中，最有用的莫过于工具栏了。工具栏一般位于主窗口的顶部，将一些常用的功能和工具栏按钮联系起来，用户使用快捷方便。

相比其他控件，工具栏的创建较为复杂。下面结合实例详细说明工具栏的创建过程。

■ 首先创建工具栏图标。打开资源管理器，在“添加资源”对话框的资源类型列表中选择“ToolBar”，如图 5.31 所示。

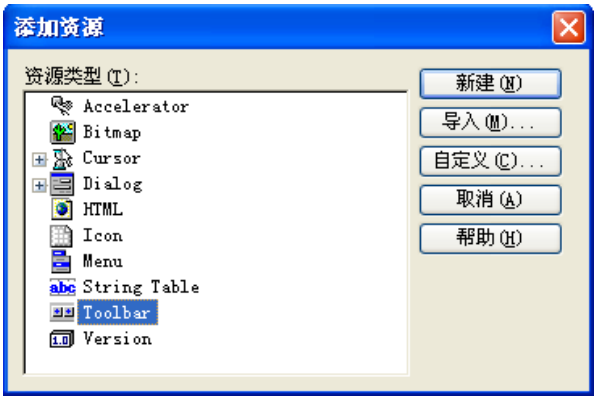


图 5.31 资源类型对话框

单击“新建”按钮，弹出如图 5.32 所示的工具栏按钮的创建界面。

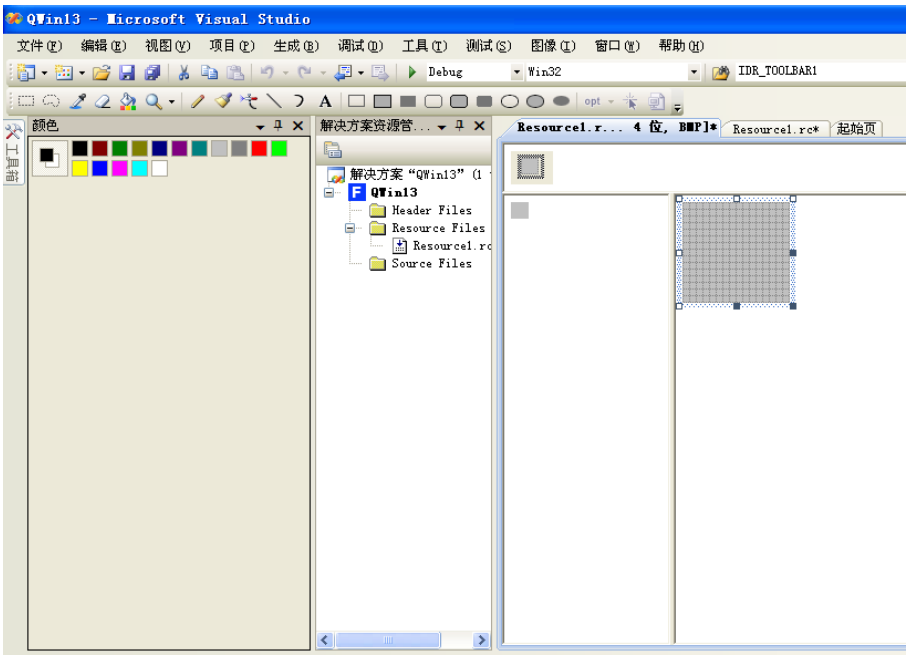


图 5.32 工具栏图标创建界面

为了创建工具栏按钮，利用“图像编辑器”工具栏和下方的“颜色”对话框，在图 5.32 最右边的正方形区域内绘制需要的按钮图标。每个正方形代表着一个工具栏按钮，如果需要创建多个工具栏按钮，就必须绘制相应个数的图标，而当所要工具栏的按钮尺寸和默认的尺寸不同时，可用如图 5.33 所示的工具栏编辑器改变按钮大小，然后绘图。



图 5.33 工具栏编辑器

经过逐一绘制后，最终的工具栏图标如图 5. 34 所示。考虑到工具栏的一个重要特征—信息提示，将每一个新绘制的按钮的属性“Prompt”修改为该工具栏按钮的信息提示文本，如第一个为“新建文件”，第二个为“保存文件”，第三个为“退出程序”。

