Roboocyte2 Scripting
# Table of Contents

1 **Introduction** 5  
1.1 Java Script 5  
   1.1.1 Roboocyte2 Specific Features 5  
   1.1.2 Oocyte Loop 5  
1.2 Variables 5  
   1.2.1 JavaScript Variables 5  
   1.2.2 User Defined Interactive Variables 6  
   1.2.3 Predefined Variables 6  
   1.2.4 Read-only Variables 7  
1.3 Complete List of Commands 9  
   1.3.1 Conditions 9  
   1.3.2 Robo2. GUI-Commands 10  
   1.3.3 Robo2. Variable Handling Commands 10  
   1.3.4 Robo2. Movement Commands 11  
   1.3.5 Robo2. Amplifier and Data Acquisition Commands 12  
   1.3.6 Robo2. Data Analysis Commands 13  
   1.3.7 Robo2. Database Commands 14  
   1.3.8 Robo2. Timing Commands 15  
   1.3.9 Robo2. Liquid Handling (Roboflow) Commands 15  
   1.3.10 Robo2. High Level Commands 16  
   1.3.11 RecDisplay. Recording Display Commands 17  
   1.3.12 ControlDisplay. Control Display Command 17  
   1.3.13 Gilson. Gilson Commands 18  
2 **Example Script "Dose-Response"** 19  
2.1 Defining Dialogue variables 19  
   2.1.1 User Defined Dialogue Variables 19  
   2.1.2 Working with Pre-defined Variables 20  
2.2 The Oocyte Loop 21  
2.3 Standard Routines executed before Recording Protocol 22  
   2.3.1 Moving the Measuring Head into Liquid 22  
   2.3.2 Switching to Current Clamp Mode 23  
   2.3.3 Electrode Offset Compensation 23  
   2.3.4 Electrode Resistance Test 23  
   2.3.5 Oocyte Impalement 24  
   2.3.6 Starting Voltage Clamp Mode 25  
   2.3.7 Initial Leak Current Test 25  
   2.3.8 Final Leak Current Test with Perfusion 26  
3 **Recording Protocol Examples** 27  
3.1 Using the Dose-Response Script as a Template 27  
3.2 Ligand-gated Channels and Electrogenic Transporters 27  
   3.2.1 Expression Test 27  
   3.2.2 Dose-Response Protocol 29  
3.3 Voltage-gated Ion Channels 30  
   3.3.1 Expression Test 30  
4 **Using the Gilson Liquid Handler** 33  
4.1 Movement Commands 33  
4.2 Peristaltic Pump Commands 33
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3</td>
<td>Transfer Port Valve Commands</td>
<td>33</td>
</tr>
<tr>
<td>4.4</td>
<td>Examples</td>
<td>34</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Differences between Roboflow and Gilson</td>
<td>34</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Solution Exchange - Roboflow vs. Gilson</td>
<td>34</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Handling of the Gilson Delay</td>
<td>35</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Java Script

Roboocyte2 scripts are written in a JavaScript like language. The syntax (control structures, variables, functions, arrays) is conforming to JavaScript, other JavaScript functionality, particularly the document (and other HTML specific things like DOM handling, libraries etc.) is NOT supported.

1.1.1 Roboocyte2 Specific Features

The Roboocyte2 is controlled via 4 JavaScript objects:

1. **Robo2** (the robot hardware)
2. **RecDisplay** (display of the recordings)
3. **ControlDisplay** (display of control recordings)
4. **Gilson** (Gilson liquid handler if applicable)

The commands for each object are listed in the tables below.

1.1.2 Oocyte Loop

In order to record from selected oocytes, functions must be executed within the so called oocyte loop. The array variable SelectedWells contains the indices of these wells and can be used in a for loop, e.g.

```javascript
for (var i = 0; i < Robo2.SelectedWells.Count; i++)
{
    var WellIndex = Robo2.SelectedWells[i];
    Robo2.Log("now moving to well: " + Robo2.SelectedWellNames[i]);
    Robo2.MoveToWell(WellIndex);
}
```

1.2 Variables

Variables come in different flavors:

1.2.1 JavaScript Variables

Variables can be created with the JavaScript keyword `var`. These variables can be used as in standard JavaScript to store values and use them later in the script.

```javascript
var test = 1
```

generates a variable “test” with the value "1"
1.2.2 User Defined Interactive Variables

There are also user defined interactively changeable variables. These variable are defined by the command `SetDialogVariable`.

```c
Robo2.SetDialogVariable("clampvoltage", -60, "Recording Voltage in mV");
```

generates the variable "clampvoltage" with the value -60. The comment "Recording Voltage in mV" will help you to identify the meaning of the variable when opening the dialog.

To interactively display and change the values the command `ShowDialog()` is used.

1.2.3 Predefined Variables

Some values which are important for the use of the Roboocyte2 are predefined (they do not have to be defined via `var`). These variables can be changed in the script source and also interactively via display of a dialog during script execution. Some of these variables are used as parameters in the high level commands, so that a script can be easily parameterized in the dialog without having to change the script source code.

All of these variables are of type `int`.

The syntax to set the values in the source is as follows: The command is a function call which starts with `Set_` and then the variable name (in upper case) is appended, e.g.

```c
Set_DCOFFSET_RANGE(5); 
```

which will assign the value 5 to the variable `DCOFFSET_RANGE`

To interactively display and change variable values the command `ShowStandardDialog()` is used.

List of all predefined variables, their default values, and respective command

<table>
<thead>
<tr>
<th>Variable</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robo2.ResistanceCheck_I(MIN_RESISTANCE_I, MAX_RESISTANCE_I)</td>
<td>MIN_RESISTANCE_I: 100 Ohm</td>
<td>minimum TEVC probe resistance of the I electrode</td>
</tr>
<tr>
<td></td>
<td>MAX_RESISTANCE_I: 1000 Ohm</td>
<td>maximum TEVC probe resistance of the I electrode</td>
</tr>
<tr>
<td></td>
<td>MIN_RESISTANCE_U: 100 Ohm</td>
<td>minimum TEVC probe resistance of the U electrode</td>
</tr>
<tr>
<td></td>
<td>MAX_RESISTANCE_U: 1000 Ohm</td>
<td>maximum TEVC probe resistance of the U electrode</td>
</tr>
<tr>
<td>Robo2.DCOffsetCorrection(DCOFFSET_RANGE, DCOFFSET_DELAY, DCOFFSET_WAIT, DCOFFSET_ATTEMPTS)</td>
<td>DCOFFSET_RANGE: 3 mV</td>
<td>max deviation of DC offset from 0</td>
</tr>
<tr>
<td></td>
<td>DCOFFSET_DELAY: 20 sec</td>
<td>delay before DC offset measurement</td>
</tr>
<tr>
<td></td>
<td>DCOFFSET_WAIT: 10 sec</td>
<td>wait after each check</td>
</tr>
<tr>
<td></td>
<td>DCOFFSET_ATTEMPTS: 3</td>
<td>number of attempts to try DC</td>
</tr>
</tbody>
</table>
### 1.2.4 Read-only Variables

These are variables the values of which cannot be changed by the user but are updated by the system.
## Variable Robo2.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SelectedWells</td>
<td>array of integer indices of the selected wells</td>
</tr>
<tr>
<td>SelectedWellNames</td>
<td>array of strings of the names of the selected wells</td>
</tr>
<tr>
<td>VALUE</td>
<td>last measured value (current or voltage)</td>
</tr>
<tr>
<td>VALUE_IC</td>
<td>last measured current at the current electrode</td>
</tr>
<tr>
<td>VALUE_UC</td>
<td>last measured voltage at the current electrode</td>
</tr>
<tr>
<td>VALUE_US</td>
<td>setpoint voltage (command potential)</td>
</tr>
<tr>
<td>VALUE_UV</td>
<td>last measured voltage at the voltage electrode</td>
</tr>
<tr>
<td>RESISTANCE_U</td>
<td>current impedance of the U electrode</td>
</tr>
<tr>
<td>RESISTANCE_I</td>
<td>current impedance of the I electrode</td>
</tr>
<tr>
<td>SCRIPT_FILE</td>
<td>file name of current script</td>
</tr>
<tr>
<td>DATE_NOW</td>
<td>current date (string) in the format yyyy-mm-dd, e.g. 2011-10-18</td>
</tr>
<tr>
<td>TIME_NOW</td>
<td>current time (string) in the format hh:mm:ss, e.g. 14:18:22, (Roboocyte2 V &gt;= 1.1.5)</td>
</tr>
<tr>
<td>MINIMUM</td>
<td>minimum calculated value in the analysis ROI (to be used after a recording is finished)</td>
</tr>
<tr>
<td>POS_MINIMUM</td>
<td>position of the calculated minimum</td>
</tr>
<tr>
<td>MAXIMUM</td>
<td>minimum calculated value in the analysis ROI</td>
</tr>
<tr>
<td>POS_MAXIMUM</td>
<td>position of the calculated maximum</td>
</tr>
<tr>
<td>AVERAGE</td>
<td>calculated average value in the analysis ROI</td>
</tr>
<tr>
<td>AREA</td>
<td>calculated area in the analysis ROI</td>
</tr>
<tr>
<td>BASELINE_AVERAGE</td>
<td>calculated baseline average in the baseline ROI</td>
</tr>
<tr>
<td>SCRIPT_FILE</td>
<td>name of the actual script file</td>
</tr>
<tr>
<td>TIME</td>
<td>the number of minutes passed since the last</td>
</tr>
<tr>
<td>TIME_S</td>
<td>the number of seconds passed since the last StartTimer command</td>
</tr>
<tr>
<td>ALIGNMENT_X</td>
<td>alignment x position in µm</td>
</tr>
<tr>
<td>ALIGNMENT_Y</td>
<td>alignment y position in µm</td>
</tr>
<tr>
<td>ALIGNMENT_Z</td>
<td>alignment z position in µm</td>
</tr>
</tbody>
</table>
1.3 Complete List of Commands

