< Back

It allows you to use OS capabilities of panels, work with timers, control the app and its events.

Updated: **July 27, 2014**
Report a mistake

Contents

Open the System Menu

Description:
The function allows you to open the system menu after the engineer password is input. After the command is executed, you will see the field for inputting the password. As soon as you input the password you will have access to the system menu.

Syntax:

```
IR.ShowSystemMenu();
```

Example:
Open the system menu by clicking Item 1 on Page 1.

```
IR.AddListener(IR.EVENT_ITEM_PRESS, IR.GetItem("Page 1").GetItem("Item 1"),
function(){

      IR.ShowSystemMenu();
});
```

Details:
It is not possible to receive access to the system menu via script without the engineer password.

Download:
Example of opening of the system menu>>

↑ Back

Receive HWIDs of Panels

Description:
The variable allows you to receive the HWID of the panel where the app is launched.

Syntax:

```
IR.HWID;
```

Result:
The line of symbols representing the panel HWID.

Example:
Write the panel HWID in the debugging console by clicking Item 1 on Page 1.

```
IR.AddListener(IR.EVENT_ITEM_PRESS, IR.GetItem("Page 1").GetItem("Item 1"),
function(){

  IR.Log(IR.HWID);
});
```

Details:
To receive a particular symbol from IR.HWID, us the function charAt(symbol_number). The numeration of symbols begins with 0.

```
IR.AddListener(IR.EVENT_ITEM_PRESS, IR.GetItem("Page 1").GetItem("Item 1"),
function(){

  IR.Log(IR.HWID.charAt(0));
});
```

↑ Back

Close the App

Description:
The function allows you to close the app. It is available for Andorid and Windows based devices only.

Syntax:

```
IR.Exit();
```

Example:
Close the app by clicking Item 1 on Page 1.

```
IR.AddListener(IR.EVENT_ITEM_PRESS, IR.GetItem("Page 1").GetItem("Item 1"),
function(){

    IR.Exit();
});
```

Details:
It is not possible to close the app on iOS based devices by using scripts. If the app closed like this before, then it was abnormal closing of the app and some data could be lost.

↑ Back

Show Keyboards

Description:
The function allows you to show keyboards. It is available for Andorid and iOS based devices only.
Syntax:

```
IR.ShowKeyboard(type);
```

Ingoing parameters

- **type** – the keyboard number: 1 - alphabet, 2 - numbers

Example:
Show the number keyboard by clicking Item 1 on Page 1.

```
IR.AddListener(IR.EVENT_ITEM_PRESS, IR.GetItem("Page 1").GetItem("Item 1"),
function(){

  IR.ShowKeyboard(2);
});
```

Details:

- To hide a keyboard on iOS based devices you can use the command IR.ShowKeyboard(0). It doesn’t work on Android based devices.
- To open a keyboard all other keyboards have to be hidden. If some keyboard (for example, numeric) is already open, then it will be impossible to open another (for example, alphabetic) one.
- To open a keyboard on Windows based devices you can use the command IR.Execute("osk");

↑ Back

Execute Item Macros

Description:
The function executes macros set up in Object Properties / Programming from the item event.

Activation:
To activate the function you need to have access to the item (page, popup or graphic item).

Syntax:

```
Item.StartAction(Event);
```

Input parameters

- **Event** – the event which macros have to be executed.

Example:
After launching the app execute macros from the PRESS event of Item 1 from Page 1.

```
IR.AddListener(IR.EVENT_START, 0, function(){

  IR.GetItem("Page 1").GetItem("Item 1").StartActions(IR.EVENT_ITEM_PRESS);
});
```

Use:

- Macros container
- Activation of actions of a button on one page on another page. It is used for scenes and hot keys.

Details:

- There is no visualization of event activation, for example clicking on the item which macros are being executed. The item actions will be executed but it won't be visually reflected in any way.
- When executing only macros set up in the editor are considered. Events activated via scripts won't be executed.
- This function **is not fully explored** so it's possible that some events won't be recognized.
- **If you** faced the problem which could be solved with the help of this function and **learnt** some **details of its use which were not documented** - **please write to us.**

↑ Back

Start Another App

Description:
The function allows you to start any other app, file or web site on the panel with the launched i2 Control app.