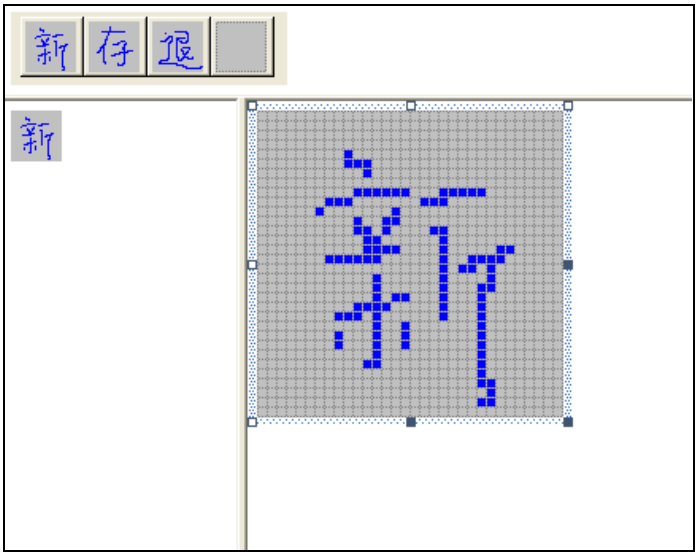


图 5.34 工具栏图标

单击“保存”按钮，完成工具栏图标的绘制和工具栏资源的保存。

■ 为了便于 Fortran 程序使用资源文件，将前面生成的资源文件稍作修改，加入 Fortran 程序。具体做法如下：

打开 resource 文件，将第一部分的代码修改为 Fortran 可识别的形式：

```
integer*4,parameter:: IDR_TOOLBAR1 = 101 ! 工具栏
integer*4,parameter:: ID_BUTTON40001 = 40001 ! 第一个按钮
integer*4,parameter:: ID_BUTTON40002 = 40002 ! 第二个按钮
integer*4,parameter:: ID_BUTTON40003 = 40003 ! 第三个按钮
integer*4,parameter:: WM_CREATETOOLBAR = 500 ! 自定义消息
```

