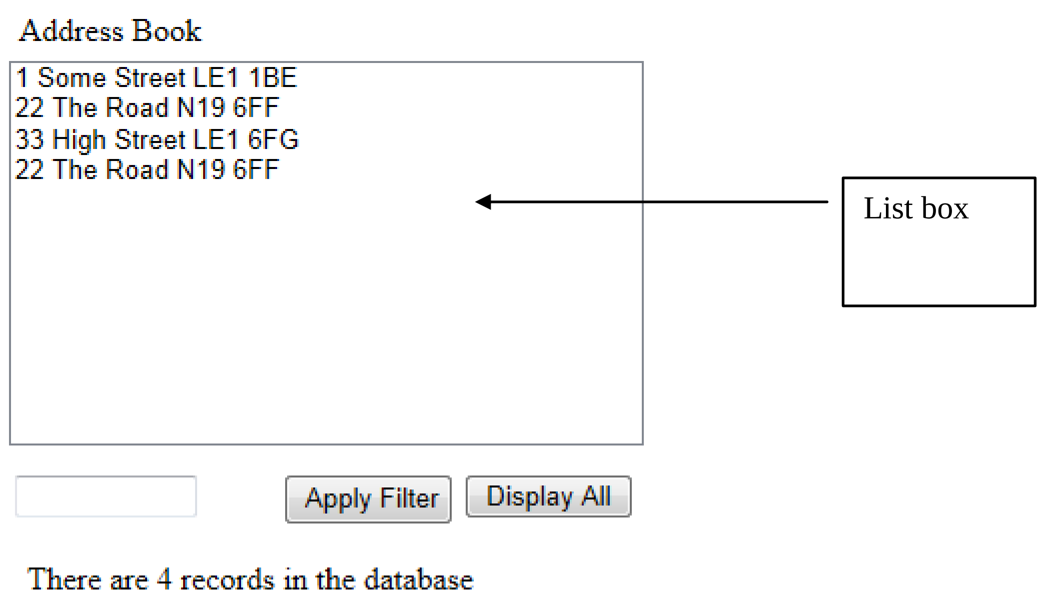
List Boxes

In this session, the plan is to cover the theory behind list boxes and drop down lists, how they work and how you may manipulate the items contained in the list.

This part of the work uses a list box to display the records in the underlying table...

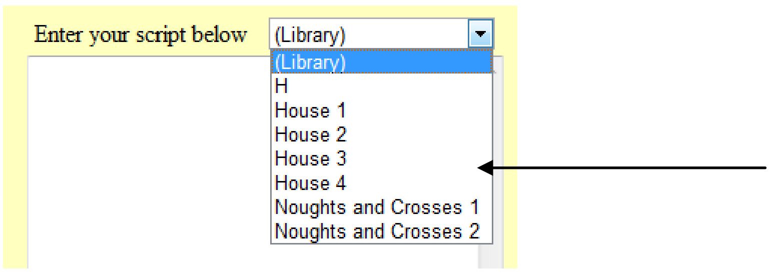


Address Book

1 Some Street LE1 1BE
22 The Road N19 6FF
33 High Street LE1 6FG
22 The Road N19 6FF

There are 4 records in the database

There is also an example of a drop-down list in GPE...



The naming conventions for these controls are...

List boxes	lst (El Es Tee – not one s t)
Drop Down Lists	cbo (for combo box – old name)
Or	ddl (whichever you use be consistent)

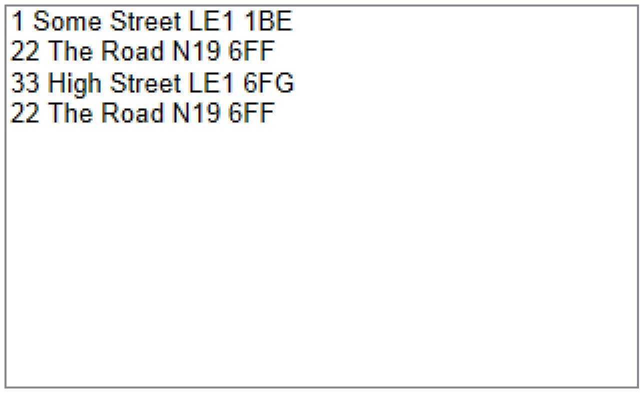
In the following notes, we shall discuss list boxes. However, all the techniques for drop down lists operate in the same way.