1.3.1 Conditions

Some Commands can be used only under certain conditions:

- **O+** must be within oocyte loop
- **O** must not be within oocyte loop

- **W+** must be with carrier set to a well
- **L+** must be with z-axis moved into liquid
- **Oo+** must be with z-axis moved into oocyte
- **Oo-** must be with z-axis moved into oocyte

- **R+** must be within a recording
- **R** must not be within a recording

Data types in Parameters:

- **string**: a text value, can either be a variable or a literal within double quotes (e.g. „this is a string“)
- **int**: an integer number (-5, 4, 5689, but not 1.4)
- **bool**: true or false
- **— no parameter needed**

Some commands give back a return value or function as described

- **R**: return value of function
### 1.3.2 Robo2. GUI-Commands

<table>
<thead>
<tr>
<th>Robo2.</th>
<th>Parameter(s)</th>
<th>Action</th>
<th>Cond.</th>
<th>Example Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log(&quot;x&quot;)</td>
<td>string</td>
<td>Plots the message &quot;x&quot; in the Log window</td>
<td>---</td>
<td>Messages_eg.js</td>
</tr>
<tr>
<td>Information(&quot;x&quot;)</td>
<td>string</td>
<td>Opens a dialog with &quot;x&quot;. Script execution is suspended until OK is pressed</td>
<td>R-</td>
<td>Messages_eg.js</td>
</tr>
<tr>
<td>Question(&quot;x&quot;)</td>
<td>string</td>
<td>Opens a dialog with &quot;x&quot;. Different control paths in the script can be executed by using the return value of the function. The return value is true if Yes was clicked, false otherwise</td>
<td>R-</td>
<td>Messages_eg.js</td>
</tr>
</tbody>
</table>

### 1.3.3 Robo2. Variable Handling Commands

<table>
<thead>
<tr>
<th>Robo2.</th>
<th>Parameter(s)</th>
<th>Action</th>
<th>Cond.</th>
<th>Example Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetDialogVariable(x, y, z)</td>
<td>string: x = name int: y = value string: z = description</td>
<td>defines and sets a value to a variable that can be changed interactively by a dialog during script execution (ShowDialog)</td>
<td>---</td>
<td>Variables_eg.js</td>
</tr>
<tr>
<td>ShowDialog()</td>
<td>---</td>
<td>shows dialog to check and change values for variables defined with SetDialogVariable()</td>
<td>R-</td>
<td>Variables_eg.js</td>
</tr>
<tr>
<td>ShowStandardDialog()</td>
<td>---</td>
<td>shows dialog to check and change values for the predefined (standard) variables</td>
<td>R-</td>
<td>Variables_eg.js</td>
</tr>
</tbody>
</table>
### 1.3.4 Robo2. Movement Commands

<table>
<thead>
<tr>
<th>Robo2.</th>
<th>Parameter(s)</th>
<th>Action</th>
<th>Cond.</th>
<th>Example Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoveToWell(x)</td>
<td>int: well number</td>
<td>moves carrier to given well ( x ), with ( x ) from 0 to 95</td>
<td>R-</td>
<td>Movement_eg.js</td>
</tr>
<tr>
<td>MoveToHomePos()</td>
<td>---</td>
<td>moves Z-axis and carrier to Home-Position</td>
<td>R-</td>
<td>Movement_eg.js</td>
</tr>
<tr>
<td>MoveToParkPos()</td>
<td>---</td>
<td>moves Z-axis and carrier to Park-Position</td>
<td>R-</td>
<td>Movement_eg.js</td>
</tr>
<tr>
<td>MoveToChangePlatePos()</td>
<td>---</td>
<td>moves Z-axis up and carrier to a position to change the plate</td>
<td>R-</td>
<td>Movement_eg.js</td>
</tr>
<tr>
<td>MoveToCoarsePos()</td>
<td>---</td>
<td>moves Z-axis and carrier to coarse position</td>
<td>R-</td>
<td>Movement_eg.js</td>
</tr>
<tr>
<td>ZMoveToLiquid()</td>
<td>---</td>
<td>moves Z-axis into the liquid z-height</td>
<td>R-</td>
<td>Movement_eg.js</td>
</tr>
<tr>
<td>ZMoveHome()</td>
<td>---</td>
<td>move Z-axis to uppermost position</td>
<td>R-</td>
<td>Movement_eg.js</td>
</tr>
<tr>
<td>ZMoveToOocyte()</td>
<td>---</td>
<td>moves Z-axis into the oocyte z-height</td>
<td>R-</td>
<td>Movement_eg.js</td>
</tr>
<tr>
<td>ZMoveStepDown(x)</td>
<td>int: 0 - 100</td>
<td>moves the z-axis down by ( x ) µm</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>ZMoveStepUp(x)</td>
<td>int: 0 - 100</td>
<td>moves the z-axis stepwise up by ( x ) µm</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>SetAxisLight()</td>
<td>---</td>
<td>turns the white LED at the axis on or off</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>GetAxisLight()</td>
<td>---</td>
<td>returns <strong>true</strong> if light is on</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>ReferenceXY()</td>
<td>---</td>
<td>performs a reference movement of the carrier</td>
<td>R-</td>
<td>---</td>
</tr>
<tr>
<td>ReferenceZ()</td>
<td>---</td>
<td>performs a reference movement of the z-axis</td>
<td>R-</td>
<td>---</td>
</tr>
</tbody>
</table>
### 1.3.5 Robo2. Amplifier and Data Acquisition Commands

<table>
<thead>
<tr>
<th>Robo2.</th>
<th>Parameter(s)</th>
<th>Action</th>
<th>Cond.</th>
<th>Example Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>SetHoldingVoltage(x)</td>
<td>int: voltage in mV</td>
<td>sets clamp voltage to ( x ) mV</td>
<td>---</td>
<td>Acquisition_eg.js</td>
</tr>
<tr>
<td>SetVoltageClamp()</td>
<td></td>
<td>sets amplifier to voltage clamp mode</td>
<td>---</td>
<td>Acquisition_eg.js</td>
</tr>
<tr>
<td>SetHoldingCurrent(x)</td>
<td>int: current in nA</td>
<td>sets clamp current to ( x ) nA</td>
<td>---</td>
<td>Acquisition_eg.js</td>
</tr>
<tr>
<td>SetCurrentClamp()</td>
<td></td>
<td>sets amplifier to current clamp mode</td>
<td>---</td>
<td>Acquisition_eg.js</td>
</tr>
<tr>
<td>SetAmplifierCoefficients(x, y)</td>
<td>int: ( x ) = proportional gain int: ( y ) = integral gain</td>
<td>sets proportional and integral gain</td>
<td>---</td>
<td>Acquisition_eg.js</td>
</tr>
<tr>
<td>SetSampleRate(x)</td>
<td>int: samplerate</td>
<td>sets sample rate to ( x ) Hz for recordings</td>
<td>R-</td>
<td>Acquisition_eg.js</td>
</tr>
<tr>
<td>StartRecording()</td>
<td></td>
<td>starts data acquisition and recording of data</td>
<td>R-</td>
<td>Acquisition_eg.js</td>
</tr>
<tr>
<td>StopRecording()</td>
<td></td>
<td>stops data acquisition</td>
<td>R+</td>
<td>Acquisition_eg.js</td>
</tr>
<tr>
<td>SetControlSampleRate(x)</td>
<td>int: sample rate</td>
<td>sets control recording sample rate to ( x ) Hz (1-50)</td>
<td>R-</td>
<td>Acquisition_eg.js</td>
</tr>
<tr>
<td>StartControlRecording()</td>
<td></td>
<td>starts data acquisition, displayed in control window (not saved to file)</td>
<td>R-</td>
<td>Acquisition_eg.js</td>
</tr>
<tr>
<td>StopControlRecording()</td>
<td></td>
<td>stops control recording</td>
<td>R+</td>
<td>Acquisition_eg.js</td>
</tr>
<tr>
<td>RecordIVProtocol(x)</td>
<td>string: protocol name</td>
<td>executes the predefined IV protocol ( x )</td>
<td>O+, R-</td>
<td>Acquisition_eg.js</td>
</tr>
</tbody>
</table>
### 1.3.6 Robo2. Data Analysis Commands

