CS 535 Object-Oriented Programming & Design
Fall Semester, 2001
Doc 22 List, Table & Dataset

References

GUI Developer's Guide, Chapter 9 Configuring Widgets

List, Table & Dataset
List

The list widget displays a list of items to the user, who can select one or more items from the list.

In this example we will create a window which looks like:

Example Source

```
Smalltalk.CS535 defineClass: #ListExample
   superclass: #{UI.ApplicationModel}
   indexedType: #none
   private: false
   instanceVariableNames: 'movies '
   classInstanceVariableNames: ''
   imports: ''
   category: 'Course-GUI-Examples'
```

Class Methods

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

    <resource: #canvas>
    ^#(#{UI.FullSpec}
        #window:
        #(#{UI.WindowSpec}
            #label: 'ListExample'
            #min: #(#{Core.Point} 225 267 )
            #bounds: #(#{Graphics.Rectangle} 512 384 737 651 ) )
        #component:
        #(#{UI.SpecCollection}
            #collection: #(
                #(#{UI.SequenceViewSpec}
                    #layout: #(#{Graphics.Rectangle} 33 19 180 177 )
                    #name: #List1
                    #model: #movies
                    #multipleSelections: false
                    #selectionType: #highlight )
                #(#{UI.ActionButtonSpec}
                    #layout: #(#{Graphics.Rectangle} 70 204 134 224 )
                    #name: #ActionButton1
                    #model: #addToList
                    #label: 'Add'
                    #defaultable: true ) ) ) )
```

Instance Methods

```
initialize
   super initialize.
   movies := self movies list: self movieList.
   movies selectionIndexHolder
         onChangeSend: #changedMovie
         to: self

movieList
   ^#('Harry Potter' 'Bandits' 'Amelie' 'Iris' 'I am Sam'
      'Monsters, Inc' 'Tape' ) asSortedCollection asList

changedMovie
   "Called by List widget when selection changes"
   movies selection isNil
      ifTrue:[Dialog warn: 'No movie selected']
      ifFalse: [Dialog warn:
            'You selected ' , movies selection printString]

movies
   "List widget calls this method to get list of movies"
   ^movies isNil
      ifTrue:
         [movies := SelectionInList new]
      ifFalse:
         [movies]

addToList
   "Called when Add button is pressed"
   movies list add: 'Cat in the Hat'
```

Comments and Variations

Basic Operation

The list widget interacts with either a SelectionInList or MultiSelectionInList. These contain an ordered collection of strings that is displayed in the list.

List verses Other Collections

(Multi)SelectionInList can use any ordered collection. Lists are special ordered collections for use with list widgets. When one changes the list collection the list widget is updated. So in "addToList" I just added a new string to list. If one uses another collection, one has to replace the collection in the SelectionInList object (movies list: aCollection).

Single verses Multiselection

A list can allow either single selection or multiselections. When you define your widget in the UI painter you can select which you want. In the UI painter select the list widget and go to the details pane of the GUI Painter Tool. In the details pane selecting the "Multi Select" button will turn on mult-selection.

If you are using single selection to find out the current selection you can use the methods "selection" or "selectionIndex" to get the string of the selected item or the index of the selected item. See the method "changedMovie" in the example.

If you are using multi-selection use the methods "selections" and selectionIndexes to get a collection of the selected items or indexes of the selected items.

Interacting with the List

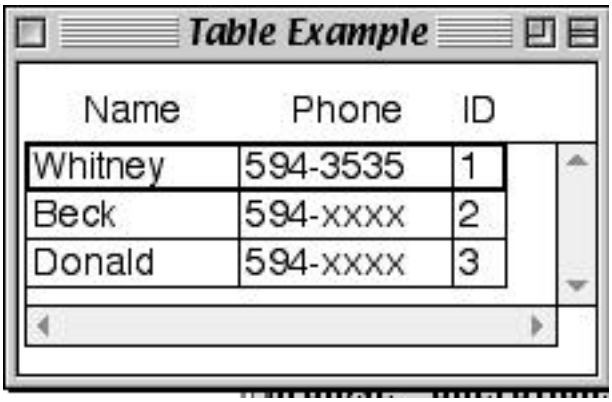
You can tell the list widget to call a method every time the user selects or deselects an item. This is done in the example in the initialize method with the code:

```
movies selectionIndexHolder
    onChangeSend: #changedMovie
    to: self
```

In many situation you do not want to know the selection until the user is done. In these cases don't send the onChangeSend:to: method to the SelectionInList object.

Tables

In this example we will display a table of data like:



Example Source

```
Smalltalk.CS535 defineClass: #TableExample
   superclass: #{UI.ApplicationModel}
   indexedType: #none
   private: false
   instanceVariableNames: ' customerTable customerTableInterface '
   classInstanceVariableNames: ''
   imports: ''
   category: 'Course-GUI-Examples'
```

Class Method

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

   <resource: #canvas>
   ^#(#{UI.FullSpec}
      #window:
      #(#{UI.WindowSpec}
         #label: 'Table Example'
         #bounds: #(#{Graphics.Rectangle} 512 384 712 584 ) )
      #component:
      #(#{UI.SpecCollection}
         #collection: #(
            #(#{UI.TableViewSpec}
               #layout: #(#{Graphics.LayoutFrame} 0 0.0 0 0.075 0 1.005 0 0.91 )
               #name: #Table1
               #model: #tableData
               #showHGrid: true
               #showVGrid: true
               #selectionStyle: #row ) ) ) )
```

Instance Methods

```
tableData
   "Called by table widget to get value holder for the table"
   ^customerTableInterface

initialize
   | list |
   super initialize.
   list := TwoDList
           on: #('Whitney' '594-3535' 1 'Beck' '594-xxxx' 2
                   'Donald' '594-xxxx' 3 )
           columns: 3
           rows: 3.
   customerTable := SelectionInTable with: list.
   customerTableInterface :=
        TableInterface new selectionInTable: customerTable.
   customerTableInterface
      columnWidths: #(80 80 20);
      columnLabelsArray: #('Name' 'Phone' 'ID')
```

Comments

TwoDList
Provides two dimensional indexing to an ordered collection
Acts like a table or ordered collection

```
list := TwoDList
      on: #( 'a' 'b' 'c' 'd' 'e' 'f')
      columns: 3
      rows: 2.
row := 2.
column := 1.
list at: column @ row.     "Returns 'd' "
list at: 4                          "Returns 'd' "
```

SelectionInTable
A value holder for a table

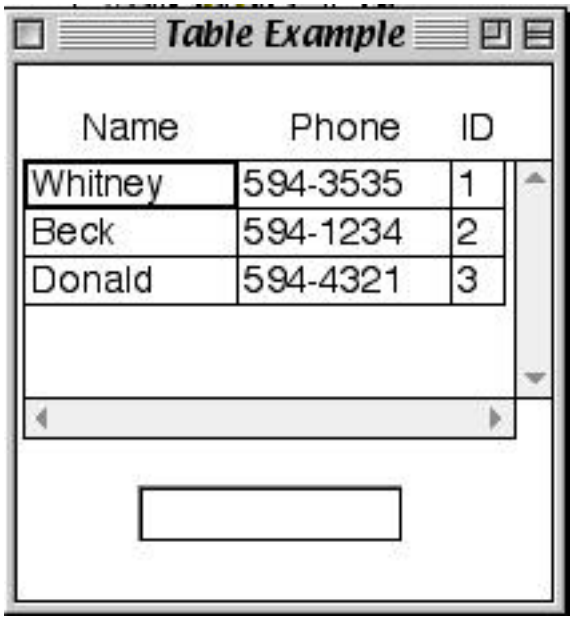
TableInterface
Contains a value holder for a table
Know how to format columns and rows

This example does display the table, but there is now way to edit the table. We will correct that in the next example.