From a programming point of view there are several things that we want to do with a list box.

1. Add an item to the list
    a. Set the list entry so it makes sense to the user
    b. Set the list entry so it makes sense to the system
2. Remove an item from the list
3. Read the value of the item selected by the user, i.e. find out what they clicked on
4. Clear the list box
5. Validate a list to make sure that an item has been selected

The Items Collection

The items in a list box are referred to as the “items collection”.



```
1 Some Street LE1 1BE
22 The Road N19 6FF
33 High Street LE1 6FG
22 The Road N19 6FF
```

Here we can see four items listed in this list’s “items collection”.

List Box Properties

The list box has several properties that we need to understand.

SelectedItem.Text	This allows us to obtain the text of the item in the list that has been selected by the user.
SelectedValue	Contains the primary key value of the selected item in the list. (If you do not set this yourself it is will be the same as SelectedItem.Text)
SelectedIndex	This tells us the index of the selected item in the list. (If this is -1 it means that no selection has been made.)
Index	Each item in a list box has an index. The first item has an index of zero, the second one and so on.

Count	The items have a count property that tells us how many items are in the list.
Value	Allows us to get or set the system ID of an item in the list.

Important List Box Methods

Add	This method allows us to add an item to a list box.
RemoveAt	Allows us to remove an item from a list box.
Clear	Clears the items collection of all entries.

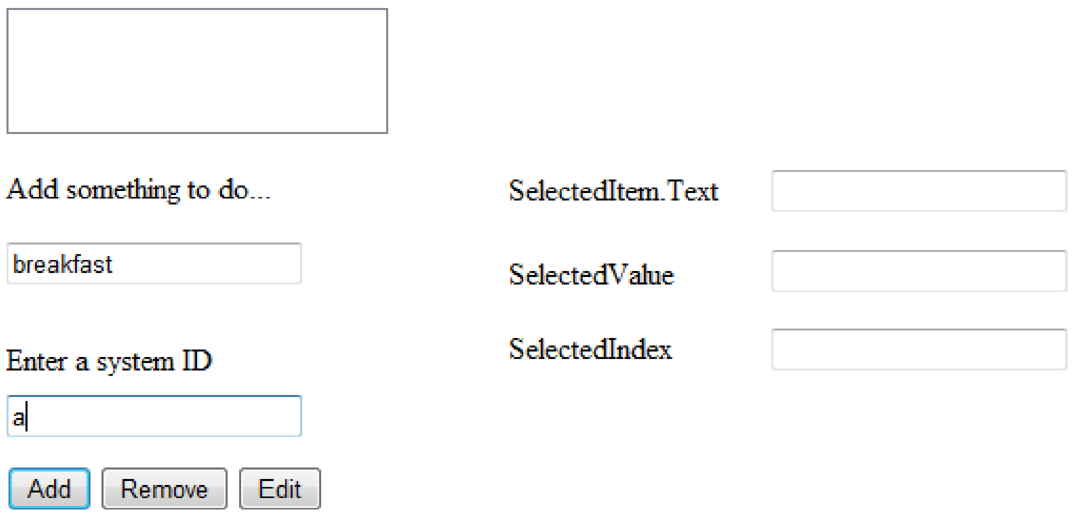
Adding Items to a List Box

To get to grips with how a list box works, we will look at the to-do list program provided on the module web site.