■ 创建工具栏及其驱动程序。为了便于说明，先给出完成的程序代码。

```
#001 ! Example 5-10
#002 module CONTROLS
#003   USE IFWIN
#004   USE IFQWIN
```

```
#005   USE IFLOGM
#006   USE COMCTL32
#007
#008   integer*4,parameter :: IDR_TOOLBAR1   =  101   ! 工具栏
#009   integer*4,parameter :: ID_BUTTON40001   =  40001 ! 第一个按钮
#010   integer*4,parameter :: ID_BUTTON40002   =  40002 ! 第二个按钮
#011   integer*4,parameter :: ID_BUTTON40003   =  40003 ! 第三个按钮
#012   integer*4,parameter ::  WM_CREATETOOLBAR  = 500  ! 自定义消息
#013
#014   integer( kind=4 ) :: hFrame ! 主框架句柄
#015   integer( kind=4 ) :: hInst ! 实例句柄
#016   integer( kind=4 ) :: hMDI ! 子窗口句柄
#017   integer( kind=4 ) :: hToolbar ! 工具栏句柄
#018   integer( kind=4 ) :: hStatus  ! 状态栏句柄
#019   integer( kind=4 ) :: FrameProc  ! 主窗口函数的地址
#020
#021   type( T_MSG ) :: mesg   ! 定义消息的结构体类数据
#022   ! 定义按纽所需的结构体数组
#023   type( T_TBBUTTON ), DIMENSION(*) :: tbrButtons( 3 )
#024 end module CONTROLS
#025
#026 LOGICAL(4) FUNCTION INITIALSETTINGS( )
#027   USE IFQWIN
#028   implicit none
#029   logical(kind=4) :: res
#030   type (qwinfo) :: winfo
#031
#032   res = appendmenuqq(1, $MENUENABLED, '文件'C, NUL)
#033   res = appendmenuqq(1, $MENUENABLED, '对话框...'C,NUL)
#034   INITIALSETTINGS= .true.
#035   return
#036 END FUNCTION INITIALSETTINGS
#037
#038 ! 主程序
#039 program main
#040   USE CONTROLS
#041   implicit None
#042   integer( kind=4 ) :: results
#043   type (qwinfo) :: winfo
#044
#045   INTERFACE
#046     INTEGER(4) FUNCTION FrameWndProc(hWnd,Msg,wParam,lParam)
#047     !DEC$ATTRIBUTES STDCALL::  FrameWndProc
#048     Integer( kind=4 ) :: hWnd,Msg,wParam,lParam
#049     END FUNCTION
#050   END INTERFACE
#051
#052   winfo%type = QWIN$MAX
#053   ! 最大化主窗口
#054   results = SETWSIZEQQ(QWIN$FRAMEWINDOW, winfo)
#055   ! 最大化子窗口
#056   results = SETWSIZEQQ (0, winfo)
#057   results = SETEXITQQ(QWIN$EXITPERSIST)
#058
#059   hFrame = GETHWNDQQ(QWIN$FRAMEWINDOW)  ! 获取主窗口句柄
#060   hInst = GetWindowLong(hFrame,GWL_HINSTANCE)  ! 获取实例句柄
#061   hMDI = GetWindow(hFrame, GW_CHILD)  ! 获取子窗口句柄
#062   ! 设置主窗口的回调函数，非常重要
```

```
#063   FrameProc = SetWindowLong(hFrame,GWL_WNDPROC,LOC(FrameWndProc))
#064   ! 发送创建工具栏的消息
#065   results = SendMessage(hFrame,WM_CREATETOOLBAR,0,0)
#066
#067   ! 激活子窗口，可不要
#068   results = SETACTIVEQQ(0)
#069   ! 更改窗口背景色
#070   results = SETBKCOLOR(7)
#071   call CLEARSCREEN($GCLEARSCREEN)
#072   ! 循环语句，其实为消息循环
#073   do while (.true.)
#074   end do
#075 end program main
#076
#077 ! 创建工具栏的子程序
#078 subroutine CreateMyToolbar()
#079   USE CONTROLS
#080   IMPLICIT NONE
#081   integer( kind=4 ) :: i,j,istyle
#082
#083   !工具栏数组赋值
#084   j = ID_BUTTON40001
#085   do  i = 1 ,3
#086     tbrButtons(i)%iBitmap   = i-1
#087     tbrButtons(i)%idCommand = j
#088     tbrButtons(i)%fsState   = TBSTATE_ENABLED
#089     tbrButtons(i)%fsStyle   = TBSTYLE_BUTTON
#090     tbrButtons(i)%dwData    = 0
#091     tbrButtons(i)%iString   = 0
#092     j =j+1
#093   enddo
#094   ! 创建工具栏
#095   hToolbar=CreateToolbarEx(hFrame,   & ! 工具栏的父窗口
#096     ! 工具栏的风格
#097     WS_CHILD.OR.WS_VISIBLE.OR.WS_CLIPSIBLINGS.OR.TBSTYLE_TOOLTIPS, &
#098       IDR_TOOLBAR1,          & ! 图片ID