Table with Editing

This example modifies the first table example to:

- Add an input field to allow user to edit the table
- Some minor refactoring of the code



Example Source

```
Smalltalk.CS535 defineClass: #TableExample
   superclass: #{UI.ApplicationModel}
   indexedType: #none
   private: false
   instanceVariableNames: 'customerTableInterface cellContents '
   classInstanceVariableNames: ''
   imports: ''
   category: 'Course-GUI-Examples'

windowSpec
   "UIPainter new openOnClass: self andSelector: #windowSpec"

   <resource: #canvas>
   ^#(#{UI.FullSpec}
      #window:
      #(#{UI.WindowSpec}
         #label: 'Table Example'
         #bounds: #(#{Graphics.Rectangle} 470 232 670 432 ) )
      #component:
      #(#{UI.SpecCollection}
         #collection: #(
            #(#{UI.TableViewSpec}
               #layout: #(#{Graphics.LayoutFrame} 0 0 0 0.075 0 1.01 0 0.7 )
               #name: #Table1
               #model: #tableData
               #showHGrid: true
               #showVGrid: true )
            #(#{UI.InputFieldSpec}
               #layout: #(#{Graphics.Rectangle} 45 157 145 179 )
               #name: #InputField1
               #model: #cellContents ) ) ) )
```

Instance Methods

```
cellContents
   "Called by inputfield for string value holder"
   ^cellContents

changedCell
   "Called by input field when it changes"
   | cellLocation |
   cellLocation := customerTableInterface selectionIndex.
   cellLocation = Point zero
      ifFalse:
         [customerTableInterface table
               at: cellLocation
               put: self cellContents value]

tableData
   "Called by table widget to get table interface"

   ^customerTableInterface

initialize
   super initialize.
   self initializeTable.
   self initializeInputCell

initializeInputCell
   cellContents := String new asValue.
   cellContents onChangeSend: #changedCell to: self
```

```
initializeTable
   | list customerTable |
   list := self customerTable.
   customerTable := SelectionInTable with: list.
   customerTableInterface := TableInterface new
            selectionInTable: customerTable.
   customerTableInterface
      columnWidths: #(80 80 20);
      columnLabelsArray: #('Name' 'Phone' 'ID')

customerTable
   ^TwoDList
      on: self sampleData
      columns: 3
      rows: 3.

sampleData
   ^#('Whitney' '594-3535' 1 'Beck' '594-1234' 2 'Donald' '594-4321'
3)
```

Comments

In the method:

```
initializeInputCell
   cellContents := String new asValue.
   cellContents onChangeSend: #changedCell to: self
```

we tell the value holder cellContents to send the message #changedCell to the TableExample object whenever the value holder changes.

This is a more flexible version of Java's listeners.

We can specify the object and method to send to the listener in Smalltalk.

Table on Customer Objects

The previous example creates a table on collection of strings.

If we have a collection of customer objects it is a pain to have to convert them to the collection of strings

We can use the TableAdaptor to do this for us.

We need to change two methods in the previous example to use collection of customer objects

```
sampleData
   ^OrderedCollection new
      add: (Customer name: 'Whitney' phone: '594-3535');
      add: (Customer name: 'Beck' phone: '594-1234');
      add: (Customer name: 'Donald' phone: '594-4321');
      yourself
```

sampleData just returns a collection of customers.

```
customerTable
   | rowMap |
   rowMap := RowAdaptor adaptForAspects: #( #name #phone #id).
   ^TableAdaptor
      on: self sampleData
      adaptors: rowMap
```

Comments

TableAdaptor

The TableAdaptor acts on a collection of objects. To the GUI widget it acts like a value holder on a two dimensional collection of strings.

To act like a two-dimensional collection of strings it needs to know how to get and set the correct values from the objects.