<table>
<thead>
<tr>
<th>Robo2.</th>
<th>Parameter(s)</th>
<th>Action</th>
<th>Cond.</th>
<th>Example Script</th>
</tr>
</thead>
</table>
| SetBaselineROI(*x*, *y*) | int: *x* = left in s  
int: *y* = right in s | sets the region of interest (ROI) used for the baseline calculation. | --- | Data_eg.js |
| SetBaselineROIMilliSec(*x*, *y*) | int: *x* = left in ms  
int: *y* = right in ms | sets the region of interest (ROI) used for the baseline calculation. | --- | --- |
| SetDriftCorrectionROI(*x*, *y*) | int: *x* = left in s  
int: *y* = right in s | sets the left and right cursor positions used to calculate the drift correction. | --- | Data_eg.js |
| SetDriftCorrectionROIMilliSec(*x*, *y*) | int: *x* = left in ms  
int: *y* = right in ms | sets the left and right cursor positions used to calculate the drift correction. | --- | --- |
| SetAnalysisROI(*x*, *y*) | int: *x* = left in s  
int: *y* = right in s | sets the region of interest (ROI) from which results are calculated | --- | Data_eg.js |
| SetAnalysisROIMilliSec(*x*, *y*) | int: *x* = left  
int: *y* = right in ms | sets the region of interest (ROI) from which results are calculated | --- | --- |
## 1.3.7 Robo2. Database Commands

<table>
<thead>
<tr>
<th>Robo2.</th>
<th>Parameter(s)</th>
<th>Action</th>
<th>Cond.</th>
<th>Example Script</th>
</tr>
</thead>
</table>
| TransmitRecording(x) | REC_TAG_VOLTAGE, REC_TAG_COMPOUND, REC_TAG_REF_VOLTAGE, REC_TAG_REF_COMPOUND | Writes recording tag to database.  
*REC_TAG_REF_COMPOUND*: not used for generating DR curves  
*REC_TAG_COMPOUND*: used for generating DR curves | R+    | valves_tags_eg.js |
| TransmitVoltage(x) | int: value                                       | writes voltage value for the current recording to the database        | R+    |                |
| TransmitCompoundValve(x) | int: valve number                              | writes the valve number to the database                               | R+    | valves_tags_eg.js |
| TransmitCompoundGilson(x,y) | int: x = slot  
int: y = tube                             | writes the Gilson slot/tube information to the database               | R+    | ---            |
| SetWellInfo(x,y)   | string: x = key  
string: y = value                               | writes any user defined text to the database. Can be used to write additional information, e.g. value of electrode resistance or leak current to the database | W+    | ---            |
| SetRecordingSeries(x) | int: series number                              | writes a series number to the database, valid for all following recordings. If one trace of the series is selected in Roboocyte2+, all other traces from the series are automatically used for plotting the dose-response curve. | R-    | ---            |
### 1.3.8 Robo2. Timing Commands

<table>
<thead>
<tr>
<th>Robo2.</th>
<th>Parameter(s)</th>
<th>Action</th>
<th>Cond.</th>
<th>Example Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait(x)</td>
<td>int: time</td>
<td>waits for x seconds before executing the next command</td>
<td></td>
<td>---</td>
</tr>
<tr>
<td>WaitMilliSec(x)</td>
<td>int: time</td>
<td>waits for x milliseconds before executing the next command</td>
<td></td>
<td>---</td>
</tr>
<tr>
<td>StartTimer()</td>
<td>---</td>
<td>starts timer, see also variables TIME and TIME_S</td>
<td></td>
<td>---</td>
</tr>
<tr>
<td>WaitForTimer(x)</td>
<td>int: time</td>
<td>waits until x seconds from timer start have been passed.</td>
<td></td>
<td>---</td>
</tr>
</tbody>
</table>

### 1.3.9 Robo2. Liquid Handling (Roboflow) Commands

<table>
<thead>
<tr>
<th>Robo2.</th>
<th>Parameter(s)</th>
<th>Action</th>
<th>Cond.</th>
<th>Example Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenValve(x)</td>
<td>int: valve number</td>
<td>opens valve x, valve index must be 1 ... 12</td>
<td></td>
<td>valves_tags_eg.js</td>
</tr>
<tr>
<td>CloseAllValves()</td>
<td>---</td>
<td>closes all valves</td>
<td></td>
<td>valves_tags_eg.js</td>
</tr>
<tr>
<td>WastePumpOn(x)</td>
<td>int: speed</td>
<td>turns waste pump on, speed x must be between 0 and 20000</td>
<td></td>
<td>valves_tags_eg.js</td>
</tr>
<tr>
<td>WastePumpOff()</td>
<td>---</td>
<td>turns off waste pump</td>
<td></td>
<td>valves_tags_eg.js</td>
</tr>
<tr>
<td>IsWastePumpOn()</td>
<td>R: bool</td>
<td>returns true if waste pump is on</td>
<td></td>
<td>---</td>
</tr>
<tr>
<td>ValvePumpOn(x)</td>
<td>int: speed</td>
<td>turns valve pump on, speed x must be between 0 and 10000</td>
<td></td>
<td>valves_tags_eg.js</td>
</tr>
<tr>
<td>ValvePumpOff()</td>
<td>---</td>
<td>turns off valve pump</td>
<td></td>
<td>valves_tags_eg.js</td>
</tr>
<tr>
<td>IsValvePumpOn()</td>
<td>R: bool</td>
<td>returns true if valve pump is on</td>
<td></td>
<td>---</td>
</tr>
</tbody>
</table>
## 1.3.10 Robo2. High Level Commands

<table>
<thead>
<tr>
<th>Robo2.</th>
<th>Parameter(s)</th>
<th>Action</th>
<th>Cond.</th>
<th>Example Script</th>
</tr>
</thead>
</table>
| ResistanceCheck_I(a,b) | a = MIN_RESISTANCE_I  
b = MAX_RESISTANCE_I  
R: bool | checks if the resistance of the current electrode is between min and max (kOhm), returns true if this was the case | L+ | HL-commands_eg.js |
| ResistanceCheck_U(a,b) | a = MIN_RESISTANCE_U  
b = MAX_RESISTANCE_U  
R: bool | checks if the resistance of the voltage electrode is between min and max (kOhm), returns true if this was the case | L+ | HL-commands_eg.js |
| DCOffsetCorrection(a,b,c,d) | a = DCOFFSET_RANGE  
b = DCOFFSET_DELAY  
c = DCOFFSET_WAIT  
d = DCOFFSET_ATTEMPTS  
R: bool | checks if the offset of the I and U electrodes is within the range given by range (mV) delay: initial delay to wait until junction potential is established in seconds wait: wait time between repeats in seconds attempts: number of repeats. | L+ | HL-commands_eg.js |
| Impale(a,b,c,d) | a = MIN_RMP  
b = IMPALEMENT_STEPS  
c = IMPALEMENT_STEP  
d = IMPALEMENT_WAIT  
R: bool | Impalement procedure  
MIN_RMP: minimum membrane potential  
IMPALEMENT_STEPS: maximum number of steps to "find" MIN_RMP on both electrodes  
IMPALEMENT_STEP: single step size in µm  
IMPALEMENT_WAIT: wait time in seconds between steps returns true if impalement was successful | W+ | HL-commands_eg.js |
| InitialLeakCurrentCheck(a,b) | a = MIN_INITIAL_LEAKCURRENT  
b = MAX_INITIAL_LEAKCURRENT  
R: bool | initial leak current test, returns true if the measured leak current is between min and max (nA) | Oo+ | HL-commands_eg.js |
| LeakCurrentCheck(a,b,c,d) | a = MIN_LEAKCURRENT  
b = MAX_LEAKCURRENT  
c = LEAKCURRENT_WAIT  
d = LEAKCURRENT_ATTEMPTS  
R: bool | leak current test, returns true if the measured leak current is between min and max (nA). Waits c seconds before determining the leak current and retries d times | Oo+ | HL-commands_eg.js |
### 1.3.11 RecDisplay. Recording Display Commands

