The specification for new driver for UniChrom Data System.

CONTENTS

CONTENTS 2
Basics 4
The Driver 4
The Callback 4
The Driver Type 4
The entry point 4
What entry points simple driver must have 4
How the instrument types are distinguished? 5
GC instrument 5
LC instrument 6
ADC instrument 7
Sampling system API 7
Timed Events (Valve control) API 7
Major API constants reference 8
The real zone names building and usage 8
Zone Object names building rules and usage 9
Object Parameter (Sensor Parameter) indices 10
Gas types for gas control zones 12
Light control state 12
HPLC pump zones (obsolete) 12
Pump indices (obsolete) 13
LC zone parameter indices (obsolete) 13
Notification indices rnXXXX used by drvSetState and InstrumentCallback 13
Detailed functions reference 15
drvOpenEx function 15
drvClose function 15
drvOpenInstrument function 15
drvCloseInstrument function 16
drvOpen[Type]ChannelEx and its variations, ... 17
Different data types and passing it to UniChrom 17
drvCloseChannelEx function 18
drvGetChannelInfo function 19

drvChannelCalibrate function 19
tcbSetZoneValueEx function 19
tcbGetZoneValueEx function 20
tcbCheckReadZoneValue function 20
tcbCheckWriteZoneValue function 20
tcbSetProgRamp function 20
tcbGetProgRamp function 21
drvSetState function 21
drvGetState function 21
fcbSetRecEx function 21
drvUpdate function 22
drvGetZoneInfo function 23
smpSetParam function 23
smpGetParam function 24
smpSetSample function 25
smpGetSample function 25
evtSetParam function 26
evtGetParam function 27
Sample driver 28

Basics

The Driver

User-mode DLL module that exports special UniChrom API functions, which are described further in this document.

The Callback

UniChrom supplied function, which the driver should “call back”. It should be called in “stdcall” calling convention.

The Driver Type

Currently the UniChrom supports three different types of devices which represent different kinds of chromatography devices. The driver types are: GC instrument, LC instrument and ADC instrument.

The entry point

Exported functions from DLL. Should be “stdcall” declared.

What entry points simple driver must have

Here are described entry points that must have every type of driver

`drvOpenEx,`

UniChrom opens the driver passing debug output callback function. Here takes place per driver initialization.

`drvClose,`

UniChrom closes the driver on shutting down the application. Here takes place per driver cleanup.

`drvOpenInstrument or tcbInitialize or fcbInitialize,`

Each driver can support several instruments. UniChrom calls this entry point if it is necessary to “OPEN” instrument and start work with it. Here takes place per instrument initialization and actual hardware connection. The function returns the handle of instrument, which should be unique per driver. This handle is passed in subsequent driver API calls.

`drvCloseInstrument,`

Per instrument cleanup after session close. Hardware should be brought into initial state. All resources bound with instrument should be freed.

`drvOpenChannelEx or its variations,`

Installation of data path callback for UniChrom. When instrument collects some data, it passes the pointer or value of data to UniChrom supplied callback. The type of passed data is determined by callback name.

`drvCloseChannelEx,`

Close the data channel which had been previously opened.

`drvGetChannelInfo,`

Get the channel description, and signal range for specified channel. Channels are per instrument fixed. Getting channel information occurs before the channel is opened

`drvGetChannelFrequency,`

Returns the data sampling frequency for specified channel.

`drvSetChannelFrequency,`

Sets the data sampling frequency for specified channel.

How the instrument types are distinguished?

The instrument types are determined upon loading the driver DLL by presents of several entry points. The GC driver must export **`t`**`cbInitialize` with same prototype as `drvOpenInstrument.` The LC driver must export **`f`**`cbInitialize` with same prototype as `drvOpenInstrument.` The ADC driver must export `drvOpenInstrument.`

Below the complete list of functions for each driver type are listed.

GC instrument

The GC instrument drivers are determined by presence of `tcbInitialize` API function.

The list of functions, the GC driver must support.