#099       3,                    & ! 图片数量
#100       hInst,                & ! 实例句柄
#101       IDR_TOOLBAR1,          & ! 工具栏ID
#102       tbrButtons,           & ! 工具栏按钮数组
#103       3,                    & ! 按钮数量，包括分隔符
#104       32,                   & ! 按钮宽度
#105       32,                   & ! 按钮高度
#106       0,                    & ! 图片宽度
#107       0,                    & ! 图片高度
#108       SizeOf(tbrButtons(1)))    ! T_TBBUTTON结构体的大小
#109
#110   ! 注意：下面的语句将原来分隔的工具条按钮给合并了，只有当鼠标移到
#111   ! 相应的按钮上后，按钮突出显示
#112   istyle = GetWindowLong( hToolbar, GWL_STYLE ) !得到原工具条的风格
#113   ! 在原风格的基础上加入TBSTYLE_FLAT风格
#114   istyle = ior( istyle,TBSTYLE_FLAT )
#115   ! 改变工具条的风格
#116   i = SetWindowLong( hToolbar, GWL_STYLE, istyle )
#117 end subroutine CreateMyToolbar
#118
#119 ! 主窗口回调函数
#120 INTEGER FUNCTION FrameWndProc(hWnd,Msg,wParam,lParam)
```

```
!DEC$ATTRIBUTES STDCALL:: FrameWndProc
USE CONTROLS

integer( kind=4 ) :: hWnd,Msg,wParam,lParam
integer( kind=4 ) :: results
integer( kind=4 ) :: itbHeight ! 工具栏的高度
integer( kind=4 ) :: StatusHeight ! 状态栏的高度
type(T_RECT) :: tbRect ! 工具栏所在的矩形坐标
type(T_RECT) :: mdiRect  ! 窗口客户区所在的矩形坐标，包括了状态栏
type(T_RECT) :: StatusRect  ! 状态栏所在的矩形坐标

type(T_NMTTDISPINFO)::lptooltiptext  ! 工具栏信息提示相关结构体
pointer(ptext, lptooltiptext) ! 不是普通的指针变量

integer,parameter::MAXLEN=20
character(MAXLEN) szTipText ! 存储工具栏文本提示信息

select case(Msg)
  case (WM_CREATETOOLBAR)
    call CreateMyToolbar
    FrameWndProc = 0
  case (WM_COMMAND)
    select case(wParam)
    case(ID_BUTTON40001)
      results=MessageBox(hFrame,"Button 1"C,"Toolbar"C,MB_OK)
    case(ID_BUTTON40002)
      results=MessageBox(hFrame,"Button 2"C,"Toolbar"C,MB_OK)
    case(ID_BUTTON40003)
      results=MessageBox(hFrame,"Button 3"C,"Toolbar"C,MB_OK)
    end select
    FrameWndProc=CallWindowProc(FrameProc,hWnd,Msg,wParam,lParam)
  case (WM_NOTIFY)
    ptext = lparam
      if(lpToolTipText%hdr%code == TTN_GETDISPINFOA ) then
        ptext = lparam
        iret  = LoadString(hInst, lpToolTipText%hdr%idfrom, &
                           szTipText, MAXLEN)
        lpToolTipText%lpszText = loc(szTipText)
      end if
    FrameWndProc = 0
  case (WM_SIZE)
    ! 获取子窗口的句柄
    results=FindWindowEx(hFrame,NULL,LOC('MDIClient'C),NULL)
    ! 获取子窗口中工具栏的句柄
    hStatus=GetWindow(results,GW_HWNDNEXT)
    ! 获取范围大小
    results = GetClientRect(hWnd, mdiRect) !
    results = GetWindowRect(hToolbar, tbRect)
    results = GetWindowRect(hStatus, StatusRect)
    ! 计算工具栏和状态栏的高度
    itbHeight = tbRect%Bottom-tbRect%Top
    StatusHeight = StatusRect%Bottom-StatusRect%Top
    ! 默认情况下，QuickWin用子窗口覆盖了工具栏。所以为了使
    ! 工具栏可见，必须将子窗口移位
    results = MoveWindow(hMDI, 0, itbHeight,  mdiRect%Right,  &
                  mdiRect%Bottom-itbHeight-StatusHeight, .TRUE.)
    ! 有上面一句就能显示工具栏，但是当窗口大小改变时，其显示不正确
    ! 所以需要时刻修正状态栏的位置
```

```
#179     results = MoveWindow(hStatus, 0, mdiRect%Bottom-StatusHeight, &
#180                   mdiRect%Right, StatusHeight, .TRUE.)
#181     FrameWndProc = 0
#182   case default
#183     FrameWndProc=CallWindowProc(FrameProc,hWnd,Msg,wParam,lParam)
#184 end select
#185 end function FrameWndProc
```