<table>
<thead>
<tr>
<th>RecDisplay.</th>
<th>Parameter(s)</th>
<th>Action</th>
<th>Cond.</th>
<th>Example Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear()</td>
<td>---</td>
<td>clears the recording display</td>
<td>---</td>
<td>valves_tags_eg.js</td>
</tr>
<tr>
<td>SetMode(x)</td>
<td>x = DISP_SINGLE or DISP_OVERLAY</td>
<td>sets recording display to overlay or single trace mode</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>SetXAxis(x,y)</td>
<td>int: min int: max</td>
<td>sets x-axis range from min to max in seconds</td>
<td>---</td>
<td>valves_tags_eg.js</td>
</tr>
<tr>
<td>SetXAxisMilliSec(x,y)</td>
<td>int: min int: max</td>
<td>sets x-axis range from min to max in milliseconds</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>SetYAxis(x,y)</td>
<td>int: min int: max</td>
<td>set y axis range from min to max in nA</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>ZoomToFitX()</td>
<td>---</td>
<td>zooms the time axis to the actual data trace</td>
<td>---</td>
<td>Acquisition_eg.js</td>
</tr>
<tr>
<td>ZoomToFitY()</td>
<td>---</td>
<td>zooms the current axis to the actual data trace</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>RecDisplay.TrackYMax()</td>
<td>bool: true or false</td>
<td>true or false switches the maximum current tracking on or off</td>
<td>---</td>
<td>valves_tags_eg.js</td>
</tr>
<tr>
<td>RecDisplay.TrackYMin()</td>
<td>bool: true or false</td>
<td>true or false switches the minimum current tracking on or off</td>
<td>---</td>
<td>valves_tags_eg.js</td>
</tr>
</tbody>
</table>

### 1.3.12 ControlDisplay. Control Display Command

<table>
<thead>
<tr>
<th>ControlDisplay.</th>
<th>Parameter(s)</th>
<th>Action</th>
<th>Cond.</th>
<th>Example Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear()</td>
<td>---</td>
<td>clears the control display</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>SetXAxis(x,y)</td>
<td>int: min int: max</td>
<td>sets x-axis range from min to max in seconds</td>
<td>---</td>
<td>valves_tags_eg.js</td>
</tr>
<tr>
<td>SetYAxis_I(x,y)</td>
<td>int: min int: max</td>
<td>sets current axis range from min to max in seconds</td>
<td>---</td>
<td>valves_tags_eg.js</td>
</tr>
<tr>
<td>SetYAxis_U(x,y)</td>
<td>int: min int: max</td>
<td>sets voltage axis range from min to max in seconds</td>
<td>---</td>
<td>valves_tags_eg.js</td>
</tr>
<tr>
<td>ZoomToFitX()</td>
<td>---</td>
<td>zooms the time axis to the actual data trace</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>ZoomToFitY_U()</td>
<td>---</td>
<td>zooms the voltage axis to the actual data trace</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
## 1.3.13 Gilson. Gilson Commands

<table>
<thead>
<tr>
<th>Gilson.</th>
<th>Parameter(s)</th>
<th>Action</th>
<th>Cond.</th>
<th>Example Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoveToTube(x,y)</td>
<td>int: x = slot int: y = tube</td>
<td>moves to tube in slot 1 .. 5, tube depending on selected rack</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>MoveToPort(x)</td>
<td>int: port number (1 or 2)</td>
<td>moves to transfer ports</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>MoveToRinse()</td>
<td>---</td>
<td>moves to the rinse station</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>MoveToDrain()</td>
<td>---</td>
<td>moves to the drain station</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>MoveUp()</td>
<td>---</td>
<td>moves the Gilson probe up</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>MoveHome()</td>
<td>---</td>
<td>moves all to home position</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Reset()</td>
<td>---</td>
<td>resets the Gilson</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>PumpBackward(x)</td>
<td>int: speed</td>
<td>starts Gilson peristaltic pump in backward direction with speed x</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>PumpForward(x)</td>
<td>int: speed</td>
<td>starts Gilson peristaltic pump in forward direction with speed x</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Gilson.PumpStop()</td>
<td>---</td>
<td>stops Gilson peristaltic pump</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Valve1On()</td>
<td>---</td>
<td>opens valve 1 from transfer port</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Valve1Off()</td>
<td>---</td>
<td>closes valve 1 from transfer port</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Valve2On()</td>
<td>---</td>
<td>opens valve 2 from transfer port</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Valve2Off()</td>
<td>---</td>
<td>closes valve 1 from transfer port</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
Example Script "Dose-Response"

In the following chapter, you will learn how a typical recording script is built. As an example we will use the script: "dose-response.js" which can be found on the Roboocyte Setup CD. Another example script which can be found on the Roboocyte CD is the script "standard_procedures.js". This script only includes the protocol independent part without the recording protocol part and without user-defined variables.

The following chapters describe a typical recording protocol sequence step-by-step:

1. Defining variables and parameter uses in the script
2. Starting the oocyte loop
3. Compensating electrode DC voltage offsets
4. Testing electrode resistance and voltage offsets
5. Oocyte impalement
6. Starting voltage-clamp and leak-current test
7. Executing the experimental protocol
8. Continue with next selected oocyte

2.1 Defining Dialogue variables

2.1.1 User Defined Dialogue Variables

The first part in a script should be used to define all necessary variables, such as holding potential, incubation times, number of used valves etc. If these parameters are defined as dialogue variables, they can be easily modified in a dialogue before starting the script recording.

```javascript
Robo2.log("Script " + SCRIPT_FILE);

// Definition of user defined variables
Robo2.SetDialogVariable("pre_agonist_s", 5, "Time before agonist application (s)");
Robo2.SetDialogVariable("agonist_s", 5, "Agonist application time (s)");
Robo2.SetDialogVariable("recoded_washout_t", 10, "Recorded wash-out time (s)");
Robo2.SetDialogVariable("nonrecorded_washout_t", 0, "Non-recorded wash time (s)");
Robo2.SetDialogVariable("preagonist_thickness", 0.01, "Thickness pump (10 - 10000)");
Robo2.SetDialogVariable("imp_steps", 50, "Maximum Number of Impalement Steps");
Robo2.SetDialogVariable("jump_step", 50, "Single Impalement Step Distance in Micrometer");
Robo2.SetDialogVariable("minmem", -6, "Minimum Membrane Potential after Impalement");
Robo2.SetDialogVariable("compvolage", -60, "Recording Voltage in mV");
Robo2.SetDialogVariable("minresponse", -100, "Minimum response in the expression test to continue the recording (mA)");
```

The command for defining a user defined (dialogue) variable has the following format:

```javascript
Robo2.SetDialogVariable("variable_name", variable value, "comment");
```

I.e. the command "Robo2.SetDialogVariable("pre_agonist_s", 5, "Time before agonist application (s)");" creates the variable "pre_agonist_s" with the default value "5" and the comment "Time before agonist application (s)"

The command "Robo2.ShowDialog();" opens a window after starting the script listing all user defined variables. All values can be changed here, but are only valid for a single script execution. Please note that whenever you start the script a second time, default values will be reloaded.
2.1.2 Working with Pre-defined Variables

Pre-defined variables are already existing variables with predefined default variable names and values (see table in chapter 1.2.3).

```c
Set_SAMPLE_RATE(500)
Set_AMPLIFIER_GAIN_P(1000)
Set_AMPLIFIER_GAIN_I(100)
Set_DC_OFFSET_RANGE(3);
Set_DC_OFFSET_DELAY(15);
Set_DC_OFFSET_WAIT(5);
Set_MIN_RANGE(minrange);
Set_IMPLEMENT_STEPS[imp_steps]
Set_IMPULSE_STEP[imp_step]
Set_IMPULSE_WAIT[1]
Set_MIN_INITIAL_IACURRENT(-10000)
Set_MAX_INITIAL_IACURRENT(500)
Set_MIN_IACURRENT(-1000)
Set_MAX_IACURRENT(100)
Set_IACURRENT_WAIT[10]
Set_IACURRENT_ATTEMPTS(3)
Set_MIN_RESISTANCE_I(100)
Set_MAX_RESISTANCE_I(1500)
Set_MIN_RESISTANCE_II(100)
Set_MAX_RESISTANCE_II(1500)
Set_MIN_RESISTANCE_III(100)
Set_MAX_RESISTANCE_III(1500)
Set_MIN_RESISTANCE_IV(100)
Set_MAX_RESISTANCE_IV(1500)
```

