TraitsUI入门

黄天羽 嵩天
www.python123.org

TraitsUI的介绍

TraitsUI介绍

Python界面开发库

- Tkinter
- wxPython
- pyQt4
- TraitsUI
  - 以traits为基础
  - 以 MVC 为设计思想

TraitsUI介绍

- Modle-View-Controller
  - Model：程序中存储数据以及对数据进行处理
  - View ：程序的界面实现数据的可视化/显示
  - Controller：控制程序流程，M/V之间组织作用

http://docs.enthought.com/traitsui/

next index

TraitsUI Documentation

- TraitsUI 5.1 User Manual
  - TraitsUI 5.1 User Manual
  - Introduction
  - The View and Its Building Blocks
  - Customizing a View
  - Advanced View Concepts
  - Controlling the Interface: the Handler
  - TraitsUI Themes
  - Introduction to Trait Editor Factories
  - The Predefined Trait Editor Factories
  - Advanced Trait Editors
  - “Extra” Trait Editor Factories
  - Tips, Tricks and Gotchas
  - Appendix I: Glossary of Terms
  - Appendix II: Editor Factories for Predefined Traits
- TraitsUI 5.1 Tutorials
  - Writing a graphical application for scientific programming using TraitsUI 5.1
- TraitsUI 5.1 Demos
  - Standard Editors
  - Advanced Demos

Table Of Contents

- TraitsUI Documentation
- Indices and tables

Next topic

TraitsUI 5.1 User Manual

This Page

Show Source

Quick search

Go

TraitsUI安装

命令提示符：pip install traitsUI

命令提示符

```
C:\Users\huangtianyu>pip install traitsUI
Collecting traitsUI
  Using cached traitsui-5.1.0.zip
Requirement already satisfied: traits in c:\python36\lib\site-packages (from traitsUI)
Requirement already satisfied: pyface in c:\python36\lib\site-packages (from traitsUI)
Requirement already satisfied: pygments in c:\python36\lib\site-packages (from pyface->traitsUI)
Installing collected packages: traitsUI
  Running setup.py install for traitsUI ... done
Successfully installed traitsUI-5.1.0

C:\Users\huangtianyu>
```

TraitsUI安装小测

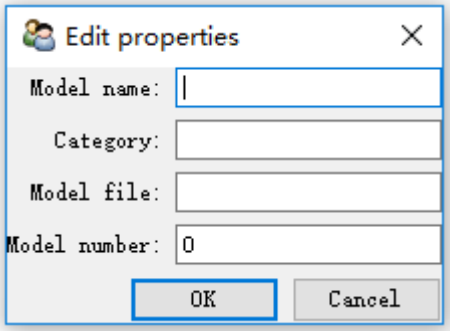
```
>>>from traitsui.api import View
```

TraitsUI的缺省界面程序框架

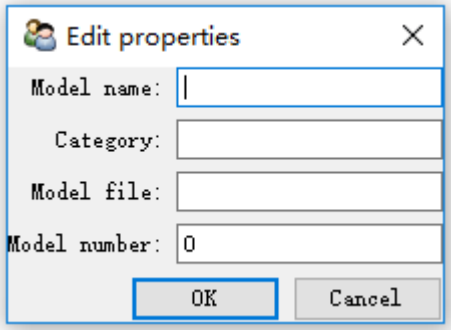
```
from traits.api import HasTraits
class TraitClass(HasTraits):
        … …
        … …
trt = TraitClass()
trt.configure_traits()
```

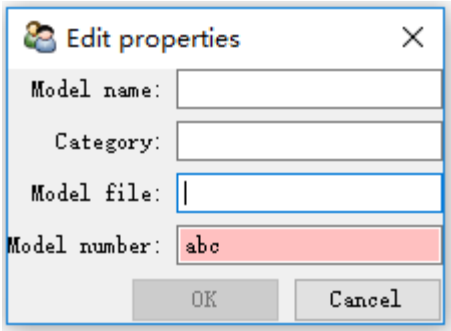
TraitsUI的小例子

```
from traits.api import HasTraits, Str, Int

class ModelManager(HasTraits):
    model_name = Str
    category = Str
    model_file = Str
    model_number = Int

model = ModelManager()
model.configure_traits()
```



