EC-Lab®
Development Package

User guide

Version 5.38 – December 2014

BioLogic
Science Instruments

History

Version	Description
V5.38	Optimized the loop technique for mandatory goto use
V5.37	Fixed xilinx version comparison at the end of LoadXilinx step.
V5.36	64-bits compatibility added. New DLLs (EClib64.dll, blfind64.dll) and new Delphi 64-bits examples. New form in Delphi examples showing how to search and select a device in a broadcast list (uses blfind.dll).
V5.35	Support for HCP-1005 and MPG-2xx
V5.34	Updated the kernel4.bin to enable new amplifiers (2A and 10A) for SP-300 Updated DLL for the new Kernel Updated SP-300 techniques for new record mode (see section 8) Updated xlx firmware to accommodate new kernel Updated C and C# examples for new kernel Timebase parameter added to some LabVIEW examples Updated Techniques description according to the new record mode
V5.33	Clarifications about BL_UpdateParameters usage Fixed references to non-existent section 4.1 Added C/C++ and C# code examples
V5.32	New USB driver for Windows 8 LabVIEW examples are now provided for LabVIEW version 12 and 8.5 User guide corrections: Reference to "section 4.2" replaced by "section *5.3. Constants*"; TchannelInfos.NbAmps, TdataInfos.MuxPad, TCurrentValues.OptErr and OptPos fields were missing in the structure declaration
V5.31	Allows connection to SP-240 device. External control bug correction for SP-300 series.
V5.29	Modular Pulse technique : add records mode and step index ECP v5.26
v5.28	Add BL_UpdateParameters_LV
v5.27	Add Modular Pulse technique Add CASG technique Add CASP technique kernel v5.25 Kernel4 v5.27 Add blfind.dll v1.1 Structure TCHANNELINFOS modified

v5.26	Record and external control options Windows Installer New USB driver for Windows XP/Vista/Seven 32bits/64bits New fields: TdataInfos.MuxPad, TCurrentValues.OptErr and OptPos
v5.23	Add techniques with limits to SP-300 series : vscanlimit, iscanlimit, calimit, cplimit. Option 4A amplifier for SP-300 Allows connection to MPG2 device Add Floating / grounded mode to SP-300 series Change name of constant
v5.22	Update LASV parameters (Record_every_dI, Record_every_dT) Kernel VMP3 v5.22 (Update change mode Potentio-Galvano with linked techniques)
v5.21	LASV (Large Amplitude sinusoidal voltammetry) Technique
V5.20	Control SP-300 and SP-200
v5.19	Description image with all technique Correction of parameter range
v5.18	First version

Table of Contents

1. Overview

The **EC-Lab® Development Package** is intended for software developers who need to integrate Bio-Logic potentiostats / galvanostats in OEM applications. This package supports the following Biologic instruments :

- VMP2 / VMP3
- BiStat
- VSP — VMP3 series
- SP-50 / SP-150
- MPG2
- HCP-803
- SP-200
- SP-300 — SP-300 series

The library accommodates some functionality offered by EC-Lab® software:

- Detection of connected instruments,
- Connection / disconnection to the instrument through Ethernet/USB,
- Channels initialization (firmware loading),
- Load techniques on selected channel(s) (OCV, CA, CP, ...),
- Start/stop selected channel(s),
- Retrieving data,
- ...

A Delphi test program and LabVIEW® VIs examples are also provided to help the user in the integration of the library in his application.

Notes:

- The instrument firmware must be in **V9.00 or higher**. If this is not the case, please refer to the "*EC-Lab software user's manual.pdf*" for firmware upgrading.

- Theoretically, any software development tool able to call a DLL is suitable for using the **EC-Lab® Development Package** (C++, Pascal, LabVIEW®...). However limitations of some compilers may prevent a proper calling of some or all the functions.

- The **EC-Lab® Development Package** is provided "AS IS". No specific support is provided to this free package. There is no warranty for damages using is. We strongly recommend you to test your techniques on a dummy cell before using them on a real cell.

- Use of the **EC-Lab® Development Package** in a commercial software is forbidden without a written authorization of Bio-Logic SAS.

- The software includes a LEPMI/ENSEEG/INPG license.

2. Files package

The **EC-Lab® Development Package** is composed with the following files:

- blfind.dll: library used to find available instruments (Ethernet and/or USB)
- blfind64.dll: same as blfind.dll provided for 64-bits compatibility
- EClib.dll: library used to communicate with the instrument
- EClib64.dll: same as Eclib.dll provided for 64-bits compatibility
- kernel.bin: channel firmware for VMP3 devices
- kernel4.bin: channel firmware for SP-300 series
- Vmp_ii_0437_a6.xlx: channel firmware for VMP3 devices
- Vmp_iv_0395_aa.xlx: channel firmware for SP-300 series
- ocv.ecc: open circuit voltage technique
- cv.ecc: Cyclic Voltammetry technique
- biovscan.ecc: Cyclic Voltammetry Advanced technique
- ca.ecc: chrono-amperometry technique
- cp.ecc: chrono-potentiommetry technique
- iscan.ecc: current scan technique
- vscan.ecc: voltage scan technique
- lasv.ecc: large amplitude sinusoidal voltammetry
- load.ecc: constant load technique
- pow.ecc: constant power technique
- peis.ecc: potentio electrochemical impedance spectroscopy technique
- geis.ecc: galvano electrochemical impedance spectroscopy technique
- seisp.ecc: potentiostatic impedance technique
- seisg.ecc: galvanostatic Impedance technique
- dpv.ecc: differential pulse voltammetry technique
- swv.ecc: square wave voltammetry technique
- npv.ecc: normal pulse voltammetry technique
- rnpv.ecc: reverse normal pulse voltammetry technique
- dnpv.ecc: differential normal pulse voltammetry technique
- dpa.ecc: differential pulse amperometry technique
- evt.ecc: ecorr. Vs time technique
- lp.ecc: linear polarization technique

- gc.ecc: generalized corrosion technique
- cpp.ecc: cyclic potentiodynamic polarization technique
- pdp.ecc: potentiodynamic pitting technique
- psp.ecc: potentiostatic pitting technique
- zra.ecc: zero resistance ammeter technique
- ircmp.ecc: manual IR technique
- pzir.ecc: IR determination with potentiostatic impedance technique
- gzir.ecc: IR determination with galvanostatic impedance technique
- loop.ecc: loop technique
- TO.ecc: trigger out technique
- TI.ecc: trigger in technique
- TOS.ecc: trigger out set technique
- vscanlimit: voltage scan technique with limits
- iscanlimit: current scan technique with limits
- calimit: chrono-amperometry technique with limits
- cplimit: chrono-potentiommetry technique with limits

These Techniques are doubled with the name *name*4.ecc only for SP-300 series. The techniques which have not these file must not to be used with SP-300 series (do not load a file *name*.ecc on SP-300 series and do not load a file *name4*.ecc on VMP3 devices).

All the files of **EC-Lab® Development Package** must be stored in the same directory.

3. General information

3.1. Calling conventions

The library uses the **stdcall** calling conventions for all exported functions.

3.2. Data alignment

All the structures used by the library are aligned on double-word to simplify the communication with others programming environment.

3.3. Multi-thread applications

All exported functions are protected by a synchronization object, they can be called in a multi-thread application.

3.4. Data types

The library is written in object Pascal under Delphi. All data type used in this document are Pascal types. The type translation table C/C++ - Object Pascal is :

Type translation table	
C/C++ Type	**ObjectPascal Type**
unsigned short [int]	Word
[signed] short [int]	SmallInt
unsigned [int]	Cardinal *{ 3.25 fix }*
[signed] int	Integer
UINT	LongInt *{ or Cardinal }*
WORD	Word
DWORD	LongInt *{ or Cardinal }*
unsigned long	LongInt *{ or Cardinal }*
unsigned long int	LongInt *{ or Cardinal }*
[signed] long	LongInt
[signed] long int	LongInt
char	Char
signed char	ShortInt
unsigned char	Byte

Type translation table	
C/C++ Type	**ObjectPascal Type**
char*	PChar
LPSTR *or PSTR*	PChar
LPWSTR *or PWSTR*	PWideChar *{ 3.12 fix }*
void*	Pointer
BOOL	Bool
float	Single
double	Double
long double	Extended
*UserType**	^*UserType*
NULL	NIL

The following data types are used by the library:

Data types		
Data types	**Format**	**Range**
int8	signed 8-bit	−128..127
int16	signed 16-bit	−32768..32767
int32	signed 32-bit	−2147483648..2147483647
uint8	unsigned 8-bit	0..255
uint16	unsigned 16-bit	0..65535
uint32	unsigned 32-bit	0..4294967295
boolean	unsigned 8-bit	FALSE=0, TRUE=1
single	Single precision floating point (32 bits, 7–8 significant digits)	[1.5 x 10^{-45}, 3.4 x 10^{38}]
double	Double precision floating point (64 bits, 15–16 significant digits)	[5.0 x 10^{-324}, 1.7 x 10^{308}]

3.5. Variables description

The following variables are used by the library:

<table>
<tr><th colspan="3">Variable description</th></tr>
<tr><th>Variable</th><th>Description</th><th>Unit</th></tr>
<tr><td>t</td><td>time</td><td>second (s)</td></tr>
<tr><td>I</td><td>Current</td><td>Ampere (A)</td></tr>
<tr><td>Ic</td><td>Current control</td><td>Ampere (A)</td></tr>
<tr><td><I></td><td>Average current</td><td>Ampere (A)</td></tr>
<tr><td>Ewe</td><td>WE potential versus REF</td><td>Volt (V)</td></tr>
<tr><td><Ewe></td><td>Average of WE potential versus REF</td><td>Volt (V)</td></tr>
<tr><td>Ece</td><td>CE potential versus REF</td><td>Volt (V)</td></tr>
<tr><td>Ewe-Ece</td><td>WE versus CE potential</td><td>Volt (V)</td></tr>
<tr><td>Ec</td><td>Potential control</td><td>Volt (V)</td></tr>
<tr><td>R</td><td>Resistor</td><td>Ohm (Ω)</td></tr>
<tr><td>power</td><td>Power</td><td>Watt (W)</td></tr>
<tr><td>cycle</td><td>Cycle number</td><td>-</td></tr>
<tr><td>Q</td><td>Electric charge from the beginning of the technique</td><td>Ampere.second (A.s)</td></tr>
<tr><td>f</td><td>Frequency</td><td>Hertz (Hz)</td></tr>
<tr><td>phase</td><td>Angle</td><td>radian (rad)</td></tr>
<tr><td>|Ewe|</td><td>Module of Ewe (V)</td><td>Volt (V)</td></tr>
<tr><td>|Ece|</td><td>Module of Ece (V)</td><td>Volt (V)</td></tr>
<tr><td>|Ice|</td><td>Module of Ice (A)</td><td>Ampere (A)</td></tr>
<tr><td>|I|</td><td>Module of I</td><td>Ampere (A)</td></tr>
<tr><td>I Range</td><td>Current range</td><td>-</td></tr>
<tr><td>E Range</td><td>WE potential range</td><td>-</td></tr>
<tr><td>tb</td><td>Timebase</td><td>Microsecond (μs)</td></tr>
</table>

4. Using the library

First of all, the function BL_CONNECT must be called to establish the connection with the selected instrument through Ethernet or USB. The function will return a device identifier (ID) which will have to be used with all the functions of the library to communicate with this instrument. Note that one can communicate with several instruments thanks to the device identifier.

After establishing the connection, the firmware must be loaded (if not already done) on channels plugged with the function BL_LOADFIRMWARE to make them operational. Use the function BL_ISCHANNELPLUGGED to list channels plugged on the instrument and the function BL_GETCHANNELINFOS to get channels informations as firmware version, board version, memory size, ...

Now the instrument is ready to receive techniques with user's parameters on selected channels thanks to the function BL_LOADTECHNIQUE. Note that the techniques are defined by the *.ecc files delivered with the library (for instance the file ocv.ecc defines the *Open Circuit Voltage* technique). See the section 7. Techniques for a complete description of parameters available for each technique.
Electrochemical techniques parameters must be carefully programmed according to the instrument hardware specifications. Be aware that wrong parameters can generate faulty operations of the technique.

Once the techniques are loaded, channels can be started (or stopped) with the function BL_STARTCHANNEL (or BL_STOPCHANNEL). One can also synchronize channels together with the functions BL_STARTCHANNELS / BL_STOPCHANNELS.

The data can be recovered from selected channels with the function BL_GETDATA. The format of the data returned depends of the technique used to generate these data. One can find the identifier of this technique in the structure TDATAINFOS returned by the function BL_GETDATA. See the section 7. Techniques for a complete description of the format of data for each technique.

Once the techniques are finished and all data recovered, one must close the connection with the instrument with the function BL_DISCONNECT.

5. Used structures and constants

5.1. Structures

The structure TDEVICEINFOS defines device information and is used by the function BL_CONNECT :

Structure **TDEVICEINFOS**		
Field name	**Data type**	**Description**
DeviceCode	int32	Device code (see section *5.3. Constants*)
RAMsize	int32	RAM size, in MBytes
CPU	int32	Computer board cpu
NumberOfChannels	int32	Number of channels connected
NumberOfSlots	int32	Number of slots available
FirmwareVersion	int32	Communication firmware version
FirmwareDate_yyyy	int32	Communication firmware date YYYY
FirmwareDate_mm	int32	Communication firmware date MM
FirmwareDate_dd	int32	Communication firmware date DD
HTdisplayOn	int32	Allow hyper-terminal prints (true/false)
NbOfConnectedPC	int32	Number of connected PC

The structure TCHANNELINFOS defines channel information and is used by the function BL_GETCHANNELINFOS :

Structure **TCHANNELINFOS**		
Field name	**Data type**	**Description**
Channel	int32	Channel (0..15)
BoardVersion	int32	Board version
BoardSerialNumber	int32	Board serial number
FirmwareCode	int32	Identifier of the firmware loaded on the channel (see section *5.3. Constants*)
FirmwareVersion	int32	Firmware version
XilinxVersion	int32	Xilinx version
AmpCode	int32	Amplifier code (see section *5.3. Constants*)
NbAmps	int32	Number of amplifiers
Lcboard	int32	Low current board present (= 1)
Zboard	int32	TRUE if channel with impedance capabilities

Structure **TCHANNELINFOS**		
Field name	**Data type**	**Description**
RESERVED	int32	not used
RESERVED	int32	not used
MemSize	int32	Memory size (in bytes)
MemFilled	int32	Memory filled (in bytes)
State	int32	Channel state : run/stop/pause (see section *5.3. Constants*)
MaxIRange	int32	Maximum I range allowed (see section *5.3. Constants*)
MinIRange	int32	Minimum I range allowed (see section *5.3. Constants*)
MaxBandwidth	int32	Maximum bandwidth allowed (see section *5.3. Constants*)
NbOfTechniques	int32	Number of techniques loaded

The structure TCURRENTVALUES defines channel current values (Ewe, Ece, I, ...) and is used by functions BL_GETCURRENTVALUES and BL_GETDATA :

Structure **TCURRENTVALUES**		
Field name	**Data type**	**Description**
State	int32	Channel state : run/stop/pause (see section *5.3. Constants*)
MemFilled	int32	Memory filled (in Bytes)
TimeBase	single	Time base (s)
Ewe	single	Working electrode potential (V)
EweRangeMin	single	Ewe min range (V)
EweRangeMax	single	Ewe max range (V)
Ece	single	Counter electrode potential (V)
EceRangeMin	single	Ece min range (V)
EceRangeMax	single	Ece max range (V)
Eoverflow	int32	Potential overflow
I	single	Current value (A)
IRange	int32	Current range (see section *5.3. Constants*)
Ioverflow	int32	Current overflow
ElapsedTime	single	Elapsed time (s)
Freq	single	Frequency (Hz)
Rcomp	single	R compensation (Ohm)
Saturation	int32	E or/and I saturation

Structure **TCURRENTVALUES**		
Field name	**Data type**	**Description**
OptErr	int32	Hardware Option Error Code (SP-300 series only see section *5.4. Error codes*)
OptPos	int32	Index of the option generating the OptErr (SP-300 series only)

The structure TDATAINFOS defines data information (i.e. information on the data saved in the data buffer) and is used by the function BL_GETDATA :

Structure **TDATAINFOS**		
Field name	**Data type**	**Description**
IRQskipped	int32	Number of IRQ skipped
NbRows	int32	Number of rows into the data buffer, i.e. number of points saved in the data buffer
NbCols	int32	Number of columns into the data buffer, i.e. number of variables defining a point in the data buffer
TechniqueIndex	int32	Index (0-based) of the technique who has generated the data. This field is only useful for linked techniques
TechniqueID	int32	Identifier of the technique who has generated the data. Must be used to identify the data format into the data buffer (see section *5.3. Constants*)
ProcessIndex	Int32	Index (0-based) of the process of the technique who has generated the data. Must be used to identify the data format into the data buffer
loop	int32	Loop number
StartTime	double	Start time (s)
MuxPad	int32	Active MP-MEA option pad number (SP-300 series only)

The array TDATABUFFER is used to retrieve data from the device by the function BL_GETDATA:

Type **TDATABUFFER**
TDataBuffer = array[1..1000] of uint32; PDataBuffer = ^TDataBuffer

The structure TECCPARAM defines an elementary technique parameter and is used by the function BL_LOADTECHNIQUE:

Structure **TECCPARAM**		
Field name	**Data type**	**Description**
ParamStr	array[1..64] of char	string who defines the parameter label (see section 7. Techniques for a complete description of parameters available for each technique)
ParamType	int32	Parameter type (0=int32, 1=boolean, 2=single)
ParamVal	int32	Parameter value (WARNING: numerical value)
ParamIndex	int32	Parameter index (0-based). Useful for multi-step parameters only.

Type **PECCPARAM**
PEccParam = ^TEccParam

The structure TECCPARAMS defines an array of elementary technique parameters and is used by the function BL_LOADTECHNIQUE:

Structure **TECCPARAMS** : array of elementary technique parameters		
Field name	**Data type**	**Description**
len	int32	Length of the array pointed by pParams
pParams	PEccParam	Pointer on the array of technique parameters (array of structure TEccParam)

Structure **THARDWARECONF** : hardware configuration		
Field name	**Data type**	**Description**
Conn	int32	Electrode connection for constant value of this parameter see 5. Used structures and constants
Ground	int32	Instrument ground for constant value of this parameter see 5. Used structures and constants

Type **PHARDWARECONF**
PHardwareConf = ^THardwareConf

5.2. Other structures

5.2.1. Labview Structures

The structures below are defined for LabVIEW compatibility for techniques parameters loading and are used by the function BL_LOADTECHNIQUE_LV.

Structure **TARRAYOFCHAR_LV** : array of char		
Field name	**Data type**	**Description**
dimSize	int32	Length of the array
FirstChar	char	First element in the array of char

Type **PPARRAYOFCHAR_LV**
PArrayOfChar_LV = ^TArrayOfChar_LV; PPArrayOfChar_LV = ^PArrayOfChar_LV;

Structure **TECCPARAM_LV** : elementary technique parameter		
Field name	**Data type**	**Description**
ParamStr	PPArrayOfChar_LV	string who defines the parameter (see section 7. Techniques for a complete description of parameters available for each technique)
ParamType	int32	Parameter type (0=int32, 1=boolean, 2=single)
ParamVal	int32	Parameter value (WARNING : numerical value)
ParamIndex	int32	Parameter index (0-based). Useful for multi-step parameters only.

Structure **TECCPARAMS_LV** : array of elementary technique parameters		
Field name	**Data type**	**Description**
dimSize	int32	Length of the array
FirstEccParam_LV	TEccParam_LV	First element in the array

Type **PPECCPARAMS_LV**
PEccParams_LV = ^TEccParams_LV PPEccParams_LV = ^PEccParams_LV

5.2.2. Vee Pro Type

The structures below are defined for Vee Pro compatibility for techniques parameters loading and are used by the function BL_UPDATEPARAMETERS_VEE, BL_LOADTECHNIQUE_VEE, and BL_GETDATA_VEE.

Type PPChar
PPChar = array of array of ansichar

TArrayDouble
TArrayDouble = array of Double

5.3. Constants

Device constants (used by the function BL_CONNECT)		
Constant	**Value**	**Description**
KBIO_DEV_VMP	0	VMP device
KBIO_DEV_VMP2	1	VMP2 device
KBIO_DEV_MPG	2	MPG device
KBIO_DEV_BISTAT	3	BISTAT device
KBIO_DEV_MCS_200	4	MCS-200 device
KBIO_DEV_VMP3	5	VMP3 device
KBIO_DEV_VSP	6	VSP device
KBIO_DEV_HCP803	7	HCP-803 device
KBIO_DEV_EPP400	8	EPP-400 device
KBIO_DEV_EPP4000	9	EPP-4000 device
KBIO_DEV_BISTAT2	10	BISTAT 2 device
KBIO_DEV_FCT150S	11	FCT-150S device
KBIO_DEV_VMP300	12	VMP-300 device
KBIO_DEV_SP50	13	SP-50 device
KBIO_DEV_SP150	14	SP-150 device
KBIO_DEV_FCT50S	15	FCT-50S device
KBIO_DEV_SP300	16	SP300 device
KBIO_DEV_CLB500	17	CLB-500 device
KBIO_DEV_HCP1005	18	HCP-1005 device
KBIO_DEV_CLB2000	19	CLB-2000 device
KBIO_DEV_VSP300	20	VSP-300 device
KBIO_DEV_SP200	21	SP-200 device
KBIO_DEV_MPG2	22	MPG2 device
KBIO_DEV_ND1	23	RESERVED
KBIO_DEV_ND2	24	RESERVED
KBIO_DEV_ND3	25	RESERVED
KBIO_DEV_ND4	26	RESERVED
KBIO_DEV_SP240	27	SP-240 device
KBIO_DEV_MPG205	28	MPG-205 (VMP3)
KBIO_DEV_MPG210	29	MPG-210 (VMP3)
KBIO_DEV_MPG220	30	MPG-220 (VMP3)
KBIO_DEV_MPG240	31	MPG-240 (VMP3)
KBIO_DEV_UNKNOWN	255	Unknown device

Firmware code constants (used by the structure TCHANNELINFOS)		
Constant	**Value**	**Description**
KIBIO_FIRM_NONE	0	No firmware loaded
KIBIO_FIRM_INTERPR	1	Firmware for EC-Lab® software
KIBIO_FIRM_UNKNOWN	4	Unknown firmware loaded
KIBIO_FIRM_KERNEL	5	Firmware for the library
KIBIO_FIRM_INVALID	8	Invalid firmware loaded
KIBIO_FIRM_ECAL	10	Firmware for calibration software

Amplifier constants (used by the structure TCHANNELINFOS)			
Constant	**Value**	**Description**	**Device Family**
KIBIO_AMPL_NONE	0	No amplifier	VMP3 series
KIBIO_AMPL_2A	1	Amplifier 2 A	VMP3 series
KIBIO_AMPL_1A	2	Amplifier 1 A	VMP3 series
KIBIO_AMPL_5A	3	Amplifier 5 A	VMP3 series
KIBIO_AMPL_10A	4	Amplifier 10 A	VMP3 series
KIBIO_AMPL_20A	5	Amplifier 20 A	VMP3 series
KIBIO_AMPL_HEUS	6	reserved	VMP3 series
KIBIO_AMPL_LC	7	Low current amplifier	VMP3 series
KIBIO_AMPL_80A	8	Amplifier 80 A	VMP3 series
KIBIO_AMPL_4AI	9	Amplifier 4 A	VMP3 series
KIBIO_AMPL_PAC	10	Fuel Cell Tester	VMP3 series
KIBIO_AMPL_4AI_VSP	11	Amplifier 4 A (VSP instrument)	VMP3 series
KIBIO_AMPL_LC_VSP	12	Low current amplifier (VSP instrument)	VMP3 series
KIBIO_AMPL_UNDEF	13	Undefinied amplifier	VMP3 series
KIBIO_AMPL_MUIC	14	reserved	VMP3 series
KIBIO_AMPL_NONE_GIL	15	No amplifier	VMP3 series
KIBIO_AMPL_8AI	16	Amplifier 8 A	VMP3 series
KIBIO_AMPL_LB500	17	Amplifier LB500	VMP3 series
KIBIO_AMPL_100A5V	18	Amplifier 100 A	VMP3 series
KIBIO_AMPL_LB2000	19	Amplifier LB2000	VMP3 series
KBIO_AMPL_1A48V	20	Amplifier 1A 48V	SP-300 series
KBIO_AMPL_4A10V	21	Amplifier 4A 10V	SP-300 series
KBIO_AMPL_5A_MPG2B	22	Amplifier 5A	MPG-205
KBIO_AMPL_10A_MPG2B	23	Amplifier 10A	MPG-210
KBIO_AMPL_20A_MPG2B	24	Amplifier 20A	MPG-220
KBIO_AMPL_40A_MPG2B	25	Amplifier 40A	MPG-240
KBIO_AMPL_COIN_CELL_HOLDER	26	Coin cell holder amplifier	
KBIO_AMPL4_10A5V	27	Amplifier 10A 5V	SP-300 series

Amplifier constants (used by the structure TCHANNELINFOS)			
Constant	**Value**	**Description**	**Device Family**
KBIO_AMPL4_2A30V	28	Amplifier 2A 30V	SP-300 series