// This opens the dialogue to change values for the built-in variables
Robo2.ShowStandardDialog()
The command "Robo2.ShowStandardDialog();" opens a window after starting the script listing all predefined variables. All values can be changed here, but again changes are not permanent and only valid for a single script execution. If you want to change values “permanently” you have to change them in the script text.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN RESISTANCE I</td>
<td>100</td>
<td>minimum TEVC probe resistance of the I electrode</td>
</tr>
<tr>
<td>MAX RESISTANCE I</td>
<td>1500</td>
<td>maximum TEVC probe resistance</td>
</tr>
<tr>
<td>MIN RESISTANCE U</td>
<td>100</td>
<td>minimum TEVC probe resistance of the U electrode</td>
</tr>
<tr>
<td>MAX RESISTANCE U</td>
<td>1500</td>
<td>maximum TEVC probe resistance</td>
</tr>
<tr>
<td>DC OFFSET RANGE</td>
<td>3</td>
<td>max deviation of DC offset from 0 [mV]</td>
</tr>
<tr>
<td>DC OFFSET_DELAY</td>
<td>15</td>
<td>delay before DC offset measurement [sec]</td>
</tr>
<tr>
<td>DC OFFSET_WAIT</td>
<td>5</td>
<td>wait after each check [sec]</td>
</tr>
<tr>
<td>DC OFFSET ATTEMPTS</td>
<td>3</td>
<td>number of attempts to try DC offset check</td>
</tr>
<tr>
<td>MIN RMP</td>
<td>45</td>
<td>minimum membrane potential [mV]</td>
</tr>
<tr>
<td>IMPALEMENT STEPS I</td>
<td>6</td>
<td>number of z axis steps to move down during impalement of the I...</td>
</tr>
<tr>
<td>IMPALEMENT STEPS U</td>
<td>2</td>
<td>number of z axis steps to move down during impalement of the U...</td>
</tr>
<tr>
<td>IMPALEMENT STEPS</td>
<td>10</td>
<td>maximum number of z axis steps to move down during impalement</td>
</tr>
<tr>
<td>IMPALEMENT_STEP</td>
<td>50</td>
<td>step size of impalement step [nm]</td>
</tr>
<tr>
<td>IMPALEMENT_WAIT</td>
<td>1</td>
<td>wait after each z axis step [sec]</td>
</tr>
<tr>
<td>IMPALEMENT_current</td>
<td>1000</td>
<td>minimum load current initial step of DC offset</td>
</tr>
</tbody>
</table>

The next section in the script - Initialization - uses some of the predefined variables.

```java
79 // ^ initialization - setting sample rate and amplifier gain values
80 // ------
81 Robo2.SetAmplifierCoefficients(AMPLIFIER_GAIN_P, AMPLIFIER_GAIN_I);
82 RecDisplay.Clear();
83 Robo2.SetSampleRate(SAMPLERATE);
```

After defining variables and their respective values, the "oocyte loop" can be initiated.

### 2.2 The Oocyte Loop

The most basic function of the Roboocyte is to move the plate carrier "from one well to the other", or more correct to move selected wells exactly below the measuring head. This movement between well 1 and 95 is initiated and controlled by the so-called "oocyte loop", a for-loop in java script.
The oocyte loop itself has only 4 lines (lines 89, 90, 91 and 105 in the example shown above), but, before terminating the oocyte loop by the final curly bracket, do not forget to stop the valve pump to avoid flooding (lines 101, 102 and 103)

### 2.3 Standard Routines executed before Recording Protocol

Before starting the recording of data, a number of steps and routines have to be started which usually will not change for different kind of recording protocols.

#### 2.3.1 Moving the Measuring Head into Liquid

The oocyte loop cares for incrementing the well number, the movement of carrier and z-axis have to be controlled by specific commands.

```csharp
for (var i = 0; i < Robo2.SelectedWells.Count; i++)
{
    var WellIndex = Robo2.SelectedWells[i];

    // Electrode offset compensation, Impalement, Leak Current Checks etc.

    // Protocol specific part

    Robo2.ValvePumpOff();
    Robo2.CloseAllValues();
    Robo2.WastePumpOff();
}
```

Before moving the measuring head into liquid, one should switch the waste pump on to guarantee a constant buffer level relative to the electrodes. This is important to avoid offset artifacts during oocyte impalement.

- `Robo2.WastePumpOn(pumpspeed*ratio/100);` starts the waste pump.
- The waste pump "speed" is defined by the values for `pumpspeed` and `ratio` as defined before.
- `Robo2.MoveToWell(WellIndex);` moves the carrier to the respective well.
- `Robo2.ZMoveToLiquid();` moves the z-axis down into the well.

The default liquid position can be changed in the Settings/Options menu of the Roboocyte software (default is 2000 µm above the well bottom).
### 2.3.2 Switching to Current Clamp Mode

After moving the electrodes into liquid, the amplifier is set to Current Clamp Mode.

**Robo2.SetHoldingCurrent(0);** sets holding current to 0 nA

**Robo2.SetCurrentClamp();** sets the amplifier to current clamp mode

```csharp
112 ControlDisplay.SetYAxis(0, 30);  
113 ControlDisplay.SetYAxis_U(-100, 10);  
114 ControlDisplay.SetYAxis_I(-50, 0);  
115
116 // Starting Current Clamp mode  
117 Robo2.Log("Current clamp at 0 nA");  
118 Robo2.SetHoldingCurrent(0);  
119 Robo2.SetCurrentClamp();  
```

Before starting control recording, scaling of the Control Display axes is performed. Units are seconds, mV and nA, respectively.

### 2.3.3 Electrode Offset Compensation

After starting current clamp, control recording is started in order to perform electrode offset compensation, to determine electrode resistances, and to perform the impalement of the oocyte.

Control recording means that data are displayed in the Control Display, but not saved to disk.

**Robo2.StartControlRecording();** starts the control recording followed by the electrode offset compensation.

```csharp
120 Robo2.StartControlRecording();  
121 // Electrode offset reset  
122 Robo2.Log("Electrode reset ...");
123
124 if (!Robo2.DCOffsetCorrection(DCOFFSET_RANGE, DCOFFSET_DELAY, DCOFFSET_WAIT, DCOFFSET_ATTEMPTS)) {  
125      continue;
126    }  
```

The offset compensation for both electrodes is performed by executing the command

**Robo2.DCOffsetCorrection(DCOFFSET_RANGE, DCOFFSET_DELAY, DCOFFSET_WAIT, DCOFFSET_ATTEMPTS)**

Variables for the offset compensation are by default the designated predefined variables, but you can also use numbers, such as **Robo2.DCOffsetCorrection(3, 10, 5, 3)** (see table in chapter 1.2.3)

The command is embedded in an if-loop, controlling what happens after the compensation: If the electrode offset compensation was successful the script proceeds, if not, the measuring head moves to the next oocyte.

The if loop is terminated by "continue" if the electrode offset compensation fails. "Continue" means that the script jumps back to the start of the respective loop (oocyte loop in this case).

### 2.3.4 Electrode Resistance Test

After compensating the electrode offsets, the resistance of both electrodes should be tested with the command

**Robo2.ResistanceCheck_I(MIN_RESISTANCE_I, MAX_RESISTANCE_I),**

where **MIN_RESISTANCE** and **MAX_RESISTANCE** are predefined variables (see table in chapter 1.2.3).
Roboocyt2 Manual

```python
if (Robo2.ResistanceCheck_I(MIN_RESISTANCE_I, MAX_RESISTANCE_I))
    {
    }
else
    (Robo2.StopControlRecording();
    continue;
    }
if (Robo2.ResistanceCheck_U(MIN_RESISTANCE_U, MAX_RESISTANCE_U))
    {
    }
else
    (Robo2.StopControlRecording();
    continue;
    }
Robo2.StopControlRecording();
``` 

If the electrode resistances are within the range (between MIN_RESISTANCE and MAX_RESISTANCE) the script proceeds; if not (else), the script breaks and continues with the next oocyte.

Now, after completing electrode offset compensation, oocyte impalement can be started.

### 2.3.5 Oocyte Impalement

The oocyte impalement is performed by using the command:

**Robo2.Impale(MIN_RMP, IMPALEMENT_STEPS, IMPALEMENT_STEP, IMPALEMENT_WAIT)**

MIN_RMP, IMPALEMENT_STEPS, IMPALEMENT_STEP and IMPALEMENT_WAIT are predefined variables (see table in chapter 1.2.3).

The impalement process is started by a movement of the z-axis to the default oocyte impalement depth (default = 800 µm above the well plate bottom). Then n = IMPALEMENT_STEPS are performed with a step size of \( \Delta z = \text{IMPALEMENT_STEP} \) and a waiting time \( t = \text{IMPALEMENT_WAIT} \) between individual steps.

Default values for IMPALEMENT_STEPS and IMPALEMENT_STEP are 8 and 50 µm, respectively.

```python
Robo2.Log("Starting Oocyte Impalement...");
Robo2.StartControlRecording();
Robo2.Wait(2);
ControlDisplay.SetXAxis(0, 30)
ControlDisplay.SetYAxis_U(-63, 10)
Robo2.Wait(5)
if (!Robo2.Impale(MIN_RMP, IMPALEMENT_STEPS, IMPALEMENT_STEP, IMPALEMENT_WAIT))
    {
    Robo2.Log("impalement failed, " + MIN_RMP + " not reached --> next oocyte");
    Robo2.ValvePumpOff();
    Robo2.CloseAllValves();
    Robo2.WastePumpOFF();
    continue;
    }
```
If the impalement was not successful, i.e. if the membrane potential \( U = \text{MIN\_RMP} \) was not reached after \( n = \text{IMPALEMENT\_STEPS} \) steps with the step distance \( d = \text{IMPALEMENT\_STEP} \) the run is continued with the next oocyte. Before moving to the next oocyte pumps and valves should be closed (lines 170 - 172). \text{IMPALEMENT\_WAIT} is the waiting time between individual steps of the z-axis in seconds.