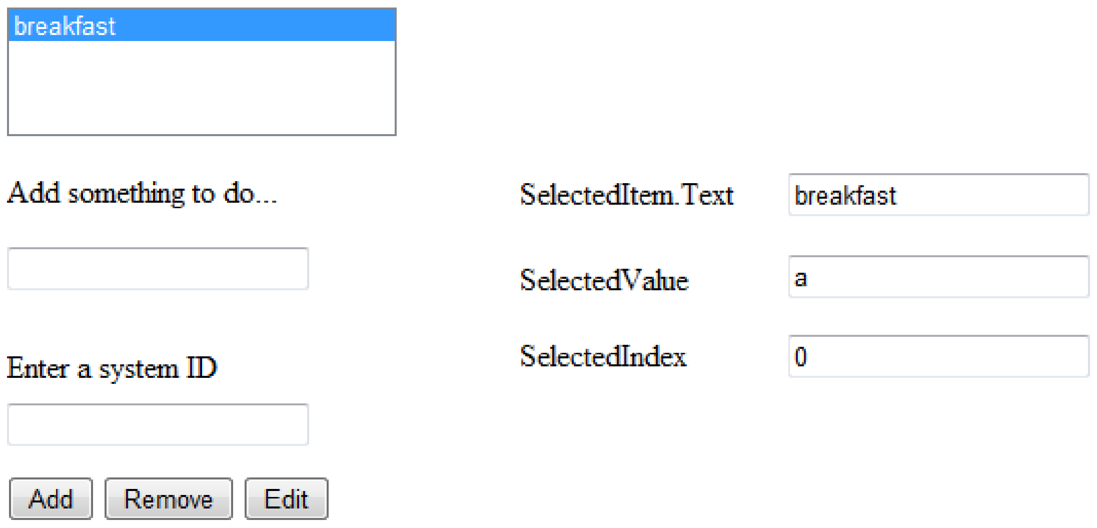
When a list box is created, it has no items listed. If you wish to display an item you need to add an item to the items collection.

In the following example, we will type two values.

Breakfast	this is the value that will appear to the user
a	this is the identifier for this entry



Once the item is added to the list we can click on it to see the values of the three properties...


breakfast

Add something to do...

Enter a system ID

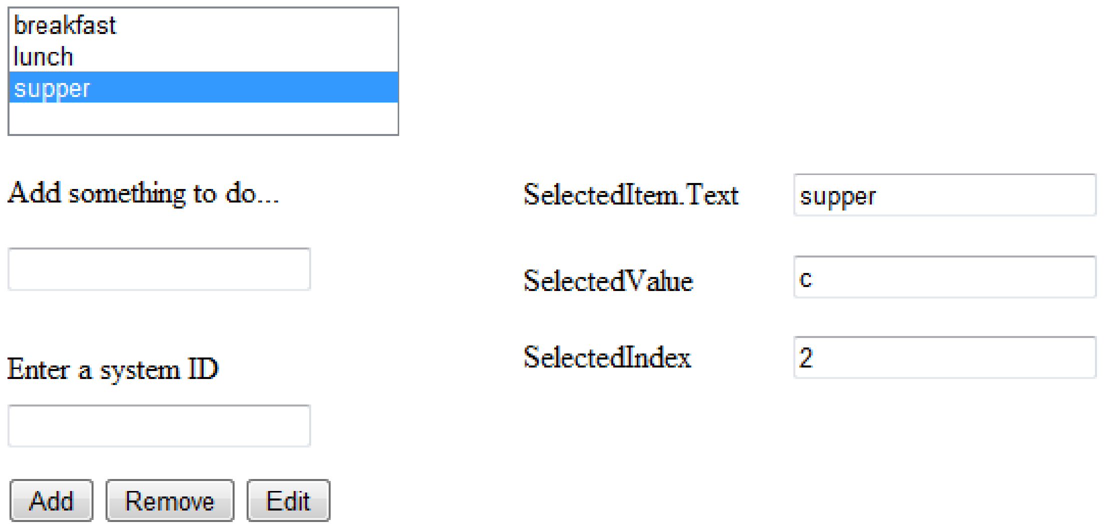
SelectedItem.Text breakfast

SelectedValue a

SelectedIndex 0

Next add two more items.

Lunch	b
Supper	c


breakfast
lunch
supper

Add something to do...

Enter a system ID

SelectedItem.Text supper

SelectedValue c

SelectedIndex 2

Clicking on each item we can see the settings for the three properties.

SelectedItem.Text as we said above is the "user friendly" text for the entry in the list.

The SelectedValue is the system's own description of the item. We can set this to anything we like. If we do not set it then it is the same as SelectedItem.Text

The SelectedIndex tells us the number of the item in the list. Notice that this starts counting from zero.

Adding Items

To make an item appear in the list we need to use the Add method of the items collection.

The following is a simple example of this...

```
//var to store the description
string ItemDescription;
//get the description entered by the user
ItemDescription = txtToDo.Text;
//add the new item to the list
lstToDo.Items.Add(ItemDescription);
```

If all we want to do is quickly add an entry to the list then the above code would work fine.

This code is not so good if we want to also include a primary key value for the item.