I range constants (used by the structures TDATAINFOS and TECCPARAM)			
Constant	**Value**	**Description**	**Device Family**
KBIO_IRANGE_100pA	0	I range 100 pA	SP-300 series
KBIO_IRANGE_1nA	1	I range 1 nA	VMP3 / SP-300 series
KBIO_IRANGE_10nA	2	I range 10 nA	VMP3 / SP-300 series
KBIO_IRANGE_100nA	3	I range 100 nA	VMP3 / SP-300 series
KBIO_IRANGE_1uA	4	I range 1 µA	VMP3 / SP-300 series
KBIO_IRANGE_10uA	5	I range 10 µA	VMP3 / SP-300 series
KBIO_IRANGE_100uA	6	I range 100 µA	VMP3 / SP-300 series
KBIO_IRANGE_1mA	7	I range 1 mA	VMP3 / SP-300 series
KBIO_IRANGE_10mA	8	I range 10 mA	VMP3 / SP-300 series
KBIO_IRANGE_100mA	9	I range 100 mA	VMP3 / SP-300 series
KBIO_IRANGE_1A	10	I range 1 A	VMP3 / SP-300 series
KBIO_IRANGE_BOOSTER	11	Booster	VMP3 / SP-300 series
KBIO_IRANGE_AUTO	12	Auto range	VMP3 / SP-300 series
KBIO_IRANGE_10pA	13	IRANGE_100pA + Igain x10	SP-300 series
KBIO_IRANGE_1pA	14	IRANGE_100pA + Igain x100	SP-300 series

E range constants (used by the structures TDATAINFOS and TECCPARAM)		
Constant	**Value**	**Description**
KBIO_ERANGE_2_5	0	±2.5 V
KBIO_ERANGE_5	1	±5 V
KBIO_ERANGE_10	2	±10 V
KBIO_ERANGE_AUTO	3	Auto range

Bandwidth constants (used by the structures TDATAINFOS and TECCPARAM)		
Constant	**Value**	**Description**
KBIO_BW_1	1	Bandwidth #1
KBIO_BW_2	2	Bandwidth #2
KBIO_BW_3	3	Bandwidth #3
KBIO_BW_4	4	Bandwidth #4
KBIO_BW_5	5	Bandwidth #5
KBIO_BW_6	6	Bandwidth #6
KBIO_BW_7	7	Bandwidth #7

Bandwidth constants (used by the structures TDATAINFOS and TECCPARAM)		
Constant	**Value**	**Description**
KBIO_BW_8	8	Bandwidth #8 (only with SP-300 series)
KBIO_BW_9	9	Bandwidth #9 (only with SP-300 series)

Electrode connection constants (used by the structure THardwareConf)		
Constant	**Value**	**Description**
KBIO_CONN_STD	0	Standard connection
KBIO_CONN_CETOGRND	1	CE to ground connection

Channel mode constants (used only with SP-300 series, by the structure THardwareConf)		
Constant	**Value**	**Description**
KBIO_MODE_GROUNDED	0	Grounded mode
KBIO_MODE_FLOATING	1	Floating mode

Technique identifier constants (used by the structure TDATAINFOS)		
Constant	**Value**	**Description**
KBIO_TECHID_NONE	0	None
KBIO_TECHID_OCV	100	Open Circuit Voltage (Rest) identifier
KBIO_TECHID_CA	101	Chrono-amperometry identifier
KBIO_TECHID_CP	102	Chrono-potentiometry identifier
KBIO_TECHID_CV	103	Cyclic Voltammetry identifier
KBIO_TECHID_PEIS	104	Potentio Electrochemical Impedance Spectroscopy identifier
KBIO_TECHID_POTPULSE	105	(unused)
KBIO_TECHID_GALPULSE	106	(unused)
KBIO_TECHID_GEIS	107	Galvano Electrochemical Impedance Spectroscopy identifier
KBIO_TECHID_STACKPEIS_SLAVE	108	Potentio Electrochemical Impedance Spectroscopy on stack identifier
KBIO_TECHID_STACKPEIS	109	Potentio Electrochemical Impedance Spectroscopy on stack identifier
KBIO_TECHID_CPOWER	110	Constant Power identifier
KBIO_TECHID_CLOAD	111	Constant Load identifier
KBIO_TECHID_FCT	112	(unused)
KBIO_TECHID_SPEIS	113	Staircase Potentio Electrochemical Impedance Spectroscopy identifier
KBIO_TECHID_SGEIS	114	Staircase Galvano Electrochemical Impedance Spectroscopy identifier

Technique identifier constants (used by the structure TDATAINFOS)		
Constant	**Value**	**Description**
KBIO_TECHID_STACKPDYN	115	Potentio dynamic on stack identifier
KBIO_TECHID_STACKPDYN_SLAVE	116	Potentio dynamic on stack identifier
KBIO_TECHID_STACKGDYN	117	Galvano dynamic on stack identifier
KBIO_TECHID_STACKGEIS_SLAVE	118	Galvano Electrochemical Impedance Spectroscopy on stack identifier
KBIO_TECHID_STACKGEIS	119	Galvano Electrochemical Impedance Spectroscopy on stack identifier
KBIO_TECHID_STACKGDYN_SLAVE	120	Galvano dynamic on stack identifier
KBIO_TECHID_CPO	121	(unused)
KBIO_TECHID_CGA	122	(unused)
KBIO_TECHID_COKINE	123	(unused)
KBIO_TECHID_PDYN	124	Potentio dynamic identifier
KBIO_TECHID_GDYN	125	Galvano dynamic identifier
KBIO_TECHID_CVA	126	Cyclic Voltammetry Advanced identifier
KBIO_TECHID_DPV	127	Differential Pulse Voltammetry identifier
KBIO_TECHID_SWV	128	Square Wave Voltammetry identifier
KBIO_TECHID_NPV	129	Normal Pulse Voltammetry identifier
KBIO_TECHID_RNPV	130	Reverse Normal Pulse Voltammetry identifier
KBIO_TECHID_DNPV	131	Differential Normal Pulse Voltammetry identifier
KBIO_TECHID_DPA	132	Differential Pulse Amperometry identifier
KBIO_TECHID_EVT	133	Ecorr vs. time identifier
KBIO_TECHID_LP	134	Linear Polarization identifier
KBIO_TECHID_GC	135	Generalized corrosion identifier
KBIO_TECHID_CPP	136	Cyclic Potentiodynamic Polarization identifier
KBIO_TECHID_PDP	137	Potentiodynamic Pitting identifier
KBIO_TECHID_PSP	138	Potentiostatic Pitting identifier
KBIO_TECHID_ZRA	139	Zero Resistance Ammeter identifier
KBIO_TECHID_MIR	140	Manual IR identifier
KBIO_TECHID_PZIR	141	IR Determination with Potentiostatic Impedance identifier
KBIO_TECHID_GZIR	142	IR Determination with Galvanostatic Impedance identifier
KBIO_TECHID_LOOP	150	Loop (used for linked techniques) identifier
KBIO_TECHID_TO	151	Trigger Out identifier
KBIO_TECHID_TI	152	Trigger In identifier
KBIO_TECHID_TOS	153	Trigger Set identifier

Technique identifier constants (used by the structure TDATAINFOS)		
Constant	**Value**	**Description**
KBIO_TECHID_CPLIMIT	155	Chrono-potentiometry with limits identifier
KBIO_TECHID_GDYNLIMIT	156	Galvano dynamic with limits identifier
KBIO_TECHID_CALIMIT	157	Chrono-amperometry with limits identifier
KBIO_TECHID_PDYNLIMIT	158	Potentio dynamic with limits identifier
KBIO_TECHID_LASV	159	Large amplitude sinusoidal voltammetry
KBIO_TECHID_MP	167	Modular Pulse
KBIO_TECHID_CASG	169	Constant amplitude sinusoidal micro galvano polarization
KBIO_TECHID_CASP	170	Constant amplitude sinusoidal micro potentio polarization

Channel state constants (used by the structure TCHANNELINFOS)		
Constant	**Value**	**Description**
KBIO_STATE_STOP	0	Channel is stopped
KBIO_STATE_RUN	1	Channel is running
KBIO_STATE_PAUSE	2	Channel is paused

Parameter type constants (used by the structure TECCPARAM)		
Constant	**Value**	**Description**
PARAM_INT32	0	Parameter type = int32
PARAM_BOOLEAN	1	Parameter type = boolean
PARAM_SINGLE	2	Parameter type = single

5.4. Error codes

All functions exported from the library returns a signed 32-bit as result. If the function succeeded the returned value is 0, otherwise the returned value is negative as described below:

General error codes

Constant	Value	Description
ERR_GEN_NOTCONNECTED	-1	no instrument connected
ERR_GEN_CONNECTIONINPROGRESS	-2	connection in progress
ERR_GEN_CHANNELNOTPLUGGED	-3	selected channel(s) unplugged
ERR_GEN_INVALIDPARAMETERS	-4	invalid function parameters
ERR_GEN_FILENOTEXISTS	-5	selected file does not exist
ERR_GEN_FUNCTIONFAILED	-6	function failed
ERR_GEN_NOCHANNELELECTED	-7	no channel selected
ERR_GEN_INVALIDCONF	-8	invalid instrument configuration
ERR_GEN_ECLAB_LOADED	-9	EC-Lab® firmware loaded on the instrument
ERR_GEN_LIBNOTCORRECTLYLOADED	-10	library not correctly loaded in memory
ERR_GEN_USBLIBRARYERROR	-11	USB library not correctly loaded in memory
ERR_GEN_FUNCTIONINPROGRESS	-12	function of the library already in progress
ERR_GEN_CHANNEL_RUNNING	-13	selected channel(s) already used
ERR_GEN_DEVICE_NOTALLOWED	-14	device not allowed
ERR_GEN_UPDATEPARAMETERS	-15	Invalid update function parameters

Instrument error codes

Constant	Value	Description
ERR_INSTR_VMEERROR	-101	internal instrument communication failed
ERR_INSTR_TOOMANYDATA	-102	too many data to transfer from the instrument (device error)
ERR_INSTR_RESPNOTPOSSIBLE	-103	selected channel(s) unplugged (device error)
ERR_INSTR_RESPERROR	-104	Instrument response error
ERR_INSTR_MSGSIZEERROR	-105	Invalid message size

Communication error codes

Constant	Value	Description
ERR_COMM_COMMFAILED	-200	communication failed with the instrument
ERR_COMM_CONNECTIONFAILED	-201	cannot establish connection with the

Communication error codes

Constant	Value	Description
		instrument
ERR_COMM_WAITINGACK	-202	waiting for the instrument response
ERR_COMM_INVALIDIPADDRESS	-203	invalid IP address
ERR_COMM_ALLOCMEMFAILED	-204	cannot allocate memory in the instrument
ERR_COMM_LOADFIRMWAREFAILED	-205	cannot load firmware into selected channel(s)
ERR_COMM_INCOMPATIBLESERVER	-206	communication firmware not compatible with the library
ERR_COMM_MAXCONNREACHED	-207	maximum number of allowed connections reached

Firmware error codes

Constant	Value	Description
ERR_FIRM_FIRMFILENOTEXISTS	-300	cannot find kernel.bin file
ERR_FIRM_FIRMFILEACCESSFAILED	-301	cannot read kernel.bin file
ERR_FIRM_FIRMINVALIDFILE	-302	invalid kernel.bin file
ERR_FIRM_FIRMLOADINGFAILED	-303	cannot load kernel.bin on the selected channel(s)
ERR_FIRM_XILFILENOTEXISTS	-304	cannot find x100_01.txt file
ERR_FIRM_XILFILEACCESSFAILED	-305	cannot read x100_01.txt file
ERR_FIRM_XILINVALIDFILE	-306	invalid x100_01.txt file
ERR_FIRM_XILLOADINGFAILED	-307	cannot load x100_01.txt file on the selected channel(s)
ERR_FIRM_FIRMWARENOTLOADED	-308	no firmware loaded on the selected channel(s)
ERR_FIRM_FIRMWAREINCOMPATIBLE	-309	loaded firmware not compatible with the library

Technique error codes

Constant	Value	Description
ERR_TECH_ECCFILENOTEXISTS	-400	cannot find the selected ECC file
ERR_TECH_INCOMPATIBLEECC	-401	ECC file not compatible with the channel firmware
ERR_TECH_ECCFILECORRUPTED	-402	ECC file corrupted
ERR_TECH_LOADTECHNIQUEFAILED	-403	cannot load the ECC file
ERR_TECH_DATACORRUPTED	-404	data returned by the instrument are corrupted
ERR_TECH_MEMFULL	-405	cannot load techniques: full memory

SP-300 series hardware options error codes (TCurrentValues.OptErr)		
Constant	**Value**	**Description**
ERR_OPT_CHANGE	1	Number of options changed
ERR_OPT_4A	100	4A amplifier unknown error
ERR_OPT_4A_OVRTEMP	101	4A amplifier temperature overflow
ERR_OPT_4A_BADPOW	102	4A amplifier bad power
ERR_OPT_4A_POWFAIL	103	4A amplifier power fail
ERR_OPT_48V	200	48V amplifier unknown error
ERR_OPT_48V_OVRTEMP	201	48V amplifier temperature overflow
ERR_OPT_48V_BADPOW	202	48V amplifier bad power
ERR_OPT_48V_POWFAIL	203	48V amplifier power fail
KBIO_OPT_10A5V_ERR	300	10A 5V amplifier error
KBIO_OPT_10A5V_OVRTEMP	301	10A 5V amplifier overheat
KBIO_OPT_10A5V_BADPOW	302	10A 5V amplifier bad power
KBIO_OPT_10A5V_POWFAIL	303	10A 5V amplifier power fail

6. Functions reference

6.1. Functions overview

Function overview of the library:

General functions	
BL_GETLIBVERSION	Return the version of the library
BL_GETVOLUMESERIALNUMBER	Return the volume serial number
BL_GETERRORMSG	Return the message corresponding to the selected error code

Communication functions	
BL_CONNECT	Open a connection with the selected instrument
BL_DISCONNECT	Close the connection
BL_TESTCONNECTION	Test the communication with the instrument
BL_TESTCOMMSPEED	Test the communication speed
BL_GETUSBDEVICEINFOS	Get informations of USB device selected

Firmware functions	
BL_LOADFIRMWARE	Load the firmware on selected channels

Channel informations functions	
BL_ISCHANNELPLUGGED	Test if selected channel is plugged
BL_GETCHANNELSPLUGGED	Return the channels plugged
BL_GETCHANNELINFOS	Return informations on selected channel
BL_GETMESSAGE	Return messages generated by the firmware of selected channel
BL_GETHARDCONF	Return ThardwareConf object with electrode connection mode and instrument ground.
BL_SETHARDCONF	Set the electrode connection mode and instrument ground with a ThardwareConf object.

Technique functions	
BL_LOADTECHNIQUE	Load a technique and its parameters on selected channel
BL_LOADTECHNIQUE_LV	loads a technique and its parameters on selected channel
BL_LOADTECHNIQUE_VEE	loads a technique and its parameters on selected channel Used with Vee Pro
BL_UPDATEPARAMETERS	Update a technique with new parameters on selected channel
BL_UPDATEPARAMETERS_LV	Update a technique with new parameters on selected channel Used with LabView
BL_UPDATEPARAMETERS_VEE	Update a technique with new parameters on selected channel Used with Vee Pro
BL_DEFINEBOOLPARAMETER	Populate TECCPARAM structure with a boolean
BL_DEFINESGLPARAMETER	Populate TECCPARAM structure with a single
BL_DEFINEINTPARAMETER	Populate TECCPARAM structure with an integer

Start/stop functions	
BL_STARTCHANNEL	Start technique(s) loaded on selected channel
BL_STARTCHANNELS	Start technique(s) loaded on selected channels
BL_STOPCHANNEL	Stops selected channel
BL_STOPCHANNELS	Stops selected channels

Data functions	
BL_GETCURRENTVALUES	Return current values (Ewe, Ece, I, t, ...) from selected channel
BL_GETDATA	Return data from selected channel
BL_GETDATA_VEE	Return data from selected channel Used with Vee Pro
BL_GETFCTDATA	Return FCT data from selected channel
BL_CONVERTNUMERICINTOSINGLE	Convert a numerical value coming from the data buffer into a single

Miscellaneous functions	
BL_SETEXPERIMENTINFOS	Save experiment informations on selected channel
BL_GETEXPERIMENTINFOS	Read experiment informations from selected channel
BL_SENDMSG	Send a message to the selected channel
BL_LOADFLASH	Update the communication firmware of the instrument

6.2. General functions

Function	**BL_GETLIBVERSION**
Syntax	function BL_GetLibVersion(pVersion: PChar; psize: puint32): int32;
Parameters	*pVersion* pointer to the buffer that will receive the text (C-string format) *psize* pointer to a uint32 who defines the maximum number of characters of the buffer. It also returns the number of characters of the copied string.
Return value	= 0 : the function succeeded < 0 : see section 5. Used structures and constants
Description	This function copies the version of the library into the buffer.

Delphi example	procedure DisplayVersion;
	var
	pVersion: PChar;
	len: int32;
	begin
	len:= 255
	pVersion:= StrAlloc(len);
	zeromemory(pVersion, len);
	BL_GetLibVersion(pVersion, @len);
	ShowMessage(pVersion);
	StrDispose(pVersion);
	end;

Function	**BL_GETVOLUMESERIALNUMBER**
Syntax	procedure BL_GetVolumeSerialNumber: uint32;
Description	This function returns volume serial number. NOTE: the serial number of a (logical) drive is generated every time a drive is formatted. When Windows formattes a drive, a drive's serial number gets calculated using the current date and time and is stored in the drive's boot sector. The odds of two disks getting the same number are virtually nil on the same machine.

Delphi example	procedure DisplayVolumeSerialNumber;
	var
	  VolumeSerialNumber: uint32;
	begin
	  VolumeSerialNumber := BL_GetVolumeSerialNumber;
	  ShowMessage(inttostr(VolumeSerialNumber));
	end;

Function	**BL_GETERRORMSG**
Syntax	function BL_GetErrorMsg(errorcode: int32; pmsg: PChar; psize: puint32): int32;
Parameters	*errorcode* error code selected *pmsg* pointer to the buffer that will receive the text (C-string format) *psize* pointer to a uint32 who defines the maximum number of characters of the buffer. It also returns the number of characters of the copied string.
Return value	= 0 : the function succeeded < 0 : see section 5. Used structures and constants
Description	This function copies into the buffer the corresponding message of the selected error code.

Delphi example	procedure DisplayErrorMessage;
	var
	  msg: PChar;
	  len: int32;
	begin
	  len := 255;
	  msg := StrAlloc(len);
	  zeromemory(msg, len);
	  BL_GetErrorMsg(-10, msg, @len);
	  ShowMessage(msg);
	  StrDispose(msg);
	end;

6.3. Communication functions

Function	**BL_CONNECT**
Syntax	function BL_Connect(pstr : PChar; TimeOut: uint8; pID: pint32; pInfos: PDeviceInfos): int32;
Parameters	*pstr* pointer to the buffer who defines the instrument selected (C-string format). Ex : 192.109.209.200, USB0, USB1, … (see section ANNEXE A. Find instruments to detect available instruments) *TimeOut* communication time-out in second (5 s recommended). This timeout will be used for the multithreading mutex that is used to control access to the network. When 2 functions are using the network, the second will be denied access until the first has finished. If the timeout is reached, the second function will return an error ERR_GEN_FUNCTIONINPROGRESS. *pID* pointer to a int32 that will receive the device identifier of the instrument *pInfos* pointer to a device informations structure (see section 5. Used structures and constants)
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function establishes the connection with the selected instrument and copies general informations (device code, RAM size, ...) into the TDEVICEINFOS structure. The identifier (ID) returned must be used in all other routines to communicate with the instrument.

Delphi example	var ID: int32; {device identifier}
	
	procedure Connect;
	var
	  IPaddress: array[0..15] of char; {IP address}
	  Infos: TDeviceInfos; {device informations}
	begin
	  IPaddress:= '192.109.209.226' + #0;
	  if BL_Connect(IPaddress, 10, @ID, @Infos) = 0 then
	    ShowMessage('Connection OK !');
	end;

Function	BL_DISCONNECT
Syntax	function BL_Disconnect(ID: int32): int32;
Parameters	ID : device identifier
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function closes the connection with the instrument.

Delphi example	var ID: int32; {device identifier}
	
	procedure Disconnect;
	begin
	  BL_Disconnect(ID);
	end;

Function	BL_TESTCONNECTION
Syntax	function BL_TestConnection(ID: int32): int32;
Parameters	ID: device identifier
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function tests the communication with the selected instrument.

Delphi example	var ID: int32; {device identifier}
	
	procedure TestConnection;
	begin
	  if BL_TestConnection(ID) = 0 then
	    ShowMessage('Connection OK')
	  else
	    ShowMessage('Connection failed !');
	end;

Function	BL_TESTCOMMSPEED
Syntax	function BL_TestCommSpeed(ID: int32; channel: uint8; spd_rcvt: pint32; spd_kernel: pint32): int32;
Parameters	*ID* device identifier *channel* selected channel (0 .. 15) *spd_rcvt* pointer to a int32 that will receive the communication speed (in ms) between the library and the device *spd_kernel* pointer to a int32 that will receive the communication speed (in ms) between the library and the selected channel
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function tests the communication speed between the library and the selected channel. This function is for advanced user only.

Delphi example	var ID: int32; {device identifier}
	
	procedure TestCommSpeed;
	var spd_rcvt, spd_kernel: int32;
	begin
	if BL_TestCommSpeed(FID,
	0,
	@spd_rcvt,
	@spd_kernel) = ERR_NOERROR then
	ShowMessage('Comm speed : ' + inttostr(spd_rcvt) + 'ms/' + inttostr(spd_kernel) + 'ms');
	end;

Function	**BL_GETUSBDEVICEINFOS**
Syntax	function BL_GetUSBdeviceinfos(USBindex: uint32; pcompany: PChar; pcompanysize: puint32; pdevice: PChar; pdevicesize: puint32; pSN: PChar; pSNsize: puint32): boolean;
Parameters	*USBindex* index of USB device selected (0-based) *pcompany* pointer to the buffer that will receive the company name (C-string format) *pcompanysize* pointer to a uint32 who defines the maximum number of characters of the buffer. It also returns the number of characters of the copied string. *pdevice* pointer to the buffer that will receive the device name (C-string format) *pdevicesize* pointer to a uint32 who defines the maximum number of characters of the buffer. It also returns the number of characters of the copied string. *pSN* pointer to the buffer that will receive the device serial number (C-string format) *pSNsize* pointer to a uint32 who defines the maximum number of characters of the buffer. It also returns the number of characters of the copied string.
Return value	= TRUE: the function succeeded = FALSE: the function failed
Description	This function returns information stored into the USB device selected. This function is for advanced user only.

6.4. Firmware functions

Function	**BL_LOADFIRMWARE**
Syntax	function BL_LoadFirmware(ID: int32; pChannels: puint8; pResults: pint32; Length: uint8; ShowGauge: boolean; ForceReload: boolean; BinFile: Pchar; XlxFile: PChar): int32;
Parameters	*ID* device identifier *pChannels* pointer to the array who represents the channels of the device. For each element of the array: = 0: channel not selected = 1: channel selected *pResults* pointer to the array that will receive the result of the function for each channels : = 0: the function succeeded < 0: see section 4.3 for error codes *Length* length of the arrays pointed by pChannels and pResults. *ShowGauge* if TRUE a gauge is shown during the firmware loading. *ForceReload* if TRUE the firmware is loaded each time, if FALSE the firmware is loaded only if not already done. *BinFile* Name of bin file (nil for default file). *XlxFile* Name of xilinx file (nil for default file).
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function loads the firmware on selected channels. Be aware that channels are unusable until the firmware is loaded.

Delphi example

```
var ID: int32;   {device identifier}

procedure LoadFirmware;
var
  Channels: array[1..16] of uint8;
  Results: array[1..16] of int32;
begin
  {Initialize array}
  BL_GetChannelsPlugged(ID, @Channels, 16);
  zeromemory(@Results, sizeof(Results));

  {Load firmware}
  BL_LoadFirmware(ID, @Channels, @Results, 16, TRUE, FALSE);
end;
```

6.5. Channel information functions

Function	**BL_ISCHANNELPLUGGED**
Syntax	function BL_IsChannelPlugged(ID: int32; ch: uint8): boolean;
Parameters	*ID* device identifier *ch* selected channel (0 .. 15)
Return value	TRUE: selected channel is plugged FALSE: selected channel is not plugged
Description	This function tests if the selected channel is plugged.