If impalement was successful, the script continues after line 175. It usually makes sense to wait at least for 10 seconds after impalement to give the oocyte membrane enough time to recover from the impalement.

2.3.6 Starting Voltage Clamp Mode

After successful impalement, the recording can be continued under voltage clamp. Before, scaling of the Control Display axes should be changed to meet the demands of the following Leak Current Test under voltage clamp.

```csharp
Robo2.SetHoldingVoltage(clampvoltage);
```

"clampvoltage" is a user defined variable created at the beginning of the script.

```csharp
Robo2.SetVoltageClamp();
```

2.3.7 Initial Leak Current Test

After starting voltage clamp, a control recording is started in order to perform the initial leak current check.

```csharp
Robo2.StartControlRecording();
Robo2.Wait(5);

if (!Robo2.InitialLeakCurrentCheck(MIN_INITIAL_LEAKCURRENT, MAX_INITIAL_LEAKCURRENT))
{
    Robo2.Log("Proceeding with next oocyte");
    Robo2.ValvePumpOff();
    Robo2.CloseAllValves();
    Robo2.WaitPumpOff();
    continue;
}
```

The leak current check is started by the command

```csharp
Robo2.InitialLeakCurrentCheck(MIN_INITIAL_LEAKCURRENT, MAX_INITIAL_LEAKCURRENT)
```
where \texttt{MIN\ INITIAL\ LEAKCURRENT} and \texttt{MAX\ INITIAL\ LEAKCURRENT} are predefined variables with default values -10000 and +200, respectively. If the leak current is within the range the script continues after line 209; if not, the run continues with the next selected oocyte. Again, commands for stopping pumps and valves should be included (lines 205 - 207).

### 2.3.8 Final Leak Current Test with Perfusion

After the successful initial leak current check, i.e. if the oocyte is not completely leaky, the script continues with the final leak current check. Although you can start this final check directly after the initial leak current test, we recommend to start the buffer perfusion of the oocyte beforehand in order to check the viability of the oocyte under perfusion.

```python
Robo2.Log("Starting Perfusion ...");
Robo2.Wait(5);
Robo2.OpenValve(1);
Robo2.ValvePumpOn(pumpspeed);
Robo2.Wait(5);
```

Then, after a waiting time of 5 seconds, the final leak current check can be initiated by the command

```python
Robo2.LeakCurrentCheck(MIN\ LEAKCURRENT, MAX\ LEAKCURRENT, LEAKCURRENT\ ATTEMPTS, LEAKCURRENT\ WAIT)
```

where \texttt{MIN\ LEAKCURRENT}, \texttt{MAX\ LEAKCURRENT}, \texttt{LEAKCURRENT\ ATTEMPTS} and \texttt{LEAKCURRENT\ WAIT} are predefined variables. Default values for \texttt{MIN\ LEAKCURRENT} and \texttt{MAX\ LEAKCURRENT} are -1000 nA and 100 nA, respectively. Default values for \texttt{LEAKCURRENT\ ATTEMPTS} and \texttt{LEAKCURRENT\ WAIT} are 3 and 10 seconds, respectively. This means that the leak current is determined up to 3 times with a waiting time of 10 seconds in between.

```python
Robo2.Log("Final leak current check minimum = " + MIN\ LEAKCURRENT + " nA ...");
if (!Robo2.LeakCurrentCheck(MIN\ LEAKCURRENT, MAX\ LEAKCURRENT, LEAKCURRENT\ ATTEMPTS, LEAKCURRENT\ WAIT))
    Robo2.Log("leak current not in range");
    Robo2.ValvePumpOff();
    Robo2.CloseValves();
    Robo2.WastePumpOff();
    continue;
```

As soon as the leak current is within the given limits, the script continues after line 237. if the cell is still too leaky after 3 trials, the script continues with moving to the next well.

After successful leak test, the control recording has to be stopped, and the result of final leak test can be sent into the log window.

```python
Robo2.StopControlRecording();
Robo2.Log("... Leak current with perfusion OK = " + Robo2.VALUE + " nA.");
Robo2.Log(" ");
Robo2.Wait(5)
```

Now, after finishing all necessary preparations and tests, the specific recording protocol can be commenced.
3 Recording Protocol Examples

3.1 Using the Dose-Response Script as a Template

The example script "dose-response.js" is a typical dose-response recording protocol which can be used as a template for all kind of dose-response like protocols. The recording protocol specific part is located between line 254 and 365.

```plaintext
// ************************************************************************************************************
// Protocol specific part starts here
// ************************************************************************************************************


// ************************************************************************************************************
// Protocol specific part ends here
// ************************************************************************************************************
```

The part of the script outside of "Protocol specific part starts here/ Protocol specific part stops here" comment can be used as a starting-point for your own scripts.

3.2 Ligand-gated Channels and Electrogenic Transporters

The example script "dose-response.js" is a typical dose-response recording protocol for ligand-gated ion channels. In addition, it can be used - after minor modifications - for electrogenic transporters, such as Na-pump, GATs, EAATs etc.

3.2.1 Expression Test

Before you start a time and compound-consuming protocol on an oocyte, it makes sense to test whether the expression i.e. the response to an agonist reference concentration is adequate. Therefore, you should usually perform an expression test before continuing the protocol.

Before the expression test is started, it is mandatory to define the ranges of baseline ROI (Region Of Interest) and analysis ROI, because these ROIs will be used for the calculation of the baseline subtracted response for expression test. They will also be used for the later analysis in Roboocyte2+. Although the ROI positions should be defined properly already in the script, they can be changed later in Roboocyte2+.

```plaintext
Robo2.Log("Starting expression test with reference compound from value " + reference + ";
RecDisplay.SetZAxis(0, pre_agonist_s + agonist_s + recorded_washout_s);
RecDisplay.TrackYMax(true);
RecDisplay.TrackYMin(true);
Robo2.SetBaselineROI(pre_agonist_s - 3, pre_agonist_s - 1);
Robo2.SetAnalysisROI(pre_agonist_s, pre_agonist_s + agonist_s);
ControlDisplay.SetZAxis(0, pre_agonist_s + agonist_s + recorded_washout_s);
ControlDisplay.SetXAxis(-10000, 10000);
```

The user-defined variables `pre_agonist_s`, `agonist_s` and `recorded_washout_s` used later for controlling the recording time and solution exchange are also used for defining the ROIs and the time axis scaling with the effect that ROI boundaries and axis change accordingly when you change variable values at script start. If the ROI boundaries are coupled to these variables, they automatically change whenever changes to these variables are made.

Example:

```plaintext
RecDisplay.SetXAxis(0, pre_agonist_s + agonist_s + recorded_washout_s);
```
Robo2.SetBaselineROI(pre_agonist_s - 3, pre_agonist_s - 1);
Robo2.SetAnalysisROI(pre_agonist_s, pre_agonist_s + agonist_s);
ControlDisplay.SetXAxis(0, pre_agonist_s + agonist_s + recorded_washout_s)
ControlDisplay.SetYAxis_I(-10000, 100)

Above JavaScript commands lead to durations listed in the following table.

<table>
<thead>
<tr>
<th>Variable values (s)</th>
<th>Example A</th>
<th>Example B</th>
<th>Example C</th>
</tr>
</thead>
<tbody>
<tr>
<td>RecDisplay x-axis (s)</td>
<td>65</td>
<td>50</td>
<td>120</td>
</tr>
<tr>
<td>Baseline ROI (s)</td>
<td>2 - 4</td>
<td>7 - 9</td>
<td>17 - 19</td>
</tr>
<tr>
<td>Analysis ROI (s)</td>
<td>5 - 35</td>
<td>10 - 30</td>
<td>20 - 60</td>
</tr>
<tr>
<td>ControlDisplay x-axis</td>
<td>65</td>
<td>50</td>
<td>120</td>
</tr>
</tbody>
</table>

After these definitions, the expression test can be initiated.

```javascript
Robo2.SetBaselineROI(pre_agonist_s - 3, pre_agonist_s - 1);
Robo2.SetAnalysisROI(pre_agonist_s, pre_agonist_s + agonist_s);
ControlDisplay.SetXAxis(0, pre_agonist_s + agonist_s + recorded_washout_s)
ControlDisplay.SetYAxis_I(-10000, 100)

Above JavaScript commands lead to durations listed in the following table.

<table>
<thead>
<tr>
<th>Variable values (s)</th>
<th>Example A</th>
<th>Example B</th>
<th>Example C</th>
</tr>
</thead>
<tbody>
<tr>
<td>RecDisplay x-axis (s)</td>
<td>65</td>
<td>50</td>
<td>120</td>
</tr>
<tr>
<td>Baseline ROI (s)</td>
<td>2 - 4</td>
<td>7 - 9</td>
<td>17 - 19</td>
</tr>
<tr>
<td>Analysis ROI (s)</td>
<td>5 - 35</td>
<td>10 - 30</td>
<td>20 - 60</td>
</tr>
<tr>
<td>ControlDisplay x-axis</td>
<td>65</td>
<td>50</td>
<td>120</td>
</tr>
</tbody>
</table>
```

After starting the recording, two transmit commands are executed to send the right information about the compound used to the database and to link this information to the actual recording.