`drvOpenEx,`

`drvClose,`

`tcbInitialize` **`instead of`** `drvOpenInstrument,`

`drvCloseInstrument,`

`drvOpenChannelEx or its variations,`

`drvCloseChannelEx,`

```
drvGetChannelInfo,
drvGetChannelFrequency,
drvSetChannelFrequency,
drvChannelCalibrate, //only if required
tcbSetZoneValueEx,
tcbGetZoneValueEx,
tcbCheckReadZoneValue,
tcbCheckWriteZoneValue,
tcbSetProgRamp,
tcbGetProgRamp,
drvSetState,
drvGetState,
```

LC instrument

The LC instrument drivers are determined by presence of `fcbInitialize` API function.

```
drvOpenEx,
drvClose,
drvOpenChannelEx, or its variations
drvCloseChannelEx,
drvGetChannelFrequency,
drvSetChannelFrequency,
drvGetChannelInfo,

fcbInitialize instead of drvOpenInstrument
drvCloseInstrument,
drvSetState,
drvGetState,
fcbSetZoneValueEx,
fcbGetZoneValueEx,
fcbSetRecEx;
```

ADC instrument

The driver must export the following entry points

```
drvOpenEx,
drvClose,
drvOpenInstrument,
drvCloseInstrument
drvOpenChannelEx or its variations
drvCloseChannelEx,
drvGetChannelInfo,
drvGetChannelFrequency,
drvSetChannelFrequency,
drvSetState
drvGetState
```

Sampling system API

Sampling system API is optional and can be present if the instrument has autosampler or probably can have it. The results of these functions are analysed to determine sampling system capabilities.

```
smpGetParam,
smpSetParam,
smpGetSample,
smpSetSample,
```

All types of drivers can have Sampling API entry points.

Timed Events (Valve control) API

Timed events API is optional and can be present only if instrument has any externally controlled devices like valves, which can change state during analysis at several time moments.

```
evtGetParam,
evtSetParam;
```

All types of drivers can have Timed Events API entry points.

Major API constants reference

All the major constants are present in DEVTYPES.*.

Any further improvements should be checked first in devtypes.

Terms:

Zone – object under control. This object represents the entity which is very similar to real heating (control) zone of the Instrument.

Zone object has fixed number of sensors. Indices (numbers) of sensors are split into several categories. Since the real sensor index and the real zone index are encoded in single **dword** value, there are constants and mask which allow encode/decode zone object index and zone sensor index.

Encoded Sensor index = (ZoneType shl 16) or ZoneIndex;

In UniChrom driver model there are following object types (prefix **zotXXXX**):

Constant mnemonic	Value	Meaning
zotTemp	0	Object is temperature control sensor
zotFlow	1	Object is gas flow sensor
zotPres	2	Object is gas pressure sensor
zotVel	3	Object is gas linear velocity sensor. Typically this cannot be measured but can be set.
zotCol	4	Object is gas column flow sensor. Typically this cannot be measured but can be set.

Zone sensors are using zotXXXX constants for building zXXXX – real sensor names

The real zone names building and usage

Zones	Real Zone Name	Zone Index building rule
Oven	zOven=0	zOven
Injectors	zInj=$10000	
	zInjA	zInjA=zInj;
	zInjB	zInjB=zInj+1
	zInjC	zInjC=zInj+2;
	zInjD	zInjD=zInj+3

Detectors	zDet=$20000	
	zDetA	zDetA=zDet
	zDetB	zDetB=zDet+1
	zDetC	zDetC=zDet+2
	zDetD	zDetD=zDet+3
Auxiliary thermal zones	zAux=$30000	
	zAux1	zAux1=zAux
	zAux2	zAux2=zAux+1
	zAux3	zAux3=zAux+2
HPLC Pump Zone	zPump=zInj	
	zPumpA	zPumpA=zInjA
	zPumpB	zPumpB=zInjB
	zPumpC	zPumpC=zInjC
	zPumpD	zPumpD=zInjD
	zSample=zAux	

Each zone has several **zone objects** (control sensors) e.g. zInjA has zoFlowA, zoFlowB, zoFlowC and zoTemp

Zone Object names building rules and usage

The single gas controller can be accessed as flow controller either as pressure (velocity) sensor. Anyway the driver should return right values of magnitude being measured, or return **InvalidZoneValue** if the zone object index is not applicable for the object.