Delphi example	var ID: int32; {device identifier}
	
	procedure TestChannelsPlugged;
	var
	  i: int32;
	begin
	  for i:= 0 to 15 do
	  begin
	    if BL_IsChannelPlugged(ID, i) then
	      ShowMessage('Channel #' + inttostr(i) + ' plugged');
	  end;
	end;

Function	**BL_GETCHANNELSPLUGGED**
Syntax	function BL_GetChannelsPlugged (ID: int32; pChPlugged: puint8; Size: uint8): int32;
Parameters	*ID* device identifier *pChPlugged* pointer to the array who represents the channels of the device. For each element of the array : 0 = channel not plugged 1 = channel plugged

	Size size of the array pointed by pChPlugged
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function returns the channels plugged.
Delphi example	See example of BL_LOADFIRMWARE function

Function	**BL_GETCHANNELINFOS**
Syntax	function BL_GetChannelInfos (ID: int32; ch: uint8; pInfos: PChannelInfos): int32;
Parameters	*ID* device identifier *ch* channel selected (0 .. 15) pInfos pointer on a channel informations structure (see section 5. Used structures and constants)
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function copies information of selected channel into the TCHANNELINFOS structure.
Delphi example	see code below

```
var ID: int32;   {device identifier}

procedure DisplayChannelInfos;
var
  Infos: TChannelInfos;
begin
  if BL_GetChannelInfos(ID, 0, @Infos) = 0 then
  begin
   ShowMessage(
        'SerialNumber=' + inttostr( Infos.BoardSerialNumber ) +
        '  MemSize=' + inttostr(Infos.MemSize));
  end;
end;
```

Function	BL_GETMESSAGE
Syntax	function BL_GetMessage(ID: int32; ch: uint8; msg: PChar; size: puint32): int32;
Parameters	*ID* device identifier *channel* selected channel (0 .. 15) *msg* pointer to the buffer that will receive the text (C-string format) *size* pointer to a uint32 who defines the maximum number of characters of the buffer. It also returns the number of characters of the copied string.
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function copies into the buffer the messages generated by the firmware of selected channel. Be aware that messages are retrieved one-by-one, this implies this function must be called several times in order to get all messages available in the message queue.
Delphi example	see below

```
var ID: int32;   {device identifier}

procedure DisplayMessages;
var
  msg: PChar;
  len: int32;
begin
  msg:= StrAlloc(255);
  repeat
    len:= 255;
    ZeroMemory(msg, len);
    if BL_GetMessage(ID, 0, msg, @len) <> 0 then Exit;
    if len > 0 then
      ShowMessage(msg);
  until (len = 0);
  StrDispose(msg);
end;
```

Function	BL_GETHARDCONF
Syntax	function BL_GetHardConf(ID: int32; ch: uint8; pHardConf : PHardwareConf): int32;
Parameters	*ID* device identifier *ch* selected channel (0 .. 15) *pHardConf* pointer to a ThardwareConf object
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function return in *pHardConf* the hardware configuration of the channel *ch*. The hardware configuration is the electrode connection and the instrument ground. This function must be used only with SP-300 series.

Delphi example	procedure GetHardConf(aId : int32;
	aCh : int32;
	var apHardConf : THardwareConf);
	var
	errcode : int32;
	
	begin
	errcode := BL_GetHardConf(aId, aCh, @apHardConf);
	end;

Function	BL_SETHARDCONF
Syntax	function BL_SetHardConf(ID: int32; ch: uint8; HardConf : THardwareConf): int32;
Parameters	*ID* device identifier *ch* selected channel (0 .. 15) *HardConf* ThardwareConf object. The attribute "Conn" is the electrode connection mode and the attribute "Ground" is the instrument ground. See 5. Used structures and constants to have the value of these attributes.
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function set the hardware configuration of a channel with *HardConf* object. This function must be used only with SP-300 series.

Delphi example	procedure SetHardConf(aId : int32;
	aCh : int32;
	var aHardConf : THardwareConf);
	var
	errcode : int32;
	
	begin
	errcode := BL_SetHardConf(aId, aCh, aHardConf);
	end;

6.6. Technique functions

Function	**BL_LOADTECHNIQUE**
Syntax	function BL_LoadTechnique (ID: int32; channel: uint8; pFName: PChar; Params: TEccParams; FirstTechnique: boolean; LastTechnique: boolean; DisplayParams: boolean): int32;
Parameters	*ID* device identifier *channel* selected channel (0 .. 15) *pFName* pointer to the buffer who contains the name of the *.ecc file who defines the technique (C-string format) *Params* structure of parameters of selected technique (see section 5. Used structures and constants). See section 7. Techniques for a complete description of parameters available for each technique. *FirstTechnique* TRUE if the technique loaded is the first one *LastTechnique* TRUE if the technique loaded is the last one *DisplayParams* Display parameters sent (for debugging purpose)
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function loads a technique and its parameters on the selected channel. Note: to run linked techniques, this function must be called for each selected technique.

Delphi example	`var ID: int32;   {device identifier}`
	
	`function LoadTechniques: boolean;`

```
var
   {OCV}
   EccParamArray_OCV: array of TEccParam;
   EccParams_OCV: TEccParams;
   fname_OCV: PChar;
   {VSCAN}
   EccParamArray_VSCAN: array of TEccParam;
   EccParams_VSCAN: TEccParams;
   fname_VSCAN: Pchar;

begin
   Result:= FALSE;
   fname_OCV:= nil;
   fname_VSCAN:= nil;
   SetLength(EccParamArray_OCV, 4);
   SetLength(EccParamArray_VSCAN, 20);
   try
      {Define OCV parameters}
      fname_OCV:= StrNew('ocv.ecc' + #0);
      BL_DefineSglParameter('Rest_time_T', 0.1, 0,
                            @EccParamArray_OCV[0]);
      BL_DefineSglParameter('Record_every_dE', 0.1, 0,
                            @EccParamArray_OCV[1]);
      BL_DefineSglParameter('Record_every_dT', 0.01, 0,
                            @EccParamArray_OCV[2]);
      BL_DefineIntParameter('E_Range', ERANGE_AUTO, 0,
                            @EccParamArray_OCV[3]);

      {Load OCV on selected channel}
      EccParams_OCV.len:= length(EccParamArray_OCV);
      EccParams_OCV.pParams:= @EccParamArray_OCV[0];
      if BL_LoadTechnique(ID,                  {device identifier}
                          0,                   {selected channel}
                          fname_OCV,           {ECC filename (c-string)}
                          EccParams_OCV,       {parameters}
                          TRUE,                {first technique}
                          FALSE,               {last technique}
                          FALSE) <> 0 then Exit; {display parameters}

      {Define VSCAN parameters}
      fname_VSCAN:= StrNew('vscan.ecc' + #0);
      {Vertex #0}
      BL_DefineSglParameter ('Voltage_step', 0.0, 0,
                             @EccParamArray_VSCAN[0]);
      BL_DefineBoolParameter('vs_initial', FALSE, 0,
                             @EccParamArray_VSCAN[1]);
      BL_DefineSglParameter ('Scan_Rate', 0.0, 0,
                             @EccParamArray_VSCAN[2]);
      {Vertex #1}
      BL_DefineSglParameter ('Voltage_step', 1.0, 1,
                             @EccParamArray_VSCAN[3]);
      BL_DefineBoolParameter('vs_initial', FALSE, 1,
                             @EccParamArray_VSCAN[4]);
      BL_DefineSglParameter ('Scan_Rate', 10.0, 1,
                             @EccParamArray_VSCAN[5]);
      {Vertex #2}
      BL_DefineSglParameter ('Voltage_step', -2.0, 2,
                             @EccParamArray_VSCAN[6]);
      BL_DefineBoolParameter('vs_initial', FALSE, 2,
                             @EccParamArray_VSCAN[7]);
      BL_DefineSglParameter ('Scan_Rate', 15.0, 2,
                             @EccParamArray_VSCAN[8]);
      {Vertex #3}
      BL_DefineSglParameter ('Voltage_step', 0.0, 3,
                             @EccParamArray_VSCAN[9]);
      BL_DefineBoolParameter('vs_initial', FALSE, 3,
                             @EccParamArray_VSCAN[10]);
      BL_DefineSglParameter ('Scan_Rate', 20.0, 3,
                             @EccParamArray_VSCAN[11]);
      {Misc.}
      BL_DefineIntParameter ('Scan_number', 2, 0,
                             @EccParamArray_VSCAN[12]);
      BL_DefineIntParameter ('N_Cycles', 0,  0,
```

```
                                @EccParamArray_VSCAN[13]);
      BL_DefineSglParameter ('Record_every_dE', 0.01, 0,
                                @EccParamArray_VSCAN[14]);
      BL_DefineSglParameter ('Begin_measuring_I', 0.4, 0,
                                @EccParamArray_VSCAN[15]);
      BL_DefineSglParameter ('End_measuring_I', 0.8, 0,
                                @EccParamArray_VSCAN[16]);
      BL_DefineIntParameter ('I_Range', IRANGE_10MA, 0,
                                @EccParamArray_VSCAN[17]);
      BL_DefineIntParameter ('E_Range', ERANGE_AUTO, 0,
                                @EccParamArray_VSCAN[18]);
      BL_DefineIntParameter ('Bandwidth', BANDWIDTH_5, 0,
                                @EccParamArray_VSCAN[19]);

      {Load VSCAN on selected channel}
      EccParams_VSCAN.len:= length(EccParamArray_VSCAN);
      EccParams_VSCAN.pParams:= @EccParamArray_VSCAN[0];
      if BL_LoadTechnique(ID,                {device identifier}
                          0,                 {selected channel}
                          fname_VSCAN,       {*.ecc filename (c-string)}
                          EccParams_VSCAN,   {parameters}
                          FALSE,             {first technique}
                          TRUE,              {last technique}
                          FALSE) <> 0 then Exit; {display parameters}
      Result:= TRUE;
   finally
      if fname_OCV <> nil then StrDispose(fname_OCV);
      if fname_VSCAN <> nil then StrDispose(fname_VSCAN);
      SetLength(EccParamArray_OCV, 0);
      SetLength(EccParamArray_VSCAN, 0);
   end;
end;
```

Function	BL_LOADTECHNIQUE_LV
Syntax	function BL_LoadTechnique_LV (ID: int32; channel: uint8; pFName: PChar; HdlParams: PPEccParams_LV; FirstTechnique: boolean; LastTechnique: boolean; DisplayParams: boolean): int32;
Parameters	*ID* device identifier *channel* selected channel (0 .. 15) *FName* pointer to the buffer who contains the name of the *.ecc file who defines the technique (C-string format) *HdlParams* structure of parameters of selected technique (see section 5. Used structures and constants). See section 7. Techniques for a complete description of parameters available for each technique. *FirstTechnique* TRUE if the technique loaded is the first one. *LastTechnique* TRUE if the technique loaded is the last one. *DisplayParams* Display parameters sent (for debugging purpose).
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function loads a technique and its parameters on the selected channel. It has been developed for LabVIEW compatibility. Note: to run linked techniques, this function must be called for each selected technique.
Delphi example	See example of BL_LOADTECHNIQUE function

Function	BL_LOADTECHNIQUE_VEE
Syntax	function BL_LoadTechnique_VEE (ID: int32; channel: uint8; pFName: PChar; appParams: PPChar; FirstTechnique: boolean; LastTechnique: boolean; DisplayParams: boolean): int32;
Parameters	*ID* device identifier *channel* selected channel (0 .. 15) *FName* pointer to the buffer who contains the name of the *.ecc file who defines the technique (C-string format) *appParams* array of parameters of selected technique (see section 5. Used structures and constants). See section 7. Techniques for a complete description of parameters available for each technique. Format of a parameter in the array of char: *ParamStr;ParamType;ParamVal;ParamIndex* *FirstTechnique* TRUE if the technique loaded is the first one. *LastTechnique* TRUE if the technique loaded is the last one. *DisplayParams* Display parameters sent (for debugging purpose).
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Desscription	This function loads a technique and its parameters on the selected channel. It has been developed for Vee Pro compatibility. Note: to run linked techniques, this function must be called for each selected technique.
Delphi example	See Vee Pro example.

Function	BL_DEFINEBOOLPARAMETER
Syntax	function BL_DefineBoolParameter(lbl: PChar; value: boolean; index: int32; pParam: PEccParam): int32;
Parameters	*lbl* parameter label (C-string format) *value* parameter value (boolean) *index* parameter index (useful only for multi-step parameters) *pParam* pointer on a elementary technique parameter structure (see section 5. Used structures and constants)
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function must be used to populate TECCPARAM structure with a boolean.
Delphi example	See example of BL_LOADTECHNIQUE function

Function	BL_DEFINESGLPARAMETER
Syntax	function BL_DefineSglParameter(lbl: PChar; value: single; index: int32; pParam: PEccParam): int32;
Parameters	*lbl* parameter label (C-string format) *value* parameter value (single) *index* parameter index (useful only for multi-step parameters) *pParam* pointer on a elementary technique parameter structure (see section 5. Used structures and constants)
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function must be used to populate TECCPARAM structure with a single.

Delphi example	See example of BL_LOADTECHNIQUE function

Function	**BL_DEFINEINTPARAMETER**
Syntax	function BL_DefineIntParameter(lbl: PChar; value: int32; index: int32; pParam: PEccParam): int32;
Parameters	*lbl* parameter label (C-string format) *value* parameter value (int32) *index* parameter index (useful only for multi-step parameters) *pParam* pointer on a elementary technique parameter structure (see section 5. Used structures and constants)
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function must be used to populate TECCPARAM structure with an integer
Delphi example	See example of BL_LOADTECHNIQUE function

Function	**BL_UPDATEPARAMETERS**
Syntax	function BL_UpdateParameters(ID: int32; channel: uint8; TechIndx: int32; Params: TeccParams; EccFileName: PAnsiChar): int32;
Parameters	*ID* device identifier *channel* selected channel (0 .. 15) *TechIndx* index of the technique if several techniques have been started.

	The first have index 0, the second have index 1, ... *Params* array of parameters of selected technique (see section 5. Used structures and constants). See section 7. Techniques for a complete description of parameters available for each technique. *EccFileName* pointer to the buffer who contains the name of the *.ecc file who defines the technique (C-string format)
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function updates a technique with new parameters on the selected channel. You should call this function only while an experiment is running.
Note	• For timing reasons, you cannot update more than 10 parameters at a time. Any call to this function with more than 10 parameters will fail. • There are parameters that cannot be changed while running. These are set before the technique begins and are unlocked only at the end, see 7.1.3 Hardware parameters.
Delphi example	See example of BL_LOADTECHNIQUE function

Function	**BL_UPDATEPARAMETERS_LV**
Syntax	function BL_UpdateParameters_LV(ID: int32; channel: uint8; TechIndx: int32; HdlParams: PPEccParams_LV; EccFileName: PAnsiChar): int32;
Parameters	*ID* device identifier *channel* selected channel (0 .. 15) *TechIndx* index of the technique if several techniques have been started. The first have index 0, the second have index 1, ... *HdlParams* array of parameters of selected technique (see section 5. Used structures and constants). See section 7. Techniques for a complete description of parameters available for each technique.

	EccFileName pointer to the buffer who contains the name of the *.ecc file who defines the technique (C-string format)
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function updates a technique with new parameters on the selected channel. It has been developed for LabView compatibility.
Note	• For timing reasons, you cannot update more than 10 parameters at a time. Any call to this function with more than 10 parameters will fail. • There are parameters that cannot be changed while running.These are set before the technique begins and are unlocked only at the end, see 7.1.3 Hardware parameters.
Delphi example	See OCV LabView examples (ocv.vi)

Function	**BL_UPDATEPARAMETERS_VEE**
Syntax	function BL_UpdateParameters_VEE(ID: int32; channel: uint8; TechIndx: int32; appParams: PPChar; EccFileName: PAnsiChar): int32;
Parameters	*ID* device identifier *channel* selected channel (0 .. 15) *TechIndx* index of the technique if several techniques have been started. The first have index 0, the second have index 1, ... *appParams* array of parameters of selected technique (see section 5. Used structures and constants). See section 7. Techniques for a complete description of parameters available for each technique. Format of a parameter in the array of char: *ParamStr;ParamType;ParamVal;ParamIndex* ParamIndex is used for step parameters. It is the index in the array of step parameters.
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function updates a technique with new parameters on the selected channel. It has been developed for Vee Pro compatibility.

Note	• For timing reasons, you cannot update more than 10 parameters at a time. Any call to this function with more than 10 parameters will fail. • There are parameters that cannot be changed while running.These are set before the technique begins and are unlocked only at the end, see 7.1.3 Hardware parameters.
Delphi example	See Vee Pro example.

6.7. Start/stop functions

Function	**BL_STARTCHANNEL**
Syntax	function BL_StartChannel(ID: int32; channel: uint8): int32;
Parameters	*ID* device identifier *channel* selected channel (0 .. 15)
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function starts technique(s) loaded on selected channel.

Delphi example	var ID: int32; {device identifier}
	
	procedure StartChannel;
	begin
	if BL_StartChannel(ID, 0) = 0 then
	ShowMessage('Channel started !');
	end;

Function	**BL_STARTCHANNELS**
Syntax	function BL_StartChannels(ID: int32; pChannels: puint8; pResults: pint32; Length: uint8): int32;
Parameters	*ID* device identifier *pChannels* pointer to the array who represents the channels of the device. For each element of the array: = 0: channel not selected = 1: channel selected *pResults* pointer to the array that will receive the result of the function for each channels : = 0: the function succeeded < 0: see section 5. Used structures and constants

	Length length of the arrays pointed by pChannels and pResults.
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function starts technique(s) loaded on selected channels.
Delphi example	see code below

```
var ID: int32;   {device identifier}

procedure StartChannels;
var
  Channels: array[1..16] of uint8;
  Results: array[1..16] of int32;
begin
  {Initialize array}
  BL_GetChannelsPlugged(ID, @Channels, 16);
  zeromemory(@Results, sizeof(Results));

  {Start channels}
  if BL_StartChannels(ID, @Channels, @Results, 16) = 0 then
    ShowMessage('Channels started !');
end;
```

Function	**BL_STOPCHANNEL**
Syntax	`function BL_StopChannel(ID: int32;` `                        channel: uint8): int32;`
Parameters	*ID* device identifier *channel* selected channel (0 .. 15)
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function stops selected channel.
Delphi example	see code below

```
var ID: int32;   {device identifier}

procedure StopChannel;
begin
  if BL_StopChannel(ID, 0) = 0 then
    ShowMessage('Channel stopped !');
end;
```

Function	BL_STOPCHANNELS
Syntax	function BL_StopChannels(ID: int32; pChannels: puint8; pResults: pint32; Length: uint8): int32;
Parameters	*ID* device identifier *pChannels* pointer to the array who represents the channels of the device. For each element of the array: = 0: channel not selected = 1: channel selected *pResults* pointer to the array that will receive the result of the function for each channels: = 0: the function succeeded < 0: see section 5. Used structures and constants *Length* length of the arrays pointed by pChannels and pResults.
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function stops selected channels.
Delphi example	see below

```
var ID: int32;   {device identifier}

procedure StopChannels;
var
  Channels: array[1..16] of uint8;
  Results: array[1..16] of int32;
begin
  {Initialize array}
  BL_GetChannelsPlugged(ID, @Channels, 16);
  zeromemory(@Results, sizeof(Results));

  {Stop channels}
  if BL_StopChannels(ID, @Channels, @Results, 16) = 0 then
    ShowMessage('Channels stopped !');
end;
```

6.8. Data functions

Function	BL_GETCURRENTVALUES
Syntax	function BL_GetCurrentValues(ID: int32; channel: uint8; pValues: PCurrentValues): int32;
Parameters	*ID* device identifier *channel* selected channel (0 .. 15) *pValues* pointer to a current values structure (see section 5. Used structures and constants)
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function copies current values (Ewe, Ece, I, t, ...) from selected channel into the structure TCURRENTVALUES.

Delphi example	var ID: int32; {device identifier}
	
	procedure DisplayCurrentValues;
	var
	curvalues: TCurrentValues;
	begin
	if BL_GetCurrentValues(ID, 0, @curvalues) = 0 then
	ShowMessage('Ewe(V)=' + FloatToStr(curvalues.Ewe) +
	' Ece(V)=' + FloatToStr(curvalues.Ece) +
	' I(A)=' + FloatToStr(curvalues.I));
	end;

Function	BL_GETDATA
Syntax	function BL_GetData (ID : int32; channel : uint8; pBuf : PDataBuffer; pInfos : PdataInfos; pValues : PCurrentValues): int32;
Parameters	*ID* device identifier *channel*

selected channel (0 .. 15)

pBuf

pointer to the buffer that will receive data.

pInfos

pointer to a data informations structure (see section 5. Used structures and constants)

pValues

pointer to TCURRENTVALUES structure (see section 5. Used structures and constants)

Return value

= 0: the function succeeded
< 0: see section 5. Used structures and constants

Description

This function copies data from selected channel into the buffer and copies informations into the TDATAINFOS structure.

The field TECHNIQUEID of the structure TDATAINFOS contains the ID of the technique used to record data. Thanks to this technique ID one can identify the format of the data saved into the buffer (see section 7. Techniques for a complete description of data format for each technique).
Be aware that techniques can also be composed of several process (PEIS and GEIS for instance). In this case one can identify the process used to record data with the field PROCESSINDEX of the structure TDATAINFOS.

Delphi example

```
var ID: int32;   {device identifier}

procedure GetData;
var
   buf: TPDataBuffer;
   infos: TdataInfos;
   values: TcurrentValues;
   idx: int32;
   row: int32;
   thigh: int32;
   tlow: int32;
   t: double;
   Ec: single;
   Ewe: single;
   Ece: single;
   I: single;
   Imoy: single;
   freq: single;
   cycle: int32;
begin
   repeat
      {Get data}
      if BL_GetData(ID, 0, @buf, @infos, @values) <> 0 then Exit;
      ShowMessage('Technique ID='     + inttostr(infos.TechniqueID) +
                  ' Technique index=' + inttostr(infos.TechniqueIndex) +
                  ' Number of points=' + inttostr(infos.NbRows));

      {Display data}
      for row := 0 to infos.NbRows - 1 do
      begin
         idx := row*infos.NbCols;

         {OCV technique}
```

```
        if (infos.TechniqueID = TECHNIQUEID_OCV) then
        begin
           thigh := buf[idx+1];
           tlow := buf[idx+2];
           t := ((thigh shl 32) + tlow)*infos.CurrentValues.TimeBase +
              infos.StartTime*0.001;
           BL_ConvertNumericIntoSingle(buf[idx+3], @Ewe);
           BL_ConvertNumericIntoSingle(buf[idx+4], @Ece);
           ShowMessage('t='    + format('%.3e', [t])    + 's ' +
                       'Ewe='  + format('%.3e', [Ewe])  + 'V ' +
                       'Ece='  + format('%.3e', [Ece])  + 'V ');
        end

        {CV technique}
        else if (infos.TechniqueID = TECHNIQUEID_CV) then
        begin
           thigh := buf[idx+1];
           tlow := buf[idx+2];
           t := ((thigh shl 32) + tlow)*infos.CurrentValues.TimeBase +
              infos.StartTime*0.001;
           BL_ConvertNumericIntoSingle(buf[idx+3], @Ec);
           BL_ConvertNumericIntoSingle(buf[idx+4], @Imoy);
           BL_ConvertNumericIntoSingle(buf[idx+5], @Ewe);
           cycle := buf[idx+6];
           ShowMessage('t='     + format('%.3e', [t])    + 's ' +
                       'Ec='    + format('%.3e', [Ec])   + 'V ' +
                       'Imoy='  + format('%.3e', [Imoy]) + 'A ' +
                       'Ewe='   + format('%.3e', [Ewe])  + 'V ' +
                       'cycle=' + inttostr(cycle));
        end

        {PEIS technique}
        else if (infos.TechniqueID = TECHNIQUEID_PEIS) then
        begin
           if infos.ProcessIndex = 0 then {PEIS, 1th process}
           begin
              thigh:= buf[idx+1];
              tlow:= buf[idx+2];
              t:= ((thigh shl 32) + tlow) *
                  infos.CurrentValues.TimeBase + infos.StartTime*0.001;
              BL_ConvertNumericIntoSingle(buf[idx+3], @Ewe);
              BL_ConvertNumericIntoSingle(buf[idx+4], @I);
              ShowMessage('t='   + format('%.3e', [t])      + 's ' +
                          'Ewe=' + format('%.3e', [Ewe]) + 'V ' +
                          'I='   + format('%.3e', [I])     + 'A ');
           end
           else if infos.ProcessIndex = 1 then {PEIS, 2th process}
           begin
              BL_ConvertNumericIntoSingle(buf[idx+1], @freq);
              {...}
              BL_ConvertNumericIntoSingle(buf[idx+5], @Ewe);
              BL_ConvertNumericIntoSingle(buf[idx+6], @I);
              {...}
              BL_ConvertNumericIntoSingle(buf[idx+14],@t);
              {...}
              ShowMessage('freq='  + format('%.3e', [freq]) + 'Hz ' +
                          't='     + format('%.3e', [t])    + 's ' +
                          'Ewe='   + format('%.3e', [Ewe])  + 'V ' +
                          'I='     + format('%.3e', [I])    + 'A ');
           end;
        end;

      end;   {for-loop}
   until (infos.CurrentValues.MemFilled = 0) and
          (infos.CurrentValues.State = STATE_STOP);
end;
```

Function	**BL_GETFCTDATA**
Syntax	function BL_GetFCTData (ID: int32; channel: uint8; pBuf: PDataBuffer; pInfos: PdataInfos; pValues : PCurrentValues): int32;
Parameters	*ID* device identifier *channel* selected channel (0 .. 15) *pBuf* pointer to the buffer that will receive data. *pInfos* pointer to a data informations structure (see section 5. Used structures and constants) *pValues* pointer to TCURRENTVALUES structure (see section 5. Used structures and constants)
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function copies data from selected channel into the buffer and copies informations into the TDATAINFOS structure. Function for advanced user only.
Delphi example	See example of BL_GETDATA function

Function	**BL_GETDATA_VEE**
Syntax	function BL_GetData_VEE (ID : int32; channel : uint8; pBuf : PDataBuffer; pInfos : TarrayDouble; pvalues : TarrayDouble): int32;
Parameters	*ID* device identifier *channel* selected channel (0 .. 15) *pBuf*

	pointer to the buffer that will receive data. *pInfos* pointer to a TarrayDouble structure (see section 5. Used structures and constants) *pvalues* pointer to a TarrayDouble structure (see section 5. Used structures and constants)
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function copies data from selected channel into the buffer and copies informations into the TArrayDouble structure. This structure is mapping to structure TDATAINFOS (pInfos) and TCURRENTVALUES (pvalues). See BL_GETDATA function description.
Delphi Example	See Vee Pro example.

