Contents

Creation of New Lists

Example of creating lists 0.4 Mb

Syntax:

```
IR.CreateItem(IR.ITEM_LISTBOX, Name, Left, Top, Width, Height);
```

Parameters:

***IR.ITEM_LISTBOX (item type</i>) - List
Name (string</i>) - the name of the created list
Left (number</i>) - the indent from the left
Top (number</i>) - the indent from the top
Width (number</i>) - the item width
Height (number</i>) - the item height***

Example:

```
// Create a List type item on Page 1
IR.GetItem("Page 1").CreateItem(IR.ITEM_LISTBOX,"List 1", 0, 0, 64, 86);


// Create a List type item on Popup 1
IR.GetItem("Popup 1").CreateItem(IR.ITEM_LISTBOX,"List 1", 0, 0, 64, 86);
```

Creation of a List type item on the opened page:

The **IR** object can be used as a link to the opened page. The **IR** object receives the link to the page when the page is opening.

The page opening occurs:

- When launching the app: the event **IR.EVENT_START**, the start page is opening;

```
// The event of page opening
IR.AddListener(IR.EVENT_START, 0, function(){
  // Create a List type item on the opened page
 IR.CreateItem(IR.ITEM_LISTBOX, "List 1", 32, 16, 128, 512);
}
```

- When using the method **IR.ShowPage**;

```
// Open Page 1
IR.ShowPage("Page 1");
// Create a List type item on the opened page
IR.CreateItem(IR.ITEM_LISTBOX, "List 1", 32, 16, 128, 512);
```

Receiving Access

Example of receiving access, 0.2 Mb

To use List methods and properties, it is required to receive the link to the List.

The link to Lists can be received:

- when creating a new List type item

```
// Create a List type item on Page 1
// and save the link to the item in the variable List_1
var List_1 = IR.GetItem("Page 1").CreateItem(IR.ITEM_LISTBOX, "List 1", 32,
16, 128, 512);
```

- after creating a List type item with the help of **iRidium Script** using the method **IR.GetItem**

```
// Create a List type item on Page 1
IR.GetItem("Page 1").CreateItem(IR.ITEM_LISTBOX, "List 1", 32, 16, 128, 512);
// receive the link to the item and save it in the variable List_1
var List_1 = IR.GetItem("Page 1").GetItem("List 1");
```

- after creating a List type item with the help of **iRidium GUI Editor** using the method **IR.GetItem**

```
// receive the link to the item created in GUI Editor and save it in the
variable List_1
var List_1 = IR.GetItem("Page 1").GetItem("List 1");
```

↑ Back

Methods

Methods used when working with Lists

CreateItem

Example of adding new items, 0.4 Mb

It adds new items to Lists. To use this method it is required to receive access to the list.

Syntax:

```
List_1.CreateItem(Item, SubItem, {Property: Value, ..});
```

Parameters:

**List_1 - the link to the list;
Item - the number of the item beginning with 0;
SubItem – the number of subitem on the item beginning with 1; 0 – the List item;
Property - the properties of subitem separated by commas, the order is not important;
Value   (number, string) – the value for the property.**

Example:

```
// Add a new item with number 0 to List_1
// write "Hello List!" in the Text property of the subitem number 1
List_1.CreateItem(0, 1, {Text: "Hello List!"});
```

↑ Back

DeleteItem

Example of deleting items 0.4 Mb

It deletes items from Lists. To use this method it is required to receive access to the list.

Syntax:

```
List_1.DeleteItem(Item);
```

Parameters:

**List_1 – the link to the list;
Item - the number of the item beginning with 0.**

Example:

```
// Delete the item number 0 from List_1
List_1.DeleteItem(0);
```

↑ Back

Clear

Example of deleting all items, 0.4 Mb

It clears the list – deletes all items. To use this method it is required to receive access to the list.