Robo2.TransmitRecording(REC_TAG_REF_COMPOUND); tags the recording as a reference recording which means that the result will not be used for dose-response fitting in Roboocyte2+.

Robo2.TransmitCompoundValve(reference); transmits the compound name and concentration to the database. This of course only works when the right entries have been made to the Liquid Configuration in the Roboocyte2 program.

pre_agonist_s seconds after recording start, the valve reference is opened. Then after agonist_s seconds valve 1 is opened and after another recorded_washout_s seconds the recording is stopped.

Then, after calculating and typing the baseline-subtracted response into the log window,
the response can be compared to the values assigned to the user-defined variables `minResponse` and `maxResponse` the result of which determines whether the script continues at line 297 or is stopped and continued with the oocyte from the next well.

Example 1: Negative response expected (inward current)

**Condition:** if (signal > minResponse || signal < maxResponse)

maxResponse = -10000

minResponse = -200

signal = -100 ==> signal > minResponse ==> next oocyte

signal = -12000 ==> signal < maxResponse ==> next oocyte

signal = -5000 ==> signal > maxResponse and signal < minResponse ==> script proceeds

Example 2: Positive response expected

If the expected response is positive (outward current) operators < and > have to be exchanged.

**Condition:** if (signal < minResponse || signal > maxResponse)

maxResponse = 10000

minResponse = 200

signal = 100 ==> signal < minResponse ==> next oocyte

signal = 12000 ==> signal > maxResponse ==> next oocyte

signal = 5000 ==> signal < maxResponse and signal > minResponse ==> script proceeds

3.2.2 Dose-Response Protocol

After the oocyte has been tested successfully for sufficient expression and stability, the recording protocol continues with the dose-response recording part of the script. Before the dose-response recording starts, the usual axis scaling and ROI definition commands are applied.

If the application of different agonist concentrations steps sequentially from valve a to valve a + 1 a for loop can be used.
The loop variable is the valve number, starting with the user defined variable first_compound and ending with last_compound.

If needed, a leak test can be placed before the application of every agonist application.

```javascript
for (var valve = first_compound; valve <= last_compound; valve++)
{
    //
}
```

If the leak test fails, the message "leak current is not in range - next oocyte" is sent to the log window, all pumps and valves are switched off and the protocol continues with the next selected oocyte. The break command in line 331 terminates the execution of the loop and the script is continued after the end of the loop at line 365. The curly bracket in line 368 designates the end of the oocyte loop, which means that the script will continue with the next selected well/oocyte at line 89.

```javascript
Robot2.StopControlRecording();
```

If the leak test fails, the message "leak current is not in range - next oocyte" is sent to the log window, all pumps and valves are switched off and the protocol continues with the next selected oocyte. The break command in line 331 terminates the execution of the loop and the script is continued after the end of the loop at line 365. The curly bracket in line 368 designates the end of the oocyte loop, which means that the script will continue with the next selected well/oocyte at line 89.

```javascript
Robot2.ValvePumpOff();
Robot2.CloseAllValves();
Robot2.WastePumpOff();
Robot2.StopControlRecording();
```

### 3.3 Voltage-gated Ion Channels

Because the external trigger for activating voltage-gated ion channels is usually a change in membrane potential and not application of an agonist, protocols usually concentrate on voltage changes instead of solution exchanges. Of course, voltage protocols and solution exchanges still can be combined in a script; e.g. to modify the activity of voltage-gated ion channels by application of different compounds.

#### 3.3.1 Expression Test

The detailed design of an expression test will depend on the channel properties, but will always have a qualitatively similar structure: A voltage step from a potential at which the channels are inactive to a potential which activates the channels will be applied. The script "expression_test_vgic_part.js" used here as an example can be found on the Roboocyte2 CD.

First, new variables should be defined. This part can be moved to the beginning of the script.

```javascript
Robot2.SetDialogVariable("pre_pulse_ms", 100, "Prepulse for expression test (ms)");
Robot2.SetDialogVariable("pulse_ms", 200, "Pulse duration for expression test (ms)");
Robot2.SetDialogVariable("post_pulse_ms", 200, "Post-pulse duration for expression test (ms)");
Robot2.SetDialogVariable("voltage_rest", -60, "Resting Voltage for expression test (mV)");
Robot2.SetDialogVariable("voltage_act", 0, "Activating Voltage for expression test (mV)");
```

Secondly, axis scaling and ROI position are set. Using the variables from above guarantees that scaling and ROI automatically change whenever the pulse timing is changed.
The analysis ROI covers the whole duration of the voltage (test) pulse except the first 10 ms in order to avoid interference with the capacitive transient caused by the voltage jump.

Next, the recording is started.

```c
RecDisplay.SetXAxisMilliSec(0, pre_pulse_ms + pulse_ms + post_pulse_ms);
RecDisplay.TrackBarMaxY(true);
RecDisplay.TrackBarMin(true);
Robo2.SetAnalysisROIMillisec(pre_pulse_ms + 10, pre_pulse_ms + pulse_ms);
```

The sequence generates the following voltage jump:

- a. Start the recording
- b. 100 ms at -60 mV
- c. 200 ms at +50 mV
- d. 200 ms at -60 mV
- e. Stop the recording

Finally, the current amplitude elicited by the (in this example depolarizing) voltage pulse is sent to the log window, and

```c
Robo2.Log(" ");
signal = Robo2.MAXIMUM
Robo2.Log("Response to reference voltage pulse to " + voltage_et + " mV = " + signal + " nA");
Robo2.Log(" ");
```

the signal is compared to the user-defined limits minResponse and maxResponse. If the elicited signal is positive (outward current), the condition

```c
if (signal < minResponse || signal > maxResponse)
```

should be used.

```c
if (signal < minResponse || signal > maxResponse)
    Robo2.Log("Response (" + signal + " nA) not in Range (" + minResponse + ", " + maxResponse + "), go to next cycle");
Robo2.Wait(0);
Robo2.ValvePumpOff();
Robo2.CloseAllValves();
Robo2.WaitPumpOff();
continue;
```

If the expected signal is negative current (inward current) the operators > and < have to be exchanged.

**Voltage Protocols (IV-dependencies)**
Although voltage step protocols can be defined line by line with the script language, voltage protocols should be generated by the built-in graphical user interface (GUI). Please refer to the Roboocyte2 manual for details.
## 4 Using the Gilson Liquid Handler

Commands controlling the Gilson liquid handler can be separated in movement commands, commands controlling the peristaltic pump connected to the Gilson liquid handler, and commands controlling the magnetic valves of the Gilson's transfer ports. If you are using the liquid handler, scripts have to be modified, using Gilson commands instead of Roboflow commands.

**Important note:** Racks (and corresponding slots) used in a specific script have to be defined and selected by means of the Liquid Configuration feature of the Roboocyte software. Otherwise the script will not work.

### 4.1 Movement Commands

<table>
<thead>
<tr>
<th>Gilson.</th>
<th>Parameter(s)</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>MoveToTube(x,y)</td>
<td>int: x = slot int: y = tube</td>
<td>moves the probe to tube y in slot x (1 .. 5); the number of available tubes depends on the type of rack</td>
</tr>
<tr>
<td>MoveToPort(x)</td>
<td>int: port number (1 or 2)</td>
<td>moves probe to transfer ports x (1 or 2)</td>
</tr>
<tr>
<td>MoveToRinse()</td>
<td>---</td>
<td>moves probe to the rinse station</td>
</tr>
<tr>
<td>MoveToDrain()</td>
<td>---</td>
<td>moves probe to the drain station</td>
</tr>
<tr>
<td>MoveUp()</td>
<td>---</td>
<td>moves the Gilson probe up</td>
</tr>
<tr>
<td>MoveHome()</td>
<td>---</td>
<td>moves all to home position</td>
</tr>
<tr>
<td>Reset()</td>
<td>---</td>
<td>resets the Gilson</td>
</tr>
</tbody>
</table>

The "movement commands" move the Gilson probe (needle) in x, y, and z direction either to tubes, transfer ports, rinse station or drain station or just up (z-axis) or home (x = y = z = 0). The Reset command can be used to "reset" the machine, e.g. after some kind of malfunction, and is usually not used in a script.

