TechnicalNOTE

GAMRY INSTRUMENTS

Redefining Electrochemical Instrumentation

Commonly Requested Changes to Explain™ Scripts

One feature that is unique to the Gamry Instruments Software is the concept of "Open Source Scripting." This means that the flow and control of your experiments is under your control. If one of Gamry Instruments' standard scripts doesn't quite do what you want, you may find that performing your customized experiment is only a few keystrokes away. This document will point out how easy it is to make some commonly requested changes.

This document will not teach you the Explain™ scripting language. If you want to know more about the Explain scripting language, go to www.Gamry.com and download *Customizing Electrochemical Experiments Explain™* or *Programming Reference for Explain™*. You can also call and ask us for them. For detailed on-line help, simply open the Gamry Framework™ environment and click **Help**!

If you need more extensive changes and don't have the time to write the script yourself, contact Gamry Instruments. We'll be happy to discuss your unique application and tell you what it will take to have one of our engineers write a custom script for you.

The changes that we will discuss here are:

1. Changing Default Setup Parameters
2. Changing the Wording of Items in the Setup Window
3. Removing Items from, or Adding Items to, the Setup Window
4. Changing the Run-time Plotting Conventions in Voltammetry
5. Moving or Removing an Open Circuit Measurement
6. Adding Limit Criteria for Stopping Data Acquisition
7. Changing Potentiostat Hardware Settings
8. Controlling an External Device
9. Sequencing Experiments

1. Changing Default Setup Parameters.

The Setup Window that is created by an Explain™ script contains default values for all objects. These default values are assigned to the objects when they are created. To change the default value for any item in the setup box, you must change the code defining that object.

For example, if the standard size of samples in the laboratory is 3 cm^2, the following change could be made to the **`Polarization Resistance.exp`** script. (changed items are **highlighted** for clarity)

```
Area = QUANT.New ("AREA", 1.0 , "Sample &Area (cm²)")
```

Would be changed to:

```
Area = QUANT.New ("AREA", 3.0 , "Sample &Area (cm²)")
```

C3 PROZESS- UND ANALYSENTECHNIK GmbH
Peter-Henlein-Str. 20
D-85540 Haar b. München
Telefon 089/45 60 06 70
Telefax 089/45 60 06 80
info@c3-analysentechnik.de
www.c3-analysentechnik.de

This would change the default value for the Area to 3.0. Since Version 4.0 remembers the last value entered, what you generally will see is the last value that you entered. However, with this change made to the **QUANT.New** statement, pressing the Default button while running the modified script will reset the area to 3.0 cm².

2. Changing the Wording of Items in the Setup Window

The descriptive text that is displayed in the setup box is also assigned to objects when those objects are created. Therefore, modifying the prompt string (usually units of the object) is done when the object is created. For example:

```
Area  = QUANT.New ("AREA", 1.0, "Sample &Area (cm²)")
```

Could be changed to:

```
Area  = QUANT.New ("AREA", 1.0, "Coupon &Area (in²)")
```

3. Removing Items from or Adding Items to the Setup Window

The setup window provided in an Explain™ script may contain information that is irrelevant to you. In that case, you may want to modify the script to remove items from that screen. The easiest way to remove items from the setup window is to comment them out using a semicolon (**;**) in the **Setup()** function. For example if you want to completely remove the **Notes** field from the setup box the following change converts the entire line specifying notes into a comment.

```
      result = Setup ("Polarization Resistance"
&                ,PstatSelect.Selector (SELECTOR_ASTERISK)
&                ,Title
&                ,Output
;&               ,Notes
etc….
```

Adding items to the setup window is a three-step process:

1. The object must be created. This is usually done in the object definition section of the script.
2. The object is added to the **Setup ()** function as another item.
3. The information stored in that object should be written out to the data file using the **Printl ()** function.

For example, you may want every sample run to have a unique sample identification number. In the object definition section of the script add the following line:

```
SampleID = LABEL.New ("SAMPLEID", 80, "S0002", "Sample Identifier")
```

The object **SampleID** must also be added to the **Setup ()** function. Note that the placement of this object in the **Setup ()** function will dictate where the **Sample ID** text will appear in the Setup Window. The following code would add the **Sample ID** setup choices to the setup window after the potentiostat selector radio button and before the title.

```
      Result = Setup ("Polarization Resistance"
&                 ,PstatSelect.Selector (SELECTOR_ASTERISK)
&                 ,SampleID
&                 ,Title
&                 ,Output
&                 ,Notes
```

Finally the information stored in the **SampleID** object should be written to the data file by adding the line below. Note: for clarity this line should be added in the section of the script that contains the other **Printl ()** functions.

```
      SampleID.Printl ()
```

