# Lesson 18 – Complex Data Types, Files & Views

* Type declaration
* Arrays of Type
* List of Type
* Map of Type
* KVS Files
* CustomListView
* B4XComboBox

What students should know

**4**

Often a developer wants to group variables related to each other. For example, a student has a First Name, Last Name, Address, Phone, etc. One can of course create separate variables for each different element of the student but it is more convenient to create a new data type that contains all these values together.

## The type statements

The described data type is called **type** and is stated as follows:

*Type Student(LastName As String, FirstName As String, Address As String, PhoneNumber As String)*

Notice that the keyword “Type” is first written, then a name is written for the new data type record that is created, and finally in parentheses all variables/elements included in the new type. Type declarations are always written in Class\_Globals and it is Public by default.

In the student's example, the data Type statement looks like this:

**Sub** Class\_Globals

**Type** Student(LastName  **As String**, FirstName **As String**, \_

Address **As String**, PhoneNumber  **As String**)

**End Sub**

A new data type has now been created called Student and you can declare new variables based on this.

**Sub** Class\_Globals

**Type** Student(LastName  **As String**, FirstName **As String**, \_

Address **As String**, PhoneNumber  **As String**)

**Public** Student1 **As Student**

**End Sub**

The “*Student1*” variable is now a Student-type variable, and to access its data you use the variable name with one period and then the name of the items included in the type statement e.g.

Student1.LastName = “Ioannidis”

Student1.FirstName = “Alkinoos”

Student1.Address = “Athens, Greece”

Student1.PhoneNumber = "+303465854234”

You can create as many variables of the Student type as you want and assign each other.

Private Student2 as Student

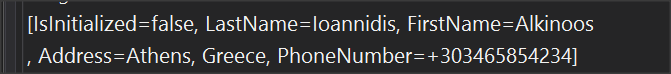
Student2 = Student1

Here the Student2 variable now contains exactly the same data as the Student1 variable.

### Show Items

You can use the Log() command to display the contents of Student1, but the result will be something like the following:

Log(Student1)



To display or use all elements of a type, use the entire name along with the item.

Log(Student1. LastName & “ “ & Student1.FirstName)

It is usually helpful to create a routine that accepts a type variable and then displays or processes the type elements.

**Private Sub** LogStudent(st  **As** Student)

**Log**(st.FirstName)

**Log**(st.LastName)

**Log**(st.Address)

**Log**(st.PhoneNumber)

**End Sub**

## Array of a type

You can also create arraysvof Type where it will include more students. For example,

**Sub** Class\_Globals

**Type** Student(LastName  **As String**, FirstName **As String**, \_

Address **As String**, PhoneNumber  **As String**)

**Public** Students(10)  **As Student**

**End Sub**

The Students(10) statement creates an array of 10 Students. Each item can be used with a for loop or with a separate reference to each with its index.

Students(0). LastName = “Paul”

Students(0). FirstName = “Belmond”

## List of Type

A list can contain variables of Type.

**Sub** Class\_Globals

**Type** Student(LastName  **As String**, FirstName **As String**, \_

Address **As String**, PhoneNumber  **As String**)

**Public** listStudents  **As List**

**End Sub**

**Private Sub B4XPage\_Created (Root1 As B4XView)**

**...**

**...**

**listStudents.Initialize**

**listStudents.Add(Student1) ‘ Add Student to list**

**‘Get list items and log**

**For i = 0 To** listStudents**.Size-1**

**Private** st  **As Student**

st **=** listStudents**.Get(i)**

**LogStudent(**st**)**

**Next**

**End Sub**

**Private Sub** LogStudent**(**st  **As Student)**

**Log**(st.FirstName)

**Log**(st.LastName)

**Log**(st.Address)

**Log**(st.PhoneNumber)

**End Sub**

## Maps of Type

Using maps to store type data is also possible as long as you have a unique key for each type variable you enter.

