

CS 535 Object-Oriented Programming & Design
Fall Semester, 2011
Doc 12 VW GUI
Sep 29 2011

Copyright ©, All rights reserved. 2011 SDSU & Roger Whitney, 5500 Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (<http://www.opencontent.org/openpub/>) license defines the copyright on this document.

References

Pattern-Oriented Software Architecture, Buschmann et al., 1996

VisualWorks GUI Developer's Guide, GUIDevGuide.pdf in the docs directory of the VW distribution

Dialog Windows

Windows used to

- Display information to the user

- Request information from user

VW Dialogs are modal

- User has to respond before application continues

Warn

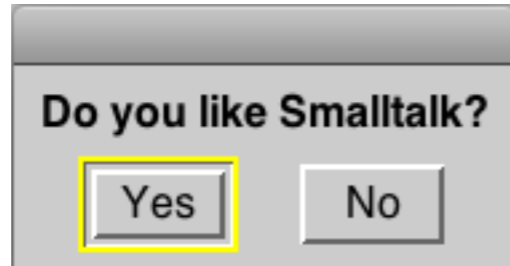
Dialog warn: 'This is a simple dialog window'.



Returns: nil

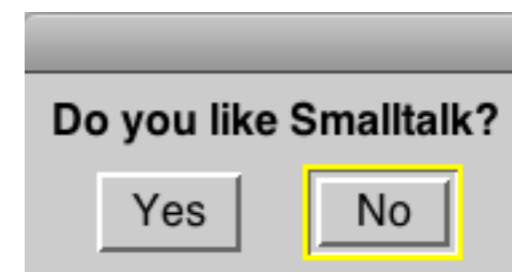
Confirm

answer := Dialog confirm: 'Do you like Smalltalk?'



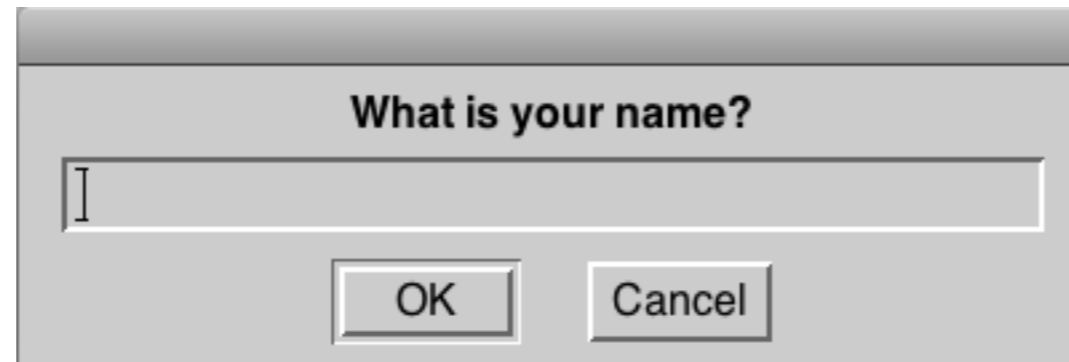
Returns: true or false

answer := Dialog
confirm: 'Do you like
Smalltalk?'
initialAnswer: false



Request

answer := Dialog request: 'What is your name?'



Returns:

Text entered on "OK"

Empty string on "Cancel"

Request Variations

Dialog

request: 'What is your name?'

initialAnswer: 'Smith'

Dialog

request: 'What is your name?'

initialAnswer: 'Smith'

onCancel: ['Jones']

Choose

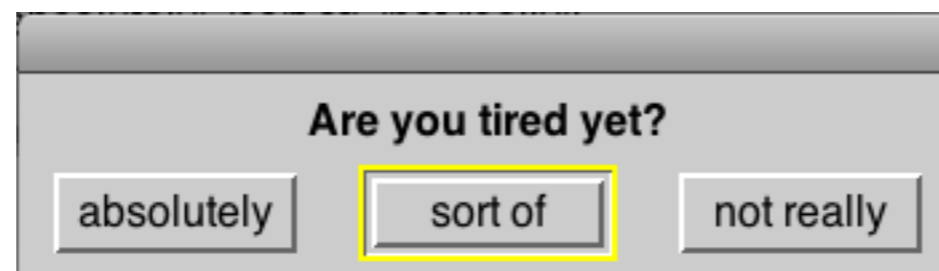
Dialog

choose: 'Are you tired yet?'

labels: #('absolutely' 'sort of' 'not really')

values: #(#yes #maybe #no)

default: #maybe



labels: what the users sees

values: value returned when user selects corresponding label

Choose from List

answer := Dialog

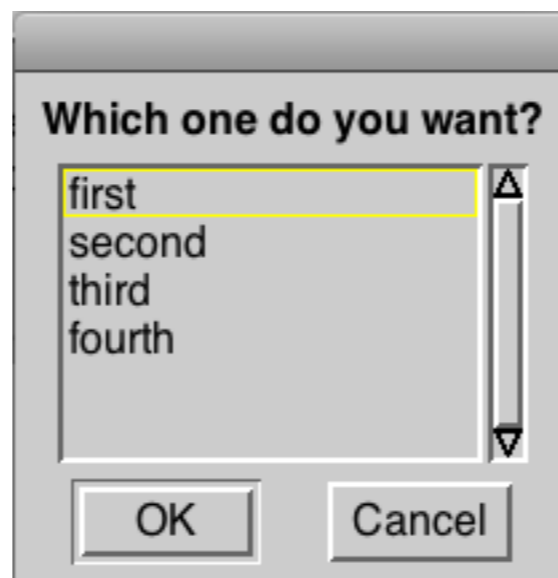
choose: 'Which one do you want?'

fromList: #('first' 'second' 'third' 'fourth')

values: #(1 2 3 4)

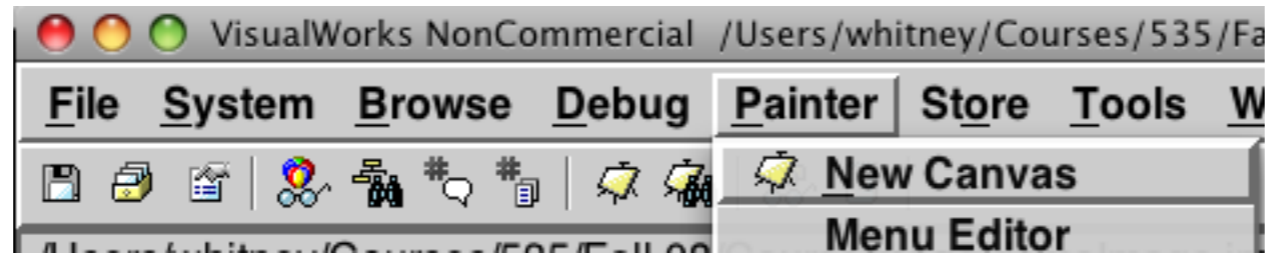
lines: 8

cancel: [#noChoice]