4. Changing the Run-time Plotting Conventions in Voltammetry

Although all electrochemists today agree on what a positive or negative potential is, some plot positive potentials to the right, others plot it to the left. The default in the PHE200™ software is to plot positive voltages to the left. The convention is set using the following lines of code.

```
function IvsEPlot (Cv)
      Cv.SetPlotView (VIEW_SINGLE)
;     Cv.SetPlotView (VIEW_DOUBLE)
      Cv.SetAxis(X_AXIS, RCV_Vf, LIN_AXIS, 1.0E-15, "E", "V")
      Cv.SetAxis(Y_AXIS, RCV_Im, LIN_AXIS, 1.0E-15, "I", "A")
;     Cv.SetAxis(Z_AXIS, RCV_T, LIN_AXIS, 1.0E-4, "T","s")
      Cv.FlipAxis(X_AXIS, TRUE)
;     Cv.FlipAxis(Y_AXIS, TRUE)
;     Cv.FlipAxis(Z_AXIS, TRUE)
      return
```

The function call `Cv.FlipAxis(X_AXIS, TRUE)` is responsible for flipping the axis to the left-positive mode. If this line is turned into a comment by placing a semicolon (;) at the start to make the line merely a comment and to restore the more commonly used right-positive convention.

```
;     Cv.FlipAxis(X_AXIS, TRUE)
```

5. Moving or Removing an Open Circuit Measurement

In some cases, you may wish to change the sequence of experimental steps in a particular experiment. For example, in standard scripts included in the DC105™ software, the open circuit potential is measured after the conditioning step and before the experimental waveform is applied. In some cases, when material is deposited on the electrode during the conditioning step, the measuring of an open circuit potential after conditioning can have adverse effects on the results. You have two options for alleviating this problem, you can move the measurement of open circuit potential to before the conditioning step, or you may completely eliminate the open circuit potential measurement.

To move the measurement of the open circuit potential to before the conditioning step the following lines:

```
; Condition the electrode
      if (Condit.Check ())
            if (Condition (Pstat, Condit.V1 (), Condit.V2 ()
&                 ,IRComp.Value (),0.1*Area.Value ()) eq FALSE)
                  return
```

```
; Measure Eoc
      if (Delay.Check ())
            OCDelay (Pstat, Delay.V1 (), Delay.V2 () * 0.001)
      else
            OCDelay (Pstat, 10.0, NIL)
      Printl ("EOC\t", POTEN.Eoc ())
```

Should be rearranged to measure the open circuit potential first:

```
; Measure Eoc
      if (Delay.Check ())
            OCDelay (Pstat, Delay.V1 (), Delay.V2 () * 0.001)
      else
            OCDelay (Pstat, 10.0, NIL)
      Printl ("EOC\t", POTEN.Eoc ())
; Condition the electrode
      if (Condit.Check ())
            if (Condition (Pstat, Condit.V1 (), Condit.V2 ()
&                 ,IRComp.Value (),0.1*Area.Value ()) eq FALSE)
                  return
```

The open circuit value may have no meaning for some experimental systems. You may wish to completely eliminate its measurement. Unfortunately even if the open circuit measurement is deleted, the user can still select potentials versus open circuit in the setup window. The Explain script must be modified so that not only is the measurement of open circuit removed, but the open circuit potential must be set to a value of 0.0 for all **POTEN** (Potential) class objects. The following changes would be made to the script:

```
; Measure Eoc
;     if (Delay.Check ())
;           OCDelay (Pstat, Delay.V1 (), Delay.V2 () * 0.001)
;     else
;           OCDelay (Pstat, 10.0, NIL)
      POTEN.SetEoc  (0.0)
```

6. Changing Potentiostat Hardware Settings

The control of hardware settings on the potentiostat is available in the function **InitializePstat()**. This function is usually defined at the end of an experimental script so that the user can quickly find the hardware controls and modify them as needed. For example, assume that the user is performing an experiment where the scan rate is sufficiently fast that a 5.0 Hz filter on the current causes distortions in the measured response. The following change would be made in the **InitializePstat** function.

```
Pstat.SetIchFilter (5.0)
```

becomes:

```
Pstat.SetIchFilter (100.0)
```

Changing the value of 5.0 to 100.0 would instruct the potentiostat to select the current channel filter appropriate to allow a signal of 100.0 Hz to pass through without any distortion. On a PC4™ potentiostat, the filter actually selected would be the 1 kHz filter since it is the lowest filter available which will not distort a signal of 100.0 Hz.