Syntax:

```
IR.Execute(path);
```

Input parameters

- **path** – the path to the app, the syntax of the path, it depends on the operating system of the device where i2 Control is launched.

Example:
Open the web site after starting the app.

```
IR.AddListener(IR.EVENT_START, 0, function(){

 IR.Execute("http://iridiummobile.net"); // Example of the link
});
```

Other examples:

```
IR.AddListener(IR.EVENT_START, 0, function(){
 // OS Windows
 IR.Execute("http://iridiummobile.net"); // Example of the link
 IR.Execute("mailto:contact@iridiummobile.ru"); // Write an e-mail through
the mail agent
 IR.Execute("cmd"); // Activate the command string
 IR.Execute("calc"); // Activate the calculator
```

```
 IR.Execute("c:\\Program Files\\iRidium\\iRidium.exe"); // Start the app


 // iOS - iPhone
 IR.Execute("http://iridiummobile.net"); // Example of the link
 IR.Execute("mailto:contact@iridiummobile.ru"); // Write an e-mail through
the mail agent
 IR.Execute("tel:+1234567890"); // Call on the phone
 IR.Execute("sms:+1234567890"); // Send an SMS
 IR.Execute("music:"); // Open the app for listening to music
 IR.Execute("videos:"); // Open the app for watching videos
 IR.Execute("ibooks://"); // Open the app for reading e-books
 IR.Execute("fb://feed"); // Open the Facebook app
 IR.Execute("twitter://"); // Open the Twitter app

 // iOS - iPad
 IR.Execute("http://iridiummobile.net"); // Example of the link
 IR.Execute("mailto:contact@iridiummobile.ru"); // Write an e-mail through
the mail agent
 IR.Execute("music:"); // Open the app for listening to music
IR.Execute("videos:"); // Open the app for watching videos
 IR.Execute("ibooks://"); // Open the app for reading e-books
 IR.Execute("fb://feed"); // Open the Facebook app
 IR.Execute("twitter://"); // Open the Twitter app

 // OS Android
 IR.Execute("http://iridiummobile.net"); // Example of the link
 IR.Execute("mailto:contact@iridiummobile.ru"); // Write an e-mail through
the mail agent
 IR.Execute("tel:+1234567890"); // Call on the phone
 IR.Execute("sms:+1234567890"); // Send an SMS
 IR.Execute("file:///tmp/android.txt"); // Open the file
});
```

Details:

- To start apps on Windows, each back slash has to be substituted by the double back slash.

```
IR.Execute("c:\\Program Files\\iRidium\\iRidium.exe")
```

↑ Back

Receive Info about Memory and Cache

Description:
The variable returns information about the memory and cache.

Syntax:

```
IR.GetSystemInfo("GC_MEMORY_INFO");
```

Example:
Write the information about the memory and cache in the console by clicking Item 1 on Page 1.

```
IR.AddListener(IR.EVENT_ITEM_PRESS, IR.GetItem("Page 1").GetItem("Item 1"),
function(){

  IR.Log(IR.GetSystemInfo("GC_MEMORY_INFO").UsedMemory);
  IR.Log(IR.GetSystemInfo("GC_MEMORY_INFO").TotalMemory);
  IR.Log(IR.GetSystemInfo("GC_MEMORY_INFO").ChunkHeaps);
  IR.Log(IR.GetSystemInfo("GC_MEMORY_INFO").ChunkCache);
  IR.Log(IR.GetSystemInfo("GC_MEMORY_INFO").MemoryHeaps);
  IR.Log(IR.GetSystemInfo("GC_MEMORY_INFO").MemoryBusyBlocks);
  IR.Log(IR.GetSystemInfo("GC_MEMORY_INFO").MemoryFreeBlocks);
});
```

↑ Back

Output Info in the Console

Description:
It is used to output of information in the debugging console while work of the i2 Control app. The debugging console is available for Windows based devices only.

Syntax:

```
IR.Log(text);
```

Input parameters:

- **text** – any data: strings, numbers, objects, arrays. Pay attention that after the data are sent as a parameter they are converted into a string.