TraitsUI的小例子





文字标签根据trait属性名自动生成：

- 第一个字母->大写
- 下划线->空格

View定义界面

TraitsUI自定义界面

traits.ui支持的后台界面库

后台界面库	程序启动时选择界面库参数
qt4	-toolkit qt4
Wx	–toolkit wx

用View定义界面

MVC类别	MVC说明
Model	HasTraits的派生类用Trait保存数据，相当于模型
View	没有指定界面显示方式时，Traits自动建立默认界面
Controller	起到视图和模型之间的组织作用，控制程序的流程

View定义界面的例子

```
from traits.api import HasTraits, Str, Int

class ModelManager(HasTraits):
    model_name = Str
    category = Str
    model_file = Str
    model_number = Int

model = ModelManager()
model.configure_traits()
```

①添加traitsUI库

②添加视图代码

用View定义界面

①添加traitsUI库

```
from traitsui.api import View, Item
```

View定义界面的例子

②添加视图代码

```
view = View(
    Item('model_name', label=u"模型名称"),
    Item('model_file', label=u"文件名", tooltip=u"路径及文件名"),
    Item('category', label=u"模型类型"),
    Item('model_number',label=u"模型数量"),
    title = u"模型资料", width=200, resizable = True)
```

Item对象属性

Item (id , name , label…)

属性	说明
id	item的唯一id
name	trait属性的名称
label	静态文本，用于显示编辑器的标签
tooltip	编辑器的提示文本

view对象属性

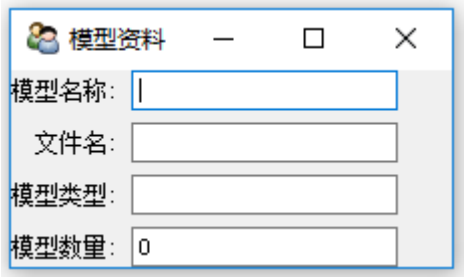
View (title , width , height , resizable…)

属性	说明
title	窗口标题栏
Width	窗口宽度
Height	窗口高度
resizable	窗口大小可变，默认为True

View定义界面的例子

```
from traits.api import HasTraits, Str, Int
from traitsui.api import View, Item

class ModelManager(HasTraits):
    model_name = Str
    category = Str
    model_file = Str
    model_number = Int
    view = View(
        Item('model_name', label=u"模型名称"),
        Item('model_file', label=u"文件名"),
        Item('category', label=u"模型类型"),
        Item('model_number',label=u"模型数量"),
        title = u"模型资料", width=220, resizable = True)

model = ModelManager()
model.configure_traits()
```

View定义界面的例子



Group对象组织界面

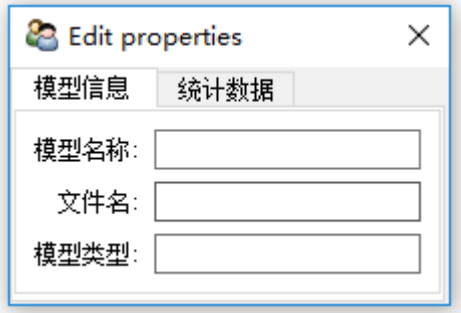
Group对象

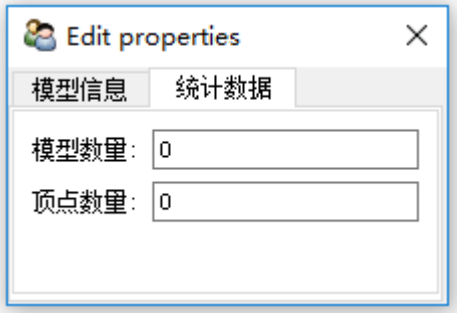
```
from traitsui.api import Group
Group(…)
```