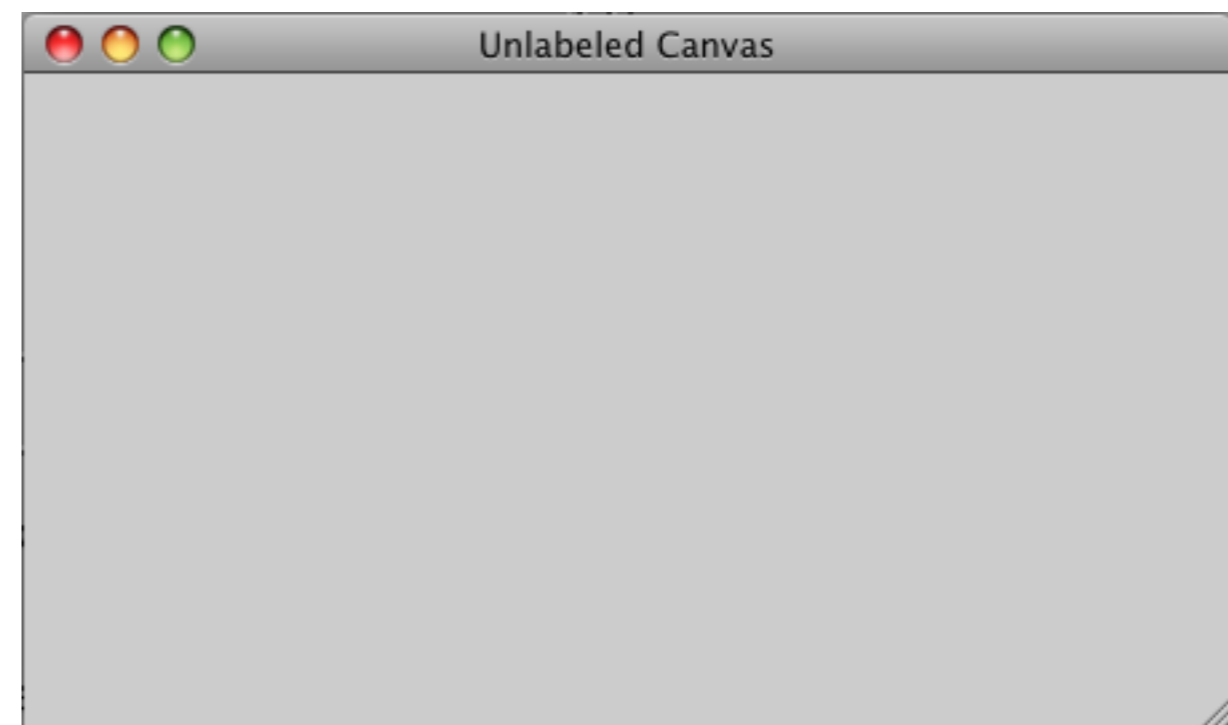
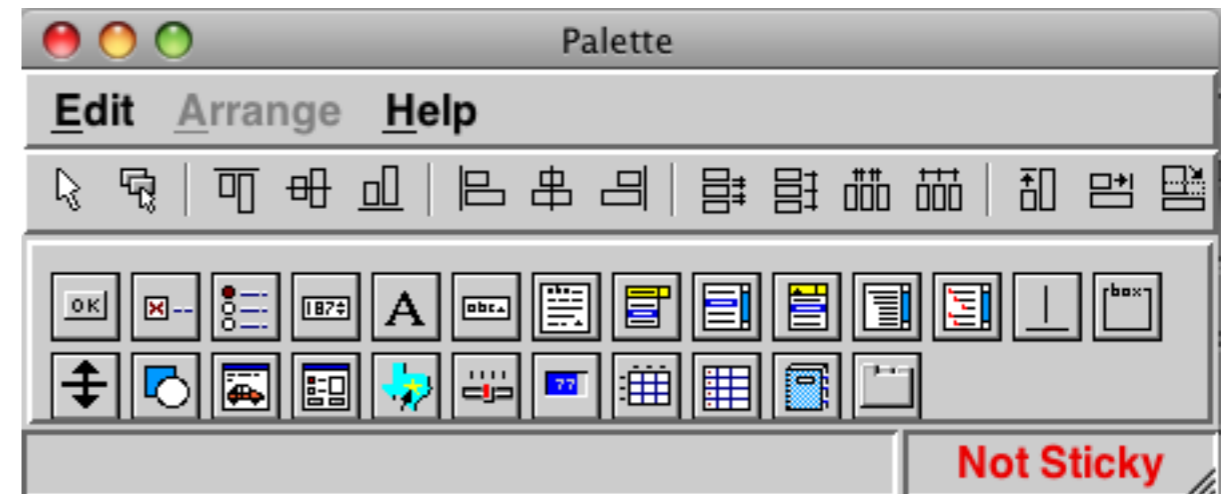
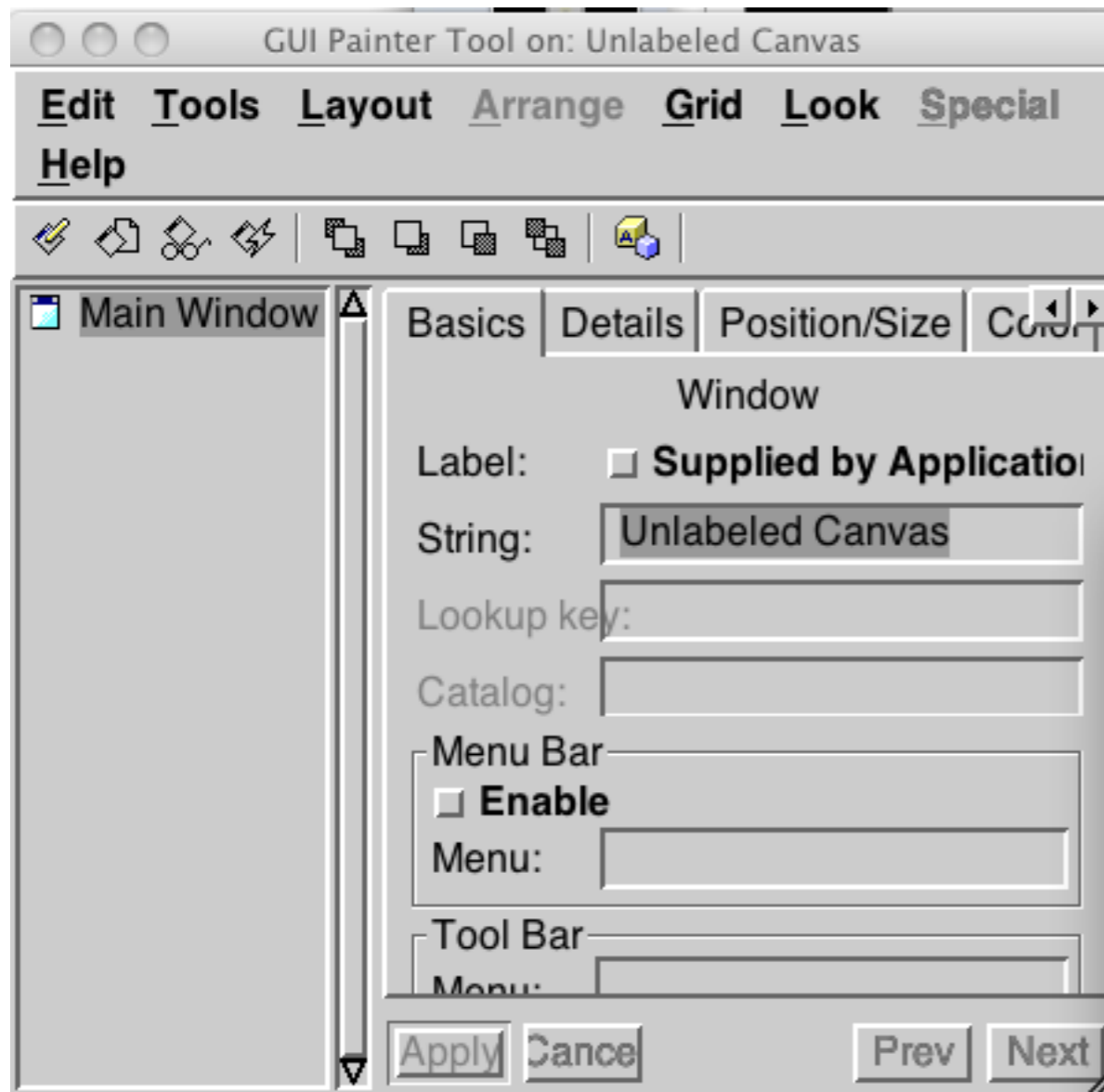


UIPainter

Graphical Tool used to develop GUIs



UI Painter Windows



11

Thursday, September 29, 2011

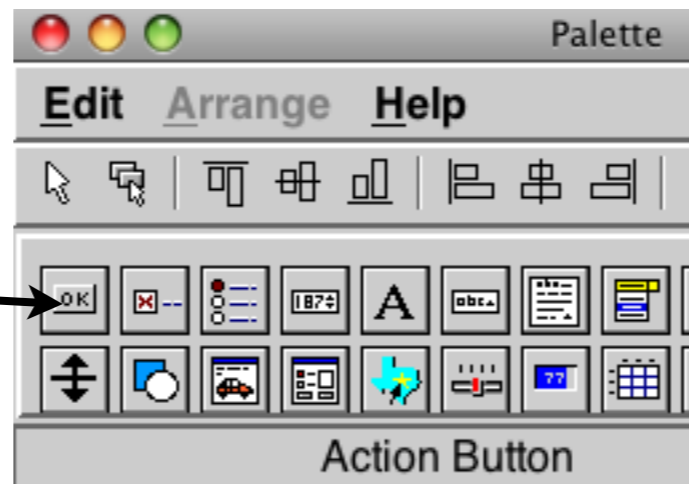
Palette – Widgets that we can put in the window

Unlabeled Canvas – Window we are constructing

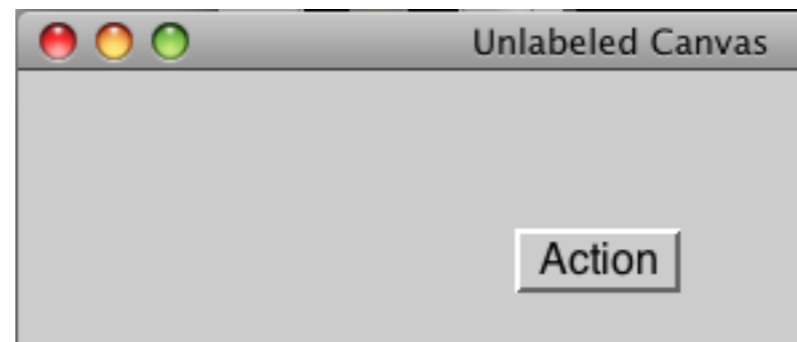
GUI Painter Tool – Details about the widgets in our new window