Example 1:
Output a sentence in the debugging console when the app is launched.

```
IR.AddListener(IR.EVENT_START, 0, function(){

    IR.Log("Hello iRidium!");
});
```

Example 2:
Output the sum of two numbers in the debugging console when the app is launched.

```
IR.AddListener(IR.EVENT_START, 0, function(){
    // create the a variable;
    var a = 8;
    // create the b variable;
    var b = 16;
    // create the c variable and sum up values of the a and b variables;
    var c = a + b;
    // output the result saved in c variable in the debugging console;
    IR.Log(c);
    //24 will be output in the console;

    // the same value can be output in the debugging console without saving
the sum
    // by summing up directly in the function parameters
    IR.Log(a + b);
    // 24 will be output in the console;

    // or even without using variables
    IR.Log(8 + 16);
    // 24 will be output in the console;

    // data output in the debugging console can be more informative as
    // you can combine output of variable values and strings, for example
    // before outputting the value of the sum a + b we'll write that it's a
"Sum", but
    // calculation of the sum should be in brackets (a + b) as it's required
for dividing  different types
    // of data – strings and numbers. If it's not done, the data will be
converted into a string
    // and catenation will be performed. After that it will be output "Sum:
816".
    IR.Log("Sum: " + (a + b));
    // "Sum: 24" will be output in the console;
});
```

↑ Back

Decode the Link

Description:
The function is used for converting the link with coded symbols. For example: %23

Syntax:

```
IR.UrlDecode("link");
```

Input parameters:

- **link** – the link with coded symbols.

Example:
After the application start we'll decode the link and output data in the debugging console.

```
IR.AddListener(IR.EVENT_START, 0, function(){

  var link =
IR.UrlDecode("http://192.168.1.128:1400/getaa?s=1&u%3as97881%3fs");

  IR.Log(link); // output the decoded link in the debugging console
});
```

Details:

- If the link does not have coded symbols as a result you will receive the same link.
- You can use the function several times to clear several coding levels.

↑ Back

Delete Timers

Description:
The function is used for deleting timers SetTimeout and SetInterval.

Syntax:

```
IR.ClearInterval(id);
```

Input parameters:

- **id** – the identifier of the timer IR.SetTimeout or IR.SetInterval. You can receive an identifier only when creating a timer.

Example:
When starting the app, create a tier and save its identifier in the id variable. Delete the timer by clicking Item 1 on Page 1.

```
IR.AddListener(IR.EVENT_START, 0, function(){

    var id = IR.SetInterval(100, function(){

         // any actions
    });
});

IR.AddListener(IR.EVENT_ITEM_PRESS, IR.GetItem("Page 1").GetItem("Item 1"),
function(){
```

```
    IR.ClearInterval(id);
});
```

↑ Back

Repeat Actions with Interval

Description:
The timer is used to repeat actions many times after some interval.

Syntax:
IR.SetInterval(time, function);

Input parameters

- **time** – the time interval, ms
- **function** – the function name

Example:
After starting the app the message function which writes messages in the debugging console will be repeated each 400 ms.

```
IR.AddListener(IR.EVENT_START, 0, function(){

  IR.SetInterval(400, message);

  function message(){

     IR.Log("Hello");
  }
});
```

↑ Back

Delayed Start

Description:
The timer is used to start functions once after the preset period of time.

Syntax:

```
IR.SetTimeout(time, function);
```

Input parameters:

- **time** (number) – the period of time (ms)

- **function** (object) – the function name

Example:
At starting the app create the delayed start timer which will activate the test function in 400 ms.

```
IR.AddListener(IR.EVENT_START, 0, function(){

  IR.SetTimeout(400, test);

  function test(){

      // any actions
  }
});
```

↑ Back

Subscribe to Events

Description:
The function allows you to subscribe to an event and execute a set of actions at event activation.

Syntax:

```
IR.AddListener(event, object, function);
```

Input parameters:

- **event** (constant) – the event name from the list of events.
- **object** (object) – the identifier for accessing the object, it might be both a graphic item and a driver
- **function** (object) – the name of the function with the set of actions

Example 1:
Subscribe to the event of clicking Item 1 on Page 1 and write that it was clicked in the debugging console.

```
IR.AddListener(IR.EVENT_ITEM_PRESS, IR.GetItem("Page 1").GetItem("Item 1"),
function(){

  IR.Log("Item 1 was clicked");
});
```

Example 2:
Subscribe to the event of the app start but unlike in "Example 1", the second parameter is 0, as

there is access to the event already.

```
IR.AddListener(IR.EVENT_START, 0, function(){

  IR.Log("The app has been started");
});
```

↑ Back

Unsubscribe from Events

Description:
The function allows you to unsubscribe from events.
Syntax:

```
IR.RemoveListener(event, object, function);
```

Input parameters:

- **event** (constant) – the event name from the list of events.
- **object** (object) – the identifier for accessing the object, it might be a pfraphic item or a driver,
- **function** (function) – the function name or the link to the function which was used when subscribing to the event.