属性	说明
orientation	编辑器的排列方向
layout	布局方式normal、flow、split、tabbed
show_labels	是否显示编辑器的标签
columns	布局的列数，范围为（1，50）

```python
from traits.api import HasTraits, Str, Int
from traitsui.api import View, Item, Group

class ModelManager(HasTraits):
    model_name = Str
    category = Str
    model_file = Str
    model_number = Int
    vertices = Int

view1 = View(
    Group(
        Item('model_name', label=u"模型名称"),
        Item('model_file', label=u"文件名"),
        Item('category', label=u"模型类型"),
        label = u'模型信息',
        show_border = True),
    Group(
        Item('model_number',label=u"模型数量"),
        Item('vertices',label=u"顶点数量"),
        label = u'统计数据',
        show_border = True)
)
model = ModelManager()
model.configure_traits(view=view1)
```

应用Group的例子

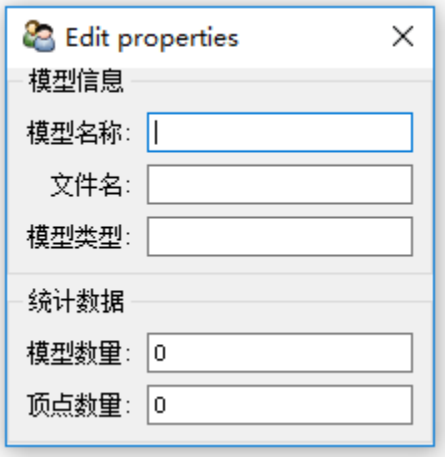

Edit properties
模型信息
统计数据
模型名称:
文件名:
模型类型:


Edit properties
模型信息
统计数据
模型数量:
0
顶点数量:
0

应用Group的例子

```
view1 = View(
    Group(
        Group(
            Item('model_name', label=u"模型名称"),
            Item('model_file', label=u"文件名"),
            Item('category', label=u"模型类型"),
            label = u'模型信息',
            show_border = True),
        Group(
            Item('model_number',label=u"模型数量"),
            Item('vertices',label=u"顶点数量"),
            label = u'统计数据',
            show_border = True)
        )
    )
```

应用Group的例子



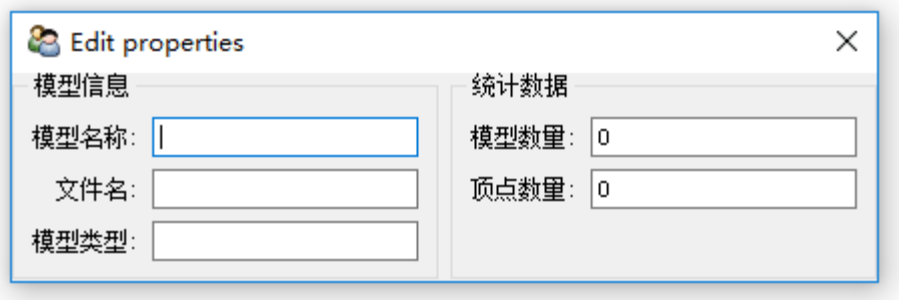
Group对象的嵌套关系

```
view = View(
    Group(
        Group(... ...),
        Group(... ...),
        orientation = 'horizontal'
        )
)
```

应用Group的例子

```
view1 = View(
    Group(
        Group(
            Item('model_name', label=u"模型名称"),
            Item('model_file', label=u"文件名"),
            Item('category', label=u"模型类型"),
            label = u'模型信息',
            show_border = True),
        Group(
            Item('model_number',label=u"模型数量"),
            Item('vertices',label=u"顶点数量"),
            label = u'统计数据',
            show_border = True),
        orientation = 'horizontal'
    )
)
```