7. Adding Limit Criteria for Stopping Data Acquisition

The user may want to add a "stop data acquisition" criterion to an experiment. The limit criterion will either trigger an end of an experiment, or to move on to the next section of the applied **SIGNAL** object. It is easily implemented by simply enabling the termination limits of the **CURVE** class object! In the example of the **Polarization Resistance.exp** script, the **CPIV** class of **CURVE** is used. There are 6 limiting conditions on which a **CPIV** curve may be prematurely terminated. Any combination of stop criteria can be combined in the following function:

CPIV.StopAt(Under, Over, Stable, Runaway, Decreasing, Increasing)

1	Under	I < Limit
2	Over	I > Limit
3	Stable	abs(dI/dt) < Limit
4	Runaway	abs(dI/dt) > Limit
5	Decreasing	dI/dt < Limit
6	Increasing	dI/dt > Limit

The following changes should be made to the script to enable a stop data acquisition criteria based on a **limiting current density**. In this example we must be careful since the **StopAt** function only accepts limits in amperes, and we wish to enter a value in mA/cm²

First, an object must be created to store the limiting current density value. The following code should be added to the object definition section of the script.

```
IApex  = QUANT.New ("IAPEX", 10.0, "Apex &I (mA/cm²)")
```

The **IApex** object should then be added to the Setup Window (see example 3 above). Then the information regarding the current limit should be added to the call of the **Cpiv** function. For example, the following addition (**highlighted**) would be added to pass an area-normalized current limit in units of Amps to the **Cpiv** function.

```
   Cpiv (Pstat, VInit.VsEref ()
&     ,VFinal.VsEref ()
&     ,Scan.Value ()*0.001
&     ,Sample.Value ()
&     ,IRComp.Value ()
&     ,IApex.Value () * 0.001 * Area.Value ()  ; Converted from mA/cm2
;                                                to Amps.
&     )
```

The **Cpiv** function definition in this script must be changed so that it accepts an **ILimit** value (in amps) as the seventh in the calling sequence.

```
function Cpiv (Pstat, VInit, VFinal, ScanRate, SampleTime, IRToggle
&     ,ILimit)
```

Finally, after the **Curve** object has been created the stop acquisition limit must be enabled using the **StopAt** function of the **CURVE** class..

```
Curve = CPIV.New ("CURVE", Pstat)
Curve.StopAt (-ILimit, ILimit, NIL, NIL, NIL, NIL)
Curve.SetPlot (CPIV_LINIV, NIL, NIL, NIL)
```

8. Controlling an External Device

It is often convenient to control an external device with a voltage from the auxiliary output of the Gamry Potentiostat. You can use this voltage to control rotating disk electrodes, scratch devices, or other mechanical devices.

In the case of a rotating disk electrode, you might enter a value for the rotation rate in RPM in the setup box and have the Explain™ script convert that value to a voltage. The following lines of code should be added to the script. First we would add to the object definition section an object that would be used to store the rotation value.

```
RotSpeed = IQUANT.New ("ROTSPEED", 1000, "Rotation (RPM)")
```

Then this object would be added to the **`Setup ()`** function so that a line for rotation speed would be added to the setup window.

```
&                   ,RotSpeed
```

Then this object would be added to the **`Stdout`** section to output the rotation rate to the **`StdOut`** window so you can see the rotation rate at which the experiment is being run.

```
      Stdout ("Rotation Rate:   ", RotSpeed.Value (), " RPM" )
```

The value of the object should be stored to the data file. This function would be added to the section of the script where the other objects were printed out to the data file.

```
      RotSpeed.Printl ()
```

In this example, the rotating electrode used has a conversion factor of 1000 RPM per 1V input to the rotator. Therefore the **`RotSpeed`** (RPM) is divided by 1000 to get volts. This conversion statement must precede the function that actually sets the analog output.

```
; Set the Rotation
; 1V/1000 RPM
      RotationVoltage = RotSpeed.Value () / 1000.0
```

The value of the voltage (**`RotationVoltage`**) is output to the analog output on the potentiostat using the **`Pstat.SetAnalogOut`** function. Placement of this code in the script is important. You may want the rotator to start rotating before the condition step, before the measurement of open circuit potential, or before the application of the applied waveform. Therefore, you should place the **`Pstat.SetAnalogOut`** function in the script accordingly.

```
;     Set the output DAC to the Rotation Voltage
      Pstat.SetAnalogOut (RotationVoltage)
```

Finally the analog output should be set to zero so that the rotator stops at the end of the experiment. This is usually done immediately before the **`Pstat.Close ()`** in the script.

```
;
      Pstat.SetAnalogOut (0.0)
      Pstat.Close ()
;
```

9. Sequencing Experiments

In some cases, you may want to string several experiments together to form a complicated series of experiments. These sequences may also include time delays (*i.e.*, wait for 30 minutes) or time triggers (*i.e.*, wait until 10:00 a.m.).