Syntax:

```
List_1.Clear();
```

Example:

```
// Clear all items from List_1
List_1.Clear();
```

↑ Back

SetPosition

Example of scrolling lists, 0.4 Mb

It scrolls the list to the indicated item. To use this method it is required to receive access to the list.

Syntax:

```
List_1.SetPosition(Item);
```

Parameters:

**List_1 - the link to the list;
Item - the number of the item beginning with 0.**

Example:

```
// Scroll List_1 to item 2
List_1.SetPosition(2);
```

Details:

For **successful** scrolling of the list to the indicated item:

- the item has to **exist**
- when executing the command **SetPosition**, the list has to be **displayed**. It means the page or popup with the list has to be **on the screen**.
- there have to be at least **100 ms** after creating a new list item. The command of scrolling will not be executed right after the command of creating a new list item.

↑ Back

Properties

Properties of Lists

Template

Example of reading /writing templates of Lists, 0.4 Mb

Items of Lists are created using a template. The template is a popup. The property is available for writing and reading. To use this property it is required to receive access to the list.

Syntax:

```
List_1.Template = PopupName;
```

Parameters:

**List_1 - the link to the list;
PopupName (string) - the popup name**

On the output:

```
 Popup name (string).
```

Example of writing:

```
// Set the template with the name "Popup 1" for List_1
List_1.Template = "Popup 1";
```

Example of reading:

```
// Read the template of List_1 and output the result in the debugging console
iRidium Log
IR.Log(List_1.Template); // in the console you will see, for example "Popup
1"
```

↑ Back

Direction

Example of reading /writing Lists orientation, 0.4 Mb

Lists orientations. There are two orientations – vertical and horizontal. The property is available for writing and reading. To use this property it is required to receive access to the list.

Syntax:

```
List_1.Direction = Direction;
```

Parameters:

**List_1 - the link to the list;
Direction (number) - orientation, 0 - vertical, 1 – horizontal.**

On the output:

```
 Orientation (number), 0 - vertical, 1 – horizontal.
```

Example of writing:

```
// Set the horizontal orientation for List_1
List_1.Direction = 1;
```

Example of reading:

```
// Read the orientation of List_1 and output the result the debugging console
```

```
iRidium Log
IR.Log(List_1.Direction); // in the console you will see, for example 1
```

↑ Back

ItemsCount

Example of reading the number of items in Lists, 0.4 Mb

The number of items in Lists. The property is available for reading. To use this property it is required to receive access to the list.

Syntax:

```
List_1.ItemsCount;
```

On the output:

```
 The number of items in Lists.
```

Example of reading:

```
// Count the number of items in List_1 and output the result the debugging
console
IR.Log(List_1.ItemsCount); // in the console you will see, for example 10
```

↑ Back

Filter

Example of working with Lists filter, 0.4 Mb

It filters the List content. It is meant for search. The property is available for writing and reading. To use this property it is required to receive access to the list.

Syntax:

```
List_1.Filter = Filter;
```

Parameters:

**List_1 - the link to the list;
Filter – the list filter.**

On the output:

Filter (string).

Example of reading:

```
// Read the filter of List_1 and output the result the debugging console
IR.Log(List_1.Filter); // in the console you will see, for example "A"
```

↑ Back

Events

EVENT_ITEM_SELECT

Example of work with the event, 0.4 Mb

Clicking on the List items. To use this event it is required to receive access to the list.

Syntax:

```
IR.AddListener(IR.EVENT_ITEM_SELECT, List_1, function(Item, Subitem){ Action
});
```

Parameters:

**EVENT_ITEM_SELECT (event) – clicking on the List item;
List_1 – the link to the list;
Item   (number) – the item which was clicked;
Subitem (number)   - the subitem which was clicked;
Action (commands) – any function, the operator of Java Sript or iRidium Script.**

Example:

```
// Subscribe to the event of clicking on the list item and output it in the
debugging console
// the number of the item which was clicked
IR.AddListener(IR.EVENT_ITEM_SELECT, List_1, function(Item, Subitem){
IR.Log(Item); // in the console you will see, for example 2
});
```

↑ Back

EVENT_LIST_ITEM_CHANGE (Beta)

Example of work with the event, 1 Mb

Changing values of List subitems. To use this event it is required to receive access to the list.