应用Group的例子



Group类继承的HSplit类

Hsplit定义：

```
Class HSplit(Group):
    #...
    #...
    layout = 'split'
    orientation = 'horizontal'
```

代码等价于：

```
Group(…, layout = ‘split’, orientation = 'horizontal')
```

HSplit类的框架示例

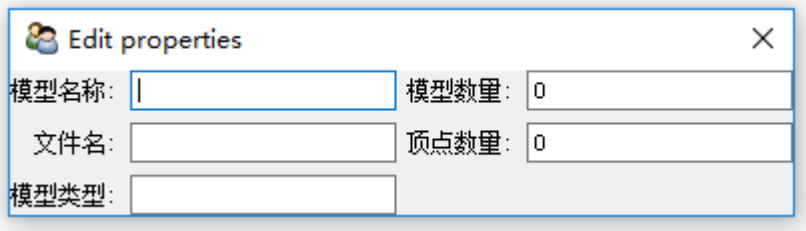
使用Hsplit代码框架：

```
view = View (
    HSplit(
        VGroup(
            ... ...
        ),
        VGroup(
            ... ...
        ),
    )
)
```

应用Group的例子

```
from traitsui.api import HSplit, VGroup

view1 = View(
    HSplit(
        VGroup(
            Item('model_name', label=u"模型名称"),
            Item('model_file', label=u"文件名"),
            Item('category', label=u"模型类型"),
            label = u'模型信息'),
        VGroup(
            Item('model_number',label=u"模型数量"),
            Item('vertices',label=u"顶点数量"),
            label = u'统计数据'),
    )
)
```

应用Group的例子



Group的各种派生类

派生类	说 明
HGroup	内容水平排列 Group(orientation=‘horizontal’)
HFlow	内容水平排列，超过水平宽度时，自动换行，隐藏标签文字。 Group(orientation=‘horizontal’,layout=‘flow’, show_labels=False)
HSplit	内容水平分隔，中间插入分隔条 Group(orientation=‘horizontal’,layout=‘flow’)

Group的各种派生类

派生类	说 明
Tabbed	内容分标签页显示 Group(orientation=‘horizontal’,layout=‘tabber’)
VGroup	内容垂直排列 Group(orientation=‘vertical’)
VFlow	内容垂直排列，超过垂直高度时，自动换列，隐藏标签文字 Group(orientation=‘vertical’,layout=‘flow’,show_labels=False)

Group的各种派生类

派生类	说 明
VFold	内容垂直排列，可折叠 Group(orientation=‘vertical’,layout=‘fold’,show_labels=False)
VGrid	按照多列网格进行垂直排列，columns属性决定网格的列数 Group(orientation=‘vertical’, columns=2)
VSplit	内容垂直排列，中间插入分隔条 Group(orientation=‘vertical’,layout=‘split’)

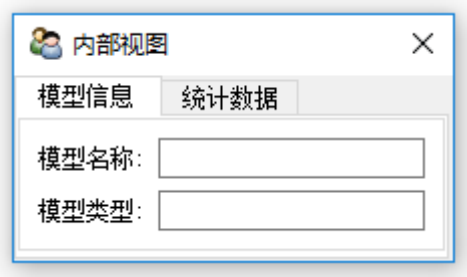
https://github.com/enthought/traitsui/blob/master/traitsui/group.py

使用多个视图对象

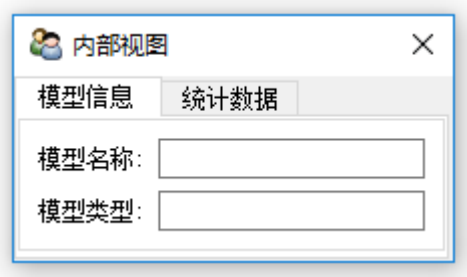
```
g1 = [Item('model_name', label=u"模型名称"),
      Item('category', label=u"模型类型")]
g2 = [Item('model_number',label=u"模型数量"),
      Item('vertices',label=u"顶点数量")]
```

```python
class ModelManager(HasTraits):
    model_name = Str
    category = Str
    model_number = Int
    vertices = Int
    traits_view = View(
        Group(*g1, label = u"模型信息", show_border = True),
        Group(*g2, label = u"统计数据", show_border = True),
        title = u"内部视图")

global_view = View(
    Group(*g1, label = u"模型信息", show_border = True),
    Group(*g2, label = u"统计数据", show_border = True),
    title = u"外部视图")

model = ModelManager()
```