Zone Object Type	**Zone Object Name**	**Building rule**
Temperature sensor zoTemp = 0;	zoTemp	
Flow sensors zotFlow = 1 zoFlow = $10000		
	zoFlowA	zoFlowA= zoFlow

	zoFlowB	zoFlowB= zoFlow + 1
	zoFlowC	zoFlowC= zoFlow + 2
Pressure sensors zotPres = 2; zoPres = $20000		
	zoPresA	zoPresA= zoPres
	zoPresB	zoPresB= zoPres + 1
	zoPresC	zoPresC= zoPres + 2
Velocity sensors zotVel = 3 zoVel = $30000		
	zoVelA	zoVelA = zoVel
	zoVelB	zoVelB = zoVel + 1
	zoVelC	zoVelC = zoVel + 2
Column flow sensor zotCol = 4 zoCol = $40000		
	zoColA	zoColA = zoCol
	zoColB	zoColB = zoCol + 1
	zoColC	zoColC = zoCol + 2

Object Parameter (Sensor Parameter) indices

Constant mnemonic	**Value**	**Description**
opNone	-1	No parameter for sensor. Unused. Just for symmetry
opMin	0	Minimal value that sensor can handle.
opMax	1	Maximal --//--
opSet	2	Methodical setpoint for that sensor
opCur	3	Current value of selected sensor
opDrift	4	Delta – absolute value for error, the selected sensor can assume. The readiness interval. If the sensor is in range Setpoint +/- Drift – then

		all right.
opGas	5	Gas type for selected sensor. Applicable for gas sensor only.
opProg	6	Boolean flag meaning the sensor is ramp programmable
opName	7	Unused. There is no way to pass PChar as long double.
opLight	8	Light/Lamp state for selected Zone, not sensor. Has meaning only for detector zones.
opEnabled	9	If the sensor is ON/OFF?
opMode	10	Gas control mode for gas sensors. See the gas control modes.
opColLen	11	Column length in meters for selected sensor.
opColIDm	12	Column internal diameter in mm for selected sensor.
opType	13	Undefined
opTime	14	time from the start in milliseconds for LC
opPumpSet	opSet	Pumping flow
opLoadSet	15	Loading eluent flow
opEluent	opGas	Type of eluent
opPumpType	opType	type of a pump - 0 - plunger 1 – syringe
opPumpState	opMode	Pump State: 0=Stopped, 1=Pumping, 2=Loading

Gas types for gas control zones

Constant mnemonic	Value	Meaning
gtNone	-1	None – no gas for this sensor
gtUnknown	0	Unknown gas – gas unlisted in this table
gtNitrogen	1	This sensor controls Nitrogen flow/pressure/velocity
gtArgon	2	Argon
gtHelium	3	Helium
gtHydrogen	4	Hydrogen
gtAir	5	Air
gtOxygen	6	Oxygen
gtArgonMethane	7	Argon+Methane

Light control state

When the zone has something like lamp or flame, which can be turned on/off or being in undetermined state, then these values are returned after calling **getZoneValueEx** with parameter index **opLight**

Constant mnemonic	Value	Meaning
ltNone	-1	This zone has not any lights (like injectors)
ltUnknown	0	The state of light is undetermined
ltOff	1	The light is not lit
ltOn	2	The light is lit

HPLC pump zones (obsolete)

The UniChrom HPLC driver model assumes that there can be up to 4 pumps. Each pump is indexed as lczPumpX and one pressure controller which is a mixer pressure (lczPress). These indices are used for addressing corresponding objects within HPLC instrument driver. Setting or getting pump parameters is taken through **drv[Set|Get]ZoneValue(hInstrument,lczPumpA,0,opMax)**

Pump indices (obsolete)

Constant mnemonic	Value	Meaning
lczPumpA	0	
lczPumpB	1	
lczPumpC	2	
lczPumpD	3	
lczPress	4	

Each object of HPLC instrument has the following:

LC zone parameter indices (obsolete)

Parameter	Value	Meaning
lcF_0	1	Setpoint for pump flow
lcF_x	2	Actual flow of pump.
lcFmax	3	Maximal flow for pump
lcPmin	4	Minimal pressure in mixer.
lcPmax	5	Maximal pressure for mixer.
lcP_x	6	Actual pressure in mixer.
lcOnline	7	Is the pump online and working? Turn on/off the pump.
lcCurTime	8	Current analysis time. Not used.
lcAlarm	9	Is the pump in alarm state. This state is defined by overranging specified Pmax, Pmin, Fmax
lcPumpType	10	type of a pump - 0 – the pump is plunger type 1 – the pump is syringe type

Notification indices rnXXXX used by drvSetState and InstrumentCallback

These notification constants are used when UniChrom calls driver for changing device state (going into pre-run, run, post-run mode). Some of the codes are used in

notification callback. When UniChrom receives rnReconfig – it should reconfigure sensor placements, because instrument has determined changes in hardware configurations. The constant rnAll indicates in DeviceCallback changes in sensor values.