Adding A Button

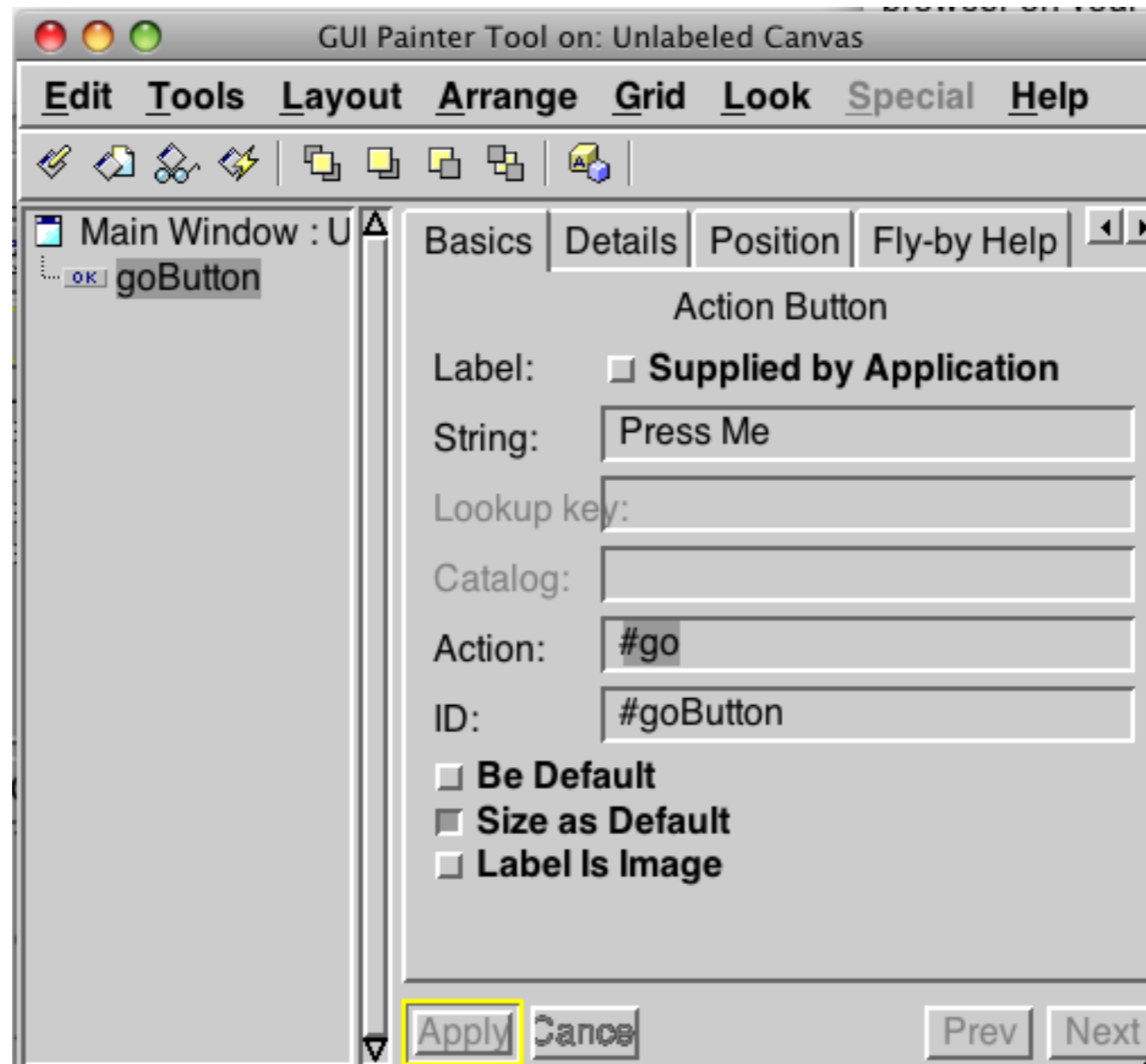
Click on "button" icon



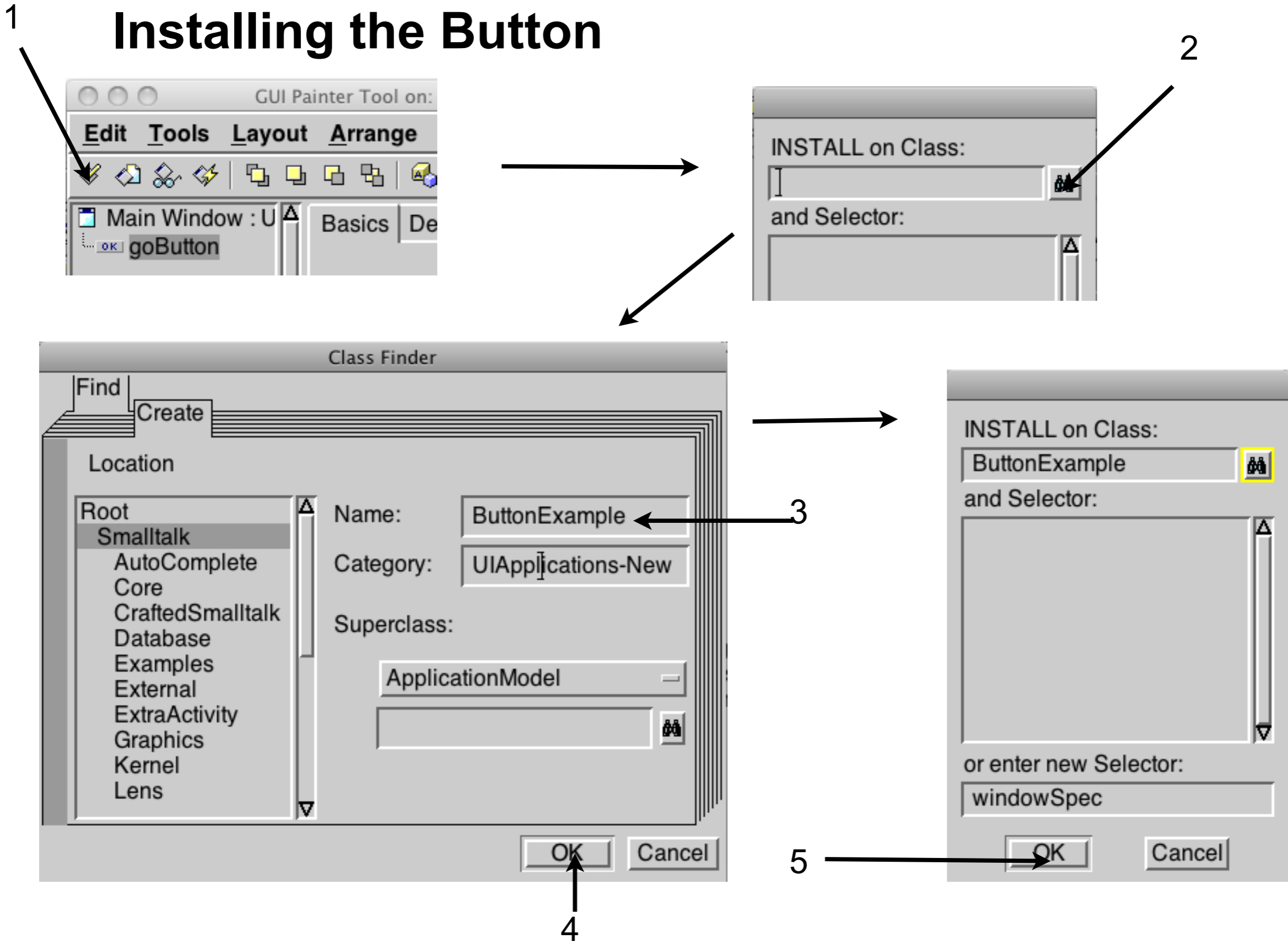
Click in the canvas
where you want the
button



Configuring Button



Installing the Button

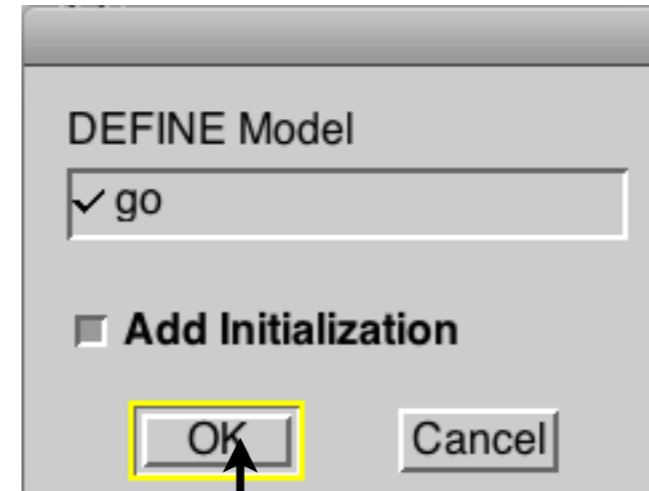
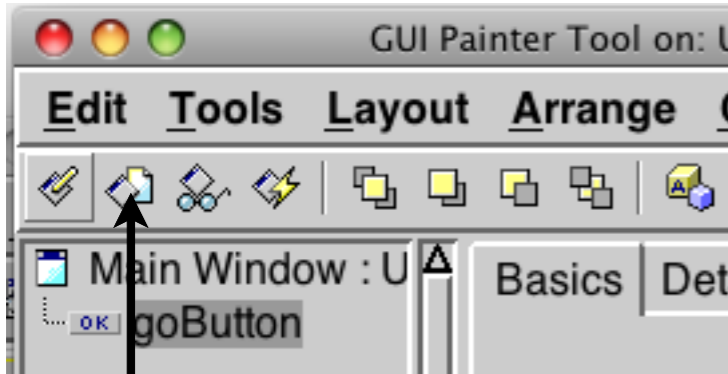


14

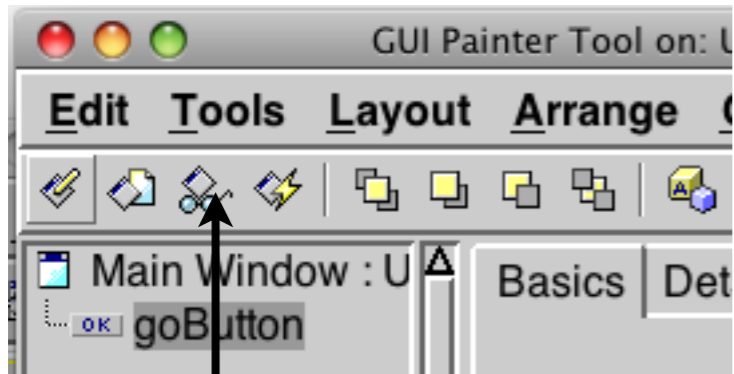
Thursday, September 29, 2011

1. In the GUI Painter Tool click on the "Install..." button.
2. Click on the magnifying glass to open the class finder. We will create a new class.
- 3 Click on the "Create" tab. Click on the Examples namespace. Then enter the name of the class. tab out of the field
- 4 & 5. Click "OK"