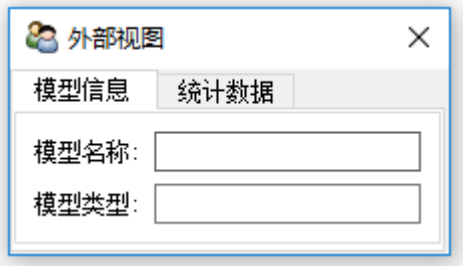
```
model.configure_traits()
```




```
model.configure_traits(view='traits_view')
```




```
model.configure_traits(view=global_view)
```



配置视图

视图类型

通过kind属性设置View显示类型

显示类型	说 明
modal	模态窗口，非即时更新
live	非模态窗口，即时更新
livemodal	模态窗口，即时更新
nonmodal	非模态窗口，非即时更新
wizard	向导类型
panel	嵌入到其它窗口中的面板，即时更新，非模式
subpanel	

视图类型

显示类型
modal
live
livemodal
nonmodal

采用窗口显示内容

模态窗口：在此窗口关闭之前，其他窗口不能激活；

即时更新：修改控件内容，立即反应到模型数据上。

视图类型

显示类型
wizard

向导窗口，模态窗口，即时更新

显示类型
panel
subpanel

嵌入窗口中的面板

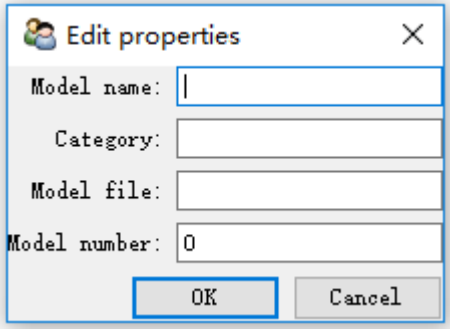
视图配置的实例

```
view = View(... ...
            kind = 'modal')
```