Example 1:
Deactivate the subscription to the event of clicking Item 1 on Page 1. The press function is a named function and it has to be defined in the code.

```
// Define the function
function press(){

     // actions
}

// Subscribe
IR.AddListener(IR.EVENT_ITEM_PRESS, IR.GetItem("Page 1").GetItem("Item 1"),
press);

// Deactivate the subscription
IR.RemoveListener(IR.EVENT_ITEM_PRESS, IR.GetItem("Page 1").GetItem("Item
1"), press);
```

Example 2:
Deactivate the subscription to the event of clicking Item 1 on Page 1. The function is not named and when subscribing to the event the link to it was saved in the link_press variable.

```
// Create the variable for saving the link to the function
var link_press;

// Subscribe and save the link to the function in the variable
IR.AddListener(IR.EVENT_ITEM_PRESS, IR.GetItem("Page 1").GetItem("Item 1"),
link_press = function(){

      // Actions
});

// Deactivate the subscription
IR.RemoveListener(IR.EVENT_ITEM_PRESS, IR.GetItem("Page 1").GetItem("Item
1"), link_press);
```

↑ Back

App Has Been Started

Name:
IR.EVENT_START

Description:
The event shows that the app has been launched.

Subscription syntax:

```
IR.AddListener(IR.EVENT_START, 0, Function_Name)
```

Unsubscription syntax:

```
IR.RemoveListener(IR.EVENT_START, 0, Function_Name)
```

Attention! It is important to have the same **Function_Name** when subscribing and unsubscribing.

Example 1:
Write the message in the debugging console after starting the app.

```
IR.AddListener(IR.EVENT_START, 0, ShowMessage);

function ShowMessage(){

   IR.Log("The app has been started");
}
```

Example 2:

Do the same with the unnamed function.

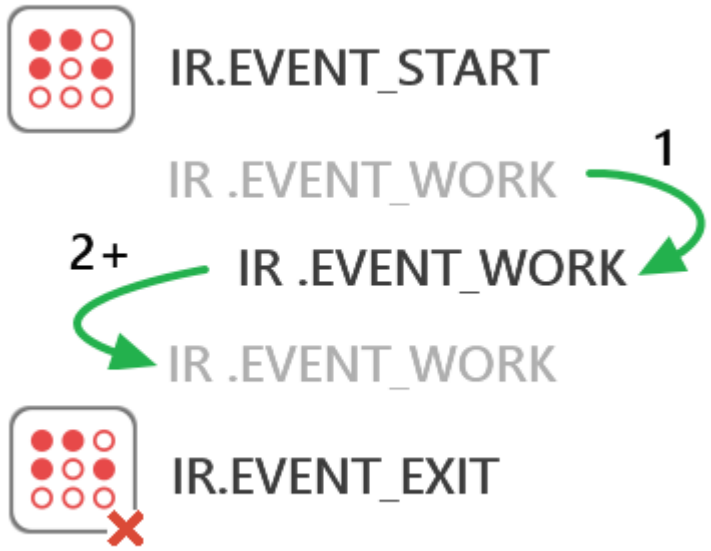
```
IR.AddListener(IR.EVENT_START, 0, function(){

   IR.Log("The app has been started");
});
```

↑ Back

App Is Running

Name:
IR.EVENT_WORK

Description:
The event shows that the app is working. So unlike other events this one is used many times.



Subscription syntax:

```
IR.AddListener(IR.EVENT_WORK, 0, Function_Name)
```

Unsubscription syntax:

```
IR.RemoveListener(IR.EVENT_WORK, 0, Function_Name)
```

Attention! It is important to have the same **Function_Name** when subscribing and unsubscribing.

Example 1:

Subscribe to the event and write the message in the log while the app is running.

```
IR.AddListener(IR.EVENT_WORK, 0, ShowMessage);

function ShowMessage(){

   IR.Log("The app is running");
}
```

Example 2:
Do the same with the unnamed function.

```
IR.AddListener(IR.EVENT_WORK, 0, function(){

   IR.Log("The app is running");
});
```

↑ Back

App Will Be Closed

Name:
IR.EVENT_EXIT

Description:
The event shows that the app will be closed. It is activated when receiving the command for closing the app.