Define the "go" method



Edit the "go" method



1

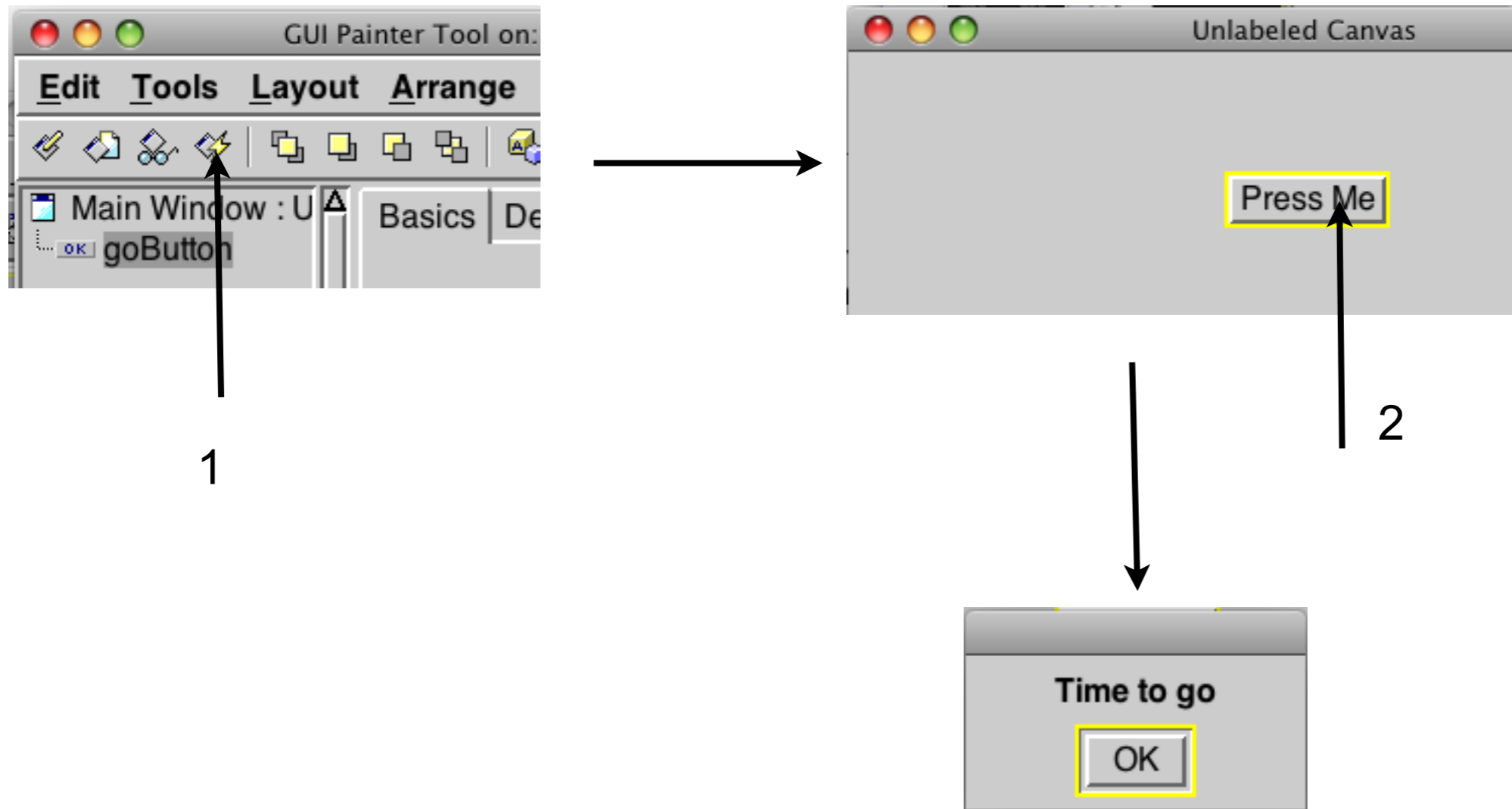
2. Edit go method in browser to be:

```
go
```

```
Dialog warn: 'Time to go'.
```

```
^self
```


Running the Example



Using Code to Run the Example

ButtonExample open

WindowSpec

ButtonExample>>windowSpec

```
"Tools.UIPainter new openOnClass: self andSelector: #windowSpec"
```

```
<resource: #canvas>
```

```
^#(#{UI.FullSpec}
```

```
  #window:
```

```
    #(#{UI.WindowSpec}
```

```
      #label: 'Unlabeled Canvas'
```

```
      #bounds: #(#{Graphics.Rectangle} 556 524 1008 770 ) )
```

```
  #component:
```

```
    #(#{UI.SpecCollection}
```

```
      #collection: #(
```

```
        #(#{UI.ActionButtonSpec}
```

```
          #layout: #(#{Graphics.Rectangle} 183 56 271 90 )
```

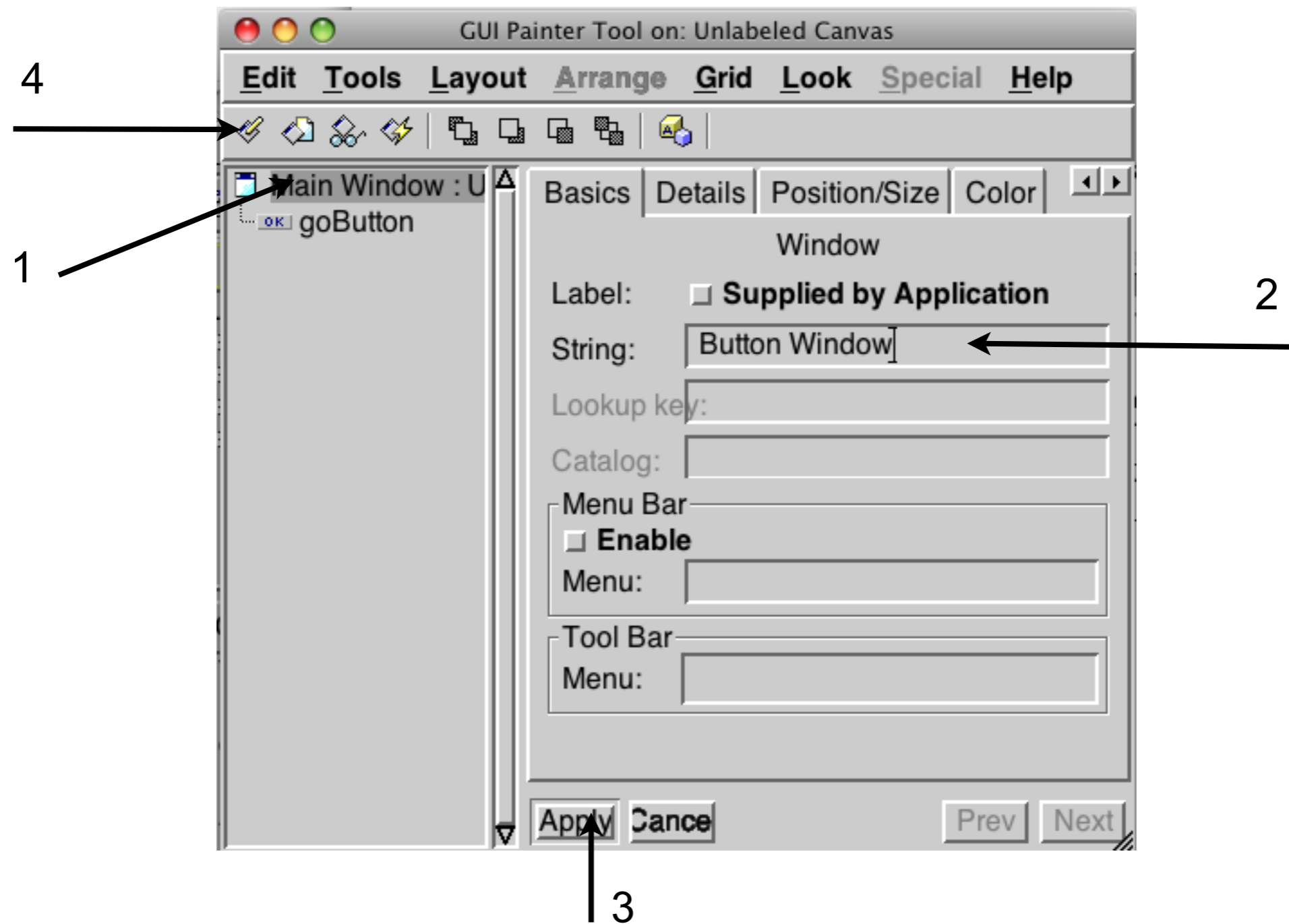
```
          #name: #goButton
```

```
          #model: #go
```

```
          #label: 'Press Me'
```