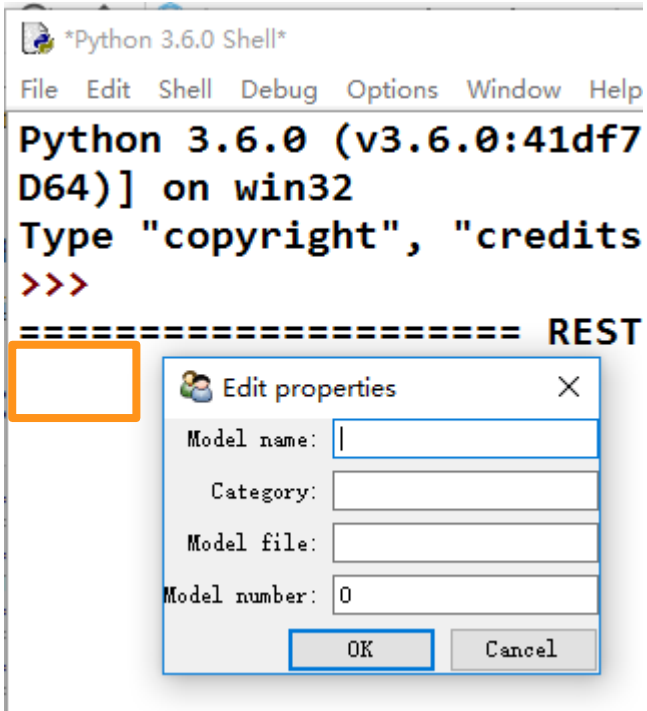
模态与非模态的小例子

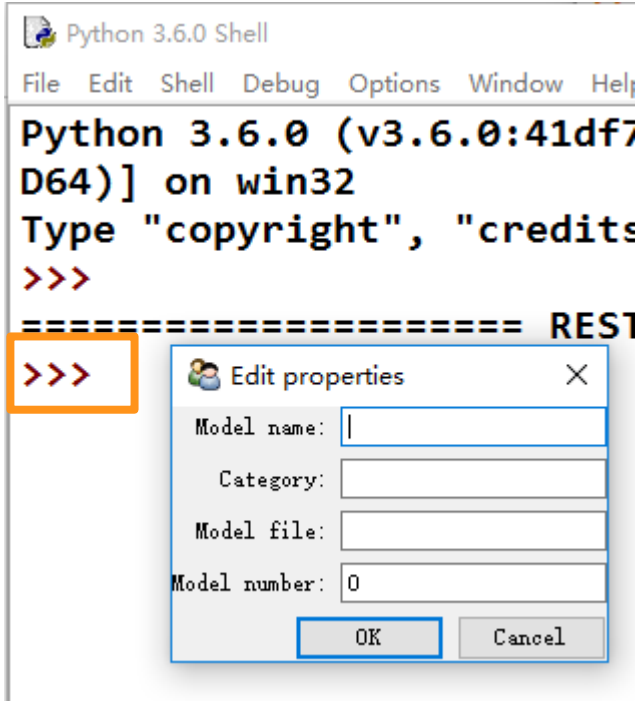
```
class ModelManager(HasTraits):
    model_name = Str
    category = Str
    model_file = Str
    model_number = Int

model = ModelManager()
model.configure_traits()
```

```
class ModelManager(HasTraits):
    model_name = Str
    category = Str
    model_file = Str
    model_number = Int

model = ModelManager()
model.edit_traits()
```



configure_traits	edit_traits()
界面显示后，进入消息循环	界面显示后，不进入消息循环。
主界面窗口或模态对话框	无模态窗口或对画框





truitsUI按钮配置

标准命令按钮
UndoButton
ApplyButton
RevertButton
OKButton
CancelButton
HelpButton

truitsUI按钮配置

traitsui.menu预定义了命令按钮：

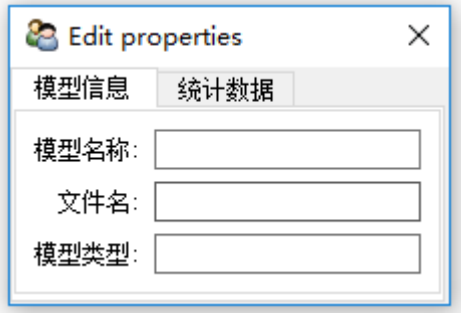
```
OKCancelButtons = [OKButton, CancelButton]
ModelButtons = [ApplyButton, RevertButton, OKButton, CancelButton,
HelpButton]
LiveButtons = [UndoButton, RevertButton, OkButton, Cancel]
```

truitsUI按钮配置的实例

```
from traitsui.menu import ModalButtons
view = View(... ...
            ... ...
            buttons = ModalButtons)
```

```python
from traits.api import HasTraits, Str, Int
from traitsui.api import View, Item, Group

class ModelManager(HasTraits):
    model_name = Str
    category = Str
    model_file = Str
    model_number = Int
    vertices = Int

view1 = View(
    Group(
        Item('model_name', label=u"模型名称"),
        Item('model_file', label=u"文件名"),
        Item('category', label=u"模型类型"),
        label = u'模型信息',
        show_border = True),
    Group(
        Item('model_number',label=u"模型数量"),
        Item('vertices',label=u"顶点数量"),
        label = u'统计数据',
        show_border = True)
)
model = ModelManager()
model.configure_traits(view=view1)
```