It uses a RowAdaptor to do this. The code:

```
RowAdaptor adaptForAspects: #( #name #phone #id).
```

Creates a RowAdaptor that uses:

- name and name: to get, set the value of the first column
- phone and phone: to get, set the value of the second column
- id and id: to get, set the value of the third column

Datasets

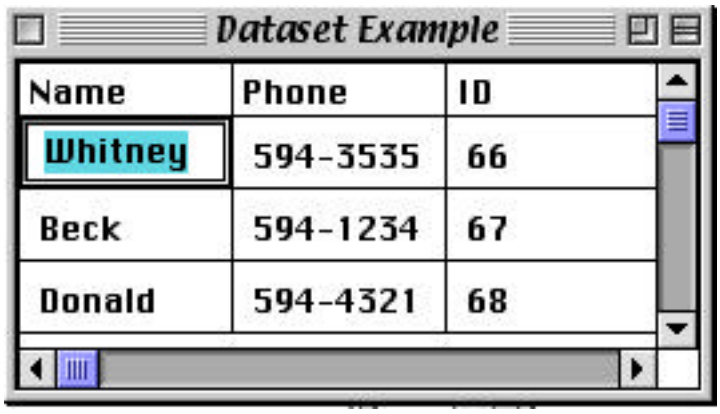
A Dataset is an improved version of a Table. It allows the user to edit items directly and change the width of columns. A Dataset operates on a collection of objects like Customer objects.

Before you can use the Dataset widget you need to download a patch. The patch corrects a bug when one edits values in a Dataset.

The patch is at:
http://www.cincomsmalltalk.com:8080/CincomSmalltalkWiki/VW+5i.4+Patches

Download patch AR43067 (file ar43067.st) and filein the file.

We will create a window that looks like:



Example Code

```
Smalltalk.CS535 defineClass: #DatasetExample
   superclass: #{UI.ApplicationModel}
   indexedType: #none
   private: false
   instanceVariableNames: 'selectedCustomer datasetData '
   classInstanceVariableNames: ''
   imports: ''
   category: 'Course-GUI-Examples'
```

Class Methods

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

    <resource: #canvas>
    ^#(#{UI.FullSpec}
        #window:
        #(#{UI.WindowSpec}
            #label: 'Dataset Example'
            #bounds: #(#{Graphics.Rectangle} 556 370 816 566 ) )
        #component:
        #(#{UI.SpecCollection}
            #collection: #(
                #(#{UI.DataSetSpec}
                    #layout: #(#{Graphics.LayoutFrame} 0 0.0269231 0 0.0918367 0 0.969231 0 0.668367 )
                    #name: #Dataset1
                    #model: #datasetData
                    #columns: #(
                        #(#{UI.DataSetColumnSpec}
                            #model: #'selectedCustomer name'
                            #label: 'Name'
                            #labelIsImage: false
                            #width: 80
                            #editorType: #InputField
                            #noScroll: false )
                        #(#{UI.DataSetColumnSpec}
                            #model: #'selectedCustomer phone'
                            #label: 'Phone'
                            #labelIsImage: false
                            #width: 80
                            #editorType: #InputField
                            #noScroll: true )
                        #(#{UI.DataSetColumnSpec}
                            #model: #'selectedCustomer id'
                            #label: 'ID'
                            #labelIsImage: false
                            #width: 80
                            #rendererType: #Text
                            #editorType: #None
                            #type: #number
                            #noScroll: false
                            #formatString: '0' ) ) )
                #(#{UI.ActionButtonSpec}
                    #layout: #(#{Graphics.LayoutFrame} 0 0.315385 0 0.765306 0 0.561538 0 0.867347 )
                    #name: #ActionButton1
                    #model: #inspect
                    #label: 'Inspect'
                    #defaultable: true ) ) ) )
```

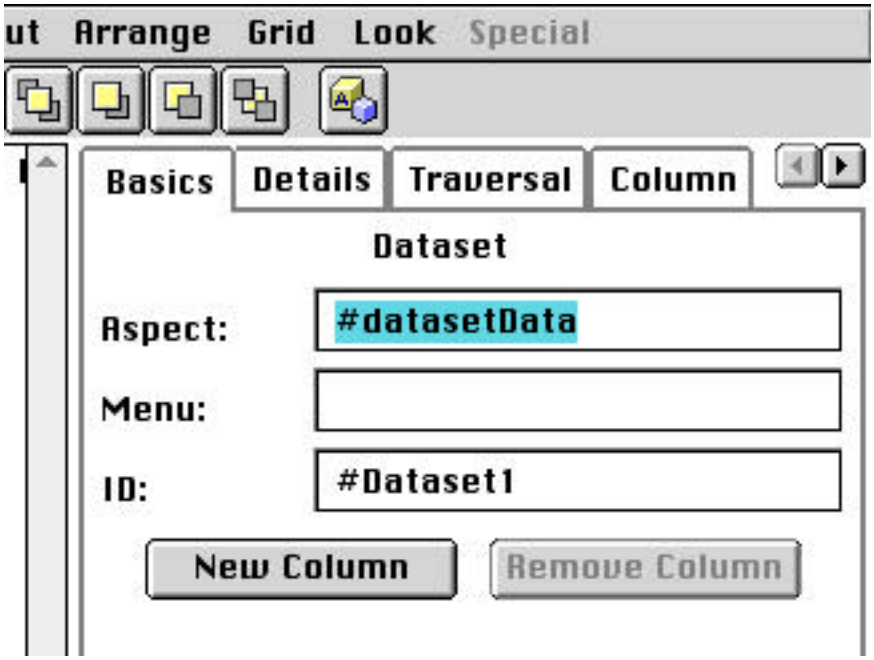
Instance Methods