Function	**BL_CONVERTNUMERICINTOSINGLE**
Syntax	function BL_ConvertNumericIntoSingle(num: uint32; psgl: psingle): int32;
Parameters	*num* numerical value *psgl* pointer to the single that will receive the result of conversion
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function converts a numerical value coming from the data buffer of BL_GETDATA function into single.
Delphi example	See example of BL_GETDATA function

6.9. Miscellaneous functions

Function	**BL_SETEXPERIMENTINFOS**
Syntax	function BL_SetExperimentInfos(ID: int32; channel: uint8; ExpInfos: TExperimentInfos): int32;
Parameters	*ID* device identifier *channel* selected channel (0 .. 15) *ExpInfos* Experiment informations (see section 5. Used structures and constants)
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function saves experiment informations on selected channel.

Delphi example	var ID: int32; {device identifier}
	
	procedure SaveExperimentInfos;
	var
	ExpInfos: TExperimentInfos;
	begin
	zeromemory(@ExpInfos, sizeof(TExperimentInfos));
	ExpInfos.Group:= 0;
	ExpInfos.PCidentifier:= 1;
	ExpInfos.TimeHMS:= 2;
	ExpInfos.TimeYMD:= 3;
	StrCopy(@ExpInfos.FileName, 'test');
	if BL_SetExperimentInfos(ID, 0, ExpInfos) = 0 then
	ShowMessage('Experiment information saved !');
	end;

Function	**BL_GETEXPERIMENTINFOS**
Syntax	function BL_GetExperimentInfos(ID: int32; channel: uint8; pExpInfos: PexperimentInfos): int32;

Parameters	*ID* device identifier *channel* selected channel (0 .. 15) *pExpInfos* pointer to an experiment information structure (see section 5. Used structures and constants)
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function copies experiment informations from selected channel into the TEXPERIMENTINFOS structure.
Delphi example	see code below

```
var ID: int32;   {device identifier}

procedure ReadExperimentInfos;
var
  ExpInfos: TExperimentInfos;
begin
  zeromemory(@ExpInfos, sizeof(TExperimentInfos));
  if BL_GetExperimentInfos(ID, 1, @ExpInfos) = 0 then
    ShowMessage('Experiment information read !');
end;
```

Function	**BL_SENDMSG**
Syntax	see code below
Parameters	*ID* device identifier *ch* selected channel (0 .. 15) *pbuf* pointer to the data buffer *plen* pointer to a uint32 who defines the length of data to transfer. It also returns the length of data copied into the buffer.
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants

```
function  BL_SendMsg(ID: int32;
                     ch: uint8;
                     pbuf: pointer;
                     plen: puint32): int32;
```

Description	This function sends a message to the selected channel. Function for advanced user only.

Function	**BL_LOADFLASH**
Syntax	function BL_LoadFlash(ID: int32; pfname: PChar; ShowGauge: boolean): int32;
Parameters	*ID* device identifier *pfname* file name (*.flash extension) of the new communication firmware (C-string format) *ShowGauge* show a gauge during transfer
Return value	= 0: the function succeeded < 0: see section 5. Used structures and constants
Description	This function updates the communication firmware of the instrument. Function for advanced users only.

7. Techniques

7.1. Notes

- See the description of the function BL_LOADTECHNIQUE for a complete example of how to load a technique and its parameters on a channel.
- See the description of the function BL_GETDATA for a complete example of data recovery and data conversion.
- PEIS, GEIS, SPEIS, SGEIS, PZIR and GZIR techniques can only be used with boards with impedance capabilities!
- Electrochemical techniques parameters must be carefully programmed according to the instrument hardware specifications. Be aware that wrong parameters can generate faulty operations of the technique.
- Please note that the name of the ECC file is *name*.ecc except to SP-300 series. For SP-300 series the name is *name*4.ecc.
- The counter-electrode value of the EIS techniques must not be taken into account with the SP-300 series.
- Some technique parameters cannot be modified when an experiment is running. They are set at the start of the technique and are unlocked only when it is finished. The immutable parameters are presented in section 7.1.3.
- The parameters names are *case-sensitive*. The BL_LOADTECHNIQUE and BL_UPDATEPARAMETERS will issue an error when a parameter is not recognized.

7.1.1 Recording additional values and XCTR changes from v5.34

From version 5.34 and onwards, the *xctr* parameter was enhanced to allow recording of some additional variables during an experiment.

As a consequence, some experiment variables are not recorded by default anymore in several techniques, and must be specified by the xctr parameter.
On the other hand, it allowed several technique timebase to be lowered since the record step does not take as much time as before. For instance, the CV technique got its timebase lowered from 50 to 45 µs.

If you want to record more than the default fields, you will have to specify the additional values to record through the XCTR parameter. Any value added to the record step will appear in the data array that is returned by the BL_GetData function in the order of the XCTR bitfield flag to set.

You will also need to manually add a small delay (depending on the value recorded, see 7.1.2) to the timebase, otherwise the technique might miss some measurements. These steps are explained in the XCTR definition and in the technique documentation.

7.1.2 Timebase calculation

The timebase that is presented in the next pages is the minimum amount of time that you should allow between two records for a technique. It provides sufficient time for the technique to record the basic set of data it has to measure.

When using XCTR to record additional experiment values, you will have to specify a timebase that is large enough for the machine to proceed to the acquisition of these extra values. The two following points will tell you how to calculate the optimal delay to add to the technique:

- Adding one extra measurement has a base cost of **5µs**. So anytime you start adding records to your technique, you should add 5µs to the original timebase.
- Each additional measurement after the first one has a cost of **0.5µs**, but you should round the final delay at the next integer value

Examples of calculation

Additional value	*Delay to add to timebase*
Ece	**5µs**
Ece, Analog IN1	5 + 0.5 = 5.5µs rounded to **6µs**
Ece, Analog IN1, IRange	5+0.5+0.5 = **6µs**

7.1.3 Hardware parameters

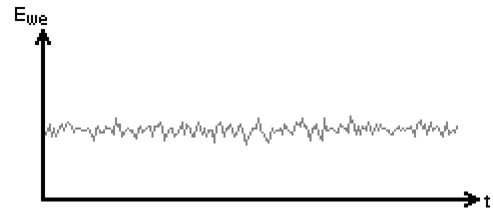
Some parameters configure the base acquisition and cannot be modified throughout the experiment. They are set at the beginning of the technique and unlocked at the end. You can set them regardless of the technique.

Hardware parameters			
Label	**Description**	**Data type**	**Data range**
I_Range	I range	integer	see IRange constants allowed on section 5. Used structures and constants
E_Range	Ewe range	integer	see ERange constants allowed on section 5. Used structures and constants
Bandwidth	Bandwidth	integer	see bandwidth constants allowed on section 5. Used structures and constants
tb	Time base (s)	single	>0, often µs range

7.2. Open Circuit Voltage technique

Technique ID: **100**

Instrument Series	VMP3	SP-300
File	**ocv.ecc**	**ocv4.ecc**
Timebase	**20µs**	**20µs**



7.2.1. Description

The Open Circuit Voltage (OCV) technique consists of a period during which no potential or current is applied to the working electrode. The cell is disconnected from the power amplifier. Only, the potential measurement is available. So the evolution of the rest potential can be recorded.

7.2.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

OCV parameters			
Label	**Description**	**Data type**	**Data range**
Rest_time_T	Rest duration (s)	single	[0..tb*2^{31}]
Record_every_dE	Record every dE (V)	single	≥ 0
Record_every_dT	Record every dT (s)	single	≥ 0

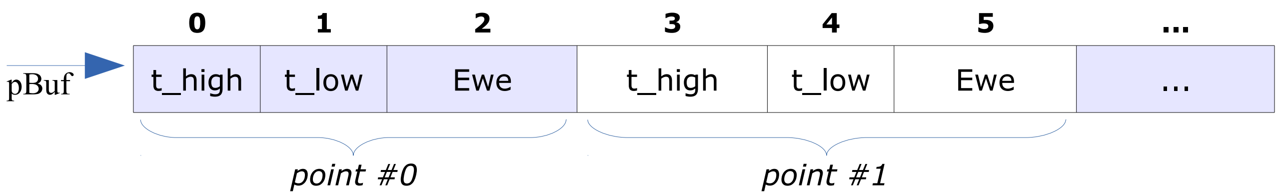
7.2.3. Data format

Data format returned by the function BL_GETDATA:

VMP3 series:

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
pBuf	t_high	t_low	Ewe	Ece	t_high	t_low	Ewe	Ece	...
	point #0				*point #1*				

SP-300 series:



The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.2.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

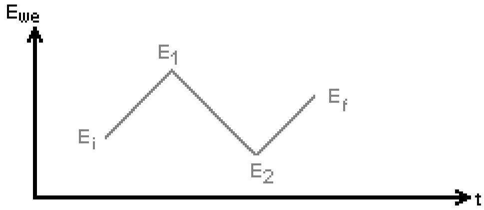
- <u>time</u>:
  The time is calculated with this formula:
  $$t\ (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh \ll 32) + tlow)$$

- <u>Float conversion</u>:
  Ewe and Ece must be converted with the function BL_CONVERTNUMERICINTOSINGLE

7.3. Cyclic Voltammetry technique

Technique ID: **103**

Instrument Series	VMP3	SP-300
File	**cv.ecc**	**cv4.ecc**
Timebase	**40µs**	**45µs**



7.3.1. Description

Cyclic voltammetry (CV) is the most widely used technique for acquiring qualitative informations about electrochemical reactions. CV provides informations on redox processes, heterogeneous electron-transfer reactions and adsorption processes. It offers a rapid location of redox potential of the electroactive species.

CV consists of scanning linearly the potential of a stationary working electrode using a triangular potential waveform. During the potential sweep, the potentiostat measures the current resulting from electrochemical reactions (consecutive to the applied potential). The cyclic voltammogram is a current response as a function of the applied potential.

7.3.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

CV parameters			
Label	**Description**	**Data type**	**Data range**
vs_initial	Current step vs initial one	Array of 5 boolean	True/False
Voltage_step	Voltage step (V)	Array of 5 single	[Ei, E1, E2, Ei, Ef] see CV picture.
Scan_Rate	slew rate array (mV/s)	Array of 5 single	≥ 0
Scan_number	Scan number	integer	= 2
Record_every_dE	recording on dE (V)	single	≥ 0
Average_over_dE	average every dE	boolean	True/False
N_Cycles	Number of cycle	integer	≥ 0
Begin_measuring_I	Begin step accumulation. “1” means 100% of step	single	[0..1]
End_measuring_I	End step accumulation. “1” means 100% of step	single	[0..1]

7.3.3. Data format

Data format returned by the function BL_GETDATA on VMP3:

	0	1	2	3	4	5	6	7	8	9	10	11	...
pBuf	t_high	t_low	Ec	<I>	<Ewe>	Cycle	t_high	t_low	Ec	<I>	<Ewe>	Cycle	...

Point #0 *Point #1*

SP-300 Series:

	0	1	3	4	5	6	7	9	10	11	...
pBuf	t_high	t_low	<I>	<Ewe>	Cycle	t_high	t_low	<I>	<Ewe>	Cycle	...

Point #0 *Point #1*

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.3.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

- time:
  The time is calculated with this formula:
  $$t\ (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)$$

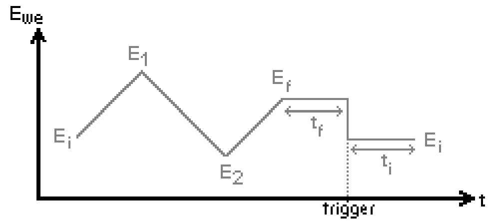
- Float conversion:
  Ewe and Ece must be converted with the function BL_CONVERTNUMERICINTOSINGLE

- cycle:
  no conversion needed

7.4. Cyclic Voltammetry Advanced technique

Technique ID: **126**

Instrument Series	VMP3	SP-300
File	**biovscan.ecc**	**biovscan4.ecc**
Timebase	**40µs**	**40µs**



7.4.1. Description

Cyclic voltammetry (CV) is the most widely used technique for acquiring qualitative information about electrochemical reactions. CV provides information on redox processes, heterogeneous electron-transfer reactions and adsorption processes. It offers a rapid location of redox potential of the electroactive species.

CV consists of scanning linearly the potential of a stationary working electrode using a triangular potential waveform. During the potential sweep, the potentiostat measures the current resulting from electrochemical reactions (consecutive to the applied potential). The cyclic voltammogram is a current response as a function of the applied potential.

7.4.1. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

CVA parameters			
Label	**Description**	**Data type**	**Data range**
vs_initial_scan	Current scan vs initial one	Array of 4 boolean	True/False
Voltage_scan	Voltage scan (V)	Array of 4 single	[Ei, E1, E2, Ef] see CVA picture.
Scan_Rate	slew rate array (mV/s)	Array of 4 single	≥ 0
Scan_number	Scan number	integer	=2
Record_every_dE	recording on dE	single	≥ 0
Average_over_dE	average every dE	boolean	True/False
N_Cycles	Number of cycle	integer	≥ 0
Begin_measuring_I	Begin step accumulation. "1" means 100% of step	single	[0..1]
End_measuring_I	End step accumulation. "1" means 100% of step	single	[0..1]

CVA parameters			
Label	**Description**	**Data type**	**Data range**
vs_initial_step	Current step vs initial one	Array of 2 boolean	True/False
Voltage_step	Voltage step (V)	Array of 2 single	≥ 0
Duration_step	Duration step (s)	Array of 2 single	$[0..tb*2^{31}]$
Step_number	Step number	integer	= 1
Record_every_dT	Recording on dT	single	≥ 0
Record_every_dI	Recording on dI	single	≥ 0
Trig_on_off	trigger	boolean	True/False

7.4.2. Data format

Data format returned by the function BL_GETDATA:

	0	1	2	3	4	5	6	7	8	9	10	11	…
pBuf	t_high	t_low	Ec	<I>	<Ewe>	Cycle	t_high	t_low	Ec	<I>	<Ewe>	Cycle	…

Point #0 (indices 0–5) *Point #1* (indices 6–11)

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.4.3. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

- <u>time</u>:
  The time is calculated with this formula:

  t (s) = TDataInfos.StartTime +
  TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)

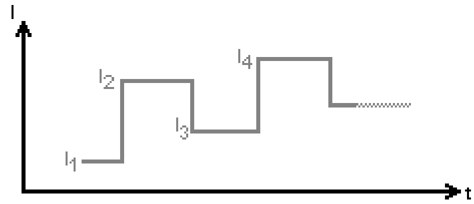
- <u>Float conversion</u>:
  Ewe and Ece must be converted with the function BL_CONVERTNUMERICINTOSINGLE

- <u>cycle</u>:
  no conversion needed

7.5. Chrono-Potentiometry technique

Technique ID: **102**

Instrument Series	VMP3	SP-300
File	**cp.ecc**	**cp4.ecc**
Timebase	**21µs**	**21µs**



7.5.1. Description

The Chronopotentiometry (CP) is a controlled current technique. The current is controlled and the potential is the variable determined as a function of time. The chronopotentiometry technique is similar to the Chronoamperometry technique, potential steps being replaced by current steps. The current is applied between the working and the counter electrode.

This technique can be used for different kind of analysis or to investigate electrode kinetics. It is considered less sensitive than voltammetric techniques for analytical uses. Generally, the curves Ewe = f(t) contains plateaus that correspond to the redox potential of electroactive species.

7.5.1. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

CP parameters			
Label	**Description**	**Data types**	**Data range**
Current_step	Current step (A)	Array of 100 single	-
vs_initial	Current step vs initial one	Array of 100 boolean	True/False
Duration_step	Duration step (s)	Array of 100 single	[0..tb*2^{31}]
Step_number	Number of steps minus 1	integer	[0..98]
Record_every_dT	Record every dt (s)	single	≥ 0
Record_every_dE	Record every dE (V)	single	≥ 0
N_Cycles	Number of times the technique is repeated	integer	≥ 0
I_Range	I range	integer	see IRange constants allowed on section 5. Used structures and constants

CP parameters			
Label	**Description**	**Data types**	**Data range**
			Warning: I Auto-range not authorized

7.5.2. Data format

Data format returned by the function BL_GETDATA :

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**...**
pBuf	t_high	t_low	Ewe	I	cycle	t_high	t_low	Ewe	I	cycle	...
	point #0					*point #1*					

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.5.3. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

- time:
  The time is calculated with this formula:
  $$t\ (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)$$

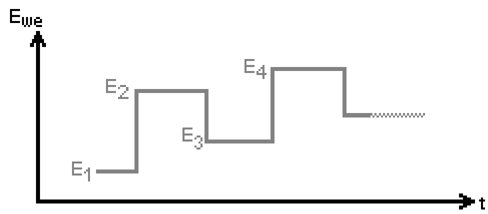
- Float conversion:
  Ewe and I must be converted with the function BL_CONVERTNUMERICINTOSINGLE

- cycle:
  no conversion needed

7.6. Chrono-Amperometry technique

Technique ID: **101**

Instrument Series	VMP3	SP-300
File	**ca.ecc**	**ca4.ecc**
Timebase	**24µs**	**21µs**



7.6.1. Description

The basis of the controlled-potential techniques is the measurement of the current response to an applied potential step.

The Chronoamperometry (CA) technique involves stepping the potential of the working electrode from an initial potential, at which (generally) no faradic reaction occurs, to a potential Ei at which the faradic reaction occurs. The current-time response reflects the change in the concentration gradient in the vicinity of the surface.
Chronoamperometry is often used for measuring the diffusion coefficient of electroactive species or the surface area of the working electrode. This technique can also be applied to the study of electrode processes mechanisms.

An alternative and very useful mode for recording the electrochemical response is to integrate the current, so that one obtains the charge passed as a function of time. This is the chronocoulometric mode that is particularly used for measuring the quantity of adsorbed reactants.

7.6.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

CA parameters			
Label	**Description**	**Data types**	**Data range**
Voltage_step	Voltage step (V)	Array of 100 single	-
vs_initial	Voltage step vs initial one	Array of 100 boolean	True/False
Duration_step	Duration step (s)	Array of 100 single	≥ 0
Step_number	Number of steps minus 1	integer	[0..98]
Record_every_dT	Record every dt (s)	single	≥ 0
Record_every_dI	Record every dI (A)	single	≥ 0
N_Cycles	Number of times the technique is repeated	integer	≥ 0

7.6.3. Data format

See CP technique data format.

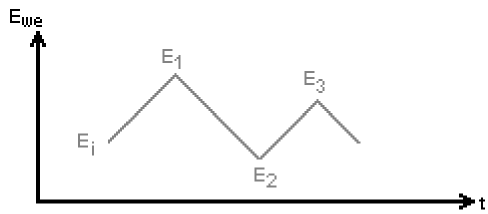
7.6.4. Data conversion

See CP technique data conversion.

7.7. Voltage Scan technique

Technique ID: **124**

Instrument Series	VMP3	SP-300
File	**vscan.ecc**	**vscan4.ecc**
Timebase	**40µs**	**50µs**



7.7.1. Description

The Potentiodynamic (PDYN) technique allows the user to perform potentiodynamic periods with different scan rates.

7.7.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

VSCAN parameters			
Parameters	**Description**	**Data types**	**Data range**
Voltage_step	Vertex potential (V)	Array of 100 single	-
vs_initial	Vertex potential vs initial one	Array of 100 boolean	True/False
Scan_Rate	Scan rate (V/s) from previous vertex potential	Array of 100 single	> 0, Value of the first scan-rate is ignored
Scan_number	Number of scans minus 1	integer	[0..98]
Record_every_dE	Record every dE (V)	single	≥ 0
N_Cycles	Number of times the technique is repeated	integer	≥ 0
Begin_measuring_I	Begin step accumulation. "1" means 100% of step	single	[0..1]
End_measuring_I	End step accumulation. "1" means 100% of step	single	[0..1]

7.7.3. Data format

Data format returned by the function BL_GETDATA for VMP300:

	0	1	2	3	4	5	6	7	8	9	10	11	...
pBuf	t_high	t_low	Ec	<I>	<Ewe>	cycle	t_high	t_low	Ec	<I>	<Ewe>	cycle	...
	point #0						*point #1*						

For SP-300:

	0	1	2	3	4	5	6	7	8	9	...
pBuf	t_high	t_low	<I>	<Ewe>	cycle	t_high	t_low	<I>	<Ewe>	cycle	...
	point #0					*point #1*					

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.7.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before :

- <u>time</u>:
  The time is calculated with this formula :
  $$t\ (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)$$

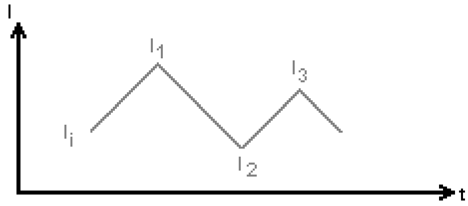
- <u>Float conversion</u>:
  Ec, <I> and Ewe must be converted with the function BL_CONVERTNUMERICINTOSINGLE

- <u>cycle</u>:
  no conversion needed

7.8. Current Scan technique

Technique ID: **125**

Instrument Series	VMP3	SP-300
File	**iscan.ecc**	**iscan 4.ecc**
Timebase	**40µs**	**50µs**



7.8.1. Description

The Galvanodynamic (GDYN) technique allows the user to perform galvanodynamic periods with different scan rates.

7.8.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

ISCAN parameters			
Label	**Description**	**Data types**	**Data range**
Current_step	Vertex current (A)	Array of 100 single	-
vs_initial	Vertex current vs initial one	Array of 100 boolean	True/False
Scan_Rate	Scan rate (A/s) from previous vertex current	Array of 100 single	> 0
Scan_number	Number of scans minus 1	integer	[0..98]
Record_every_dI	Record every dI (A)	single	≥ 0
N_Cycles	Number of times the technique is repeated	integer	≥ 0
Begin_measuring_E	Select the part of the current step (1 = 100%) used for data averaging	single	[0..1]
End_measuring_E		single	[0..1]
I_Range	I range	integer	see IRange constants authorized on section 5. Used structures and constants **Warning** : I Auto-range not authorized

7.8.3. Data format

Data format returned by the function BL_GETDATA for VMP3:

	0	1	2	3	4	5	6	7	8	9	10	11	...
pBuf	t_high	t_low	Ic	<I>	Ewe	cycle	t_high	t_low	Ic	<I>	Ewe	cycle	...
	point #0						*point #1*						

For SP-300:

	0	1	2	3	4	5	6	7	8	9	...
pBuf	t_high	t_low	<I>	Ewe	cycle	t_high	t_low	<I>	Ewe	cycle	...
	point #0					*point #1*					

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.8.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

- <u>time</u>:
  The time is calculated with this formula:
  t (s) = TDataInfos.StartTime +
  TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)

- <u>Float conversion</u>:
  Ic, <I> and Ewe must be converted with the function BL_CONVERTNUMERICINTOSINGLE

- <u>cycle</u>:
  no conversion needed

7.9. Constant Power technique

Technique ID: **110**

Instrument Series	VMP3	SP-300
File	**pow.ecc**	**pow4.ecc**
Timebase	**100µs**	**100µs**

7.9.1. Description

The Constant Power technique is designed to study the discharge (eventually the charge) of a cell at constant power.

7.9.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

Constant Power parameters			
Label	**Description**	**Data types**	**Data range**
Power_step	Power step (W)	Array of 100 single	-
vs_initial	Power step vs initial one	Array of 100 boolean	True/False
Duration_step	Duration step (s)	Array of 100 single	[0..tb*2^{31}]
Step_number	Number of steps minus 1	integer	[0..98]
Record_every_dT	Record every dt (s)	single	≥ 0
Record_every_dE	Record every dE (V)	single	≥ 0
N_Cycles	Number of times the technique is repeated	integer	≥ 0
I_Range	I range	integer	see IRange constants authorized on section 5. Used structures and constants **Warning** : I Auto-range not authorized

7.9.3. Data format

Data format returned by the function BL_GETDATA:

	0	1	2	3	4	5	6	7	8	9	10	11	…
pBuf	t_high	t_low	Ewe	I	P	cycle	t_high	t_low	Ewe	I	P	cycle	…
	point #0						*point #1*						

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.9.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

- <u>time</u>:
  The time is calculated with this formula:
  $$t\ (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)$$

- <u>Float conversion</u>:
  Ewe, I and P must be converted with the function BL_CONVERTNUMERICINTOSINGLE

- <u>cycle</u>:
  no conversion needed

7.10. Constant Load technique

Technique ID: **111**

Instrument Series	VMP3	SP-300
File	**load.ecc**	**load4.ecc**
Timebase	**100µs**	**100µs**

7.10.1. Description

The Constant Load technique is designed to discharge a battery at a constant resistance.

7.10.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

Constant Load parameters

Label	Description	Data types	Data range
Load_step	Resistor step value (Ohm)	Array of 100 single	-
vs_initial	Resistor step value vs initial one	Array of 100 boolean	True/False
Duration_step	Duration step (s)	Array of 100 single	[0..tb*2^{31}]
Step_number	Number of steps minus 1	integer	[0..98]
Record_every_dT	Record every dt (s)	single	≥ 0
Record_every_dE	Record every dE (V)	single	≥ 0
N_Cycles	Number of times the technique is repeated	integer	≥ 0
I_Range	I range	integer	see IRange constants authorized on section 5. Used structures and constants **Warning** : I Auto-range not authorized

7.10.3. Data format

Data format returned by the function BL_GETDATA :

	0	1	2	3	4	5	6	7	8	9	10	11	...
pBuf	t_high	t_low	Ewe	I	R	cycle	t_high	t_low	Ewe	I	R	cycle	...

point #0 (columns 0–5) *point #1* (columns 6–11)

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.10.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

- time :
  The time is calculated with this formula:
  $$t\ (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)$$

- Float conversion:
  Ewe, I and R must be converted with the function BL_CONVERTNUMERICINTOSINGLE

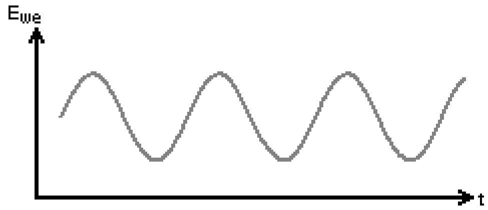
- cycle:
  no conversion needed

7.11. Potentio Electrochemical Impedance Spectroscopy technique

Technique ID: **104**

Instrument Series	VMP3	SP-300
File	**peis.ecc**	**peis4.ecc**
Timebase	**24µs***	**24µs***

* Timebase is for first process only.