### 4.2 Peristaltic Pump Commands

<table>
<thead>
<tr>
<th>PumpBackward(x)</th>
<th>int: x = speed</th>
<th>starts the Gilson peristaltic pump in backward direction with speed x</th>
</tr>
</thead>
<tbody>
<tr>
<td>PumpForward(x)</td>
<td>int: x = speed</td>
<td>starts the Gilson peristaltic pump in forward direction with speed x</td>
</tr>
<tr>
<td>Gilson.PumpStop()</td>
<td>---</td>
<td>stops the Gilson peristaltic pump</td>
</tr>
</tbody>
</table>

Whenever the Gilson liquid handler is used, the Minipulse peristaltic pump will be used. This pump will be then controlled by the Roboocyte software with the script commands listed above.

### 4.3 Transfer Port Valve Commands

<table>
<thead>
<tr>
<th>Valve1On()</th>
<th>---</th>
<th>opens valve 1 from transfer port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valve1Off()</td>
<td>---</td>
<td>closes valve 1 from transfer port</td>
</tr>
<tr>
<td>Valve2On()</td>
<td>---</td>
<td>opens valve 2 from transfer port</td>
</tr>
<tr>
<td>Valve2Off()</td>
<td>---</td>
<td>closes valve 2 from transfer port</td>
</tr>
</tbody>
</table>
Roboocyte2 Manual

The transfer ports offer you the option to work with larger solution volumes, e.g. for often used buffers or compounds. Using the transfer ports would mean moving the needle to the transfer port with the respective movement commands, opening the respective valves with the commands listed above, and finally switching the Minipulse pump on to aspirate solution from the transfer port.

4.4 Examples

Scripts that were written for using the Roboflow can be easily modified to work with the Gilson liquid handler instead. The following examples demonstrate how Roboflow solution exchange commands correspond to those of the Gilson liquid handler.

1. Opening a Roboflow valve corresponds to moving the Gilson needle into a tube or transfer port.

2. Turning on the Roboflow valve pump corresponds to starting the Minipulse pump in backward direction.

4.4.1 Differences between Roboflow and Gilson

1. Instead of using a manifold the Gilson liquid handler connects the different solution reservoirs and the measuring head with a single tubing. Therefore, an air gap has to be formed between two different solutions in the tubing to avoid mixing during transport from the probe to measuring head.

2. When using Roboflow, there is a delay of 2 to 3 s between the solution exchange command in the script and the arrival of the solution in the well (i.e. at the oocyte) when using pump speed 5000.

When using the Gilson liquid handler, this delay is much larger due to the tubing length between Gilson probe and measuring head. Depending on tubing length and the selected pump speed of the Minipulse pump, this delay can be tens of seconds and has to be considered in the design of the script.

4.4.2 Solution Exchange - Roboflow vs. Gilson

Defining a solution exchange sequence where solution1 runs for 5 seconds, followed by solution 2 for 10 seconds and finally solution 1 for 10 seconds will be realized by the following sequences of script commands.

The following table shows how Gilson commands correspond to Roboflow commands.

<table>
<thead>
<tr>
<th>Roboflow</th>
<th>Gilson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robo2.OpenValve(1);</td>
<td>Gilson.MoveToTube(1, 1);</td>
</tr>
<tr>
<td>Robo2.ValvePumpOn(5000);</td>
<td>Gilson.PumpBackward(2600);</td>
</tr>
<tr>
<td>Robo2.Wait(5);</td>
<td>Robo2.Wait(5);</td>
</tr>
<tr>
<td>solution separating air gap (Gilson only)</td>
<td>Gilson.PumpStop();</td>
</tr>
<tr>
<td></td>
<td>Gilson.MoveUp();</td>
</tr>
<tr>
<td></td>
<td>Gilson.PumpBackward(2600);</td>
</tr>
<tr>
<td></td>
<td>Robo2.WaitMilliSec(600);</td>
</tr>
<tr>
<td></td>
<td>Gilson.PumpStop();</td>
</tr>
<tr>
<td>Robo2.OpenValve(2);</td>
<td>Gilson.MoveToTube(1, 2);</td>
</tr>
<tr>
<td></td>
<td>Gilson.PumpBackward(2600);</td>
</tr>
<tr>
<td>Robo2.Wait(10);</td>
<td>Robo2.Wait(10);</td>
</tr>
<tr>
<td>solution separating air gap</td>
<td>Gilson.PumpStop();</td>
</tr>
<tr>
<td></td>
<td>Gilson.MoveUp();</td>
</tr>
</tbody>
</table>
### Appendix

<table>
<thead>
<tr>
<th>Roboflow</th>
<th>Gilson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robo2.OpenValve(1);</td>
<td>Gilson.MoveToTube(1, 1);</td>
</tr>
<tr>
<td>Robo2.ValvePumpOn(5000);</td>
<td>Gilson.PumpBackward(2600);</td>
</tr>
<tr>
<td>Robo2.Wait(5);</td>
<td>Robo2.Wait(5);</td>
</tr>
<tr>
<td>Robo2.OpenValve(2);</td>
<td>Gilson.PumpStop();</td>
</tr>
<tr>
<td>Robo2.Wait(10);</td>
<td>Gilson.MoveUp();</td>
</tr>
<tr>
<td>Robo2.OpenValve(1);</td>
<td>Gilson.PumpBackward(2600);</td>
</tr>
<tr>
<td>Robo2.Wait(10);</td>
<td>Robo2.WaitMilliSec(600);</td>
</tr>
</tbody>
</table>

Same sequence from above listed sequentially.

Synchronizing this sequence with a recording is straightforward when using the Roboflow; put the command `Robo2.StartRecording();` and `Robo2.StopRecording();` before and after the sequence of commands, respectively.

#### 4.4.3 Handling of the Gilson Delay

When using the Gilson liquid handler, the significant delay between switching to a respective solution and its arrival at the oocyte has to be considered. There are different ways to include this delay, but the easiest way is to add an additional wait command `Robo2.Wait(delay);` directly before the `Robo2.StopRecording();` command. The delay time depends on the used Minipulse pump speed and the length of the tubing between Gilson needle and measuring head. Therefore, delay time has to be determined experimentally beforehand.

Usually, buffer solution (solution 1) is already running before starting a solution exchange protocol. Therefore, the sequence after the `Robo2.StartRecording();` command starts with the first wait command `Robo2.Wait(5);`.

In addition, the respective transmit commands have to be included in order to send the appropriate compound and concentration information to the database.

Likewise, ROIs have to be defined in a way that the response of interest will be localized within the ROI.
<table>
<thead>
<tr>
<th>Roboflow</th>
<th>Gilson</th>
</tr>
</thead>
<tbody>
<tr>
<td>--&gt;Robo2.OpenValve(1); --&gt;Robo2.ValvePumpOn(5000);</td>
<td>--&gt;Gilson.MoveToTube(1, 1); --&gt;Gilson.PumpBackward(2600);</td>
</tr>
<tr>
<td>Robo2.SetBaselineROI(2, 5); Robo2.SetAnalysisROI(5, 15);</td>
<td>Robo2.SetBaselineROI(2+delay, 4+delay); Robo2.SetAnalysisROI(5+delay, 15+delay);</td>
</tr>
<tr>
<td><strong>Robo2.StartRecording();</strong></td>
<td><strong>Robo2.StartRecording();</strong></td>
</tr>
<tr>
<td>Robo2.TransmitRecording(REC_TAG_COMPOUND); Robo2.TransmitCompoundValve(2);</td>
<td>Robo2.TransmitRecording(REC_TAG_COMPOUND); Robo2.TransmitCompoundGilson(1, 2);</td>
</tr>
<tr>
<td>Robo2.Wait(5); Robo2.OpenValve(2); Robo2.Wait(10); Robo2.OpenValve(1); Robo2.Wait(10); <strong>Robo2.StopRecording();</strong></td>
<td>Robo2.Wait(5); Gilson.PumpStop(); Gilson.MoveUp(); Gilson.PumpBackward(2600); Robo2.WaitMilliSec(600); Gilson.PumpStop(); Gilson.MoveToTube(1, 2); Gilson.PumpBackward(2600); Robo2.Wait(10); Gilson.PumpStop(); Gilson.MoveUp(); Gilson.PumpBackward(2600); Robo2.WaitMilliSec(600); Gilson.PumpStop(); Gilson.MoveToTube(1, 1); Gilson.PumpBackward(2600); Robo2.Wait(10); <strong>Robo2.Wait(delay) Robo2.StopRecording();</strong></td>
</tr>
</tbody>
</table>

Alternatively, you can work with the script commands **Robo2.StartTimer();** and **Robo2.WaitForTimer();**