Constant mnemonic	**Value**	**Description**
rnAll	-1	Notify that any of the zone parameters have changed. UniChrom should re-read device set points.
rnStart	-2	Start the instrument. This takes place when user forcibly starts the instrument. In general conditions. The instrument detects start conditions, shows them in data callback and notifies by DeviceCallback.
rnStop	-3	Stop the instrument. Actually break the run and stop the sequence.
rnReady	-4	Not used.
rnSeqFinish	-5	Not Used
rnReconfig	-100	Driver request UniChrom reconfiguration and re-reading of device configuration.

Detailed functions reference

drvOpenEx function

```
function drvOpenEx(ADebugFunction:TDebugWriteStr;AClient:Pointer):BOOL;stdcall;
```

The function is intended for per driver initialization and installation of debug callbacks. Please note, that the driver can use STDOUT either as ADebugFunction for printing diagnostic messages. So the **write**() and **printf**() are allowed and welcome in drivers for getting detailed description of in-driver processes. The STDOUT is captured by UniChrom and stored in internal debug buffer also painted in internal console, printed to Win32 console and written to the file if desired. Enabling of debug console is described in UniChrom users manual and typically consist of running **uwin32.exe –debug[session][:out filename]**.
The word “debug” enables only internal graphic console. The word “session” brings Win32 console. Please note, that some output function are buffered and STDOUT is shared by several threads, so please **flush()** the output to make your print operations atomic.

drvClose function

```
function drvClose()BOOL;stdcall;
```

The function must perform per driver cleanup. For instance the LabNET, since it uses only one com port performs it’s initialization in drvOpen and cleanup in drvClose.

drvOpenInstrument function

```
function drvOpenInstrument(AProfile:integer;
      AnAction:TPageNotify;
      AClient:pointer):integer;stdcall;
```

The most important function in instrument configuration, it is used for instrument initialization.
AProfile – is the number of registry hive, where the instrument configuration is located. The UniChrom supports up to 16 instruments.
It’s configurations are stored in:
HKLM\Software\New Analytical Systems\UniChrom\I0 ..I15

The keys I0 … I15 are the folders for instrument configurations. E.g. I0 stores config for first instrument, I1 – for second etc. The configuration hives are built by Configuration Editor (CE) upon driver installation. The parameters of corresponding hive are determined by driver *.INF file which is general Windows INF file, described in MS documentation. The UniChrom particularities are only in parameters only. The UniChrom expects the following parameter in instrument hive:

@ aka Default value	string	Instrument name. CE generates it from instrument type.
DriverName	string	The path for driver module. Should be fully qualified for modules located in non-standard folders
InfName	string	CE writes here the name of INF file, which installs your instrument.

The other parameters are for driver developer. Write and read here what you want, but please remember – this is HKLM and on NT platform only admin can write here.

AnAction - UniChrom supplied callback which driver calls upon changing its state. E.g. – going into PRERUN, RUN, POSTRUN, IDLE state must be informed to UniChrom. The device states are set/get through drvGetState/drvSetState. The callback should be called with notification codes, which are named rnXXX (remote notification). Should be stored and then passed in further calling of TPageNotify(AnAction)(AClient…)

AClient – the client handle – should be stored and then passed in further calling of TPageNotify(AnAction)(AClient…)

The function must return non NULL handle of instrument initialized and ready to work. The NULL return value means problem.

drvCloseInstrument function

```
function drvCloseInstrument(HInstrument:integer):BOOL;stdcall;
```

This function closes the instrument previously opened with drvOpenInstrument.