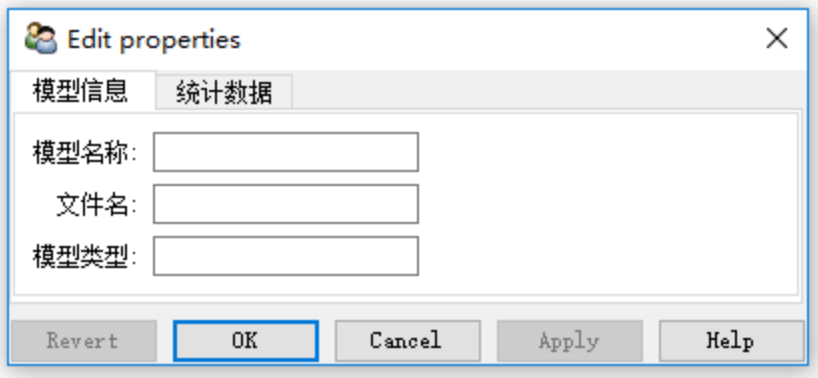
回顾例子：



视图和按钮配置的实例

```
from traitsui.menu import ModalButtons

view = View(... ...
            kind = 'Modal', #模态对话框
            buttons = ModalButtons) #采用模态对话框的按钮
```

```python
from traits.api import HasTraits, Str, Int
from traitsui.api import View, Item, Group
from traitsui.menu import ModalButtons

class ModelManager(HasTraits):
    model_name = Str
    category = Str
    model_file = Str
    model_number = Int
    vertices = Int
    view = View(
        Group(
            Item('model_name', label=u"模型名称"),
            Item('model_file', label=u"文件名"),
            Item('category', label=u"模型类型"),
            label = u'模型信息',
            show_border = True),
        Group(
            Item('model_number',label=u"模型数量"),
            Item('vertices',label=u"顶点数量"),
            label = u'统计数据',
            show_border = True),
        kind = "modal",
        buttons = ModalButtons
    )

model = ModelManager()
model.configure_traits()
```


Edit properties
模型信息
统计数据
模型名称:
文件名:
模型类型:
Revert
OK
Cancel
Apply
Help

TraitsUI控件

文本编辑器

定义文本编辑器变量：

```
string_trait = Str("sample string")
password     = Password #密码trait
```

设置文本编辑器风格：

```
Item('string_trait', style='simple', label='Simple'),
```

'password'

'custom'：多行显示

'readonly'：只读

```python
from traits.api import HasTraits, Str, Password
from traitsui.api import Item, Group, View

class TextEditor( HasTraits ):
    # 定义文本编辑器的变量
    string_trait = Str("sample string")
    password     = Password
    # 定义布局
    text_str_group = Group(
        Item('string_trait', style='simple', label='Simple'),
        Item('_'),
        Item('string_trait', style='custom', label='Custom'),
        Item('_'),
        Item('password', style='simple', label='password'))
    # 定义视图
    traits_view = View(
        text_str_group,
        title   = 'TextEditor',
        buttons = [ 'OK' ])

text = TextEditor()
text.configure_traits()
```

文本编辑器

输入单行文本

输入多行文本

密码符号

按钮Button Editor一般程序框架

```
from traits.api import Button # 导入控件模块
class ButtonEditor(HasTraits):
    TraitName = Button() # 定义按钮变量名
    def _TraitName_fired(self)   # 定义监听函数
        ... ...
    view = View()  # 定义视图

button = ButtonEditor()
button.configure_traits()
```

按钮Button Editor

监听方法对比

	Event属性	Trait属性
触发监听事件	对Event属性赋值	值被改变后
监听函数名	`_event_fired()`	`_trait_changed()`