```
datasetData
   "Generated by UIPainter"
   ^datasetData isNil
      ifTrue:
         [datasetData :=  SelectionInList new.
         datasetData selectionIndexHolder compute:
            [:v |
            self selectedCustomer value: datasetData selection].
         datasetData]
      ifFalse:
         [datasetData]

selectedCustomer
   "Generated by UIPainter"
   ^selectedCustomer isNil
      ifTrue:
         [selectedCustomer := nil asValue]
      ifFalse:
         [selectedCustomer]

initialize
   super initialize.
   self datasetData list: self sampleData

sampleData
   ^List new
      add: (Customer name: 'Whitney' phone: '594-3535');
      add: (Customer name: 'Beck' phone: '594-1234');
      add: (Customer name: 'Donald' phone: '594-4321');
      yourself
```

Comments

As you can see there is very little code. Only the last two methods of the example were written by hand. Most of the work is in setting the values in the GUI Painter tool.

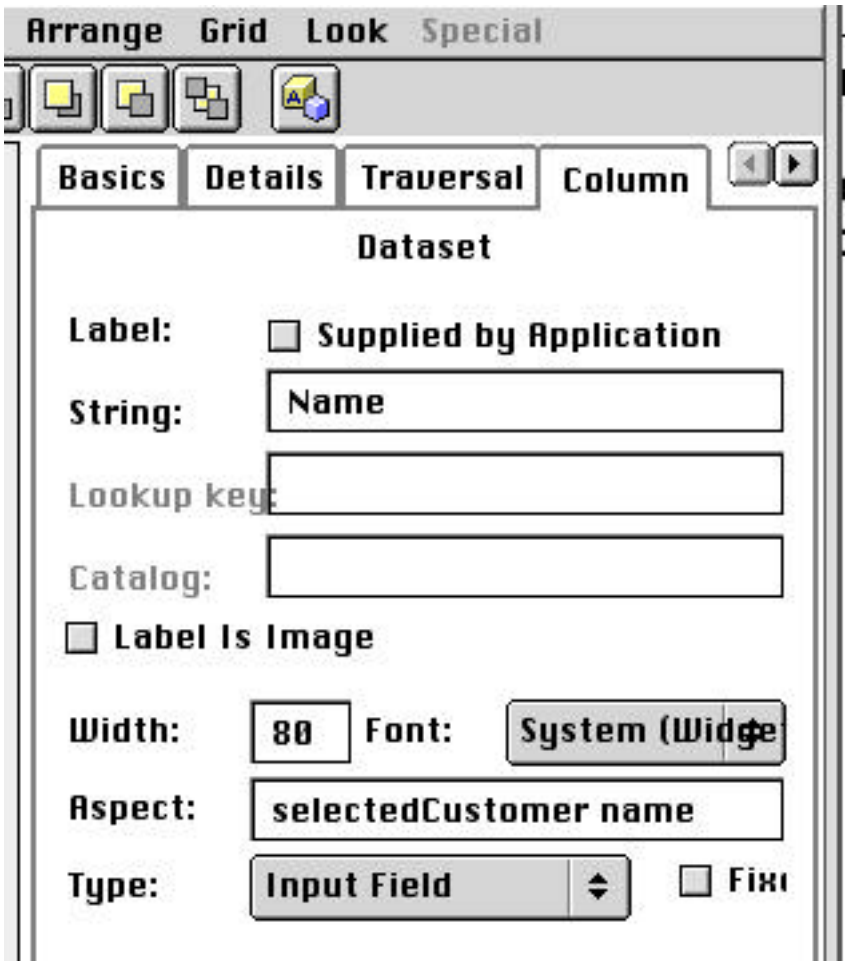
When you add a Dataset widget to the canvas the GUI Painter tool looks like:



The aspect is the name of the method called by the Dataset widget to get the values to display.