In the case of Students it could be for example an ID number, a registration number, an email account, etc.

**Sub** Class\_Globals

**Type** Student(id **As String**, LastName  **As String**, \_

FirstName **As String**, Address **As String**, PhoneNumber  **As String**)

**Public** mapStudents  **As**  **Map**

**End Sub**

**Private Sub B4XPage\_Created (Root1 As B4XView)**

Student1.ID = "FXA47345S3"

Student1.LastName = "Ioannidis"

Student1.FirstName = "Alkinoos"

Student1.Address = "Athens, Greece"

Student1.PhoneNumber = "+303465854234"

mapStudents**.Initialize**

mapStudents**.Put(**Student1**.** ID**,**  Student1**)**

**End Sub**

In this way it is now easy to find a student with a key ID number by calling the Get method.

## Save to KVS files

The files discussed in Chapter 17 refer to text files that often do not serve to store complex structures such as type declarations, Lists, and Maps. Another type of file supported by B4J is key value store (KVS) files.

The way they work is similar to that of Maps. All you need is a key and the structure you want to save. KVS files essentially hide a database, but the important thing is that the developer doesn't need to know any of them as long as he uses the appropriate methods.

The commands in the KVS files are as follows:

### Declare a KVS file

To declare a KVS file, you must use the KeyValueStore library to which you can add to your project from the Libraries tab.

Then create a **KeyValueStore** variable**.**

**Private** kvsFile  **As KeyValueStore**

### Initialize the KVS file

The Initialize method defines the folder in which the file will be saved and the name it will have.

File.**MakeDir**(File.**DirTemp**, "lesson18")

kvsFile.**Initialize**(File.DirTemp & "lesson18", "kvsData.dat")

**Log**(File. **DirTemp** & "lesson18")

The above commands create a folder named lesson18 within the temp folder, and a file named "kvsData.dat” and finally displays the path of the file on the Log screen.

**Remember**

The file to be created cannot be opened with another viewer (for example notepad++) but must always be used by the program you built.



### Insert items into a KVS file

The Put method is used to write the data into a KVS file. Each insertion that is **made** is called a Record. A prerequisite is that you have decided on a unique key so that you can refer to the record later. The following example inserts the Student1 with the student's ID number as a key.

kvsFile.**Put**(Student1.ID, Student1)

This way you can write any data type such as Lists, Maps, Strings, simple variables (numbers), types, and tables (only tables typed Byte or Object), as well as combinations (one list of maps for example).

Type declarations should be always be declared in the B4XMainPage module.

Maps can also be stored using the PutMapAsync method. It is also the best way to save Map elements since it enters the Map elements key as the key of the record.

kvsFile.**PutMapAsync**(mapStudents)

### Retrieving items from a KVS file

The Get method retrieves an item from a KVS file. The returned value is an object. In other words, be careful to assign the returned value to a variable of the same object type.

The following example reads the record of a Student from a KVS file.

**Student3 = kvsFile.Get("FS23534X21")**

LogStudent(Student3)

The value "FS23534X21" is the key to the record and the subroutine LogStudent has described previously retrieves a Student and displays its contents with the Log () command.

**Remember**

If the key does not exist, then there is a problem with the operation of the program. The existence of the record must be checked before you assign it to a variable. This can be done with the ContainsKey method.



You can read an entire map using the **GetMapAsync** method**.** This command accepts a list of keys as a parameter and returns a map with the keys and their corresponding values.

**Log**("Show keys")

**Private** keys **As List** = kvsFile.ListKeys

**For** i = 0 **To** keys.**Size**-1

**Log**(keys.**Get**(i))

**Next**

Finally, following the above code, GetMapAsync becomes:

**Wait** **For** (StudentFile.**GetMapAsync**(keys)) **Complete** (mapSt **As** **Map**)

To write a map in a KVS file use:

**Wait For** (StudentFile.PutMapAsync(keys)) **Complete** (Succes **As Boolean**)

### Check the existence of a record

To check the existence of a key in a KVS file, you must use the ContainsKey method.