Subscription syntax:

```
IR.AddListener(IR.EVENT_EXIT, 0, Function_Name)
```

Unsubscription syntax:

```
IR.RemoveListener(IR.EVENT_EXIT, 0, Function_Name)
```

Attention! It is important to have the same **Function_Name** when subscribing and unsubscribing.

Example 1:
Write the message in the Test global variable when exiting the app.

```
IR.AddListener(IR.EVENT_EXIT, 0, SaveMessage);

function SaveMessage(){
```

```
   IR.SetVariable("Global.Test", "The app is closed successfully");
}
```

Example 2:
Do the same with the unnamed function.

```
IR.AddListener(IR.EVENT_EXIT, 0, function(){

   IR.SetVariable("Global.Test", "The app is closed successfully");
});
```

↑ Back

App Has Changed Orientation

Name:
IR.EVENT_ORIENTATION

Description:
The event shows that the app has changed the orientation. The event can be activated by the turn of the panel or by the F5 hot key on Windows.

Subscription syntax:

```
IR.AddListener(IR.EVENT_ORIENTATION, 0, Function_Name)
```

Unsubscription syntax:

```
IR.RemoveListener(IR.EVENT_ORIENTATION, 0, Function_Name)
```

Attention! It is important to have the same **Function_Name** when subscribing and unsubscribing.

Received arguments

- **orientation** – the number of the new app orientation;

Value range

- **1** – vertical, the Home button is on the bottom, constant:

IR.ORIENTATION_PORTRAIT

- **2** - vertical, the Home button is on the top, constant: **IR.ORIENTATION_PORTRAIT_UPSIDE_DOWN**
- **3** – horizontal, the Home button is to the right, constant: **IR.ORIENTATION_LANDSCAPE_LEFT**
- **4** - horizontal, the Home button is to the left, constant: **IR.ORIENTATION_LANDSCAPE_RIGHT**

Example 1:
Write the message with the orientation number in the debugging console after the orientation is changed.

```
IR.AddListener(IR.EVENT_ORIENTATION, 0, Message);

function Message(orientation){

   IR.Log("Caramba! Somebody has turned the iPad, orientation number: " +
orientation);
}
```

Example 2:
Do the same with the unnamed function.

```
IR.AddListener(IR.EVENT_ORIENTATION, 0, function(orientation){

   IR.Log("Caramba! Somebody has turned the iPad, number of the current
orientation: " + orientation);
});
```

Details:
After starting the app, it's not possible to learn the current orientation of the device until the app changes the orientation for the first time.

Download:
Example of working with the event >>

↑ Back

Keyboard Has Been Shown

Name:
IR.EVENT_KEYBOARD_SHOW

Description:
The event is activated when showing the keyboard of the panel OS.

Subscription syntax:

```
IR.AddListener(IR.EVENT_KEYBOARD_SHOW, 0, Function_Name)
```

Unsubscription syntax:

```
IR.RemoveListener(IR.EVENT_KEYBOARD_SHOW, 0, Function_Name)
```

Attention! It is important to have the same **Function_Name** when subscribing and unsubscribing.

Example 1:
Write the message in the debugging console when showing the keyboard.

```
IR.AddListener(IR.EVENT_KEYBOARD_SHOW, 0, ShowMessage);

function ShowMessage(){

   IR.Log("The keyboard has been shown!");
}
```

Example 2:
Do the same with the unnamed function.

```
IR.AddListener(IR.EVENT_KEYBOARD_SHOW, 0, function(){

   IR.Log("The keyboard has been shown!");
});
```

↑ Back

Create Effects

Description:
The function allows you to create an effect when working with the app and use it in the group of effects for hiding or showing a popup.

Use:

- It is required to show/hide one popup in the project with different effects.
- Adding effect groups created in the editor.