7.11.1. Description

The Potentio Electrochemical Impedance Spectroscopy (PEIS) technique performs impedance measurements into potentiostatic mode in applying a sinus around a DC potential E that can be set to a fixed value or relatively to the cell equilibrium potential.

For very capacitive or low impedance electrochemical systems, the potential amplitude can lead to a current overflow that can stop the experiment in order to protect the unit from overheating. Using GEIS instead of PEIS can avoid this inconvenient situation.

Moreover, during corrosion experiment, a potential shift of the electrochemical system can occur. PEIS technique can lead to impedance measurements far from the corrosion potential while GEIS can be performed at a zero current.

7.11.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

PEIS parameters			
Label	**Description**	**Data types**	**Data range**
vs_initial	Voltage step vs initial one	boolean	True/False
vs_final	Voltage step vs initial one	boolean	= vs_initial
Initial_Voltage_step	Initial voltage step (V)	single	-
Final_Voltage_step	Final voltage step (V)	single	= Initial_Voltage_step
Duration_step	Step duration (s)	single	[0..tb*2^{31}]
Step_number	Number of voltage steps	integer	= 1
Record_every_dT	Record every dt (s)	single	≥ 0
Record_every_dI	Record every dI (A)	single	≥ 0
Final_frequency	Final frequency (Hz)	single	Depend on instrument

PEIS parameters			
Label	**Description**	**Data types**	**Data range**
Initial_frequency	Initial frequency (Hz)	single	Depend on instrument
sweep	sweep linear/logarithmic (TRUE for linear points spacing)	boolean	True/False
Amplitude_Voltage	Sinus amplitude (V)	single	Depend on instrument
Frequency_number	Number of frequencies	integer	≥ 1
Average_N_times	Number of repeat times (used for frequencies averaging)	integer	≥ 1
Correction	Non-stationary correction	boolean	True/False
Wait_for_steady	Number of period to wait before each frequency	single	≥ 0

7.11.3. Data format

Data format depends of the technique process used to record data. The process index is returned in the field TDATAINFOS.PROCESSINDEX.

- Data format of process 0:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
t_high	t_low	Ewe	I	t_high	t_low	Ewe	I	...

point #0 *point #1*

- Data format of process 1:

VMP3 Series:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**...**
freq	|Ewe|	|I|	Phase Zwe	Ewe	I	-	|Ece|	|Ice|	Phase Zce	...

point #0

...	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**...**
...	Ece	-	-	t	IRange	freq	|Ewe|	|I|	Phase Zwe	...

point #0 *point #1*

...	19	20	21	22	23	24	25	26	27	28	29	...
...	Ewe	I	-	|Ece|	|Ice|	Phase Zce	Ece	-	-	t	IRange	...

point #1

SP-300 Series:

0	1	2	3	4	5	6	7	8	9	...
freq	|Ewe|	|I|	Phase Zwe	Ewe	I	-	|Ece|	|Ice|	Phase Zce	...

point #0

...	10	11	12	13	14	15	16	17	...
...	Ece	-	-	t	freq	|Ewe|	|I|	Phase Zwe	...

point #0 *point #1*

...	18	19	20	21	22	23	24	25	26	27	...
...	Ewe	I	-	|Ece|	|Ice|	Phase Zce	Ece	-	-	t	...

point #1

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.11.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

- time:
  The time for the process 0 is calculated with this formula:
  t (s) = TDataInfos.StartTime +
  *TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)*

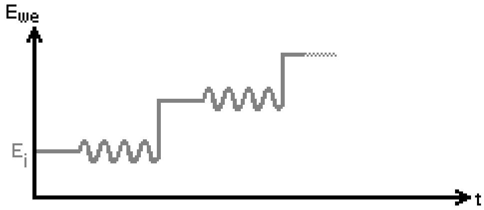
- Float conversion:
  time (only time of process 1), IRange, freq, Ewe, |Ewe|, Ece, |Ece|, I, |I|, Phase Zwe, Phase Zce must be converted with the function BL_CONVERTNUMERICINTOSINGLE

- cycle:
  no conversion needed

7.12. Staircase Potentio Electrochemical Impedance Spectroscopy technique

Technique ID: **113**

Instrument Series	VMP3	SP-300
File	**speis.ecc**	**speis4.ecc**
Timebase	**24µs**	**24µs**



7.12.1. Description

The Staircase Potentio Electrochemical Impedance Spectroscopy (SPEIS) technique is designed to perform successive impedance measurements (on a whole frequency range) during a potential sweep. The main application of this technique is to study electrochemical reaction kinetics along voltamperometric (I(E)) curves in analytical electrochemistry. Thus this technique finds all its interest to study the complexity of non-stationary interfaces with faradic processes where the total AC response (whole frequency range) is required.

Another common application of such a technique is semi-conductor materials study. For these stationary systems only two or three frequencies for each potential step are required to determine the donor density and the flat band potential.

The SPEIS technique consists in a staircase potential sweep (potential limits and number of steps defined by the user). An impedance measurement (with an adjustable number of frequencies) is performed on each potential step. For all these applications a Mott-Schottky plot (1/C2 vs. Ewe) can be displayed and a special linear fit is applied to extract the semi-conductor parameters.

7.12.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

SPEIS parameters			
Label	**Description**	**Data types**	**Data range**
vs_initial	Voltage step vs initial one	boolean	True/False
vs_final	Voltage step vs initial one	boolean	True/False
Initial_Voltage_step	Initial voltage step (V)	single	-
Final_Voltage_step	Final voltage step (V)	single	-
Duration_step	Step duration (s)	single	[0..tb*2^{31}]
Step_number	Number of voltage steps	integer	[0..98]

SPEIS parameters			
Label	**Description**	**Data types**	**Data range**
Record_every_dT	Record every dt (s)	single	≥ 0
Record_every_dI	Record every dI (A)	single	≥ 0
Final_frequency	Final frequency (Hz)	single	Depend on instrument
Initial_frequency	Initial frequency (Hz)	single	Depend on instrument
sweep	sweep linear/logarithmic (TRUE for linear points spacing)	boolean	True/False
Amplitude_Voltage	Sinus amplitude (V)	single	Depend on instrument
Frequency_number	Number of frequencies	integer	≥ 1
Average_N_times	Number of repeat times (used for frequencies averaging)	integer	≥ 1
Correction	Non-stationary correction	boolean	True/False
Wait_for_steady	Number of period to wait before each frequency	single	≥ 0

7.12.3. Data format

Data format depends of the technique process used to record data. The process index is returned in the field TDATAINFOS.PROCESSINDEX.

- Data format of process 0:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**...**
t_high	t_low	Ewe	I	step	t_high	t_low	Ewe	I	step	...
point #0					*point #1*					

- Data format of process 1:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**...**
freq	|Ewe|	|I|	Phase Zwe	Ewe	I	-	|Ece|	|Ice|	Phase Zce	...
point #0										

...	10	11	12	13	14	15	16	17	18	19	...
...	Ece	-	-	t	Irange	step	freq	|Ewe|	|I|	Phase Zwe	...
	point #0						point #1				

...	20	21	22	23	24	25	26	27	28	29	...
...	Ewe	I	-	|Ece|	|Ice|	Phase Zce	Ece	-	-	t	...
	point #1										

...	30	31	...
...	Irange	step	...
	point #1		

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.12.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

- time:
  The time for the process 0 is calculated with this formula:
  *t (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)*

- Float conversion:
  time (only time of process 1), IRange, freq, Ewe, |Ewe|, Ece, |Ece|, I, |I|, Phase Zwe, Phase Zce must be converted with the function BL_CONVERTNUMERICINTOSINGLE

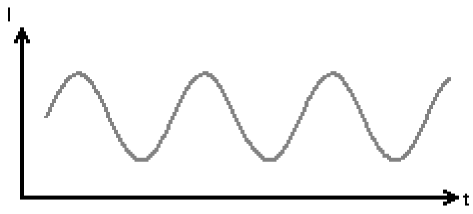
- cycle:
  no conversion needed

7.13. Galvano Electrochemical Impedance Spectroscopy technique

Technique ID: **107**

Instrument Series	VMP3	SP-300
File	**geis.ecc**	**geis4.ecc**
Timebase	**24µs***	**24µs***

* Timebase is for first process only.



7.13.1. Description

The Galvano Electrochemical Impedance Spectroscopy (GEIS) technique performs impedance measurements into galvanostatic mode in applying a sinus around a DC current I that can be set to a fixed value or relatively to a previous controlled current. In the case of particular non-linear systems it can be necessary to use PEIS instead of GEIS.

The Electrochemical Impedance spectroscopy (EIS) finds many of applications in corrosion, battery, fuel cell development, sensors and physical electrochemistry. It can provide information on reaction parameters, corrosion rates, electrode surfaces porosity, coating, mass transport, interfacial capacitance measurements.

7.13.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

GEIS parameters			
Label	**Description**	**Data types**	**Data range**
vs_initial	Current step vs initial one	boolean	True/False
vs_final	Current step vs initial one	boolean	= vs_initial
Initial_Current_step	Initial current step (A)	single	-
Final_Current_step	Final current step (A)	single	= Initial_Current_step
Duration_step	Step duration (s)	single	[0..tb*2^{31}]
Step_number	Number of voltage steps	integer	= 1
Record_every_dT	Record every dt (s)	single	≥ 0
Record_every_dE	Record every dE (V)	single	≥ 0
Final_frequency	Final frequency (Hz)	single	Depend on instrument
Initial_frequency	Initial frequency (Hz)	single	Depend on instrument
sweep	sweep linear/logarithmic (TRUE for linear points	boolean	True/False

GEIS parameters			
Label	**Description**	**Data types**	**Data range**
	spacing)		
Amplitude_Current	Sinus amplitude (A)	single	[0..50% of the Irange]
Frequency_number	Number of frequencies	integer	≥ 1
Average_N_times	Number of repeat times (used for frequencies averaging)	integer	≥ 1
Correction	Non-stationary correction	boolean	True/False
Wait_for_steady	Number of period to wait before each frequency	single	≥ 0
I_Range	I range	integer	see IRange constants authorized on section 5. Used structures and constants **Warning** : I Auto-range not authorized

7.13.3. Data format

See PEIS technique data format.

7.13.4. Data conversion

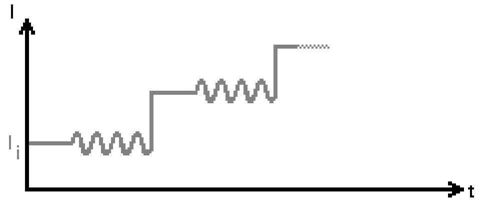
See PEIS technique data conversion.

7.14. Staircase Galvano Electrochemical Impedance Spectroscopy technique

Technique ID: **114**

Instrument Series	VMP3	SP-300
File	**sgeis.ecc**	**sgeis4.ecc**
Timebase	**24µs***	**24µs***

* Timebase for first process only



7.14.1. Description

The Staircase Galvano Electrochemical Impedance Spectroscopy (SGEIS) technique is close to the SPEIS technique. The difference is that the potentiostat works as a galvanostat and applies a current sweep (staircase shape). In the same way an impedance measurement (whole frequency range) can be performed on each current step. The user can also select several frequencies.

7.14.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

SGEIS parameters

Label	Description	Data types	Data range
vs_initial	Current step vs initial one	boolean	True/False
vs_final	Current step vs initial one	boolean	True/False
Initial_Current_step	Initial current step (A)	single	-
Final_Current_step	Final current step (A)	single	-
Duration_step	Step duration (s)	single	[0..tb*2^{31}]
Step_number	Number of voltage steps	integer	≥ 0
Record_every_dT	Record every dt (s)	single	≥ 0
Record_every_dE	Record every dE (V)	single	≥ 0
Final_frequency	Final frequency (Hz)	single	Depend on instrument
Initial_frequency	Initial frequency (Hz)	single	Depend on instrument
sweep	sweep linear/logarithmic (TRUE for linear points spacing)	boolean	True/False
Amplitude_Current	Sinus amplitude (A)	single	[0..50% of range]
Frequency_number	Number of frequencies	integer	≥ 1

SGEIS parameters			
Label	**Description**	**Data types**	**Data range**
Average_N_times	Number of repeat times (used for frequencies averaging)	integer	≥ 1
Correction	Non-stationary correction	boolean	True/False
Wait_for_steady	Number of period to wait before each frequency	single	≥ 0
I_Range	I range	integer	see IRange constants authorized on section 5. Used structures and constants **Warning** : I Auto-range not authorized

7.14.3. Data format

See PEIS technique data format.

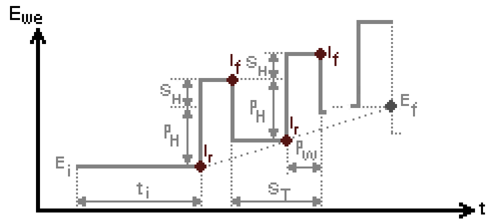
7.14.4. Data conversion

See PEIS technique data conversion.

7.15. Differential Pulse Voltammetry technique

Technique ID: **127**

Instrument Series	VMP3	SP-300
File	**dpv.ecc**	**dpv4.ecc**
Timebase	**40µs**	**40µs**



7.15.1. Description

DPV is very useful technique for analytical determination (for example metal ion quantification in a sample). The differential measurements discriminate faradic current from capacitive one.

In this technique, the applied waveform is the sum of a pulse train and a staircase from the initial potential (Ei) to a final potential (Ef). The current is sampled just before the pulse and near the end of the pulse. The resulting current is the difference between these two currents. It has a relatively flat baseline. The current peak height is directly related to the concentration of the electroactive species in the electrochemical cell.

7.15.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

DPV parameters			
Label	**Description**	**Data types**	**Data range**
Ei	Initial potential (V)	single	-
OCi	Initial potential vs initial one	boolean	True/False
Rest_time_Ti	Ei duration	single	[0..tb*2^{31}]
Ef	Final potential (v)	single	-
OCf	Final potential vs initial one	boolean	True/False
PH	Pulse height (mV)	single	≥ 0
PW	Pulse width (ms)	single	≥ 0
SH	Step height (mV)	single	≥ 0
ST	Step width (ms)	single	≥ 0
Begin_measuring_I	Select the part of the current step. 1 = 100 % used for data averaging	single	[0..1]
End_measuring_I		single	[0..1]

DPV parameters			
Label	**Description**	**Data types**	**Data range**
I_Range	I range	integer	see IRange constants authorized on section 5. Used structures and constants **Warning**: I Auto-range not authorized

7.15.3. Data format

Data format returned by the function BL_GETDATA for VMP3:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**7**	**8**	**...**
t_high	t_low	<Ewe>	<I>	Q	t_high	t_low	<Ewe>	<I>	Q	...

Point #0 *Point #1*

For SP-300:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
t_high	t_low	<Ewe>	<I>	t_high	t_low	<Ewe>	<I>	...

Point #0 *Point #1*

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.15.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

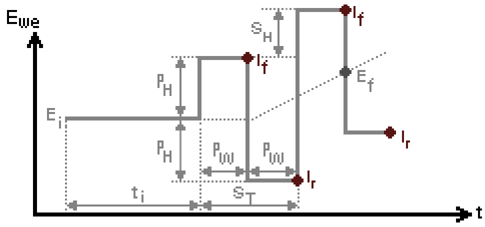
- <u>time</u>:
  The time for the process 0 is calculated with this formula:
  $$t\ (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)$$

- <u>Float conversion</u>:
  time (only time of process 1), IRange, freq, Ewe, |Ewe|, Ece, |Ece|, I, |I|, Phase Zwe, Phase Zce, Q-Qo must be converted with the function BL_CONVERTNUMERICINTOSINGLE

7.16. Square Wave Voltammetry technique

Technique ID: **128**

Instrument Series	VMP3	SP-300
File	**swv.ecc**	**swv4.ecc**
Timebase	**40µs**	**40µs**



7.16.1. Description

Among the electroanalytical techniques, the Square Wave Voltammetry (SWV) combines the background suppression, the sensitivity of DPV and the diagnostic value of NPV.

The SWV is a large amplitude differential technique characterized by a pulse height (PH) and a pulse width (PW).The pulse width can be expressed in term of square wave frequency f=1/(2PW).The scan rate is v = PH/(2PW). The current is sampled twice, once at the end of the forward pulse and once at the end of the reverse pulse. The difference between the two measurements is plotted versus the base staircase potential. The resulting peak-shaped voltammogram is symmetrical about the half-wave potential and the peak current is proportional to the concentration.

Excellent sensitivity accrues from the fact that the net current is larger than either the forward or reverse components (since it is the difference between them).

7.16.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

SWV parameters

Label	Description	Data types	Data range
Ei	Initial potential (V)	single	-
OCi	Initial potential vs initial one	boolean	True/False
Rest_time_Ti	Ei duration	single	[0..tb*2^{31}]
Ef	Final potential (v)	single	-
OCf	Final potential vs initial one	boolean	True/False
PH	Pulse height (mV)	single	≥ 0
PW	Pulse width (ms)	single	≥ 0
SH	Step height (mV)	single	≥ 0
Begin_measuring_I	Select the part of the	single	[0..1]

SWV parameters			
Label	**Description**	**Data types**	**Data range**
End_measuring_I	current step (1 = 100%) used for data averaging	single	[0..1]
I_Range	I range	integer	see IRange constants authorized on section 5. Used structures and constants **Warning** : I Auto-range not authorized

7.16.3. Data format

Data format returned by the function BL_GETDATA for VMP3:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**...**
t_high	t_low	<Ewe>	<I>	Q	t_high	t_low	<Ewe>	<I>	Q	...

Point #0 (columns 0–4) *Point #1* (columns 5–9)

For SP-300:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
t_high	t_low	<Ewe>	<I>	t_high	t_low	<Ewe>	<I>	...

Point #0 (columns 0–3) *Point #1* (columns 4–7)

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.16.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

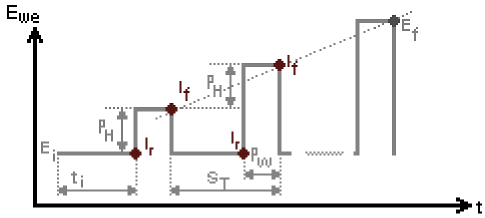
- <u>time</u>:
  The time for the process 0 is calculated with this formula:
  *t (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)*

- <u>Float conversion</u>:
  time (only time of process 1), IRange, freq, Ewe, |Ewe|, Ece, |Ece|, I, |I|, Phase Zwe, Phase Zce, Q-Qo must be converted with the function BL_CONVERTNUMERICINTOSINGLE

7.17. Normal Pulse Voltammetry technique

Technique ID: **129**

Instrument Series	VMP3	SP-300
File	**npv.ecc**	**npv4.ecc**
Timebase	**40µs**	**40µs**



7.17.1. Description

Pulsed techniques have been introduced to increase the ratio between the faradic and nonfaradic currents in order to permit a quantification of species to very low concentration levels.

The Normal Pulse Voltammetry is one of the first pulsed techniques elaborated for polarography needs. An essential idea behind the NPV is the cyclic renewal of the diffusion layer. With a DME, this is achieved by the stirring accompanying the fall of the mercury drop. But at other electrodes renewal may not be so easily accomplished.

NPV consists of a series of pulses of linear increasing amplitude (from Ei to Ef). The potential pulse is ended by a return to the base value Ei. The usual practice is to select Ei in a region where the electroactive species of interest does not react at the electrode. The current is sampled at a time t near the end of the pulse and at a time t' before the pulse. The plotted current is the difference of both currents measured at the end of the pulse (forward) and at the end of the period previous to the pulse (reverse).

7.17.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

NPV parameters

Label	Description	Data types	Data range
Ei	Initial potential (V)	single	-
OCi	Initial potential vs initial one	boolean	True/False
Rest_time_Ti	Ei duration	single	[0..tb*2^31]
Ef	Final potential (v)	single	-
OCf	Final potential vs initial one	boolean	True/False
PH	Pulse height (mV)	single	≥ 0
PW	Pulse width (ms)	single	≥ 0
ST	Step width (ms)	single	≥ 0
Begin_measuring_I	Select the part of the	single	[0..1]

NPV parameters			
Label	**Description**	**Data types**	**Data range**
End_measuring_I	current step (1 = 100%) used for data averaging	single	[0..1]
I_Range	I range	integer	see IRange constants authorized on section 5. Used structures and constants **Warning**: I Auto-range not authorized

7.17.3. Data format

Data format returned by the function BL_GETDATA for VMP3:

0	1	2	3	4	5	6	7	8	9	...
t_high	t_low	<Ewe>	<I>	Q	t_high	t_low	<Ewe>	<I>	Q	...

Point #0 *Point #1*

For SP-300:

0	1	2	3	4	5	6	7	...
t_high	t_low	<Ewe>	<I>	t_high	t_low	<Ewe>	<I>	...

Point #0 *Point #1*

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.17.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

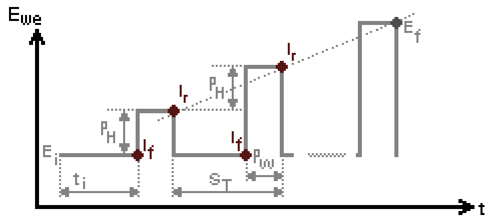
- <u>time</u>:
  The time for the process 0 is calculated with this formula:
  $$t\ (s) = \mathit{TDataInfos.StartTime} + \mathit{TDataInfos.CurrentValues.TimeBase} * ((\mathit{thigh} \ll 32) + \mathit{tlow})$$

- <u>Float conversion</u>:
  time (only time of process 1), IRange, freq, Ewe, |Ewe|, Ece, |Ece|, I, |I|, Phase Zwe, Phase Zce, Q-Qo must be converted with the function BL_CONVERTNUMERICINTOSINGLE

7.18. Reverse Normal Pulse Voltammetry technique

Technique ID: **130**

Instrument Series	VMP3	SP-300
File	**rnpv.ecc**	**rnpv4.ecc**
Timebase	**40µs**	**40µs**



7.18.1. Description

The Reverse Normal Pulse Voltammetry is a derivative technique from the NPV. The main difference is that the initial (base) potential Ei is placed in the diffusion-limited region for electrolysis of the species present in the bulk solution. The pulses are made through the region where the species in solution is not electroactive.

The RNPV experiment involves a significant faradic current. This method is a reversal experiment because of the detection of the product from a prior electrolysis.

7.18.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

RNPV parameters			
Label	**Description**	**Data types**	**Data range**
Ei	Initial potential (V)	single	-
OCi	Initial potential vs initial one	boolean	True/False
Rest_time_Ti	Ei duration	single	[0..tb*2^{31}]
Ef	Final potential (v)	single	-
OCf	Final potential vs initial one	boolean	True/False
PH	Pulse height (mV)	single	≥ 0
PW	Pulse width (ms)	single	≥ 0
ST	Step width (ms)	single	≥ 0
Begin_measuring_I	Select the part of the current step (1 = 100%) used for data averaging	single	[0..1]
End_measuring_I		single	[0..1]
I_Range	I range	integer	see IRange constants authorized on section 5. Used structures and constants **Warning**: I Auto-range not authorized

7.18.3. Data format

Data format returned by the function BL_GETDATA for VMP3:

0	1	2	3	4	5	6	7	8	9	...
t_high	t_low	<Ewe>	<I>	Q	t_high	t_low	<Ewe>	<I>	Q	...
Point #0					*Point #1*					

For SP-300:

0	1	2	3	4	5	6	7	...
t_high	t_low	<Ewe>	<I>	t_high	t_low	<Ewe>	<I>	...
Point #0				*Point #1*				

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.18.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

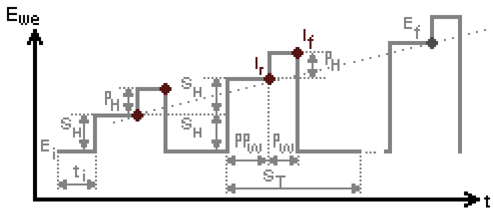
- <u>time</u>:
  The time for the process 0 is calculated with this formula:
  $$t\ (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)$$

- <u>Float conversion</u>:
  time (only time of process 1), IRange, freq, Ewe, |Ewe|, Ece, |Ece|, I, |I|, Phase Zwe, Phase Zce, Q-Qo must be converted with the function BL_CONVERTNUMERICINTOSINGLE

7.19. Differential Normal Pulse Voltammetry technique

Technique ID: **131**

Instrument Series	VMP3	SP-300
File	**dnpv.ecc**	**dnpv4.ecc**
Timebase	**40µs**	**40µs**



7.19.1. Description

Originally introduced as a polarographic technique (performed at a DME), the Differential Normal Pulse Voltammetry is a sensitive electroanalytical technique very close to the DPV technique with a pulsed potential sweep. The potential pulse is swept from an initial potential Ei to a final potential Ef. There are two main differences with the DPV technique: first the pulse waveform is made with a prepulse (SH amplitude with PPW duration) before the pulse (PH amplitude with PW duration) and second the potential always comes back to the initial potential (Ei) after the pulsed sequence. Ei is assumed to be the potential where no faradic reaction occurs. The plotted current is the difference of both currents measured at the end of the pulse (I forward) and at the end of the prepulse (I reverse).