HInstrument – Handle of instrument, which was previously returned by drvOpenInstruement.

drvOpen[Type]ChannelEx and its variations,

```
function drvOpen[Type]ChannelEx(HInstrument:IHandle;
      AnAction:T[Type]Callback;
      AClient:pointer;
      AChannel:integer):BOOL;stdcall;
```

This function is called when user presses the “Start” button in UniChrom, and opens the selected instrument channels. The instrument channel numbers have no special meanings. Typically the channel numbers 0...N-1 are intended for the analytical signals. After that channels can be any diagnostic or special channels. Please note, that UniChrom calculates measurement time from the number of point it actually got and from frequency, the instrument reports for selected channel.

Please note, that in UniChrom user interface channels are numbered from 1 to N.

Please note, that [Type] – must be one of the following, which UniChrom supports: Point, Block, Int32, Int64, Float, and Double.

Channels type:

- Point – Data is the single 32 bit signed integer. Point means single signal point at one function call.
- Block – Data is the pointer to array of 32 bit values containing 24 bit integers formatted according LNet specification.
- Int32 – Data is the pointer to array of 32 bit integers.
- Int64 – Data is the pointer to array of 64 bit integers.
- Float – Data is the pointer to array of 4-byte IEEE float numbers.
- Double – Data is the pointer to array of 8-byte IEEE float numbers.

Different data types and passing it to UniChrom

The driver writer decides what type of data comes from driver. The data type is determined by name of driver–exported entry points. The driver must export only one of these routines.

```
drvOpenPointChannelEx
drvOpenBlockChannelEx
drvOpenInt32ChannelEx
drvOpenInt64ChannelEx
drvOpenFloatChannelEx
drvOpenDoubleChannelEx
```

If the driver declares that it produce the data of [Type] then it must export **drvOpen[Type]ChannelEx** entry point. When the data is ready for passing it to UniChrom, the driver must call **T[Type]Callback** with parameters which came from **drvOpen[Type]ChannelEx** entry point.

The callback prototypes are placed below:

```
TPointCallback=Procedure(Self:pointer;data:integer;stat:integer);stdcall;
TBlockCallback=Procedure(Self:pointer;PBuf:PIntArray;cbLen:integer);stdcall;

TInt32Callback=procedure(Self:pointer;PData:PDwordArray;cbLen:integer;stat:integer);st
dcall;
TInt64Callback=procedure(Self:pointer;PData:PQwordArray;cbLen:integer;stat:integer);st
dcall;
TFloatCallback=procedure(Self:pointer;PData:PFloatArray;cbLen:integer;stat:integer);st
dcall;
TDoubleCallback=procedure(Self:pointer;PData:PDoubleArray;cbLen:integer;stat:integer);
stdcall;
```

The values meaning and its purpose

Self - the Object instance pointer of Method window which actually controls the instrument. Should be interpreted as handle, which is unique per application. Must not be zero. The pointer to callback function driver and object handle the driver gots in **drvOpen[Type]ChannelEx** . The callback function can be called in context of arbitrary thread. Preferable it would be called once or twice per second for quasi-realtime data display. Obviously there is no need in calling the callback, when no data is arrived after the last call.

PData – driver internal pointer to data, ready for passing to UniChrom. It is desirable not to change the data until the function returns. The UniChrom callbacks are written for fast return if data portions are not so big.

Stat – the status of current operation. The actual data collection startup takes place when here is placed **dsStarting**. Please see the **dsXXXX** constants in **devtypes.***

drvCloseChannelEx function

```
function drvCloseChannelEx(HInstrument:IHandle;AChannel:integer):BOOL;stdcall;
```

The function must close the instrument channel previously opened with the call of **drvOpen[Type]ChannelEx.** Do per-channel cleanup. The comm. resources can be freed if the last channel closes.

drvGetChannelInfo function

```
function drvGetChannelInfo (HInstrument:IHandle;
      AChannel:integer;
      var DevInfo:TDeviceInfo):BOOL;stdcall;
```