默认情况下，Quickwin 应用程序中只有菜单和状态栏，为了在应用程序中加入工具栏，编写了工具栏创建程序 CreateMyToolbar。但是当程序运行后，我们发现在主窗口中并没有显示工具栏，这是因为默认情况下的 Quickwin 应用程序中子窗口完全覆盖了工具栏。为了正常显示工具栏，程序代码 175、176 行将子窗口向下移动了工具栏的高度值，运行程序后工具栏正常显示。但此时状态栏却消失了，根据前面的步骤，程序将子窗口向下移动了一定的距离，这恰好又覆盖了状态栏，为了显示状态栏，在程序移动子窗口的同时，必须将子窗口的高度减小，也就是将子窗口的大小变为原高度值减去状态栏的高度，程序代码为 179、180 行。

工具栏的位置调整工作都是在响应同一个系统消息 WM_SIZE 时完成。对 Windows 系统来说，窗口程序的运行过程就是事件驱动的过程。当程序的窗口大小有变化时，系统马上将 WM_SIZE 消息发送给程序，此时程序响应消息同时调整工具栏和状态栏位置。至于消息如何发送，这是系统和程序之间的事，读者可以暂时不用考虑。

程序的第 63 行代码非常重要。它设置了程序的主窗口回调函数为 FrameWndProc，也就是说当程序在运行的过程中随时都在调用该函数，如当窗口的大小发生改变时，操作系统通知程序响应 WM_SIZE 消息，以便调整窗口内控件之间的位置；或者当鼠标移动到工具栏上，系统通知程序响应 WM_NOTIFY 消息，从而弹出工具栏信息提示窗口；或者当用户选择菜单项和工具栏按钮时，系统通知程序响应 WM_COMMAND 消息，从而执行特定的任务；或者响应程序自身发送的消息 WM_CREATETOOLBAR，完成工具栏的创建。

子程序 CreateMyToolbar 中，为了创建工具栏，首先定义了 T_TBBUTTON 类结构体数组，用来设置工具栏按钮相关的属性。其结构组成如下：

```
typedef struct _TBBUTTON {
    int iBitmap;   //按钮图片的索引值
    int idCommand; //按钮 ID
    BYTE fsState; //按钮的状态
    BYTE fsStyle; //按钮的风格
    DWORD dwData; //程序自定义的值
    int iString; //按钮字符串索引
} TBBUTTON, NEAR* PTBBUTTON, FAR* LPTBBUTTON;
```

程序中 84~93 行程序针对每一个工具栏按钮的属性进行了定义。其次利用函数 CreateToolbarEx 创建了工具栏，函数语法为：

```
HWND CreateToolbarEx(
    HWND hwnd,          //工具栏父窗口句柄
    DWORD ws,           //工具栏的样式，必须指定 WS_CHILD
```

```
    UINT wID,               //工具栏 ID
    int nBitmaps,           //工具栏图片的数目
    HINSTANCE hBMInst,      //程序实例
    UINT wBMID,             //工具栏 ID
    LPCTBBUTTON lpButtons,      //工具栏结构数组
    int iNumButtons,        //工具栏按钮的数量
    int dxButton,           //工具栏按钮宽度
    int dyButton,           //工具栏按钮高度
    int dxBitmap,           //工具栏图片宽度
    int dyBitmap,           //工具栏图片高度
    UINT uStructSize        //T_TBBUTTON 类结构体的大小
);
```

为了改善工具栏按钮的外观，程序中添加了 110~116 行代码。当鼠标移动到工具栏按钮上方的时候，按钮自动响应鼠标事件，同时自动弹起。

该程序中，可能有许多地方和以前的有所不同。如果读者不能理解，请阅读 Windowing Application 程序设计相关部分后再去理解其中的原因。那时相信读者会有一个清晰的思路。