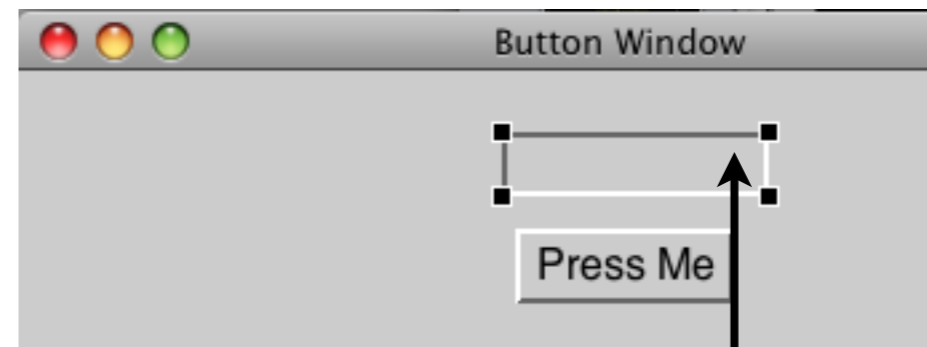
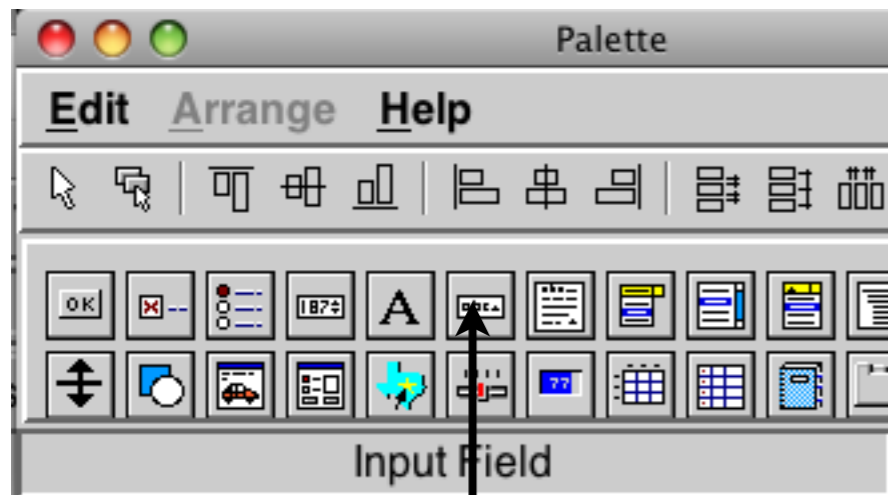
```
          #defaultable: true ) ) ) )
```

Window label



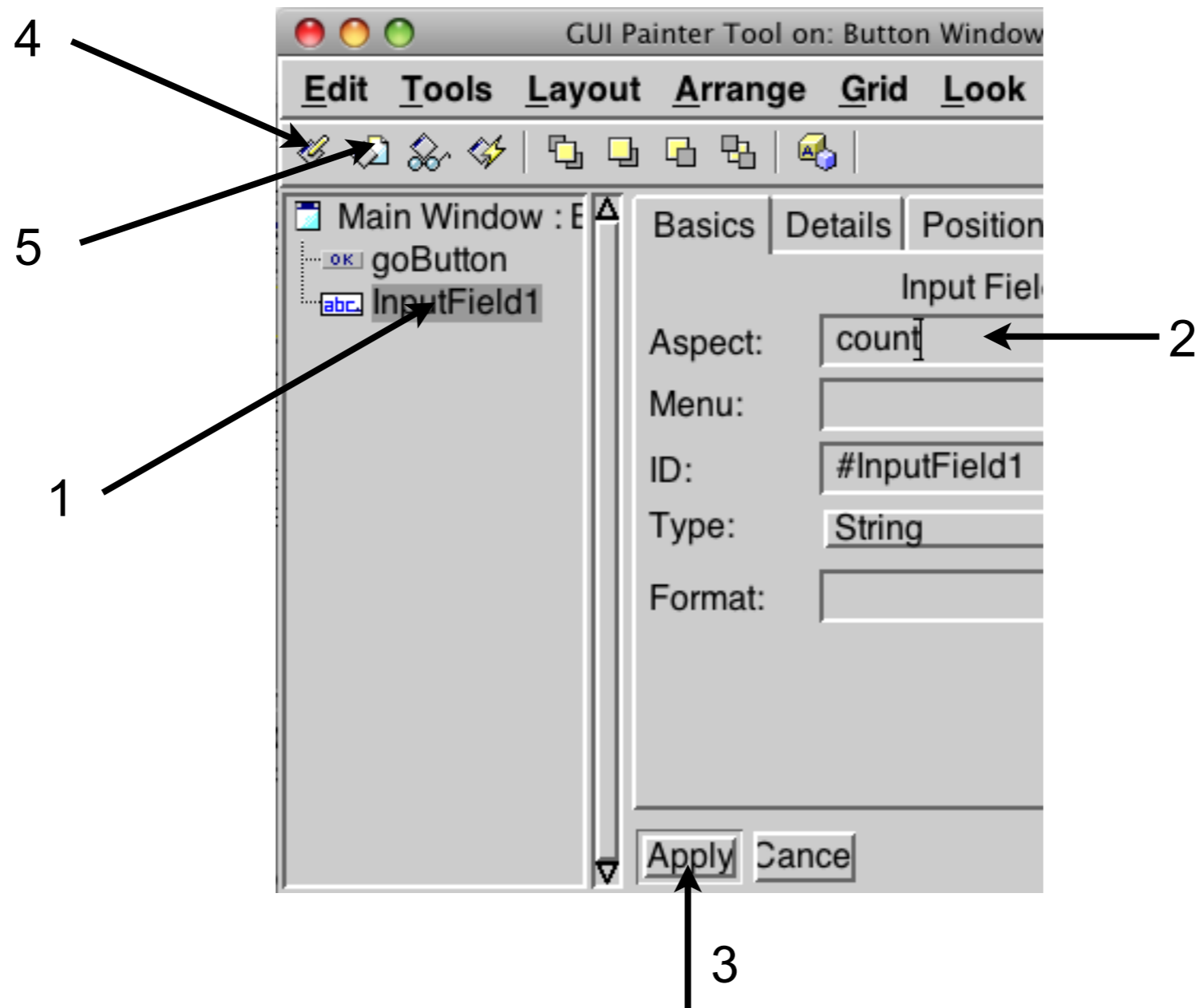
1. Select "Main window"
2. Enter Window label
3. Apply the change
4. Install the change in the ButtonExample class

Adding Text Input



1. Click on the "input Field" icon
2. Click in the Button Window where you would like the input field

Configuring Input Field



count method

ButtonExample>>count

"This method was generated by UIDefiner. Any edits made here may be lost whenever methods are automatically defined. The initialization provided below may have been preempted by an initialize method."

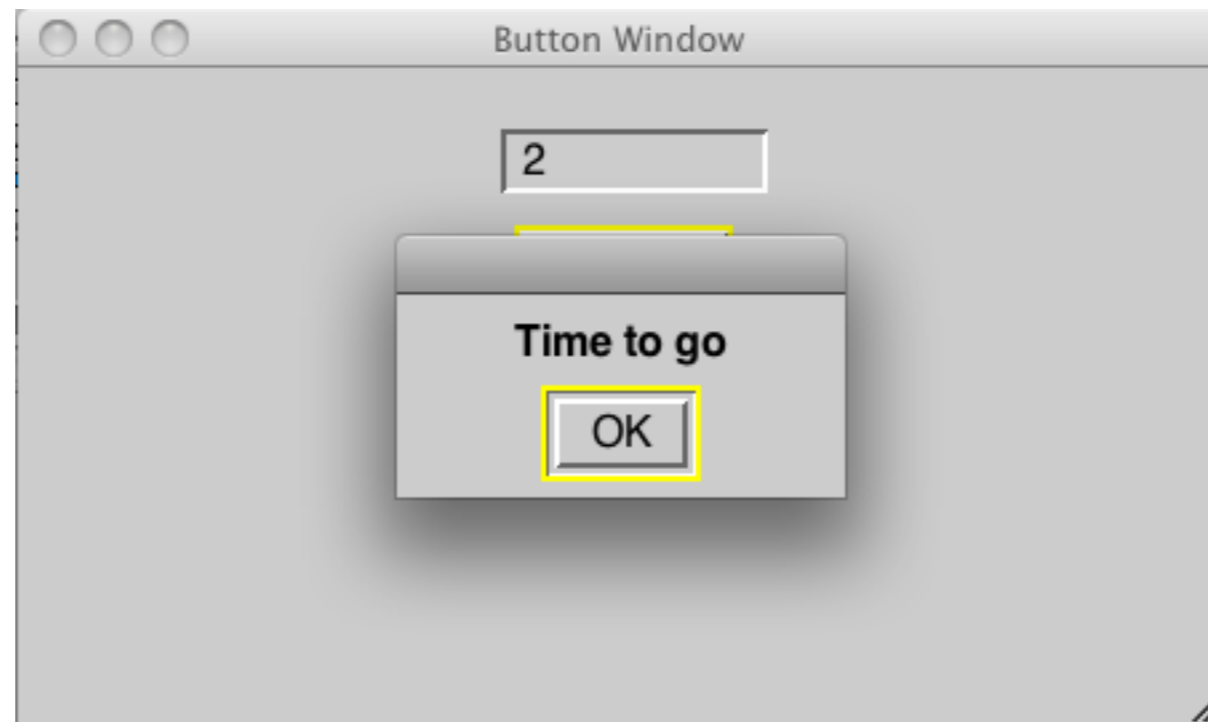
```
^count isNil
  ifTrue:
    [count := '0' asValue]
  ifFalse:
    [count]
```