This technique is often used in polarography and by biologists to define the most appropriate potential for the electrochemical detection to a fixed potential with the DPA technique.

7.19.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

DNPV parameters

Label	**Description**	**Data types**	**Data range**
Ei	Initial potential (V)	single	-
OCi	Initial potential vs initial one	boolean	True/False
Rest_time_Ti	Ei duration	single	[0..tb*2^31]
Ef	Final potential (v)	single	-
OCf	Final potential vs initial one	boolean	True/False
PH	Pulse height (mV)	single	≥ 0
PPW	PrePulse Width	single	≥ 0
PW	Pulse width (ms)	single	≥ 0
SH	Step height (mV)	single	≥ 0
ST	Step width (ms)	single	≥ 0

DNPV parameters			
Label	**Description**	**Data types**	**Data range**
Begin_measuring_I	Select the part of the current step (1 = 100%) used for data averaging	single	[0..1]
End_measuring_I		single	[0..1]
I_Range	I range	integer	see IRange constants authorized on section 5. Used structures and constants **Warning**: I Auto-range not authorized

7.19.3. Data format

Data format returned by the function BL_GETDATA for VMP3:

0	1	2	3	4	5	6	7	8	9	…
t_high	t_low	<Ewe>	<I>	Q	t_high	t_low	<Ewe>	<I>	Q	…

Point #0 *Point #1*

For SP-300:

0	1	2	3	4	5	6	7	…
t_high	t_low	<Ewe>	<I>	t_high	t_low	<Ewe>	<I>	…

Point #0 *Point #1*

The charge can be recorded additionnally using XCTR.

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.19.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

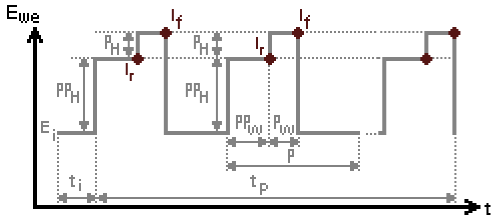
- time:
  The time for the process 0 is calculated with this formula:
  *t (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)*

- Float conversion:
  time (only time of process 1), IRange, freq, Ewe, |Ewe|, Ece, |Ece|, I, |I|, Phase Zwe, Phase Zce, Q-Qo must be converted with the function BL_CONVERTNUMERICINTOSINGLE

7.20. Differential Pulse Amperometry technique

Technique ID: **132**

Instrument Series	VMP3	SP-300
File	**dpa.ecc**	**dpa4.ecc**
Timebase	**40µs**	**40µs**



7.20.1. Description

The Differential Pulse Amperometry results from the DNPV technique without increasing pulse steps. The potential waveform and the current sampling are the same as for DNPV. A DPA experiment is often used as a sensitive method for the quantification of electrochemical species at a defined potential (Es). This potential value is often determined with a DNPV experiment (using a potential sweep with the same waveform) previously performed. This technique is dedicated to the quantification of biological electroactive species.

7.20.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

DPA parameters

Label	**Description**	**Data types**	**Data range**
Ei	Initial potential (V)	single	-
OCi	Initial potential vs initial one	boolean	True/False
Rest_time_Ti	Ei duration	single	$[0..tb*2^{31}]$
PPH	PrePulse height	single	≥ 0
PPW	PrePulse Width	single	≥ 0
PH	Pulse height (mV)	single	≥ 0
PW	Pulse width (ms)	single	≥ 0
P	Period (mV)	single	≥ 0
Tp	Duration (s)	single	≥ 0
Begin_measuring_I	Select the part of the current step (1 = 100%) used for data averaging	single	[0..1]
End_measuring_I		single	[0..1]
I_Range	I range	integer	see IRange constants authorized on section 5. Used structures and constants

DPA parameters			
Label	**Description**	**Data types**	**Data range**
			Warning: I Auto-range not authorized

7.20.3. Data format

Data format returned by the function BL_GETDATA for VMP3:

0	1	2	3	4	5	6	7	8	9	...
t_high	t_low	<Ewe>	<I>	Q	t_high	t_low	<Ewe>	<I>	Q	...

Point #0 (columns 0–4) *Point #1* (columns 5–9)

For SP-300:

0	1	2	3	4	5	6	7	...
t_high	t_low	<Ewe>	<I>	t_high	t_low	<Ewe>	<I>	...

Point #0 (columns 0–3) *Point #1* (columns 4–7)

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.20.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

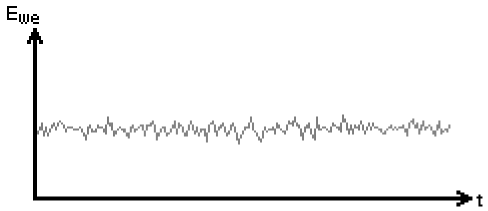
- <u>time</u>:
  The time for the process 0 is calculated with this formula:
  $$t\,(s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh \ll 32) + tlow)$$

- <u>Float conversion</u>:
  time (only time of process 1), IRange, freq, Ewe, |Ewe|, Ece, |Ece|, I, |I|, Phase Zwe, Phase Zce, Q-Qo must be converted with the function BL_CONVERTNUMERICINTOSINGLE

7.21. Ecorr. Vs Time technique

Technique ID: **133**

Instrument Series	VMP3	SP-300
File	**evt.ecc**	**evt4.ecc**
Timebase	**20µs**	**20µs**



7.21.1. Description

This technique corresponds to the follow up of the corrosion potential (when the circuit is open) versus time. During the measurement no potential or current is applied to the cell.

7.21.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

EVT parameters			
Label	**Description**	**Data type**	**Data range**
Record_every_dEr	Record every dE (V)	single	≥ 0
Rest_time_T	Rest duration (s)	single	[0..tb*2^{31}]
Record_every_dTr	Record every dT (s)	single	≥ 0

7.21.3. Data format

Data format returned by the function BL_GETDATA:
VMP3 series:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**...**
t_high	t_low	Ewe	Ece	t_high	t_low	Ewe	Ece	...

Point #0 *Point #1*

SP-300 series:

0	**1**	**2**	**3**	**4**	**5**	**...**
t_high	t_low	Ewe	t_high	t_low	Ewe	...

Point #0 *Point #1*

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.21.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

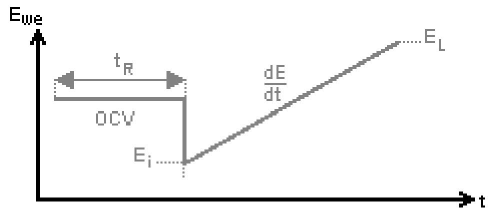
- time:
  The time is calculated with this formula:
  *t (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)*

- Float conversion:
  Ewe and Ece must be converted with the function BL_CONVERTNUMERICINTOSINGLE

7.22. Linear Polarization technique

Technique ID: **134**

Instrument Series	VMP3	SP-300
File	**lp.ecc**	**lp4.ecc**
Timebase	**40µs**	**40µs**



7.22.1. Description

The linear polarization technique is used in corrosion monitoring. This technique is especially designed for the determination of a polarization resistance Rp of a material and Icorr through potential steps around the corrosion potential.

Rp is defined as the slope of the potential-current density curve at the free corrosion potential: Rp = (dE/dI) dE->0

Rp is determined using the ''Rp fit'' graphic tool. Contrary to the Potentiodynamic Pitting (PDP) technique, no current limitation is available with the linear polarization technique.

7.22.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

LP parameters

Label	Description	Data types	Data range
Record_every_dEr	Record every dE (V)	single	≥ 0
Rest_time_T	Rest duration (s)	single	[0..tb*2^{31}]
Record_every_dTr	Record every dT (s)	single	≥ 0
OC1	Step voltage vs initial one (not used)	boolean	True/False
E1	Step voltage (V) (not used)	single	≥ 0
T1	Step duration (s) (not used)	single	≥ 0
vs_initial_scan	Current scan vs initial one	Array of 2 boolean	True/False
Voltage_scan	Voltage scan (V)	Array of 2 single	-
Scan_Rate	slew rate array (mV/s)	Array of 2 single	≥ 0
Scan_number	Scan number	integer	= 0

LP parameters			
Label	**Description**	**Data types**	**Data range**
Record_every_dE	recording on dE	single	≥ 0
Average_over_dE	average every dE	boolean	True/False
Begin_measuring_I	Select the part of the current step (1 = 100%) used for data averaging	single	[0..1]
End_measuring_I		single	[0..1]

7.22.3. Data format

Data format returned by the function BL_GETDATA
Data format depends of the technique process used to record data. The process index is returned in the field TDATAINFOS.PROCESSINDEX.

- Data format of process 0:

0	1	2	3	4	5	...
t_high	t_low	Ewe	t_high	t_low	Ewe	...
Point #0			*Point #1*			

- Data format of process 1:

VMP3 Series:

0	1	2	3	4	5	6	7	8	9	...
t_high	t_low	Ec	<I>	<Ewe>	t_high	t_low	Ec	<I>	<Ewe>	...
Point #0					*Point #1*					

SP-300 Series:

0	1	2	3	4	5	6	7	...
t_high	t_low	<I>	<Ewe>	t_high	t_low	<I>	<Ewe>	...
Point #0				*Point #1*				

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.22.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

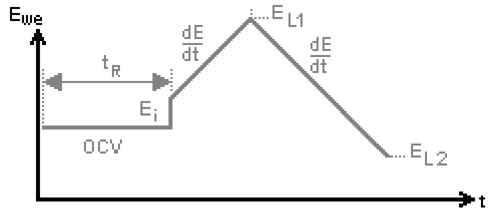
- time:
  The time for the process 0 is calculated with this formula:
  *t (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)*

- Float conversion:
  time (only time of process 1), IRange, freq, Ewe, |Ewe|, Ece, |Ece|, I, |I|, Phase Zwe, Phase Zce, Q-Qo must be converted with the function BL_CONVERTNUMERICINTOSINGLE

7.23. Generalized Corrosion technique

Technique ID: **135**

Instrument Series	VMP3	SP-300
File	**gc.ecc**	**gc4.ecc**
Timebase	**40µs**	**40µs**



7.23.1. Description

The generalized corrosion technique is applied for general corrosion (sometimes called uniform corrosion) study. For this corrosion, anodic dissolution is uniformly distributed over the entire metallic surface. The corrosion rate is nearly constant at all locations.

Microscopic anodes and cathodes are continuously changing their electrochemical behavior from anode to cathode cells for a uniform attack.

7.23.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

GC parameters

Label	Description	Data types	Data range
Record_every_dEr	Record every dE (V)	single	≥ 0
Rest_time_T	Rest duration (s)	single	[0..tb*2^{31}]
Record_every_dTr	Record every dT (s)	single	≥ 0
OC1	Step voltage vs initial one (not used)	boolean	True/False
E1	Step voltage (V) (not used)	single	≥ 0
T1	Step duration (s) (not used)	single	≥ 0
vs_initial_scan	Current scan vs initial one	Array of 3 boolean	True/False
Voltage_scan	Voltage scan (V)	Array of 3 single	-
Scan_Rate	slew rate array (mV/s)	Array of 3 single	≥ 0
Scan_number	Scan number	integer	= 1
Record_every_dE	recording on dE	single	≥ 0
Average_over_dE	average every dE	boolean	True/False
Begin_measuring_I	Select the part of the current step (1 = 100%) used for data averaging	single	[0..1]
End_measuring_I		single	[0..1]

7.23.3. Data format

Data format returned by the function BL_GETDATA
Data format depends of the technique process used to record data. The process index is returned in the field TDATAINFOS.PROCESSINDEX.

- Data format of process 0:

0	1	2	3	4	5	...
t_high	t_low	Ewe	t_high	t_low	Ewe	...
Point #0			*Point #1*			

- Data format of process 1:

For VMP3:

0	1	2	3	4	5	6	7	8	9	...
t_high	t_low	Ec	<I>	<Ewe>	t_high	t_low	Ec	<I>	<Ewe>	...
Point #0					*Point #1*					

For SP-300:

0	1	2	3	4	5	6	7	...
t_high	t_low	<I>	<Ewe>	t_high	t_low	<I>	<Ewe>	...
Point #0				*Point #1*				

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.23.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

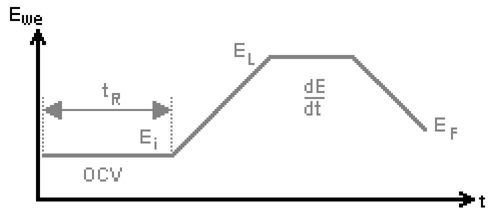
- time:
  The time for the process 0 is calculated with this formula:
  $$t\ (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)$$

- Float conversion:
  time (only time of process 1), IRange, freq, Ewe, |Ewe|, Ece, |Ece|, I, |I|, Phase Zwe, Phase Zce, Q-Qo must be converted with the function BL_CONVERTNUMERICINTOSINGLE

7.24. Cyclic PotentioDynamic Polarization technique

Technique ID: **136**

Instrument Series	VMP3	SP-300
File	**cpp.ecc**	**cpp4.ecc**
Timebase	**40µs**	**44µs**



7.24.1. Description

The Cyclic Potentiodynamic Polarization is often used to evaluate pitting susceptibility. It is the most common electrochemical test for localized corrosion resistance. The potential is swept in a single cycle or slightly less than one cycle. The size of the hysteresis is examined along with the difference between the values of the starting Open circuit corrosion potential and the return passivation potential. The existence of hysteresis is usually indicative of pitting, while the size of the loop is often related to the amount of pitting. This technique can be used to determine the pitting potential and the repassivation potential.

This technique is based both on the PDP and PSP techniques. It begins with a potentiodynamic phase where the potential increases. This phase is limited either with a limit potential (EL) or with a pitting current (Ip) defined by the user. If the pitting current is not reached during the potentiodynamic phase, then a potentiostatic phase is applied until pitting (Ip is reached). Ip can be used as a safety parameter in order to avoid damages on the working electrode. Then an additional potentiodynamic phase is done as a reverse scan.

7.24.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

CPP parameters

Label	Description	Data types	Data range
Record_every_dEr	Record every dE (V)	single	≥ 0
Rest_time_T	Rest duration (s)	single	$[0..tb*2^{31}]$
Record_every_dTr	Record every dT (s)	single	≥ 0
vs_initial_scan	Current scan vs initial one	Array of 3 boolean	True/False
Voltage_scan	Voltage scan (V)	Array of 3 single	-
Scan_Rate	slew rate array (mV/s)	Array of 3 single	≥ 0
Scan_number	Scan number	integer	= 1

CPP parameters			
Label	**Description**	**Data types**	**Data range**
I_pitting	Pitting current	single	≥ 0
t_b	Check condition |I|>Ip after time t_b	single	≥ 0
Record_every_dE	recording on dE	single	≥ 0
Average_over_dE	average every dE	boolean	True/False
Begin_measuring_I	Select the part of the current step (1 = 100%) used for data averaging	single	[0..1]
End_measuring_I		single	[0..1]
Record_every_dT	recording on dt	single	≥ 0

7.24.3. Data format

Data format returned by the function BL_GETDATA
Data format depends of the technique process used to record data. The process index is returned in the field TDATAINFOS.PROCESSINDEX.

- Data format of process 0:

0	1	2	3	4	5	...
t_high	t_low	Ewe	t_high	t_low	Ewe	...
Point #0			*Point #1*			

- Data format of process 1:

VMP3 Series:

0	1	2	3	4	5	6	7	8	9	...
t_high	t_low	Ec	<I>	<Ewe>	t_high	t_low	Ec	<I>	<Ewe>	...
Point #0					*Point #1*					

SP-300 Series:

0	1	2	3	4	5	6	7	...
t_high	t_low	<I>	<Ewe>	t_high	t_low	<I>	<Ewe>	...
Point #0				*Point #1*				

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.24.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

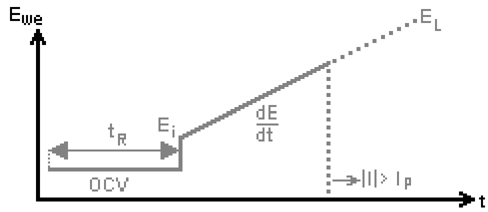
- time:
  The time for the process 0 is calculated with this formula:
  t (s) = TDataInfos.StartTime +
  *TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)*

- Float conversion:
  time (only time of process 1), IRange, freq, Ewe, |Ewe|, Ece, |Ece|, I, |I|, Phase Zwe, Phase Zce, Q-Qo must be converted with the function BL_CONVERTNUMERICINTOSINGLE

7.25. PotentioDynamic Pitting technique

Technique ID: **137**

Instrument Series	VMP3	SP-300
File	**pdp.ecc**	**pdp4.ecc**
Timebase	**40µs**	**42µs**



7.25.1. Description

Pitting corrosion occurs when discrete areas of a material undergo rapid attack while the vast majority of the surface remains virtually unaffected. The basic requirement for pitting is the existence of a passive state for the material in the environment of interest. Pitting of a given material depends strongly upon the presence of an aggressive species in the environment and a sufficiently oxidizing potential.

This technique corresponds to the pitting potential determination of a material, using a potential sweep.The experiment stops when a pitting current (Ip) defined by the user is reached. Ip can be used as a safety parameter in order to avoid damages on the working electrode.

7.25.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

PDP parameters			
Label	**Description**	**Data types**	**Data range**
Record_every_dEr	Record every dE (V)	single	≥ 0
Rest_time_T	Rest duration (s)	single	$[0..tb*2^{31}]$
Record_every_dTr	Record every dT (s)	single	≥ 0
vs_initial_scan	Current scan vs initial one	Array of 2 boolean	True/False
Voltage_scan	Voltage scan (V)	Array of 2 single	-
Scan_Rate	slew rate array (mV/s)	Array of 2 single	≥ 0
Scan_number	Scan number	integer	= 1
I_pitting	Pitting current	single	≥ 0
t_b	Check condition |I|>Ip after time t_b	single	≥ 0
Record_every_dE	recording on dE	single	≥ 0

PDP parameters			
Label	**Description**	**Data types**	**Data range**
Average_over_dE	average every dE	boolean	True/False
Begin_measuring_I	elect the part of the current step (1 = 100%) used for data averaging	single	[0..1]
End_measuring_I		single	[0..1]
Record_every_dT	recording on dt	single	≥ 0
Hold	Hold potential	boolean	True/False

7.25.3. Data format

Data format returned by the function BL_GETDATA
Data format depends of the technique process used to record data. The process index is returned in the field TDATAINFOS.PROCESSINDEX.

- Data format of process 0:

0	1	2	3	4	5	...
t_high	t_low	Ewe	t_high	t_low	Ewe	...
Point #0			*Point #1*			

- Data format of process 1:

VMP3 Series:

0	1	2	3	4	5	6	7	8	9	...
t_high	t_low	Ec	<I>	<Ewe>	t_high	t_low	Ec	<I>	<Ewe>	...
Point #0					*Point #1*					

SP-300 Series:

0	1	2	3	4	5	6	7	...
t_high	t_low	<I>	<Ewe>	t_high	t_low	<I>	<Ewe>	...
Point #0				*Point #1*				

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.25.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

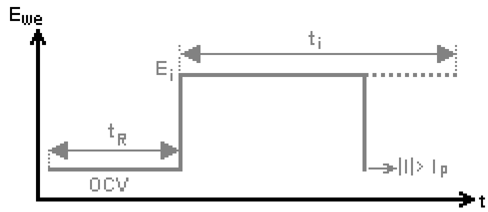
- time:
  The time for the process 0 is calculated with this formula:
  *t (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)*

- Float conversion:
  time (only time of process 1), IRange, freq, Ewe, |Ewe|, Ece, |Ece|, I, |I|, Phase Zwe, Phase Zce, Q-Qo must be converted with the function BL_CONVERTNUMERICINTOSINGLE

7.26. PotentioStatic Pitting technique

Technique ID: **138**

Instrument Series	VMP3	SP-300
File	**psp.ecc**	**psp4.ecc**
Timebase	**24µs**	**24µs**



7.26.1. Description

The PSP technique corresponds to studying pitting occurrence under applied constant potential.

The experiment stops when a pitting current (Ip) defined by the user is reached. Ip can be used as a safety parameter in order to avoid damages on the working electrode.

7.26.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

PSP parameters			
Label	**Description**	**Data types**	**Data range**
Record_every_dEr	Record every dE (V)	single	≥ 0
Rest_time_T	Rest duration (s)	single	[0..tb*2^{31}]
Record_every_dTr	Record every dT (s)	single	≥ 0
Ei	Initial Potential (V)	single	-
OCi	Initial Potential vs initial one	boolean	True/False
Rest_time_Ti	Ei duration (s)	single	≥ 0
Record_every_dT	recording on dt	single	≥ 0
Record_every_dI	recording on dI (A)	single	≥ 0
I_pitting	Pitting current	single	≥ 0
t_b	Check condition |I|>Ip after time t_b	single	≥ 0
I_Range	I range	integer	see IRange constants authorized on section 5. Used structures and constants **Warning** : I Auto-range not authorized

7.26.3. Data format

Data format returned by the function BL_GETDATA
Data format depends of the technique process used to record data. The process index is returned in the field TDATAINFOS.PROCESSINDEX.

- Data format of process 0:

0	1	2	3	4	5	...
t_high	t_low	Ewe	t_high	t_low	Ewe	...
Point #0			*Point #1*			

- Data format of process 1:

0	1	2	3	4	5	6	7	...
t_high	t_low	Ewe	I	t_high	t_low	Ewe	I	...
Point #0				*Point #1*				

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.26.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

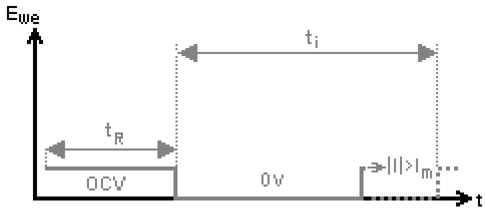
- time:
  The time for the process 0 is calculated with this formula:
  $$t\ (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)$$

- Float conversion:
  time (only time of process 1), IRange, freq, Ewe, |Ewe|, Ece, |Ece|, I, |I|, Phase Zwe, Phase Zce, Q-Qo must be converted with the function BL_CONVERTNUMERICINTOSINGLE

7.27. Zero Resistance Ammeter technique

Technique ID: **139**

Instrument Series	VMP3	SP-300
File	**zra.ecc**	**zra4.ecc**
Timebase	**40µs**	**40µs**



7.27.1. Description

The Zero Resistance Ammeter technique is used to examine the effects of coupling dissimilar metals and to perform some types of electrochemical noise measurement.

This technique consists into applying zero volts between the working electrode (WE) and the counter electrode (CE) and then measures the current and the potentials (Ewe, Ece) versus the reference electrode (REF).

7.27.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

ZRA parameters			
Label	**Description**	**Data types**	**Data range**
Record_every_dEr	Record every dE (V)	single	≥ 0
Rest_time_T	Rest duration (s)	single	[0..tb*2^{31}]
Record_every_dTr	Record every dT (s)	single	≥ 0
Ei	Initial Potential (V)	single	-
OCi	Initial Potential vs initial one	boolean	True/False
Rest_time_Ti	Ei duration (s)	single	≥ 0
Record_every_dT	recording on dt	single	≥ 0
Record_every_dI	recording on dI (A)	single	≥ 0
I_max	Pitting current	single	≥ 0
t_b	Check condition |I|>Ip after time t_b	single	≥ 0

7.27.3. Data format

Data format returned by the function BL_GETDATA
Data format depends of the technique process used to record data. The process index is returned in the field TDATAINFOS.PROCESSINDEX.

- Data format of process 0:

VMP3 series:

0	1	2	3	4	5	6	7	...
t_high	t_low	Ewe	Ece	t_high	t_low	Ewe	Ece	...
Point #0				*Point #1*				

SP-300 series:

0	1	2	3	4	5	...
t_high	t_low	Ewe	t_high	t_low	Ewe	...
Point #0			*Point #1*			

- Data format of process 1:

VMP3 series:

0	1	2	3	4	5	6	7	8	9	10	11	...
t_high	t_low	<Ewe>	<I>	Ece	Q	t_high	t_low	<Ewe>	<I>	Ece	Q	...
Point #0						*Point #1*						

SP-300 series:

0	1	2	3	4	5	6	7	...
t_high	t_low	<Ewe>	<I>	t_high	t_low	<Ewe>	<I>	...
Point #0				*Point #1*				

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.27.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

- time:
  The time for the process 0 is calculated with this formula:

$$t\ (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)$$

- Float conversion:
  time (only time of process 1), IRange, freq, Ewe, |Ewe|, Ece, |Ece|, I, |I|, Phase Zwe, Phase Zce, Q-Qo must be converted with the function BL_CONVERTNUMERICINTOSINGLE

7.28. Manual IR technique

Technique ID: **140**

Instrument Series	VMP3	SP-300
File	**IRcmp.ecc**	**IRcmp4.ecc**
Timebase	**20µs**	**20µs**

7.28.1. Description

The ohmic drop iRu is the voltage drop developed across the solution resistance Ru between the reference electrode and the working electrode, when current is flowing through. When the product iRu gets significant it introduces an important error in the control of the working electrode potential and should be compensated.

In controlled potential techniques, The Manual IR (MIR) can be used to compensate the ohmic drop when the uncompensated solution resistance value (Ru) is known or measured before the experiment start. This technique will not measure Ru.

When used with linked techniques and loops, this technique allow the user to keep the same Ru value for each loop. The user can select the percentage of compensation. It is highly recommended to not exceed 85% of the Ru measured value in order to avoid oscillations of the instrument.