到此为止，读者已经掌握了 Quickwin 工程中常用的 15 种控件的用法。读者完全可以借助于这些强大的控件创造出完整的应用程序。

第六章 图形程序设计

Fortran 程序设计中，设计者经常涉到数值计算之后大量的数据处理工作，其中的处理方式之一是用图形展示数据。Fortran 中图形的显示方式有三种，其一是借助于现有的图形软件(如 Matfor、Dislin)和 Fortran 的接口进行图形的实时处理；其二是借助于微软的图形函数库 GDI 进行图形的绘制；其三是利用 Fortran 编译器封装的 Opengl 三维图形函数库进行绘图。其中第一种方式不需要程序设计者从底层编写绘图程序代码，所以大大减小了工作量，至于后两种，需要程序设计者在掌握相关函数库的基础上进行底层开发。本章将重点介绍后两种开发方法。

6-1 GDI 绘图

说到绘图，坐标系就显得非常重要。Fortran 的 Quickwin 工程支持三种坐标系统，分别为物理坐标、视窗坐标和窗口坐标。下面分别做一说明。

■ 物理坐标 指的是窗口客户区内以左上角为坐标原点，X 轴正方向向右，Y 轴正方向向下的坐标系统，物理坐标中，坐标以像素为单位，所有的坐标都是整数。从屏幕像素的角度讲，物理坐标就是显示器的当前分辨率，以像素为计量单位的坐标。

下面的例子说明了物理坐标的使用。按照前面的章节，笔者的电脑屏幕像素为 1366 x768，这个例子中为了说明物理像素的使用方法，在整个屏幕范围内绘制了一条线段，代码如下：

```
#001 ! Example 6-1
#002 program main
#003   USE IFQWIN
#004   implicit none
#005   integer( kind = 4 ) :: results, bkcolor,graphcolor
#006   type (qwinfo) :: winfo
#007   type (xycoord) :: xy
#008   integer( kind = 2 ) :: status
#009
#010   ! 背景色
#011   bkcolor = RGBTOINTEGER(125,125,125)
#012   ! 图形的颜色
#013   graphcolor = RGBTOINTEGER(0,255,0)
#014
#015   results = SETBKCOLORRGB (bkcolor)
#016   ! 真正改变背景颜色
#017   call CLEARSCREEN($GCLEARSCREEN)
#018
#019   winfo%type = QWIN$MAX
#020   ! 最大化主窗口
#021   results = SETWSIZEQQ(QWIN$FRAMEWINDOW, winfo)
#022   ! 最大化子窗口
#023   results = SETWSIZEQQ (0, winfo)
#024
#025   ! 设置图形的颜色
```

```
#026   results = SETCOLORRGB (graphcolor)
#027   ! 移动画笔到坐标(0,0)
#028   call MOVETO (0,0,xy)
#029   ! 绘制直线到坐标(1366,768)
#030   status = LINETO (1366,768)
#031   ! 取消提示信息
#032   results = SETEXITQQ (QWIN$EXITPERSIST)
#033 end program main
```

程序运行如图 6.1 所示。为了检验程序的正确性，请读者用鼠标移动窗口滚动条观察直线所在范围。

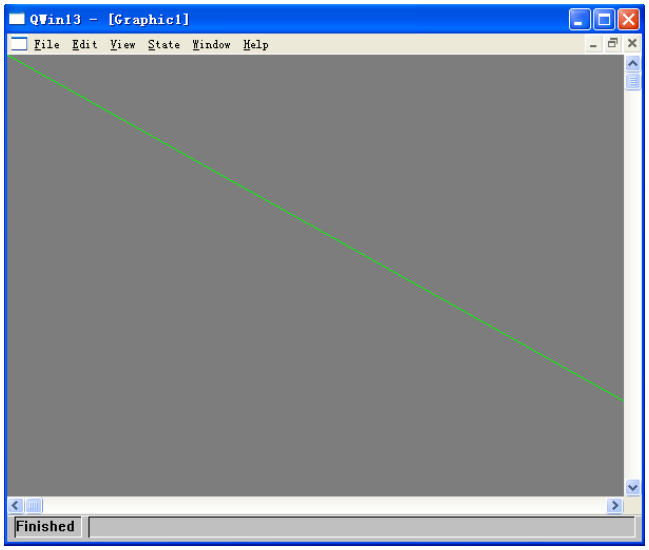


图 6.1 Example 6-1 运行结果