go method

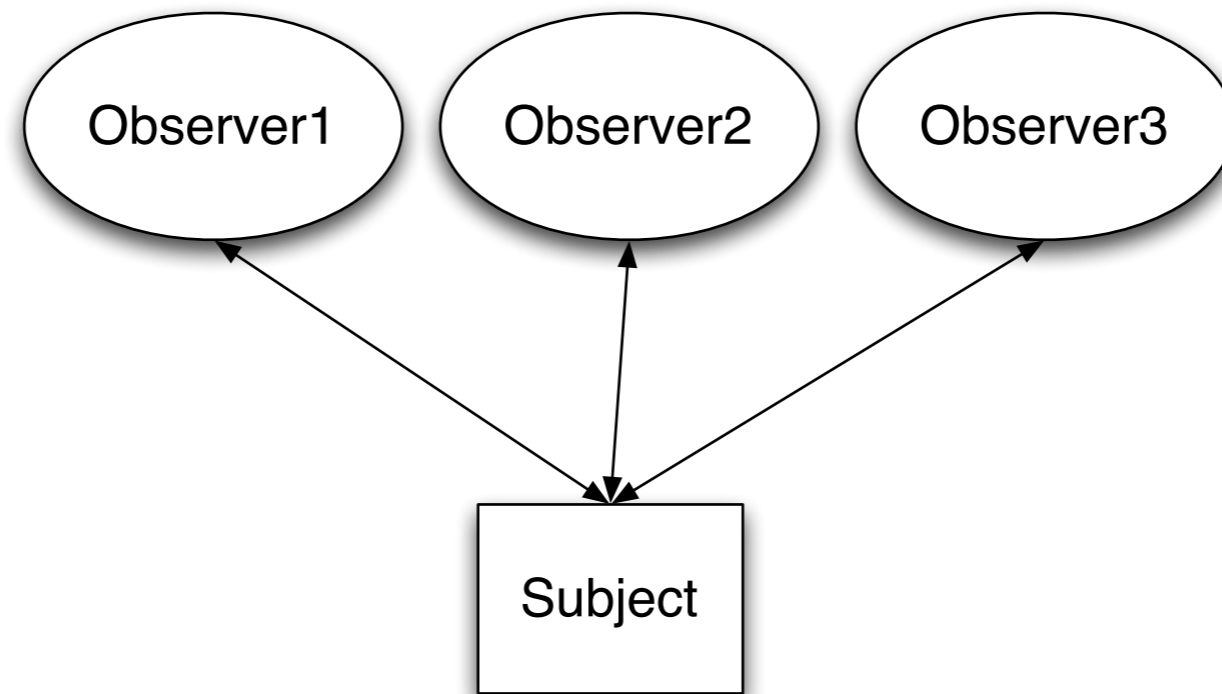
```
ButtonExample>>go
```

```
self count value: ((self count value asNumber) + 1) printString.  
Dialog warn: 'Time to go'.  
^self
```


Window in Action



Observer



Subject notifies all observers when it changes

Coupling

Measure of the interdependence among modules

"Unnecessary object coupling needlessly decreases the reusability of the coupled objects "

"Unnecessary object coupling also increases the chances of system corruption when changes are made to one or more of the coupled objects"

Notify With Coupling

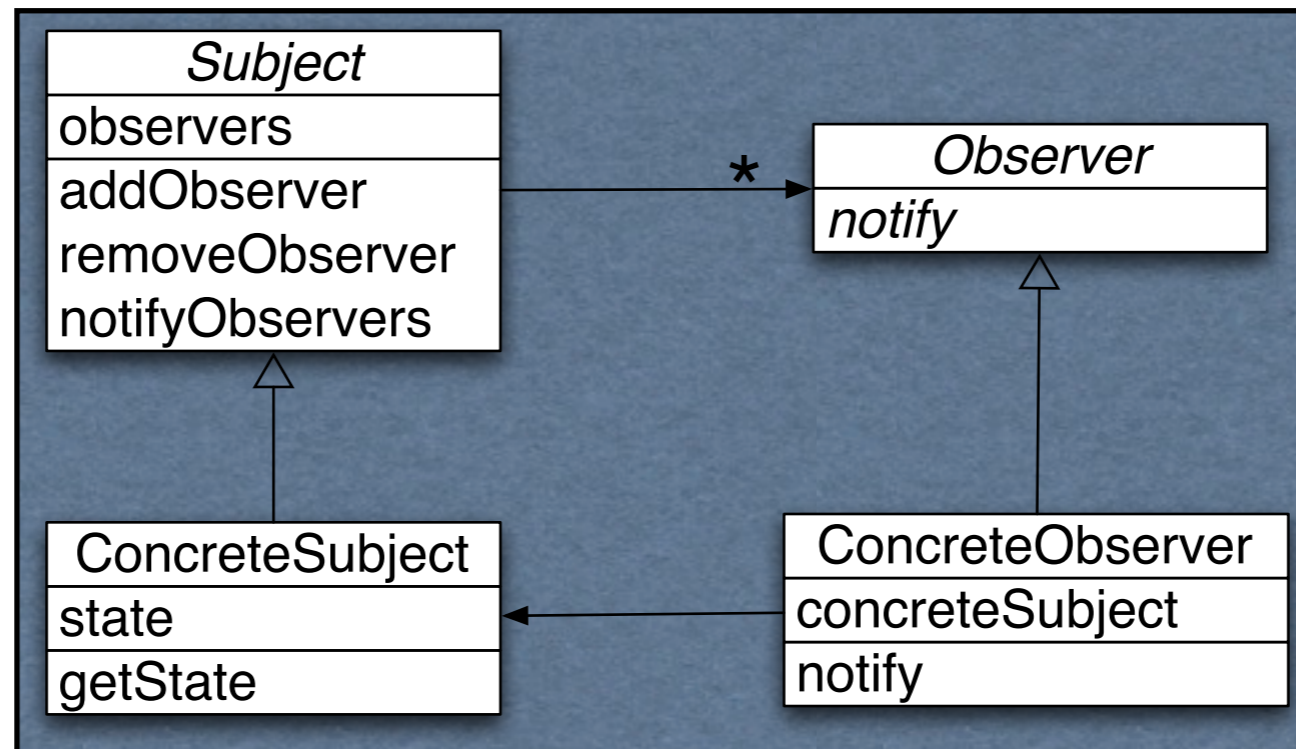
Subject>>notifyObservers

observer1 myNewValue: self myValue.

observer2 displayValue: self myValue.

observer3 dispalyAndSave: self myValue.

Keeping it Flexible



```
Subject>>notifyObservers
  observers do: [:each | each notify]
```

ValueHolder

A subject

When value changes it notifies observers

foo asValue

Returns ValueHolder on foo

valueHolder value: newValue

Changes the value

Notifies observers

Button Example

```
ButtonExample>>count
```

```
  ^count isNil
```

```
    ifTrue:
```

```
      [count := '0' asValue]
```

```
    ifFalse:
```

```
      [count]
```

```
ButtonExample>>go
```

```
self count value: (self count value asNumber + 1) printString.
```

```
Dialog warn: 'Time to go'.
```

```
^self
```

Coupling and Transcript

```
Smalltalk.CS535 defineClass: #Customer  
  superclass: #{Core.Object}  
  instanceVariableNames: 'name phone id '
```

```
Customer>>display  
Transcript  
  show: 'Customer(';  
  print: name;  
  show: ', ';  
  print: phone;  
  show: ', ';  
  print: id;  
  show: ')'
```

```
foo := Customer new.  
...  
foo display.
```


Separate display device from Customer

```
Customer>>printOn: aStream
aStream
  print: 'Customer(';
  print: name;
  print: ', ';
  print: phone;
  print: ', ';
  print: id;
  print: ')'
```

```
foo := Customer new.
```

```
...
```

```
Transcript
```

```
  show: foo printString.
```

```
bar := 'bar' asFilename writeStream.
```

```
bar
```

```
  nextPutAll: foo printString
```

GUIs & Coupling

Domain information

- Customer records
- Inventory
- Names
- Reports
- Addresses

Application/GUI information

- Menus
- Error Messages
- Help information
- Labels

Keep domain and application information separate

Application information changes faster

Often there is multiple view of domain information

Model-View-Controller (MVC)