7.28.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

MIR parameters			
Label	**Description**	**Data type**	**Data range**
Rcmp_Value	R value to compensate	single	≥ 0
Rcmp_Mode	Ohmic compensation mode: 0 = software 1 = hardware (SP-300 only)	integer	0 or 1

To deactivate the compensation, you can simply set the Rcmp_value to 0 and Rcmp_Mode to software.

7.28.3. Data format

No data recorded by this technique, it has been designed for linked experiments.

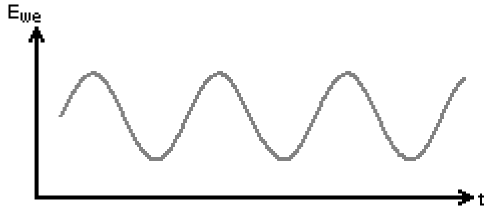
7.28.4. Data conversion

No data conversion.

7.29. IR Determination with PotentioStatic Impedance technique

Technique ID: **141**

Instrument Series	VMP3	SP-300
File	**pzir.ecc**	**pzir4.ecc**
Timebase	**24µs**	**24µs**



7.29.1. Description

The ohmic drop iRu is the voltage drop developed across the solution resistance Ru between the reference electrode and the working electrode, when current is flowing through. When the product iRu gets significant it introduces an important error in the control of the working electrode potential and should be compensated.

The IR Determination with Potentiostatic Impedance (PZIR) technique utilizes Impedance measurements to determine the Ru Value. This technique applies a sinusoidal excitation around the DC potential measured at the beginning of the technique. PZIR technique determines the solution resistance Ru, for one high frequency value, as the real part of the measured impedance. A percentage of the Ru value will be used to compensate next potentio techniques. It is highly recommended to not exceed 85% of the Ru measured value in order to avoid oscillations of the instrument. The Rcmp_Mode parameter will allow to specify the compensation mode for the next potentio techniques (only for SP-300 series).

When used in linked techniques including loops, Ru value can change during the experiment. PZIR can be an ideal tool to do a dynamic ohmic drop compensation between repeated techniques.
For low impedance electrochemical systems it is recommended to use GZIR instead of PZIR.

7.29.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

PZIR parameters			
Label	**Description**	**Data types**	**Data range**
Final_frequency	Final frequency (Hz)	single	= Initial_frequency
Initial_frequency	Initial frequency (Hz)	single	Depend on instrument
Amplitude_Voltage	Sinus amplitude (V)	single	Depend on instrument
Average_N_times	Number of repeat times (used for frequencies	integer	≥ 1

	averaging)		
Wait_for_steady	Number of period to wait before each frequency	single	≥ 0
sweep	sweep linear/logarithmic (TRUE for linear points spacing)	boolean	= True
Rcomp_Level	% IR compensation	single	≥ 0
Rcmp_Mode	Ohmic drop compensation mode. 0 = software 1 = hardware (SP-300 only)	integer	0 or 1

To deactivate the compensation, you can simply set the Rcomp_Level to 0 and Rcmp_Mode to software.

7.29.3. Data format

Data format returned by the function BL_GETDATA for VMP3:

0	1	2	3	4	5	6	7	8	9	...
freq	|Ewe|	|I|	Phase Zwe	Ewe	I	-	|Ece|	|Ice|	Phase Zce	...

Point #0

...	10	11	12	13	14	15	16	17	18	...
...	Ece	-	-	t	Irange	freq	|Ewe|	|I|	Phase Zwe	...

Point #0 *Point #1*

...	19	20	21	22	23	24	25	26	27	28	...
...	Ewe	I	-	|Ece|	|Ice|	Phase Zce	Ece	-	-	t	...

Point #1

...	29	...
...	Irange	...

Point #1

For SP-300:

0	1	2	3	4	5	6	7	8	9	...
freq	|Ewe|	|I|	Phase Zwe	Ewe	I	-	|Ece|	|Ice|	Phase Zce	...
				Point #0						

...	10	11	12	13	14	15	16	...
...	Ece	-	t	freq	|Ewe|	|I|	Phase Zwe	...
		Point #0			*Point #1*			

...	17	18	19	20	21	22	23	24	25	...
...	Ewe	I	-	|Ece|	|Ice|	Phase Zce	Ece	-	t	...
					Point #1					

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.29.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

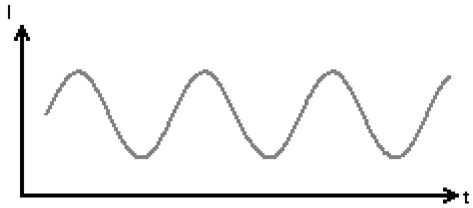
- <u>time</u>:
  The time for the process 0 is calculated with this formula:
  $$t\ (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)$$

- <u>Float conversion</u>:
  time (only time of process 1), IRange, freq, Ewe, |Ewe|, Ece, |Ece|, I, |I|, Phase Zwe, Phase Zce must be converted with the function BL_CONVERTNUMERICINTOSINGLE

7.30. IR Determination with GalvanoStatic Impedance technique

Technique ID: **142**

Instrument Series	VMP3	SP-300
File	**gzir.ecc**	**gzir4.ecc**
Timebase	**24µs**	**24µs**



7.30.1. Description

The ohmic drop iRu is the voltage drop developed across the solution resistance Ru between the reference electrode and the working electrode, when current is flowing through. When the product iRu gets significant it introduces an important error in the control of the working electrode potential and should be compensated.

The IR Determination with Galvanostatic Impedance (GZIR) technique utilizes Impedance measurements to determine the Ru Value. This technique applies a sinusoidal excitation around the DC current measured at the beginning of the technique. GZIR technique determines the solution resistance Ru, for one high frequency value, as the real part of the measured impedance. A percentage of the Ru value will be used to compensate next potentiostatic techniques. It is highly recommended to not exceed exceed 85% of the Ru measured value in order to avoid oscillations of the instrument.

When used in linked techniques including loops, Ru value can change during the experiment. GZIR can be an ideal tool to do a dynamic ohmic drop compensation between repeated techniques.
In the case of particular non-linear systems it can be necessary to use PZIR instead of GZIR.

7.30.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

GZIR parameters			
Label	**Description**	**Data types**	**Data range**
Final_frequency	Final frequency (Hz)	single	= Initial_frequency
Initial_frequency	Initial frequency (Hz)	single	Depend on instrument
Amplitude_Current	Sinus amplitude (A)	singleDepend on instrument	Depend on instrument
Average_N_times	Number of repeat times (used for frequencies	integer	≥ 1

	averaging)		
Rcomp_Level	% IR compensation	single	≥ 0
Wait_for_steady	Number of period to wait before each frequency	single	≥ 0
sweep	sweep linear/logarithmic (TRUE for linear points spacing)	boolean	= True
I_Range	I range	integer	see IRange constants authorized on section 5. Used structures and constants **Warning** : I Auto-range not authorized
Rcmp_Mode	Ohmic drop compensation mode. 0 = software 1 = hardware	integer	0 or 1

To deactivate the compensation, you can simply set the Rcomp_Level to 0 and Rcmp_Mode to software.

7.30.3. Data format

Data format returned by the function BL_GETDATA for VMP3:

0	1	2	3	4	5	6	7	8	9	...
freq	|Ewe|	|I|	Phase Zwe	Ewe	I	-	|Ece|	|Ice|	Phase Zce	...

Point #0

...	10	11	12	13	14	15	16	17	18	...	...
...	Ece	-	-	t	Irange	freq	|Ewe|	|I|	Phase Zwe	...	...

Point #0 *Point #1*

...	20	21	22	23	24	25	26	27	28	29	...
...	Ewe	I	-	|Ece|	|Ice|	Phase Zce	Ece	-	-	t	...

Point #1

...	29	...
...	Irange	...

Point #1

For SP-300:

0	1	2	3	4	5	6	7	8	9	...
freq	|Ewe|	|I|	Phase Zwe	Ewe	I	-	|Ece|	|Ice|	Phase Zce	...

Point #0

...	10	11	12	13	14	15	16	...
...	Ece	-	t	freq	|Ewe|	|I|	Phase Zwe	...

Point #0 (10–12) *Point #1* (13–16)

...	17	18	19	20	21	22	23	24	25	...
...	Ewe	I	-	|Ece|	|Ice|	Phase Zce	Ece	-	t	...

Point #1

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.30.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

- <u>time</u>:
  The time for the process 0 is calculated with this formula:
  t (s) = TDataInfos.StartTime +
  *TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)*

- <u>Float conversion</u>:
  time (only time of process 1), IRange, freq, Ewe, |Ewe|, Ece, |Ece|, I, |I|, Phase Zwe, Phase Zce must be converted with the function BL_CONVERTNUMERICINTOSINGLE

7.31. Loop technique

Technique ID: **150**

Instrument Series	VMP3	SP-300
File	**loop.ecc**	**loop4.ecc**
Timebase	**20µs**	**20µs**

7.31.1. Description

The loop technique has been built to repeat all or a part of an experiment made with several linked techniques. The user has to define the technique number *Ne* where he wants to go back to (*Ne* = 0 for the first technique of the experiment). Then all the techniques linked after the selected one will be repeated. The user has to choose the number of time *nt* that the experiment will be looped. An experiment with *nt* = 2 will have three loops. The loop technique can be also used as a mandatory goto technique when the experiment will be looped for unlimited number of times. To activate this mode the user has to put *nt* = −1.

7.31.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

Loop parameters			
Parameters	**Description**	**Data type**	**Data range**
loop_N_times	Loop N times	integer	≥ 1 for conditional goto
			−1 for mandatory goto
protocol_number	Index of the technique to be linked (index 0-based)	integer	≥ 0

7.31.3. Data format

No data recorded by this technique, it has been designed for linked experiments.

7.31.4. Data conversion

No data conversion.

7.32. Trigger Out technique

Technique ID: **151**

Instrument Series	VMP3	SP-300
File	**TO.ecc**	**TO4.ecc**
Timebase	**20µs**	**20µs**

7.32.1. Description

The 'Trigger Out' technique can be used to synchronize a potentiostat channel with an external instrument. A trigger out pulse is generated by the potentiostat during the technique placed after the 'Trigger Out' technique. The pulse duration and level can be programmed by the user. The pulse cannot last more than the next technique duration. Before and after the pulse the potentiostat drives the trigger out signal to the default level set by the 'Trigger Out Set' technique.

7.32.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

TO parameters			
Parameters	**Description**	**Data type**	**Data range**
Trigger_Logic	Trigger out level	integer	0 or 1
Trigger_Duration	Trigger out duration (s)	integer	≥ 0

7.32.3. Data format

No data recorded by this technique, it has been designed for linked experiments.

7.32.4. Data conversion

No data conversion.

7.33. Trigger In technique

Technique ID: **152**

Instrument Series	VMP3	SP-300
File	**TI.ecc**	**TI4.ecc**
Timebase	**20µs**	**20µs**

7.33.1. Description

The 'Trigger In' technique can be used to synchronize a potentiostat channel with an external instrument. The potentiostat waits an external trigger to continue the experiment with the technique set after the 'Trigger In' technique. Before receiving the trigger the potentiostat goes to the next technique control mode. The trigger in signal is level sensitive and can be set to be either logic low or high. For the potentiostat to recognize the trigger a pulse must be set and held for a minimum of 100 µs.

7.33.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

TI parameters			
Parameters	**Description**	**Data type**	**Data range**
Trigger_Logic	Trigger in level	integer	0 or 1

7.33.3. Data format

No data recorded by this technique, it has been designed for linked experiments.

7.33.4. Data conversion

No data conversion.

7.34. Trigger Out Set technique

Technique ID: **153**

Instrument Series	VMP3	SP-300
File	**TOS.ecc**	**TOS4.ecc**
Timebase	**20µs**	**20µs**

7.34.1. Description

The 'Trigger Out Set' technique can be used together with the 'Trigger Out' technique to synchronize the potentiostat with an external instrument. 'Trigger Out Set' technique sets the default level of the trigger out signal to be either at a logic low or high level. Before and after a pulse generated by the 'Trigger Out' technique the potentiostat drives the trigger out signal to the default level. The trigger out default level can be changed only by another execution of a 'Trigger Out Set' technique or by a power-up or reset of the potentiostat.

7.34.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

TOS parameters			
Parameters	**Description**	**Data type**	**Data range**
Trigger_Logic	Trigger out value	integer	0 or 1

7.34.3. Data format

No data recorded by this technique, it has been designed for linked experiments.

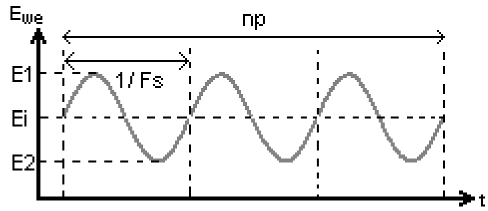
7.34.4. Data conversion

No data conversion.

7.35. Large Amplitude Sinusoidal Voltammetry technique

Technique ID: **159**

Instrument Series	VMP3	SP-300
File	**lasv.ecc**	**lasv4.ecc**
Timebase	**50µs**	**50µs**



7.35.1. Description

Large Amplitude Sinusoidal Voltammetry (LASV) is an electrochemical technique where the potential excitation of the working electrode is a large amplitude sinusoidal waveform. Similar to the cyclic voltammetry (CV) technique, it gives qualitative and quantitative information on redox processes. In contrast to the CV, the double layer capacitive current is not subject to sharp transitions at reverse potentials. Since the electrochemical systems are non-linear the current response exhibits higher order harmonics at large sinusoidal amplitudes. Valuable information can be found from data analysis in the frequency domain.

7.35.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

LASV parameters			
Label	**Description**	**Data types**	**Data range**
Ei	Initial potential	single	-
Ei_vs_initial	Initial potential vs initial one	boolean	True/False
Fs	Frequency of applied sinusoidal	Array of 20 single	≥ 0
E1	High Potential of sinusoidal (V)	Array of 20 single	-
E1_vs_initial	Voltage E1 vs initial one	Array of 20 boolean	True/False
E2	Low Potential of sinusoidal (V)	Array of 20 single	-
vs_initial	Voltage E2 vs initial one	Array of 20 boolean	True/False
Period_number	Number of periods	Array of 20 integer	≥ 0
Record_every_dT	Record every dt (s)	Array of 20 single	≥ 0
Record_every_dI	Record every dI (A)	Array of 20 single	≥ 0
Step_number	Number of steps minus 1	integer	[0..19]

LASV parameters			
Label	**Description**	**Data types**	**Data range**
N_Cycles	Number of times the technique is repeated	integer	≥ 0

This parameters cannot be updated with BL_UPDATEPARAMETERS (or BL_UPDATEPARAMETERS_VEE) function.

7.35.3. Data format

See CP technique data format.

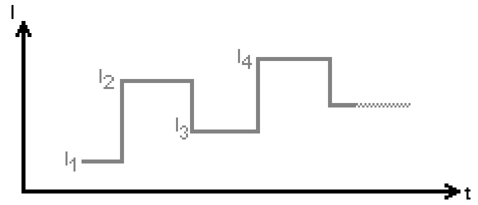
7.35.4. Data conversion

See CP technique data conversion.

7.36. Chrono-Potentiometry technique with limits

Technique ID: **155**

Instrument Series	VMP3	SP-300
File	**cplimit.ecc**	**cplimit4.ecc**
Timebase	**50µs**	**34µs**



7.36.1. Description

The Chronopotentiometry (CP) is a controlled current technique. The current is controlled and the potential is the variable determined as a function of time. The chronopotentiometry technique is similar to the Chronoamperometry technique, potential steps being replaced by current steps. The current is applied between the working and the counter electrode.

This technique can be used for different kind of analysis or to investigate electrode kinetics. It is considered less sensitive than voltammetric techniques for analytical uses. Generally, the curves Ewe = f(t) contains plateaus that correspond to the redox potential of electroactive species.

3 limits (tests) are available. These 3 limits can be combined with logical operator (AND , OR). An limit is defined with a variable, a compare operator, and a value.
The variables are potential (E), current (I), potential on Auxiliary 1 (AUX1) or potential on auxiliary 2 (AUX2).
The logical operators are "<" or ">".
The value is a single data type value.

7.36.1. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

CPLIMIT parameters			
Label	**Description**	**Data types**	**Data range**
Current_step	Current step (A)	Array of 20 single	-
vs_initial	Current step vs initial one	Array of 20 boolean	True/False
Duration_step	Duration step (s)	Array of 20 single	[0..tb*2^31]
Step_number	Number of steps minus 1	integer	[0..98]
Record_every_dT	Record every dt (s)	single	≥ 0
Record_every_dE	Record every dE (V)	single	≥ 0

CPLIMIT parameters			
Label	**Description**	**Data types**	**Data range**
N_Cycles	Number of times the technique is repeated	integer	≥ 0
Test1_Config	Configuration of Test1 by step	Array of 20 integer	See format below
Test1_Value	Value of Test1 by step	Array of 20 single	[0..tb*2^{31}]
Test2_Config	Configuration of Test1 by step	Array of 20 integer	See format below
Test2_Value	Value of Test2 by step	Array of 20 single	[0..tb*2^{31}]
Test3_Config	Configuration of Test3 by step	Array of 20 integer	See format below
Test3_Value	Value of Test3 by step	Array of 20 single	[0..tb*2^{31}]
Exit_Cond	Exit condition	Array of 20 integer	0: Next Step 1: Next Technique 2: STOP Experiment
I_Range	I range	integer	see IRange constants allowed on section 5. Used structures and constants **Warning** : I Auto-range not authorized

Test configuration format:

the test configuration is an integer on 32 bits.

31	...	5	4	3	2	1	0
Variable			Sign			Logic	Active
E (potentiel) = 0 AUX1 (Auxiliaire 1) = 1 AUX2 (Auxiliaire 2) = 2 I (current) = 3			< = 0 > = 1			OR = 0 AND = 1	Active = 1 / not active = 0

If Test3 is active, Test1 and Test2 must be active. The Test3 is ignored if Test2 and Test1 are inactive.
If Test2 is active, Test1 must be active. The Test2 is ignored if Test1 is inactive.

7.36.2. Data format

See CP technique data format.

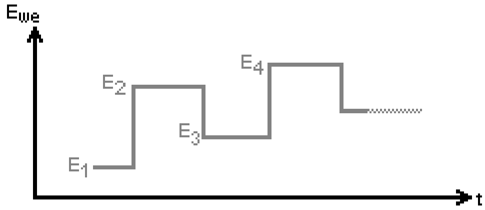
7.36.3. Data conversion

See CP technique data format.

7.37. Chrono-Amperometry technique with limits

Technique ID: **157**

Instrument Series	VMP3	SP-300
File	**calimit.ecc**	**calimit4.ecc**
Timebase	**50µs**	**34µs**



7.37.1. Description

The basis of the controlled-potential techniques is the measurement of the current response to an applied potential step.

The Chronoamperometry (CA) technique involves stepping the potential of the working electrode from an initial potential, at which (generally) no faradic reaction occurs, to a potential Ei at which the faradic reaction occurs. The current-time response reflects the change in the concentration gradient in the vicinity of the surface.
Chronoamperometry is often used for measuring the diffusion coefficient of electroactive species or the surface area of the working electrode. This technique can also be applied to the study of electrode processes mechanisms.

An alternative and very useful mode for recording the electrochemical response is to integrate the current, so that one obtains the charge passed as a function of time. This is the chronocoulometric mode that is particularly used for measuring the quantity of adsorbed reactants.

3 limits (tests) are available. These 3 limits can be combined with logical operator (AND, OR). A limit is defined with a variable, a compare operator, and a value.
The variables are potential (E), current (I), potential on Auxiliary 1 (AUX1) or potential on auxiliary 2 (AUX2).
The logical operators are “<” or “>”.
The value is a single data type value.

7.37.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

CALIMIT parameters			
Label	**Description**	**Data types**	**Data range**
Voltage_step	Voltage step (V)	Array of 100 single	-
vs_initial	Voltage step vs initial one	Array of 100 boolean	True/False
Duration_step	Duration step (s)	Array of 100 single	≥ 0

CALIMIT parameters			
Label	**Description**	**Data types**	**Data range**
Step_number	Number of steps minus 1	integer	[0..98]
Record_every_dT	Record every dt (s)	single	≥ 0
Record_every_dI	Record every dI (A)	single	≥ 0
Test1_Config	Configuration of Test1 by step	Array of 20 integer	See format below
Test1_Value	Value of Test1 by step	Array of 20 single	$[0..tb*2^{31}]$
Test2_Config	Configuration of Test1 by step	Array of 20 integer	See format below
Test2_Value	Value of Test2 by step	Array of 20 single	$[0..tb*2^{31}]$
Test3_Config	Configuration of Test3 by step	Array of 20 integer	See format below
Test3_Value	Value of Test3 by step	Array of 20 single	$[0..tb*2^{31}]$
Exit_Cond	Exit condition	Array of 20 integer	0: Next Step 1: Next Technique 2: STOP Experiment
N_Cycles	Number of times the technique is repeated	integer	≥ 0

Test configuration format:

the test configuration is an integer on 32 bits.

31	…	5	4	3	2	1	0
Variable			Sign			Logic	Active
E (potentiel) = 0 AUX1 (Auxiliaire 1) = 1 AUX2 (Auxiliaire 2) = 2 I (current) = 3			< = 0 > = 1			OR = 0 AND = 1	Active = 1 / not active = 0

If Test3 is active, Test1 and Test2 must be active. The Test3 is ignored if Test2 and Test1 are inactive.
If Test2 is active, Test1 must be active. The Test2 is ignored if Test1 is inactive.

7.37.3. Data format

See CP technique data format.

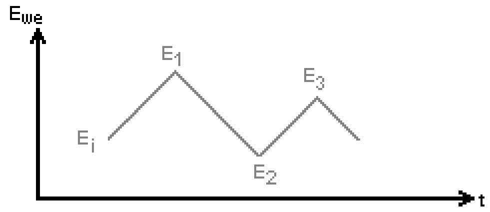
7.37.4. Data conversion

See CP technique data conversion.

7.38. Voltage Scan technique with limits

Technique ID: **158**

Instrument Series	VMP3	SP-300
File	**vscanlimit.ecc**	**vscanlimit4.ecc**
Timebase	**60µs**	**60µs**



7.38.1. Description

The Potentiodynamic (PDYN) technique allows the user to perform potentiodynamic periods with different scan rates.

3 limits (tests) are available. These 3 limits can be combined with logical operator (AND , OR). An limit is defined with a variable, a compare operator, and a value.
The variables are potential (E), current (I), potential on Auxiliary 1 (AUX1) or potential on auxiliary 2 (AUX2).
The logical operators are "<" or ">".
The value is a single data type value.

7.38.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

PDYNLIMIT parameters			
Parameters	**Description**	**Data types**	**Data range**
Voltage_step	Vertex potential (V)	Array of 100 single	-
vs_initial	Vertex potential vs initial one	Array of 100 boolean	True/False
Scan_Rate	Scan rate (V/s) from previous vertex potential	Array of 100 single	> 0, Value of the first scan-rate is ignored
Scan_number	Number of scans minus 1	integer	[0..98]
Record_every_dE	Record every dE (V)	single	≥ 0
N_Cycles	Number of times the technique is repeated	integer	≥ 0
Begin_measuring_I	Select the part of the potential step. 1 = 100% used for data	single	[0..1]
End_measuring_I		single	[0..1]

PDYNLIMIT parameters			
Parameters	**Description**	**Data types**	**Data range**
	averaging		
Test1_Config	Configuration of Test1 by step	Array of 20 integer	See format below
Test1_Value	Value of Test1 by step	Array of 20 single	[0..tb*2^{31}]
Test2_Config	Configuration of Test1 by step	Array of 20 integer	See format below
Test2_Value	Value of Test2 by step	Array of 20 single	[0..tb*2^{31}]
Test3_Config	Configuration of Test3 by step	Array of 20 integer	See format below
Test3_Value	Value of Test3 by step	Array of 20 single	[0..tb*2^{31}]
Exit_Cond	Exit condition	Array of 20 integer	0: Next Step 1: Next Technique 2: STOP Experiment

Test configuration format:

the test configuration is an integer on 32 bits.

31	...	5	4	3	2	1	0
Variable			Sign			Logic	Active
E (potentiel) = 0 AUX1 (Auxiliaire 1) = 1 AUX2 (Auxiliaire 2) = 2 I (current) = 3			< = 0 > = 1			OR = 0 AND = 1	Active = 1 / not active = 0

If Test3 is active, Test1 and Test2 must be active. The Test3 is ignored if Test2 and Test1 are inactive.
If Test2 is active, Test1 must be active. The Test2 is ignored if Test1 is inactive.

7.38.3. Data format

See PDYN technique data format.

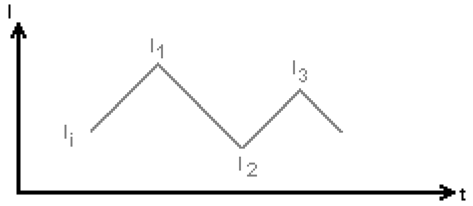
7.38.4. Data conversion

See PDYN technique data conversion.

7.39. Current Scan technique with limits

Technique ID: **156**

Instrument Series	VMP3	SP-300
File	**iscanlimit.ecc**	**iscanlimit4.ecc**
Timebase	**60µs**	**60µs**



7.39.1. Description

The Galvanodynamic (GDYN) technique allows the user to perform galvanodynamic periods with different scan rates.

3 limits (tests) are available. These 3 limits can be combined with logical operator (AND , OR). An limit is defined with a variable, a compare operator, and a value.
The variables are potential (E), current (I), potential on Auxiliary 1 (AUX1) or potential on auxiliary 2 (AUX2).
The logical operators are "<" or ">".
The value is a single data type value.