Click on the "New Column" button to create as many columns as you need.

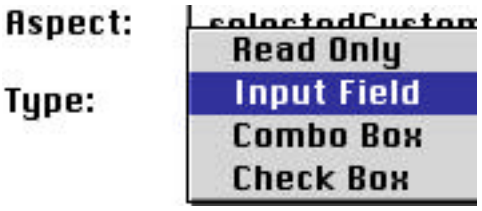
You need to configure each column. To do this you must first select a column by <command>-clicking on the column in the canvas. In the GUI Painter view click on the column tab. You will see:



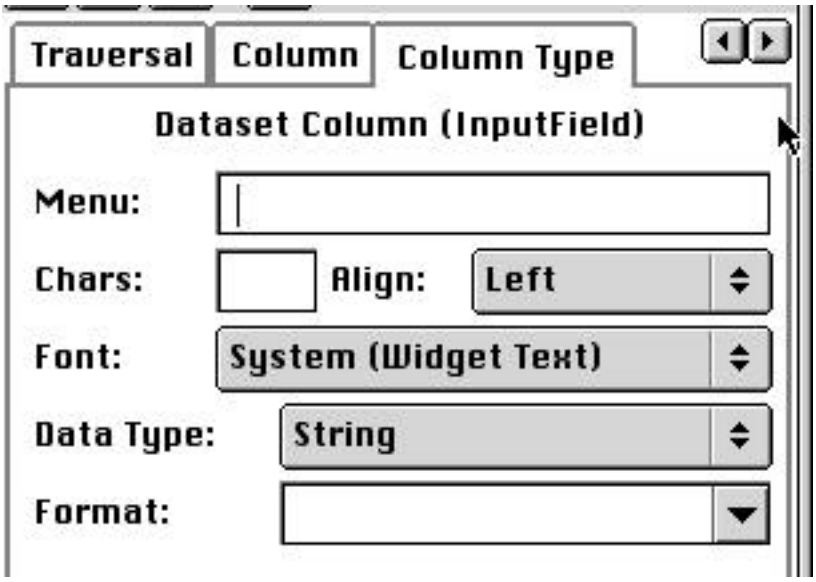
The string is the label used for the column. Leave it blank if you do not want column labels.

The aspect is how the widget will get the value for the cells in this column. Notice the aspect has two methods in it. The first one, selectedCustomer, is used to get the customer for a row. The method will be generated for you the UIPainter. Make sure you use the same method for each column. The second method, name, is the one sent to the customer object to get/set the value for the cells in this column.

The type allows you select between Read only, Input Field, Combo Box, and Check Box.



In the GUI Painter view click on the column type tab. You may have to make the window wider or click on arrow buttons on the upper right to make the column type tab visible. The GUI Painter tool will look like:



You need to set the data type so that editing will work correctly.