Model

Encapsulates

Domain information
Core data and functionality

Independent of

Specific output representations
Input behavior

View

Display data to the user

Obtains data from the model

Multiple views of the model are possible

Controller

Handles input

- Mouse movements and clicks
- Keyboard events

Each view has it's own controller

Programmers commonly don't see controllers

Button Example

Model

ButtonExample

View

Created dynamically from
window spec

Controller

Hidden

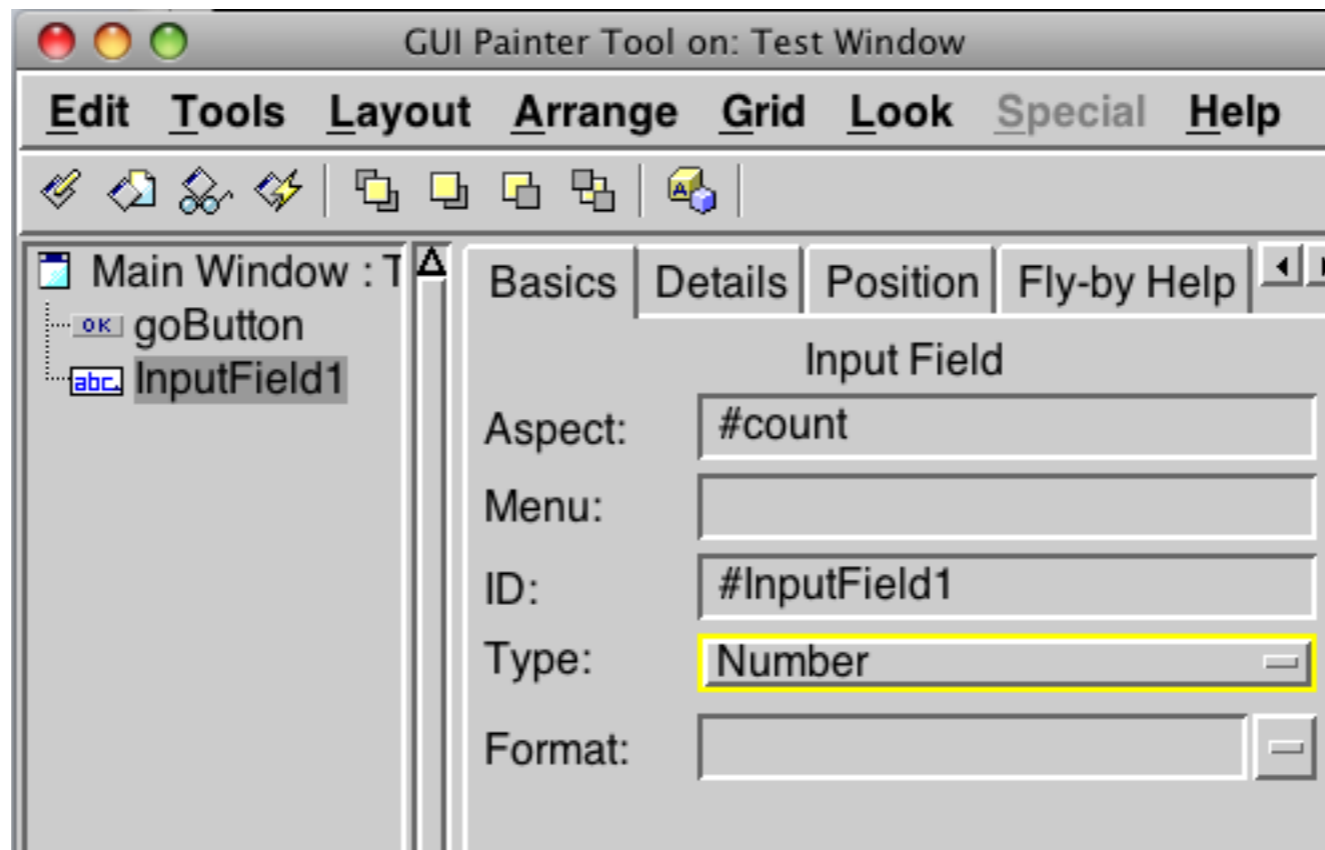
Issues with ButtonExample

Strings rather than numbers

Dealing with ValueHolders rather than with values

View code/logic in domain code

Configuring the Widget for Numbers



Using Numbers in Button Example

ButtonExample>>count

```
^count isNil
  ifTrue:
    [count := 0 asValue]
  ifFalse:
    [count]
```

ButtonExample>>go

```
self count value: (self count value + 1) .
Dialog warn: 'Hi'.
^self
```


Adapters



ButtonExample Adapter

```
ButtonExample>>countAdapter
```

```
| countAdapter |  
countAdapter := AspectAdaptor subject: self.  
countAdapter  
    forAspect: #count;  
    subjectSendsUpdates: true.  
^countAdapter
```

```
ButtonExample>>initialize
```

```
    count := 0
```

```
ButtonExample>>count
```

```
    ^count
```

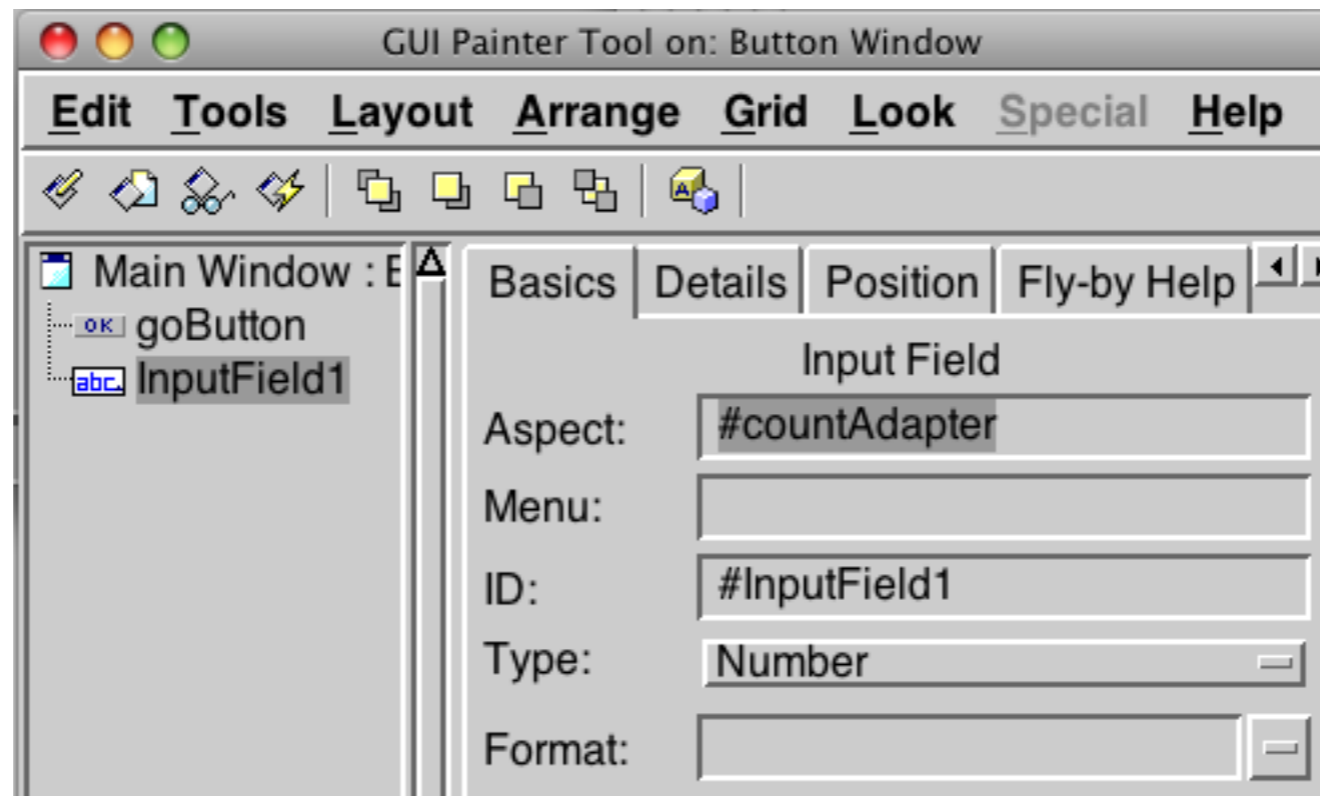
```
ButtonExample>>go
```

```
    count := count + 1.  
    self changed: #count.  
    Dialog warn: 'Time to go'.  
^self
```

```
ButtonExample>>count: anInteger
```

```
    count := anInteger
```

Using a Number rather than a String



View & Domain Logic Mixed

ButtonExample

Simple example

Designed to show how to use a widget

It handles both view logic and domain logic

Simple Domain Class

```
Smalltalk defineClass: #Counter  
  superclass: #{Core.Object}  
  instanceVariableNames: 'count '
```

```
Counter class>>new  
  ^super new initialize
```

```
Counter>>count  
  ^count
```

```
Counter>>count: anInteger  
  count := anInteger
```

```
Counter>>increment  
  self count: count + 1
```

```
Counter>>initialize  
  count := 0
```

Using the Domain Object

```
Smalltalk defineClass: #ButtonExample  
  superclass: #{UI.ApplicationModel}  
  instanceVariableNames: 'count '
```

```
initialize
```

```
  count := Counter new
```

```
go
```

```
  count increment.
```

```
  count changed: #count.
```

```
  Dialog warn: 'Time to go'.
```

```
  ^self
```

```
countAdapter
```

```
  | countAdapter |
```

```
  countAdapter := AspectAdaptor subject: count.
```

```
  countAdapter
```

```
    forAspect: #count;
```

```
    subjectSendsUpdates: true.
```

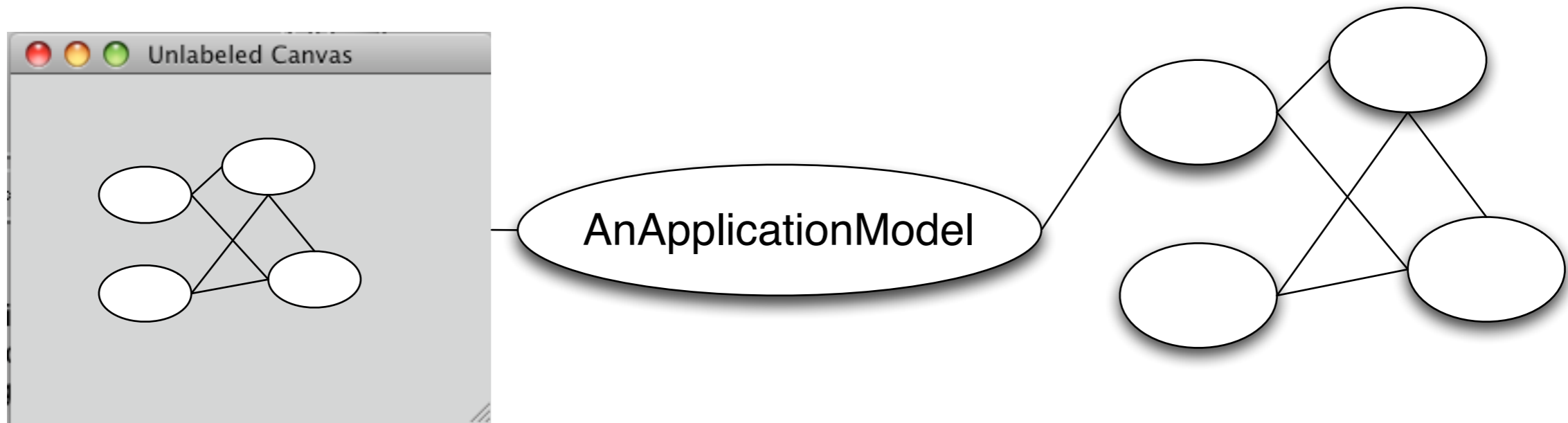
```
  ^countAdapter
```

Issue - Who changes count?