Unlike EVENT_ITEM_SELECT it reacts to the event Press, Release and Move and provides reading and writing access to all subitem properties. It improves Lists capabilities and makes work more convenient.

Syntax:

```
IR.AddListener(IR.EVENT_LIST_ITEM_CHANGE, List_1, function(Item, Subitem,
TypeEvent, object){ Action });
```

Parameters:

**IR.EVENT_LIST_ITEM_CHANGE (event) - changing values of the List subitem;
List_1 - the link to the list;
Item (number) – the item which was clicked;
To receive the order number it is required to use the mask 0xFFFFFF
Subitem (number) - the subitem which was clicked;
TypeEvent (number) - the event type, it returns numbers: 11 - Press, 12 - Release, 13 - Move;
Object (item) - the link to the subitem which was changed, it gives access to properties for reading and writing
Action (commands) - any function, the operator of Java Sript or iRidium Script.**

↑ Back

Reading

Example of reading

```
// Subscribe to the event of changing the value of the List subitem and
output the subitem name
// in the debugging console
IR.AddListener(IR.EVENT_LIST_ITEM_CHANGE, List_1, function(Item, Subitem,
TypeEvent, object){
        IR.Log(object.Value); // in the console you will see, for example
"Item 1"
});
```

↑ Back

Writing

Example of writiting

```
// Subscribe to the event of changing the value of the List subitem and write
for the subitem
// value 32 in the property X
```

```
IR.AddListener(IR.EVENT_LIST_ITEM_CHANGE, List_1, function(Item, Subitem,
TypeEvent, object){
        // The property X will be rewritten and the item will be moved to the
corresponding position
        object.X = 32;
});
```

↑ Back

Links to Objects

Example of receiving the link to the object and using the object in another event.

```
var MyItem = null; // create and describe the global variable MyItem and
write an empty value in it
// subscribe to the event of changing the value of the List subitem and to
the variable MyItem
// write the link to the changed subitem
IR.AddListener(IR.EVENT_LIST_ITEM_CHANGE, List, function(Item, Subitem,
TypeEvent, object)
{
        // write the link of the changed subitem in the global variable
MyItem
        MyItem = object;
});


// subscribe to the event EVENT_WORK and if the variable MyItem is not empty,
increase
// the value Angle of the List subitem the link to which you wrote
// in the event EVENT_LIST_ITEM_CHANGE
IR.AddListener(IR.EVENT_WORK, 0, function(){
        if(MyItem)
                MyItem.Angle++; // increase the value of the Angle property
of the subitem
});
```

↑ Back

Reference by Names

Example of referring to List subitems by their names

```
// Subscribe to the event of changing the value of the List subitem
// and depending on the subitem name output different messages
// in the debugging console
IR.AddListener(IR.EVENT_LIST_ITEM_CHANGE, List, function(Item, Subitem,
TypeEvent, object){
        switch(object.Name){
                case "Item 1":
```

```
                        IR.Log("The item with the name Item 1 was clicked");
                        break;

                case "Item 2":
                        IR.Log("The item with the name Item 2 was clicked");
                        break;
        }
});
```

↑ Back

Reference by Items Numbers

Example of using numbers of items

```
// Subscribe to the event of changing the value of the List subitem
// and depending on the item number output different messages
// in the debugging console
IR.AddListener(IR.EVENT_LIST_ITEM_CHANGE, List, function(Item, Subitem,
TypeEvent, object){
        switch(Item & 0xFFFFFF){ // apply the mask to Item to receive the
number
                case 1:
                        IR.Log("The item with number 1 was clicked");
                        break;
                case 2:
                        IR.Log("The item with number 2 was clicked");
                        break;
                case 3:
                        IR.Log("The item with number 3 was clicked");
                        break;
        }
});
```