Setting the Item's Value via the List Item Class

Much list box code uses the above approach. However, when we start linking our list boxes to a database we need to know how to set two properties of each item.

1. The text (that the user sees)
2. The value (that the system uses)

The ListItem class gives us a way of doing this.

Look at the following code...

```
//var to store the description
string ItemDescription;
//var to store the system ID of the item
string PrimaryKeyValue;
//get the item's system ID entered by the user
PrimaryKeyValue = txtPrimaryKey.Text;
//get the description entered by the user
ItemDescription = txtToDo.Text;
//create a new instance of ListItem
ListItem NewEntry = new ListItem(ItemDescription, PrimaryKeyValue);
//add the new item to the list
lstToDo.Items.Add(NewEntry);
//clear the text boxes
txtToDo.Text = "";
txtPrimaryKey.Text = "";
```

Now when we access the SelectedValue property for these entries the primary key value will be returned.

Identifying what the User Selects

Once we have the list populated with data, we then want to know which item the user has selected.

When the user clicks on the list box the SelectedValue property is set.

The SelectedValue property contains the identifier of the item (the value) that the user has selected.

If we do not explicitly set the value of an item ourselves (as shown above), the SelectedValue contains the text of the entry in the list.

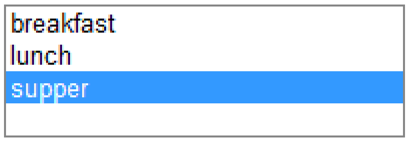
If we change the value of each item it will contain whatever it is we set it to.

Deleting Items on the List

There will be times when the user clicks an item on the list and wishes for that item to be removed.

When an item is added to the list it is given an index by the list box.

So in the following example…



Breakfast	Index 0
Lunch	Index 1
Supper	Index 2

We can obtain the numeric value of each item clicked by referencing the SelectedIndex property.

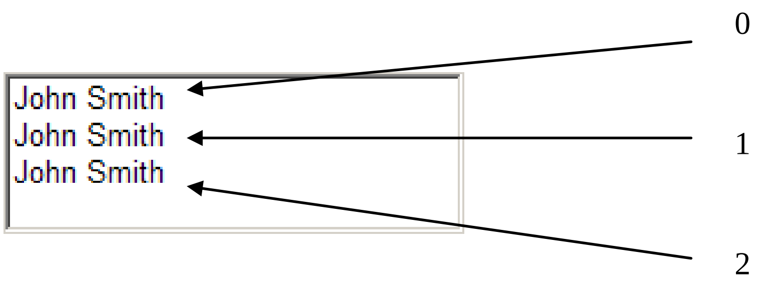
So if the user clicks on "breakfast" the SelectedIndex will be 0.

We need to know the number given to an item if we want to delete an entry to remove ambiguity.

For example, consider a list contained people's names.

If there were 3 John Smiths in the list we run the risk of deleting them all if we sent a command to the list…

"Delete all entries called John Smith"



Since each entry has a unique number, we can use this to specify…

"Delete only John Smith number 2."

So, the first thing we need to do is identify the selected index of whatever the user clicked on.

Once we know which item the user has selected we can now get on with deleting the entry.

To remove an item from the Items Collection we use the RemoveAt method, stating which number item in the collection we want to get rid of.

```
//declare a variable to store the selected index
Int32 AnIndex;
//obtain the selected index from the list
AnIndex = lstToDo.SelectedIndex;
//remove the item identified by the selected index
lstToDo.Items.RemoveAt(AnIndex);
```

Ensuring that a User Has Selected an Item

One problem with deleting items on a list occurs when a user presses delete without first selecting an item on the list.

If no item is selected on the list, the selected index property is set to -1.

We may make use of this in our code as follows.

```
//declare a variable to store the selected index
Int32 AnIndex;
//obtain the selected index from the list
AnIndex = lstToDo.SelectedIndex;
//if the user has made a selection
if (AnIndex != -1)
{
     //remove the item identified by the selected index
     lstToDo.Items.RemoveAt(AnIndex);
}
else
{
     //if no selection made show an error message
     lblError.Text = "You must select an item on the list to remove it";
}
```

Clearing the List Box

Should you wish to clear the entire contents of the list box, you would use the Clear method.

For example, the following code would clear a list box called lstListExample.

```
lstListExample.Items.Clear;
```