**If** kvsFile.**ContainsKey**("FS23534X21") **Then**

Student3 = kvsFile.**Get**("FS23534X21")

**Else**

**Log**("Wrong id key")

**End If**

ContainsKey returns True after it finds the key in the file.

### Delete KVS file content

The Remove method deletes a key along with its value from a KVS file, and the DeleteAll method deletes all data.

kvsFile.Remove ("FS23534X21")

kvsFile.DeleteAll

**Remember**

It’s always a good practice to close file you don’t need any more with method.close.

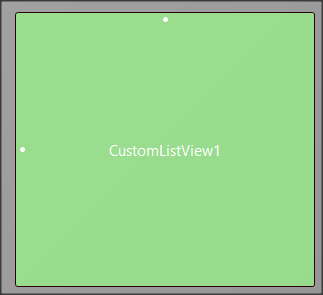
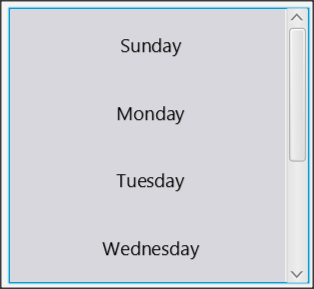


## More Views

Two important views that will help to display or select elements from complex types, lists, and maps structures are **CustomListView** and **B4XComboBox**.

### CustomListView

Creates a list of items that you can select by clicking the mouse.



The returned value is a value that you specified at the time ListView was created.

Sub **Class**\_**Globals**

Private Root As **B4XView**

Private xui As **XUI**

Private lstItems As **List**

1

Private CustomListView1 **As CustomListView**

Private lblDate As **Label**

End Sub

**Private** Sub B4XPage\_Created (Root1 As **B4XView**)

Root = Root1

Root.**LoadLayout**("MainPage")

lstItems.**Initialize**

2

lstItems.**AddAll**(**Array As String**("Sunday", "Monday", \_

"Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"))

**For** i = 0 **To** lstItems.Size-1

3

CustomListView1.AddTextItem(lstItems.**Get**(i), i)

**Next**

**End Sub**

4

**Private** Sub CustomListView1\_ItemClick(Index As Int, Value As Object)

lblDate.Text = Value

**End Sub**

1. Declare a CustomListView variable
2. Create a list structure that contains the items that will appear in CustomListView
3. Scan the entire list and each item is placed in CustomListView1 using the  **AddTextItem** method**.**

**AddTextItem** accepts and displays a value you want to display. The above example corresponds to the values from 0 to 6 for the days from Sunday to Saturday.

1. When an item in the list is clicked,the **\_ItemClick** eventtriggered and adds an index order number corresponding to**.** The example shows the index of the item that was clicked on the lblDatetag.

Additional methods of CustomListView are the following:

* **Clear** As String

Deletes all CustomListView components

* **GetValue** (Index As Int) As Object

Returns the value for the index value specified in the parenthesis.

* **Size** As Int [read only]

Returns the number of items.

#### Mark a list item

As mentioned earlier when clicking an item in a list, the event \_ItemClick is triggered. In order for this item to remain marked, the developer must either use some of the ready-made tools provided by the language (e.g. the CLVSelections library) or write his own code. The following algorithm assumes that you have set a variable in Class\_Global named selectedItem of type Int.