↑ Back

Reference by Subitems Numbers

Example of using numbers of subitems

```
// Subscribe to the event of changing the value of the List subitem
// and depending on the subitem number output different messages
// in the debugging console
IR.AddListener(IR.EVENT_LIST_ITEM_CHANGE, List, function(Item, Subitem,
TypeEvent, object){
        switch(Subitem){
                case 1:
                        IR.Log("The subitem with number 1 was clicked");
                        break;
                case 2:
```

```
                        IR.Log("The subitem with number 2 was clicked");
                        break;
                case 3:
                        IR.Log("The subitem with number 3 was clicked");
                        break;
        }
});
```

↑ Back

Types of Events

Example of using event types

```
// Subscribe to the event of changing the value of the List subitem
// and depending on the event type output different messages
// in the debugging console
IR.AddListener(IR.EVENT_LIST_ITEM_CHANGE, List, function(Item, Subitem,
TypeEvent, object){
        switch(TypeEvent){
                case 11:
                        IR.Log("The Press event");
                        break;

                case 12:
                        IR.Log("The Release event");
                        break;
                case 13:
                        IR.Log("The Move event");
                        break;
        }
});
```

↑ Back

Access to Item Parent

The property for receiving access to the item parent.

The tree of items parentage:

- Subitem;
  - Item;
    - List;
      - Page or Popup;

For example: The List subitem has a parent – the List item.

Example of changing the value of one List sibitem by another List subitem with the help of the Parent property.

```
// Subscribe to the event of changing the value of the List subitem
// and change the value of one subitem by changing another.
IR.AddListener(IR.EVENT_LIST_ITEM_CHANGE, List, function(Item, Subitem,
TypeEvent, object){
   // If the subitem has the Up name, then increase the item value with the
name Level
   // which is located on the same List item.
   if(object.Name == "Up")
      object.Parent.GetItem("Level").Value++;

    // in this case Parent is a link to the List item
    // where all List subitems are located
    // and they can be referred to by name using the method GetItem
});
```

↑ Back

Use

As an example of use – animation at deleting Lists

```
var AnimObject = null; // the variable for storing the link to the List item
var aTime = 400;       // the activation time
var timer = 0;         // timer
var animrun = false;   // the flag of the animation state true – in work,
false – not executed

// Subscribe to the event of changing the value of the List subitem
IR.AddListener(IR.EVENT_LIST_ITEM_CHANGE, List, function(ItemID, SubItemID,
Type, object){
    if(object.Name == "Delete" && !animrun){  // if the subitem name is
Delete and animation is not executed
       AnimObject = object.Parent;            // receive the parent – the
List item
       AnimID = ItemID;                       // and save its identifier, it
can be done without a mask
    }
});
// Subscribe to the event EVENT_WORK
IR.AddListener(IR.EVENT_WORK, 0, function(time){
   if(AnimObject){        // if AnimObject is not empty and has the link to
the subitem
      animrun = true;     // install the animation state to true
      if(timer <= aTime){ // if the timer didn't exceed the animation time
         AnimObject.GetState(0).Opacity = IR.Tween("TWEEN_LINEAR", timer,
255, -255, aTime);
         AnimObject.ScaleX = IR.Tween("TWEEN_LINEAR", timer, 1, -0.5, aTime);
         AnimObject.ScaleY = IR.Tween("TWEEN_LINEAR", timer, 1, -0.5, aTime);
```

```
         timer += time;   // increase the timer
      }
      else{
                  // as soon as the animation time is finished
         List.DeleteItem(AnimID); // delete the List item
         AnimObject = null;       // clear the variable AnimObject
         animrun = false;         // set the animation state to false

         timer = 0;               // clear the timer
      }
   }
});
```

↑ Back

Application

Example of using Lists, 0.4 Mb

↑ Back