按钮Button Editor

监听方法：

```
_TraitName_fired()
```

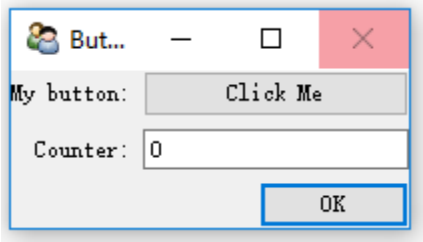
TraitName是Button Trait的名字，即my_button

```
def _my_button_fired(self):
        self.counter += 1
```

按钮Button Editor

```python
from traits.api import HasTraits, Button, Int
from traitsui.api import View

class ButtonEditor(HasTraits):
    # 定义一个Button trait:
    my_button = Button(u'点击我')
    counter = Int
    # 当按钮点击后，处理当按钮被点击后，触发的事件
    def _my_button_fired(self):
        self.counter += 1
    # 创建视图
    traits_view = View(
        'my_button',
        'counter',
        title     = 'ButtonEditor',
        buttons   = [ 'OK' ],
        resizable = True)

button = ButtonEditor()
button.configure_traits()
```

按钮Button Editor



Range滑动条

定义控件变量：

```
from traits.api import Int, Range, Property

a = Range(1, 10)
b = Range(1, 10)
c = Property(Int)
```

```
_get_c()
```

Range滑动条

```
from traits.api import property_depends_on

@property_depends_on('a,b', settable = True)
def _get_c(self):
        return (self.a + self.b)
```

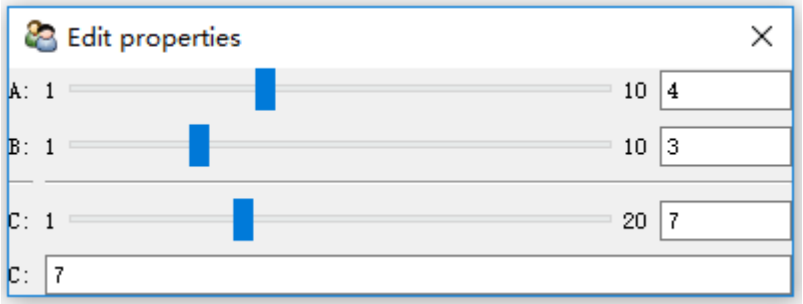
Range滑动条

定义视图布局：

```
from traitsui.api import View, Item, RangeEditor

view = View(
    Item('a'),
    Item('b'),
    '_',
    Item('c',editor=RangeEditor(low = 1, high = 20, mode = 'slider')),
    Item('c'),
    width = 0.3
)
```

```python
from traits.api import HasTraits, Int, Range, Property, property_depends_on
from traitsui.api import View, Item, RangeEditor

class RangeDemo(HasTraits):
    a = Range(1, 10)
    b = Range(1, 10)
    c = Property(Int)
    view = View(
        Item('a'),
        Item('b'),
        '_',
        Item('c',editor=RangeEditor(low = 1, high = 20, mode = 'slider')),
        Item('c'),
        width = 0.3
    )

    @property_depends_on('a,b', settable = True)
    def _get_c(self):
        return (self.a + self.b)

ran = RangeDemo()
ran.configure_traits()
```

Range滑动条



菜单、工具栏

```
from traitsui.menu import Action…
```

对象	说 明
Action	在Menu对象中，通过Action对象定义菜单中的每个选项
ActionGroup	对菜单中的选项进行分组
Menu	定义菜单栏中的一个菜单
MenuBar	菜单栏对象，由多个Menu对象组成
ToolBar	工具栏对象，它由多个Action对象组成，每个Action对应工具条中的一个按钮

控件列表

对象	说 明
Array	数组空间
Bool	单选框、复选框
Button	按钮
Code	代码编辑器
Color	颜色对话框

其他控件

对象	说 明
Dircetory	目录控件
Enum	枚举控件
File	文件控件
Font	字体选择控件
Html	Html网页控件

其他控件

对象	说 明
List	列表框
Str	本文框
Password	密码控件
Tuple	元组控件

traitsui_editors.py见网络资源