**Private** **Sub** CustomListView1\_ItemClick (Index **As** **Int**, Value **As** **Object**)

**If** selectedItem = -1 **Then**

**1**

**Private** p **As B4XView** = CustomListView1.**GetPanel**(Index)

p.GetView(0).**Color** = xui.**Color\_Blue**

selectedItem = Index

**Else**

**2**

**Private** p **As B4XView** = CustomListView1.**GetPanel**(selectedItem)

p.GetView(0).Color = xui.**Color\_White**

**If** selectedItem = Index **Then**

selectedItem = -1

**Else**

**3**

**Private** p **As B4XView** = CustomListView1.**GetPanel**(Index)

p.GetView(0).Color = xui.**Color\_Blue**

selectedItem = Index

**End If**

**End If**

**End Sub**

The original value of selectedItem was set to -1 where it means that nothing is selected. The code uses two commands:

**Private** p **As B4XView** = CustomListView1.**GetPanel**(Index)

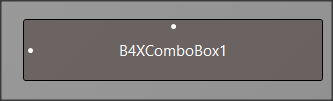
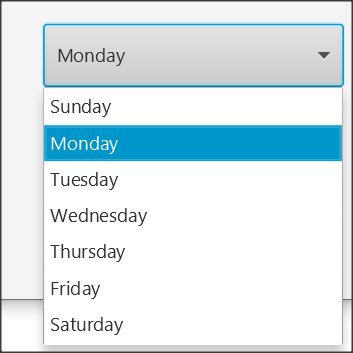
p.GetView(0).**Color** = xui.**Color\_Blue**

Each item in a list is created by the language within a box called a Panel. You can set the color by accessing the box using the GetPanel(Index) method where Index equals the current value of the line that was clicked. Then the p.GetView(0).Color defines the color the developer wants to set. Then:

1. If an item in the list is clicked, the routine checks the value of the selectedItem, and if that is -1 then sets a Blue background color in the clicked line and sets selectedItem to the value of the Index that gives the event \_ItemClick
2. If selectedItem already has a value, a white color is set as the background in the box, and then there are two cases
   1. If Clicked the already selected item, the item stops being selected and selectedItem becomes -1
   2. If Clicked another item, the new item gets blue color and selectedItem gets the value of the Index.

### B4XComboBox

B4XComboBox displays a drop-down list of items. The user can click one of them and select it. Unlike CustomListView, the returned value is always a number for the order in which the item is placed in the ComboBox, so the developer must then be able to map the value to a structure.



Sub Class\_Globals

Private Root As **B4XView**

Private xui As **XUI**

Private lstItems As **List**

1

Private B4XComboBox1 As **B4XComboBox**

Private lblCmbDate As **Label**

End Sub

Private Sub B4XPage\_Created (Root1 As **B4XView**)

Root = Root1

Root.**LoadLayout**("MainPage")

lstItems.**Initialize**

2

lstItems.**AddAll**(Array As String("Sunday", "Monday", \_

"Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"))

3

B4XComboBox1.**SetItems**(lstItems)

End Sub

4

Private Sub B4XComboBox1**\_SelectedIndexChanged**(Index As  **Int**)

lblCmbDate.**Text** = Index

End Sub

1. Declare a type **B4XComboBox** variable
2. Create a list structure that contains the items that will appear in B4XComboBox
3. Place the items in the list in B4XComboBox. **Caution** does not require iterative structure as in CustomListView
4. If an item is selected, the **\_SelectedIndexChanged** event is triggered and returns the index of the selected item.

## Exercises

1. Create a type named **Customer** and variables

ID, FirstName, LastName, Phone

1. Create a list of Customers with the following items

|  |  |  |  |
| --- | --- | --- | --- |
| **Id** | **FirstName** | **LasName** | **Phone** |
| A3473 | John | Smith | 4563454 |
| B1753 | Selim | Al Huarizmi | 6532578 |
| C6544 | Mateo | Sandor | 7345346 |
| C6323 | Lucía | Graham | 1231345 |
| F1462 | Noam | Bell | 6978323 |

1. Save the list to a KVS file
2. Build a button that when clicked fills a CustomListView with customers information.
3. Build an insert button that, when clicked, displays an appropriate DialogBox that prompts for new customer information and add them into CustomListView.
4. Save the new client to the KVS file as well.
5. When a customer is clicked display a confirmation message for deleting customer and in case of positive feedback delete the customer from CustomListView and KVS File.