7.39.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

GDYNLIMIT parameters			
Label	**Description**	**Data types**	**Data range**
Current_step	Vertex current (A)	Array of 100 single	-
vs_initial	Vertex current vs initial one	Array of 100 boolean	True/False
Scan_Rate	Scan rate (A/s) from previous vertex current	Array of 100 single	> 0
Scan_number	Number of scans minus 1	integer	[0..98]
Record_every_dI	Record every dI (A)	single	≥ 0
N_Cycles	Number of times the technique is repeated	integer	≥ 0
Begin_measuring_E	Select the part of the current step (1 = 100%) used for data averaging	single	[0..1]
End_measuring_E		single	[0..1]
Test1_Config	Configuration of	Array of 20 integer	See format below

GDYNLIMIT parameters			
Label	**Description**	**Data types**	**Data range**
	Test1 by step		
Test1_Value	Value of Test1 by step	Array of 20 single	[0..tb*2^{31}]
Test2_Config	Configuration of Test1 by step	Array of 20 integer	See format below
Test2_Value	Value of Test2 by step	Array of 20 single	[0..tb*2^{31}]
Test3_Config	Configuration of Test3 by step	Array of 20 integer	See format below
Test3_Value	Value of Test3 by step	Array of 20 single	[0..tb*2^{31}]
Exit_Cond	Exit condition	Array of 20 integer	0: Next Step 1: Next Technique 2: STOP Experiment
I_Range	I range	integer	see IRange constants authorized on section 5. Used structures and constants **Warning** : I Auto-range not authorized

Test configuration format:

the test configuration is an integer on 32 bits.

31	...	5	4	3	2	1	0
Variable			Sign			Logic	Active
E (potentiel) = 0 AUX1 (Auxiliaire 1) = 1 AUX2 (Auxiliaire 2) = 2 I (current) = 3			< = 0 > = 1			OR = 0 AND = 1	Active = 1 / not active = 0

If Test3 is active, Test1 and Test2 must be active. The Test3 is ignored if Test2 and Test1 are inactive.
If Test2 is active, Test1 must be active. The Test2 is ignored if Test1 is inactive.

7.39.3. Data format

See GDYN technique data format.

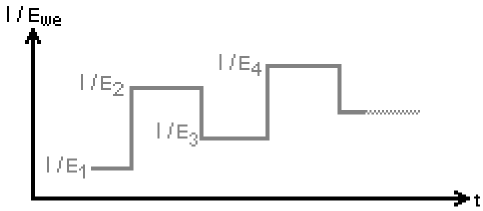
7.39.4. Data conversion

See GDYN technique data format.

7.40. Modular Pulse technique

Technique ID: **167**

Instrument Series	VMP3	SP-300
File	**mp.ecc**	**mp4.ecc**
Timebase	**100µs**	**100µs**



7.40.1. Description

The Modular pulse technique (MOD) allows the user to control successively in different sequences the current and/or the voltage of the cell. With this technique including galvanostatic and potentiostatic sequences, the switch from one mode to the other is very fast. The recording conditions included in the sequence (r_c) offer the possibility to record only few sequences in a long time experiment. This technique is particularly useful for electrochemical coating.

7.40.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

MOD parameters

Label	Description	Data types	Data range
Value_step	Voltage step (V) in Potentiostatic mode or Current step (A) in Galvanostatic mode	Array of 20 single	-
vs_initial	Voltage / Current step vs initial one	Array of 20 boolean	True/False
Duration_step	Duration step (s)	Array of 20 single	≥ 2*tb
Record_every_dT	Record every dt (s)	Array of 20 single	≥ 0
Record_every_dM	Record every dI (A) in Potentiostatic Record every dE (V) in Galvanostatic	Array of 20 single	≥ 0
Mode_step	Potentiostatic or Galvanostatic	Array of 20 integer	0 : Potentiostatic 1 : Galvanostatic
Step_number	Number of steps minus 1	integer	[0..19]
Record_every_rc	Record every cycle	integer	≥ 0

MOD parameters			
Label	**Description**	**Data types**	**Data range**
N_Cycles	Number of times the technique is repeated	integer	≥ 0

7.40.3. Data format

Data format returned by the function BL_GETDATA :

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	...
pBuf	t_high	t_low	Ewe	I	cycle	mode	step	t_high	t_low	Ewe	I	cycle	mode	step	...
	Point #0							Point #1							

The number of points saved in the buffer is returned in the field TDATAINFOS.NBROWS. The number of variables defining a point is returned in the field TDATAINFOS.NBCOLS.

7.40.4. Data conversion

Data returned into the buffer are not usable as-is, one must convert the data before:

- time:
  The time is calculated with this formula:
  $$t\ (s) = TDataInfos.StartTime + TDataInfos.CurrentValues.TimeBase * ((thigh << 32) + tlow)$$

- Float conversion:
  Ewe and I must be converted with the function BL_CONVERTNUMERICINTOSINGLE

- cycle:
  no conversion needed

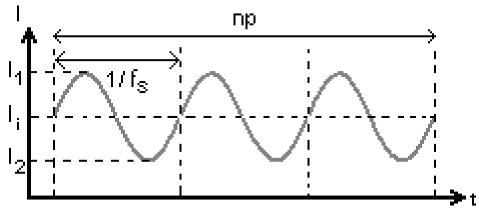
- mode:
  no conversion needed. '0' is the potentio mode and '1' is the galvano mode.

- step:
  no conversion needed

7.41. Constant Amplitude Sinusoidal micro Galvano polarization technique

Technique ID: **169**

Instrument Series	VMP3	SP-300
File	**casg.ecc**	**casg4.ecc**
Timebase	**50µs**	**50µs**



7.41.1. Description

Constant Amplitude Sinusoidal micro Galvano polarization (CASG) is a technique similar than CASP. But in that case, the perturbation is performed around an initial current (*I*i) with a small amplitude (Ia) and a constant low frequency (fs).Thanks to a Direct Fourier Transform the amplitudes of the fundamental frequency (fs), 1st (2 fs) and 2nd (3 fs) harmonics are determined. This technique can also be used for other applications such as battery, fuel cell, …

7.41.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

CASG parameters			
Label	**Description**	**Data types**	**Data range**
Ii	Initial current	single	-
Ii_vs_initial	Initial current vs initial one	boolean	True/False
Fs	Frequency of applied sinusoidal	Array of 20 single	≥ 0
I1	High current of sinusoidal (A)	Array of 20 single	-
I1_vs_initial	Current I1 vs initial one	Array of 20 boolean	True/False
I2	Low Current of sinusoidal (A)	Array of 20 single	-
I2_vs_initial	Current I2 vs initial one	Array of 20 boolean	True/False
Period_number	Number of periods	Array of 20 integer	≥ 0
Record_every_dT	Record every dt (s)	Array of 20 single	≥ 0
Record_every_dE	Record every dE (V)	Array of 20 single	≥ 0
Step_number	Number of steps minus 1	integer	[0..19]

CASG parameters			
Label	**Description**	**Data types**	**Data range**
N_Cycles	Number of times the technique is repeated	integer	≥ 0

7.41.3. Data format

See CP technique data format.

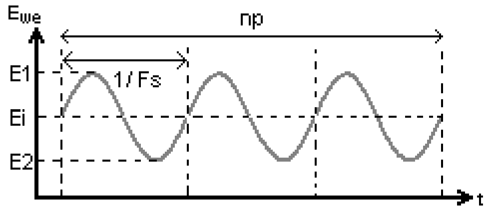
7.41.4. Data conversion

See CP technique data conversion.

7.42. Constant Amplitude Sinusoidal micro Potentio polarization technique

Technique ID: **170**

Instrument Series	VMP3	SP-300
File	**casp.ecc**	**casp4.ecc**
Timebase	**50µs**	**50µs**



7.42.1. Description

Constant Amplitude Sinusoidal micro Potentio polarization (CASP) is a technique used to determine the corrosion current and the*Tafel* coefficients. In this technique, a sinusoidal voltage is applied around a potential (Ei) with a small amplitude (Va) and a constant low frequency (fs).Thanks to a Direct Fourier Transform, the amplitudes of the fundamental frequency (fs), 1st (2 fs) and 2nd (3 fs) harmonics are determined and used to calculate the corrosion current and the *Tafel* coefficients. This technique *was* designed to be faster than the usual linear polarization around the corrosion potential and, *compared to the Tafel fit, does not require an adjustment of the Tafel parameters to have access to Icorr* .

7.42.2. Technique parameters

Technique parameters available for the function BL_LOADTECHNIQUE:

CASP parameters			
Label	**Description**	**Data types**	**Data range**
Ei	Initial potential	single	-
Ei_vs_initial	Initial potential vs initial one	boolean	True/False
Fs	Frequency of applied sinusoidal	Array of 20 single	≥ 0
E1	High Potential of sinusoidal (V)	Array of 20 single	-
E1_vs_initial	Voltage E1 vs initial one	Array of 20 boolean	True/False
E2	Low Potential of sinusoidal (V)	Array of 20 single	-
vs_initial	Voltage E2 vs initial one	Array of 20 boolean	True/False
Period_number	Number of periods	Array of 20 integer	≥ 0
Record_every_dT	Record every dt (s)	Array of 20 single	≥ 0

CASP parameters			
Label	**Description**	**Data types**	**Data range**
Record_every_dI	Record every dI (A)	Array of 20 single	≥ 0
Step_number	Number of steps minus 1	integer	[0..19]
N_Cycles	Number of times the technique is repeated	integer	≥ 0

This parameters cannot be updated with BL_UPDATEPARAMETERS (or BL_UPDATEPARAMETERS_VEE) function.

7.42.3. Data format

See CP technique data format.

7.42.4. Data conversion

See CP technique data conversion.

8. Global parameters for hardware configuration

8.1 Electrode connection

The configuration parameters can be used with all techniques. The configuration parameters modification follows the same method as the technique parameters one.

CE to ground Configuration :

Electrode connection parameters (only for VMP3 series)			
Parameters	**Description**	**Data type**	**Data range**
ce	CE to ground	integer	CE to ground mode : ce = 1 and em = 3 Standard mode : ce = em = 0
em	Controled potential mode	integer	

Note:

The functions BL_SetHardConf and BL_GetHardConf must be only used with SP-300 series in order to change electrode connection mode.
See 5. Used structures and constants and 6. Functions reference.

8.2 Instrument ground

This parameters is only used with SP-300 series. The functions BL_SetHardConf and BL_GetHardConf must be used to modify or get the value of the parameter. See 6. Functions reference.

8.3 Record and external control options

These options can be used only with SP-300 series.

Record and external control options parameters (only for SP-300 series)			
Parameters	**Description**	**Data type**	**Data range**
Rcmp_Mode	Solution ohmic drop mode ofcompensation. The SP-300 has an harware control of this compensation that can be enabled with this flag.	integer	0 = software based (via the *Rcmp_Value* or *Rcomp_Level* parameter) 1 = Hardware based (faster)

<table>
<tr><th colspan="4">Record and external control options parameters (only for SP-300 series)</th></tr>
<tr><td>xctr</td><td>Bitfield controling some extra options:

Activation of External control, or record of additional values: Ece, Analog IN1 & 2, Ramp, Charge, Control and IRange values

Conversion to binary value to activate each options (see data range).

1 : activated

0 : not activated</td><td>Integer / bitfield</td><td><table><tr><th>Bit</th><th>Option</th></tr><tr><td>1</td><td>Record Ece</td></tr><tr><td>2</td><td>Record Analog IN1</td></tr><tr><td>3</td><td>Record Analog IN2</td></tr><tr><td>4</td><td>Enable External ctrl</td></tr><tr><td>5</td><td>Reserved</td></tr><tr><td>6</td><td>Record Control</td></tr><tr><td>7</td><td>Record Charge</td></tr><tr><td>8</td><td>Record IRange</td></tr></table>
All remaining bits are reserved.

For example, activate External and record Charge. Value in:

binary = 0b01001000

integer = 72

hex = 0x0048</td></tr>
<tr><td>R32</td><td>Ece ERange</td><td>integer</td><td>see ERange constants authorized on section 5. Used structures and constants</td></tr>
<tr><td>raux1</td><td>Analog IN 1 ERange</td><td>integer</td><td>see ERange constants authorized on section 5. Used structures and constants</td></tr>
<tr><td>raux2</td><td>Analog IN 2 ERange</td><td>integer</td><td>see ERange constants authorized on section 5. Used structures and constants</td></tr>
</table>

Note:

- The external control option allows to control the potentiostat / galvanostat from an external signal source through the Analog IN 2 input. For the potentio techniques the input voltage simply adds on the technique waveform. For the galvano techniques the input voltage is converted into a current. A simply rule of thumb is to consider 1V as the full scale of the selected current range. For more precise control one should divide the input voltage, (which should not exceed ± 1V) by the value of the shunt resistor. The input voltage has no effect if a technique is not running.

ANNEXE A. Find instruments

The **EC-Lab® Development Package** includes a library (**blfind.dll**) to find available instruments (Ethernet and USB).

1. Calling conventions

The library uses the **stdcall** calling conventions for all exported functions.

2. Multi-thread applications

All exported functions are protected by a synchronization object, they can be called in a multi-thread application.

3. Data types

see section 3.4. Data types

4. Functions reference

Function	**BL_FINDECHEMDEV**
Syntax	function BL_FindEChemDev(pLstdev: PChar; psize: puint32; pNbrDev : puint32): int32;
Parameters	pLstdev pointer to the buffer that will receive the serialization of instruments description (C-string format) (see section 5 Serializations format) *psize* pointer to a uint32 who defines the maximum number of characters of the buffer. It also returns the number of characters of the copied serialization. *pNbrDev* pointer to a uint32 who returns the number of detected ethernet and USB instruments.
Return value	= 0 : the function succeeded < 0 : see section 6 Error codes
Description	This function finds ethernet and USB electrochemistry instruments and copies into the buffer a serialization of descriptions of detected instruments.
Delphi example	procedure FindEchemDevice; var

```
  pSerialization: PChar;
  len, nbDev: uint32;
  Err: int32;
begin
  len := 8192;
  pSerialization := StrAlloc(len);
  zeromemory(pSerialization, len);
  Err := BL_FindEChemDev( pSerialization, @len, @nbDev);
  ShowMessage('Instuments detected : ' + IntToStr(nbDev));
  StrDispose(pSerialization);
end;
```

Function	**BL_FINDECHEMETHDEV**

Syntax

```
function  BL_FindEChemEthDev(pLstdev: PChar;
                             psize: puint32;
                             pNbrDev : puint32): int32;
```

Parameters

pLstdev
: pointer to the buffer that will receive the serialization of instruments description (C-string format) (see section 5 Serializations format)

psize
: pointer to a uint32 who defines the maximum number of characters of the buffer. It also returns the number of characters of the copied serialization.

pNbrDev
: pointer to a uint32 who returns the number of detected ethernet instruments.

Return value

= 0 : the function succeeded
< 0 : see section 6 Error codes

Description

This function finds ethernet electrochemistry instruments and copies into the buffer a serialization of descriptions of detected instruments.

Delphi example

```
procedure FindEchemEthDevice;
var
  pSerialization: PChar;
  len, nbDev: uint32;
  Err: int32;
begin
  len := 4096;
  pSerialization := StrAlloc(len);
  zeromemory(pSerialization, len);
  Err := BL_FindEChemEthDev( pSerialization, @len, @nbDev);
  ShowMessage('Instuments detected : ' + IntToStr(nbDev));
  StrDispose(pSerialization);
end;
```

Function	**BL_FINDECHEMUSBDEV**
Syntax	function BL_FindEChemUsbDev(pLstdev: PChar; psize: puint32; pNbrDev : puint32): int32;
Parameters	pLstdev pointer to the buffer that will receive the serialization of instruments description (C-string format) (see section 5 Serializations format) *psize* pointer to a uint32 who defines the maximum number of characters of the buffer. It also returns the number of characters of the copied serialization. *pNbrDev* pointer to a uint32 who returns the number of detected USB instruments.
Return value	= 0 : the function succeeded < 0 : see section 6 Error codes
Description	This function finds USB electrochemistry instruments and copies into the buffer a serialization of descriptions of detected instruments.

Delphi example	procedure FindEchemUsbDevice;
	var
	pSerialization: PChar;
	len, nbDev: uint32;
	Err: int32;
	begin
	len := 4096;
	pSerialization := StrAlloc(len);
	zeromemory(pSerialization, len);
	Err := BL_FindEChemUsbDev(pSerialization, @len, @nbDev);
	ShowMessage('Instuments detected : ' + IntToStr(nbDev));
	StrDispose(pSerialization);
	end;

Function	**BL_SETCONFIG**
Syntax	function BL_SetConfig(pIP: PChar; pCfg: PChar): int32;
Parameters	*pIP* pointer to the buffer who defines the IP address of the instrument to configure.

	pCfg pointer to the buffer who defines the new TCP/IP parameters of the instrument (see section 5 Serializations format)
Return value	= 0 : the function succeeded < 0 : see section 6 Error codes
Description	This function sets new TCP/IP parameters of selected instrument. IP address, netmask and gateway could be modify.
Delphi example	see code below

```
procedure SetTCPIP;
var
  IPaddress: array[0..15] of char;
  newCfg: array[0..57] of char;
  vErr : int32;
begin
  IPaddress := '192.109.209.220' + #0;
  newCfg := 'IP%192.109.209.22$' +
            'NM%255.255.255.0$' +
            'GW%192.109.209.170$' + #0;

  vErr := BL_SetConfig(IPaddress, newCfg);
end;
```

Procedure	**BL_GETERRORMSG**
Syntax	procedure BL_GetErrorMsg(errorcode: int32; pmsg: PChar; psize: puint32);
Parameters	*errorcode* error code selected *pmsg* pointer to the buffer that will receive the text (C-string format) *psize* pointer to a uint32 who defines the maximum number of characters of the buffer. It also returns the number of characters of the copied string.
Return value	None
Description	This function copies into the buffer the corresponding message of the selected error code.

Delphi example	procedure DisplayErrorMessage;
	var
	  msg: PChar;
	  len: uint32;
	begin
	  len := 255;
	  msg := StrAlloc(len);
	  zeromemory(msg, len);
	  BL_GetErrorMsg(-20, msg, @len);
	  ShowMessage(msg);
	  StrDispose(msg);
	end;

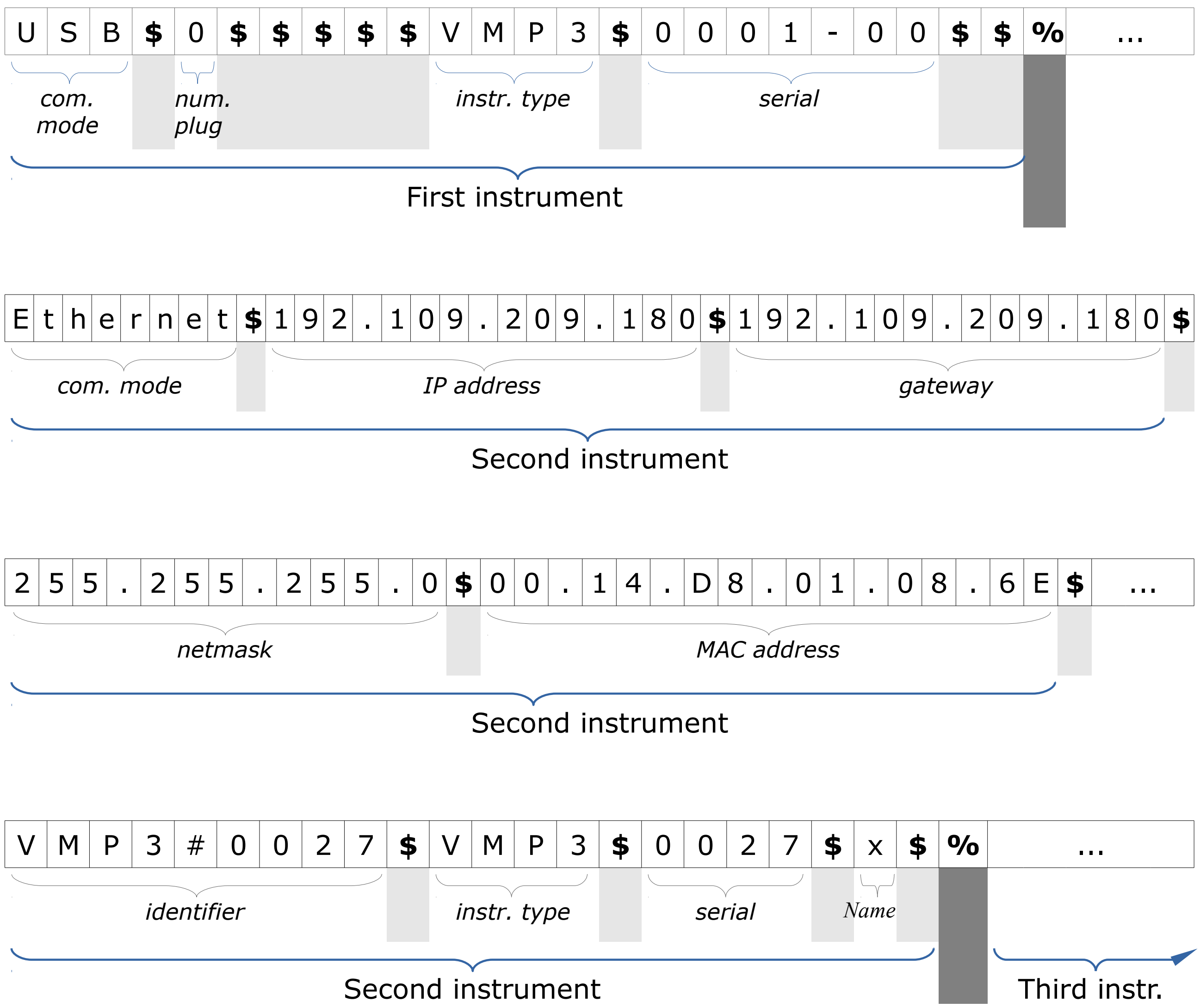
5. Serializations format

5.1. Instruments descriptions

This serialization consists of an array of characters containing the descriptions of detected instruments. If more than one instruments are detected, serialized descriptions are separated by the character **'%'**.
An instrument description is defined by a set of 9 strings descriptors.
In the serialization, descriptors are separated by the character **'$'** and are always serialized in the same order :

(1) Connection mode
(2) **IP address** or **USB plug index**
(3) **Gateway** (always empty with USB)
(4) **Netmask** (always empty with USB)
(5) **MAC address** (always empty with USB)
(6) **Identifier** (always empty with USB)
(7) Instrument type
(8) Serial number
(9) **Name** (always empty with USB)

Example of instruments description serialization :



5.2. TCP/IP parameters

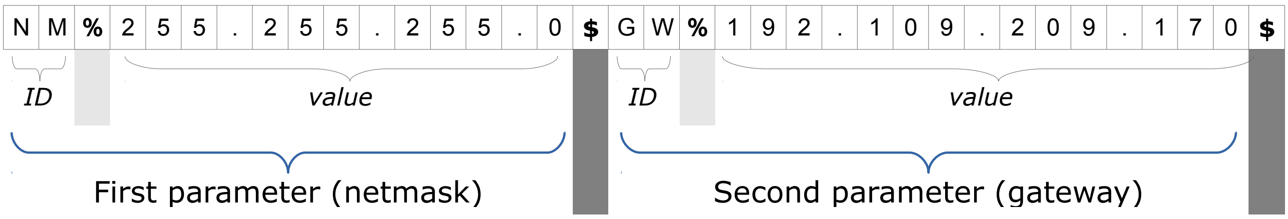
This serialization consists of an array of characters containing the definitions of TCP/IP parameters. If more than one parameters are defined, they must be separated by the character **'$'**.
Each parameter must be defined by an identifier and a value, separated by the character **'%'**.

Identifier must be one of the followings key words :

- **IP** : for IP address
- **NM** : for netmask
- **GW** : for gateway

There is no order in the definition of TCP/IP parameters.
Key words must be in upper case.

Example of TCP/IP parameters serialization :



6. Error codes

General error codes		
Constant	**Value**	**Description**
BLFIND_ERR_UNKNOWN	-1	unknown error
BLFIND_ERR_INVALID_PARAMETER	-2	invalid function parameters

Instrument error codes		
Constant	**Value**	**Description**
BLFIND_ERR_ACK_TIMEOUT	-10	instrument response timeout
BLFIND_ERR_EXP_RUNNING	-11	experiment is running on instrument
BLFIND_ERR_CMD_FAILED	-12	instrument do not execute command

Find error codes		
Constant	**Value**	**Description**
BLFIND_ERR_FIND_FAILED	-20	find failed
BLFIND_ERR_SOCKET_WRITE	-21	cannot write the request of the descriptions of ethernet instruments
BLFIND_ERR_SOCKET_READ	-22	cannot read descriptions of ethernet instrument

Modify error codes		
Constant	**Value**	**Description**
BLFIND_ERR_CFG_MODIFY_FAILED	-30	set TCP/IP parameters failed
BLFIND_ERR_READ_PARAM_FAILED	-31	deserialization of TCP/IP parameters failed
BLFIND_ERR_EMPTY_PARAM	-32	not any TCP/IP parameters in serialization
BLFIND_ERR_IP_FORMAT	-33	invalid format of IP address
BLFIND_ERR_NM_FORMAT	-34	invalid format of netmask address
BLFIND_ERR_GW_FORMAT	-35	invalid format of gateway address
BLFIND_ERR_IP_NOT_FOUND	-38	instrument to modify not found
BLFIND_ERR_IP_ALREADYEXIST	-39	new IP address in TCP/IP parameters already exists