ButtonExample class controls when count changes

ButtonExample can then inform window of changes
Keeps Counter class independent of GUI

Issue - Who changes count?



What if other objects can change count?

ButtonExample will not be able to inform window of changes

Domain Objects Updated

```
Smalltalk defineClass: #Counter  
  superclass: #{Core.Object}  
  instanceVariableNames: 'count '
```

```
Counter class>>new  
  ^super new initialize
```

```
Counter>>count  
  ^count
```

```
Counter>>count: anInteger  
  count := anInteger
```

```
Counter>>increment  
  self count: count + 1.  
  self changed: #count ←
```

```
Counter>>initialize  
  count := 0
```

ButtonExample Updated

```
Smalltalk defineClass: #ButtonExample  
  superclass: #{UI.ApplicationModel}  
  instanceVariableNames: 'count '
```

initialize

```
count := Counter new
```

countAdapter

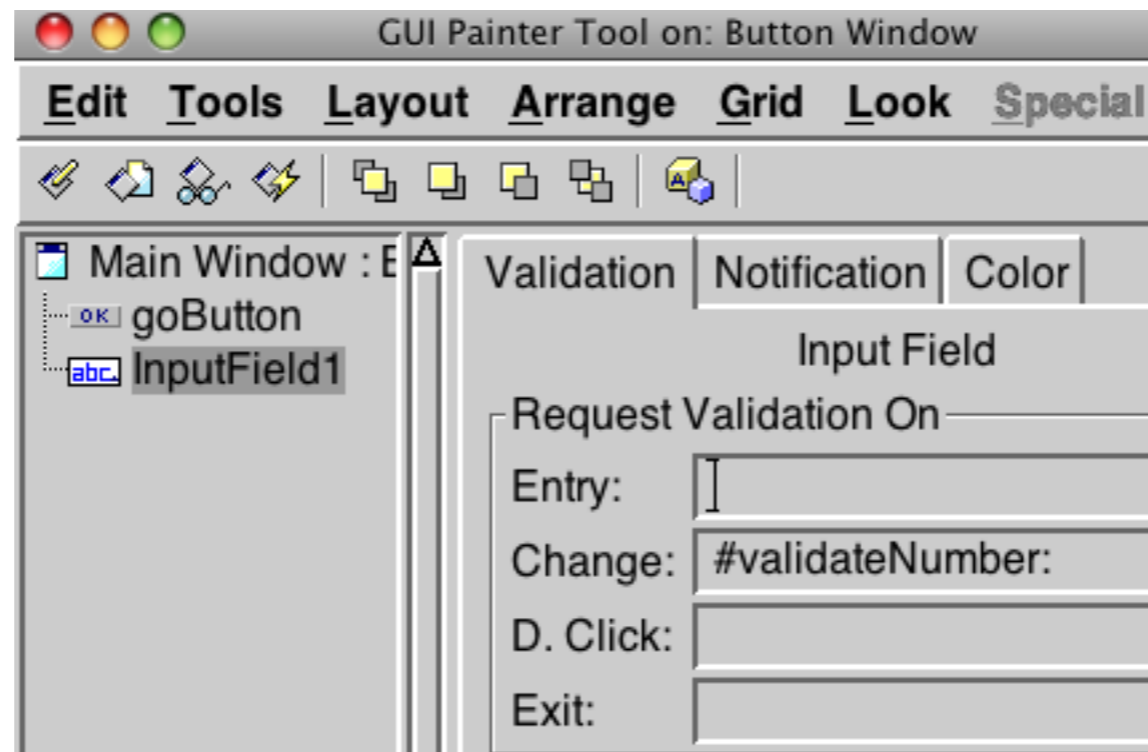
```
| countAdapter |  
countAdapter := AspectAdaptor subject: count.  
countAdapter  
  forAspect: #count;  
  subjectSendsUpdates: true.  
^countAdapter
```

go

```
count increment.  
Dialog warn: 'Time to go'.  
^self
```



Validating Input

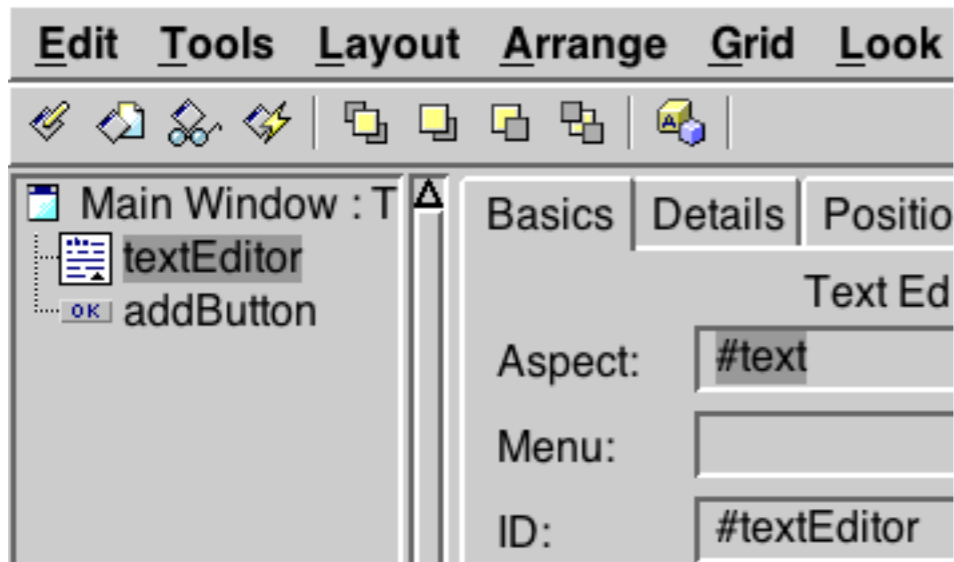


```
ButtonExample>>validateNumber: aController  
| entry |  
entry := aController editValue.  
^entry >= 0
```

Text Editor Example



Text Editor Example



```
TextExample>>text
```

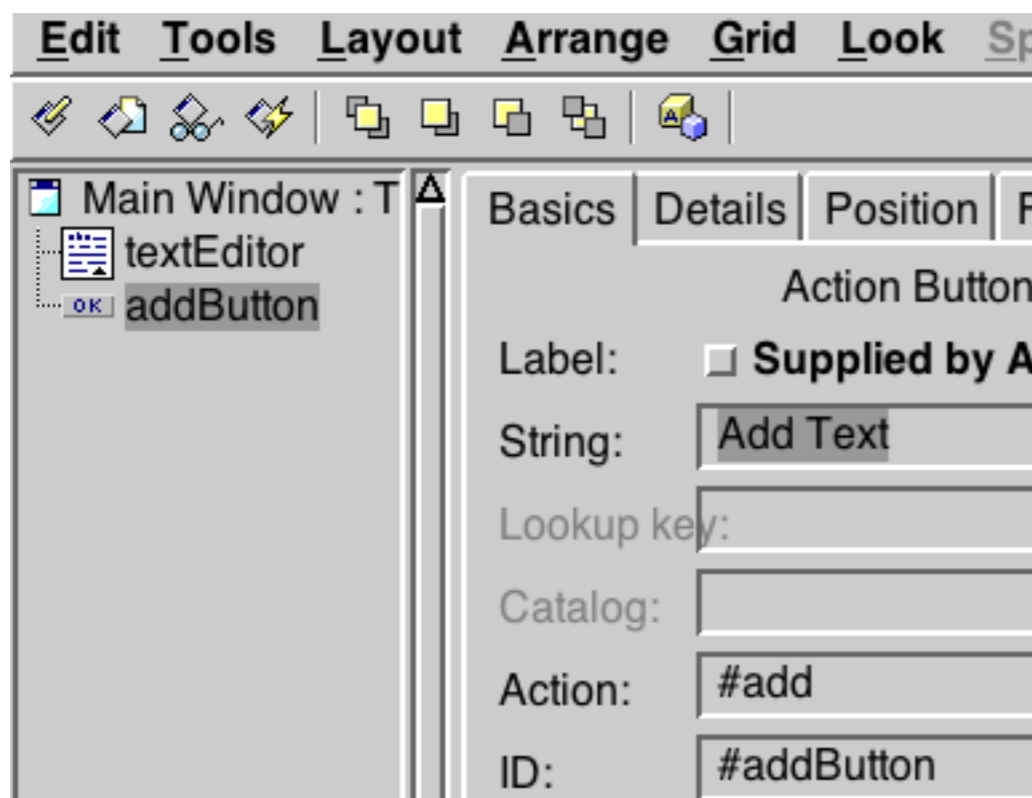
```
  ^text isNil
```

```
    ifTrue:
```

```
      [text := 'Hi Mom' asValue]
```

```
    ifFalse:
```

```
      [text]
```



```
TextExample>>add
```

```
  self text
```

```
  value: self text value , '\Add more text\' withCRs
```