In the Explain™ language, two special types of scripts are used to create sequences. These are called the "master" script and the "auto" script. A master script is an Explain script that contains a list of other scripts to be run and when to run them. Since the master script is written as an Explain script, you can create very flexible experiment sequences.

The master script calls the auto scripts that, in turn, run the experiments. The master script provides a way to schedule the timing and repetitions of the auto scripts. The script **Run Many.exp** is included with the Gamry Framework™ software and is shown in the Appendix. It is an example of a master script that calls several auto scripts. This script consists primarily of a number of calls to the function **LaunchWait()**. The format for each **LaunchWait()** function call is:

```
      if LaunchWait(AutoScriptFile,SetupFile,SetupName
&           ,OutputFile,PstatNo,ChannelNo)) eq  FALSE)
            return
```

Note that the use of an "if" statement before the **LaunchWait()** test requires that the experiment was successfully completed before moving on the next experiment.

The **AutoScriptFile** must be an auto script, which will be discussed below. The **SetupFile** and **SetupName** are created by saving setup information using the save button in the setup window generated by the regular script. For example, the setup information for an automated polarization resistance experiment would be saved using the setup box from the **Polarization Resistance.exp** script. **OutputFile** is the name of the data file that will be created. **PstatNo** is the designation of the potentiostat in the computer that should be used to run the experiment (1 through 4 are available options; 1 is the default for a single potentiostat in a computer). **ChannelNo** is the channel on the ECM8™ multiplexer that will be used: A value of **NIL** is used when no multiplexer is present.

Auto scripts are experiments that must be launched by a master script. The major difference between an auto script and the corresponding normal script is that the auto script is designed to run with minimal operator interaction. Instead of asking the operator to supply experiment parameters, the parameters are read from setup files. You'll need to create setup files for each type of experiment to be performed. In addition, warning messages and queries requiring user input are avoided so the sequence does not pause waiting for an "OK". The Auto scripts are easy to find: The filenames all begin with **AUTO**!

Another difference is that the master script can pass parameters to the auto script. This allows parameter sets, file names, etc. to be controlled by the master script. A side effect is that you cannot call the auto scripts directly from the Gamry Framework™ since they expect input parameters.

You can create your own specialized auto scripts or use the auto scripts that Gamry Instruments has added to the application packages. Both original and automated versions must create the same data file formats so the analysis routines will work with data files from either one.

Conclusion

The Explain™ scripting language provides an unrivaled degree of flexibility in customizing experiments while preserving a simple point-and-click interface. Standard experiments provided by Gamry Instruments can be performed with the click of a button. More complicated experiments may be created from the scripts provided with the Gamry system by copying those scripts and making small modifications. The Explain scripting language and user-accessible object-oriented scripts provide an easily modified platform for electrochemical experiments.

Appendix
Example script: Run Many.exp

```
; Explain Script for automatic scheduling of experiments.
; Copyright (c) Gamry Instruments, Inc., 1995-2001
; Version 4.0
;
; Use:  Runs a sequence of single experiments.
;
include "explain4.exp"
include "Auto Utilities.exp"
;
function Main()
;
      if (LaunchWait("Auto Corrosion Potential.exp"
&                ,"auto.set"       ; Set File
&                ,"CPOT"           ; Set Name
&                ,"autocpot.dta"   ; Filename where data will be saved
&                ,1                ; Potentiostat number
&                ,NIL              ; Mux Channel (NIL = No mux used)
&                ) eq FALSE)
            return
;
      if (LaunchWait("Auto RpEc Trend.exp"
&                ,"auto.set"       ; Set File
&                ,"RP"             ; Set Name
&                ,"autorp.dta"     ; Filename where data will be saved
&                ,1                ; Potentiostat number
&                ,NIL              ; Mux Channel (NIL = No mux used)
&                ) eq FALSE)
            return
;
      if (LaunchWait("Auto Potentiostatic EIS.exp"
&                ,"auto.set"       ; Set File
&                ,"EIS"            ; Set Name
&                ,"autoeis.dta"    ; Filename where data will be saved
&                ,1                ; Potentiostat number
&                ,NIL              ; Mux Channel (NIL = No mux used)
&                ) eq FALSE)
            return
;
      Notify("All Experiments Done")
;
      Dawdle()
```