Syntax:

```
var Variable_Name = IR.CreateEffect(Effect);
```

Input parameters:

- **Effect** (number) – the effect name available in iRidium. See the names of effects below:

Effects:

- IR.EFFECT_FADE – gradual appearing /disappearing;
- IR.EFFECT_ROTATE – rotation round its own axis, the center of rotation is the middle of the

popup;
- IR.EFFECT_SCALE – scaling in/scaling out;
- IR.EFFECT_SLIDE – sliding to the sides;
- IR.EFFECT_TVSCAN – compressing horizontally;

Effect properties:

- **Group** (number) - the number of effect group where the effect will be. It is recommended to use the numbers of effect groups beginning with 1000. It will help you to avoid using the numbers of the effect groups created by the editor. If you need to add some effects to the effect groups created by the editor, please do it knowingly.
- **Delay** (number, ms) – the delay before the effect;
- **Duration** (number, ms) – duration of the effect;
- **Tween** (number) – the calculation formula, see the list of calculation formulas

Example 1:
Create the Fade effect at the start of the app, set up its properties, place it in the effect group and show the popup with this effect group.

```
IR.AddListener(IR.EVENT_START, 0, function(){

   // Create and save in the variable
   var Fade = IR.CreateEffect(IR.EFFECT_FADE);

   // Group number
   Fade.Group = 1000;

   // Delay
   Fade.Delay = 0;

   // Duration
   Fade.Duration = 400;

   // Calculation formula
   Fade.Tween = 0;

   // Showing the popup
   IR.ShowPopup("Popup 1", 1000)
});
```

↑ Back

Specify Values

Description:
The function which allows you to specify values on the specified section.

Syntax:

```
var Variable_Name = IR.Tween(Calculation, Number, Beginning, End,
Specifying);
```

Input parameters:

- **Calculation** (number) – the calculation formula name, see the list of calculation formulas
- **Number** (number) – the number of the value you need to learn
- **Beginning** (number) – the first value for the calculation
- **End** (number) - the last value for the calculation
- **Specifying** (number) – it defines the degree of specification. The more the number, the more accurate is the value.

Use

- Specifying of ranges
- Animation of graphic items

Example 1:
When starting the app, write the value of the 10th element from the range from 0 to 100, specified to 255 pixels in the debugging console.

```
IR.AddListener(IR.EVENT_START, 0, function(){

    var value = IR.Tween("TWEEN_LINEAR", 10, 0, 100, 255);
    IR.Log(value);
});
```

Example 2:
While working with the app, change the item opacity from 0 to 255 in 500 ms.

```
// Declare the timer variable for building up time
// Declare the Tween variable for storing values of the math function
var timer = 0;
var Tween = 0;
// Create the listener for the Work event and send the current time
IR.AddListener(IR.EVENT_WORK, 0, function(time){

   // Build up time
   timer += time;

   // Save the value of the math function
   Tween = IR.Tween("TWEEN_LINEAR", timer, 0, 255, 500);

   // Write in the Opacity property of the graphic item
   // the math function
   IR.GetPage("Page 1").GetItem("Item 1").GetState(0).Opacity = Tween;
});
```

↑ Back

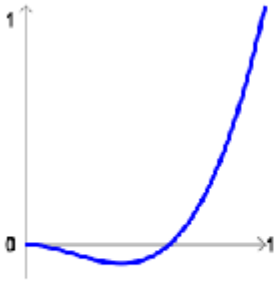
Calculation Formulas

Description:
The formula defines how the intermediate value on the selected value section will be calculated. Each formula has its number which should be sent to the function as the first IR.Tween property.

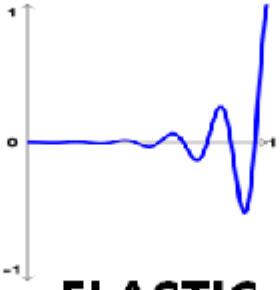
Linear
линейная

Quad
квадратичная

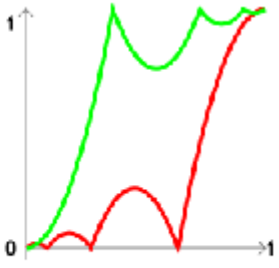
CIRC
дуга



BACK
оттягивание



ELASTIC
упругая



EASE
движение мяч

Formula names and numbers:

- TWEEN_LINEAR: 0,
- TWEEN_SINE_IN: 1,
- TWEEN_SINE_OUT: 2,
- TWEEN_SINE_IN_OUT: 3,
- TWEEN_QUINT_IN: 4,
- TWEEN_QUINT_OUT: 5,
- TWEEN_QUINT_IN_OUT: 6,
- TWEEN_QUART_IN: 7,
- TWEEN_QUART_OUT: 8,
- TWEEN_QUART_IN_OUT: 9,
- TWEEN_QUAD_IN : 10,
- TWEEN_QUAD_OUT: 11,
- TWEEN_QUAD_IN_OUT: 12,
- TWEEN_EXPO_IN: 13,
- TWEEN_EXPO_OUT: 14,
- TWEEN_EXPO_IN_OUT: 15,
- TWEEN_ELASTIC_IN: 16,
- TWEEN_ELASTIC_OUT: 17,
- TWEEN_ELASTIC_IN_OUT:18,
- TWEEN_CUBIC_IN: 19,
- TWEEN_CUBIC_OUT: 20,
- TWEEN_CUBIC_IN_OUT: 21,
- TWEEN_CIRC_IN: 22,
- TWEEN_CIRC_OUT: 23,
- TWEEN_CIRC_IN_OUT: 24,
- TWEEN_BOUNCE_IN: 25,
- TWEEN_BOUNCE_OUT: 26,
- TWEEN_BOUNCE_IN_OUT: 27,
- TWEEN_BACK_IN: 28,

- TWEEN_BACK_OUT: 29,
- TWEEN_BACK_IN_OUT: 30

Note:
For more convenient use, without remembering numbers, use the script which enables using formula names instead of numbers. The script should be added in the project and it shouldn't be changed.

Download the script:
https://s3.amazonaws.com/iridium2_downloads/Script_Drivers/IR_Tweens.js

↑ Back

Copying Data in the Buffer

Description:
The function enables copying of strings in the device buffer. Works on iOS, Android, Windows

Syntax:

```
IR.CopyToClipboard(text);
```

Input parameters:

- text - the data which can be converted into a string

Example:

```
IR.AddListener(IR.EVENT_START, 0, function(){
// Create the HWID variable
var HWID = IR.HWID;
// Copy the HWID of the panel with the running app
IR.CopyToClipboard(IR.GetVariable("System.Device.Name") + " HWID:" + HWID);
});
```

↑ Back

Updating Project via Commands

Description:
The function allows you to change the name of the project uploaded via HTTP and use the Multi-design function.

Syntac:

```
IR.DownLoadProject(
{
```

```
      type:parsed_url.scheme,
      host:parsed_url.authority,
      path:parsed_url.path + '?' + parsed_url.query,
      multiproject: Value,
      name: Text
});
```

Input parameters:

- **type** – the protocol type (https/http)
- **host** – the address
- **path** – the path to the project file, in the beginning of the path you should add "/"
- **multiproject** – the Multi-design function (1 - on, 0 - off)
- **name** – the project new name

Example:

```
function HTTPUpdate(url)
{
IR.Log(url);

   var parsed_url = parse_url(url);
   var downloader = IR.DownLoadProject(
                    {
                type:parsed_url.scheme,
                host:parsed_url.authority,
                path:parsed_url.path + '?' + parsed_url.query,
                multiproject: 1,
                name: "TestNew"
                     });
                  }
```

Details:

- The name is set in the name property. When you start to upload the project, the file system gets a new name and creates a file with a new name. Then it writes data in this file. If you create a file with the same name and then rename the file, you can damage the file with the previous name.

Note:

- The uploaded project receives the name set in the name property.

Download:
Example of working with the function >>

↑ Back

Calculate HASH and CRC

IR.CalculateCRC - the method to calculate CRC summ.
IR.CalculateHash - the method to calculate Hash summ.

Example:

```
var crc8 = IR.CalculateCRC(IR.CRC_8,  "Test"));
var crc16 = IR.CalculateCRC(IR.CRC_16, "Test"));
var crc32 = IR.CalculateCRC(IR.CRC_32, "Test"));

var hashMD5 = IR.CalculateHash(IR.HASH_MD5, "Test"));
var hashSHA1 = IR.CalculateHash(IR.HASH_SHA1, "Test"));
var hashSHA256 = IR.CalculateHash(IR.HASH_SHA256, "Test"));
var hashSHA384 = IR.CalculateHash(IR.HASH_SHA384, "Test"));
var hashSHA512 = IR.CalculateHash(IR.HASH_SHA512, "Test"));
```

↑ Back