This function is called by UniChrom to determine information on particular channel. The DevInfo structure contains several members which have the following meaning:
DevInfo.FullScale – full scale in bits e.g. 24 bit device should place here 0x00FFFFFF;
DevInfo.DevMin – minimal value of units the channel can measure e.g. -2.5 Volts or 0 picoamperes.
DevInfo.DevMax - maximal value of units the channel can measure e.g. +2.5 Volts or 3000 picoamperes.
DevInfo.Flags – currently interpreted as const PChar name for selected channel number. E.g. some chromatograph can return the string like “FID” or “Unknown”

drvChannelCalibrate function

```
function drvChannelCalibrate(HInstrument:IHandle;
      AChannel:integer;Adata:double):double;stdcall;
```

This function is optional and is used for postprocessing the measured data. The main task for this function is to remove float calculation from driver callback. When the driver stores measured data calling UniChrom with **`T[Type]CallBack(…)`** the UniChrom must store data ASAP and return immediately without stopping driver data measuring loop. The calibration (even filtering) takes place later by calling for each obtained point the function **`drvChannelCalibrate.`** When no additional processing is necessary, the driver must not export this entry point.
AData – the double data which the unichrom had stored internally. Typically it is the int which comes from driver but converted into double representation.

tcbSetZoneValueEx function

```
function tcbSetZoneValueEx(HInstrument:IHandle;
      AZone,AZoneObject,AParamIndex:integer; AValue:extended):BOOL;stdcall;
```

GC instrument API function. Used for setting zone sensor parameters.
Zone –the zone of GC device which can be controlled e.g. Detector, Injector, Oven, etc. Each zone can have several sensors – temperature, flow, pressure and each sensor can have different functions (carrier gas, makeup gas, fuel gas etc.).

AZone is the index of control zone. Indices have prefix **zXXXX**. E.g. zInjA, zOven, zAux1.
AZoneObject is index of control object in zone. Objects have prefix **zoXXXX**. E.g. zoTemp, zoFlowA, zoPress.
AParamIndex is the index of parameter which is set for selected sensor. Parameter indices have prefix **opXXXX**. E.g. opMin, opMax, opSet, opGas.
AValue – long double parameter of sensor. E.g. flow setting for Inlet 1 carrier gas flow of 10.5 would look like
`tcbSetZoneValueEx(hIns, zInjA, zoFlowA, opSet, 10.5)`.
If the parameter is not suitable for object, the function must return FALSE.

tcbGetZoneValueEx function

```
function tcbGetZoneValueEx(HInstrument:IHandle;
      AZone,AZoneObject,AParamIndex:integer):extended;stdcall;
```

The function is intended for getting object parameter. See the description of **`tcbSetZoneValueEx.`** The function returns long double. If the parameters passed in function are not suitable, the driver must return **`InvalidValue=-1000`**

tcbCheckReadZoneValue function

```
function tcbCheckReadZoneValue(HInstrument:IHandle;
      Zone,AZoneObject,AParamIndex:integer):BOOL;stdcall;
```

The function is used to check the zone for presence or for readability. The driver should return FALSE if the sensor being asked could not be read.

tcbCheckWriteZoneValue function

```
function tcbCheckWriteZoneValue(HInstrument:IHandle;
      AZone,AZoneObject,AParamIndex:integer):BOOL;stdcall;
```

The function is intended for checking the write ability of sensor. The function must return FALSE if sensor could not be written;

tcbSetProgRamp function

```
function TtcbSetProgRamp(HInstrument:IHandle;
      AZone,AZoneObject,ARampIndex:integer;PRec:PRamp):BOOL;stdcall;
```

The function is intended for setting the parameter change ramp. Some of GC instruments allow temperature or flow parameters to be ramped e.g. broken into several time-dependent changes.

PRec – the pointer to a structure, which has the following members.

PRec.ValRate – the change rate of selected sensor till the Plato reached.

PRec.FinalVal – the value of plato section which should be reached after growing parameter with ValRate.

PRec.IsoTime – the period which the plato should continue.

The function must return true if the ramp is accepted. After sending all ramps to driver, the UniChrom calls this entry with ARampIndex equal to –(RampCount) (negative number of ramps). This is used for checking. The function must return TRUE after check is done.

tcbGetProgRamp function

```
function tcbGetProgRamp(HInstrument:IHandle;
      AZone,AZoneObject,ARampIndex:integer;PRec:PRamp):BOOL;stdcall;
```

The function is intended for getting program ramps from instrument.

drvSetState function

```
function drvSetState= (HInstrument:IHandle;AState:integer):BOOL;stdcall;
```

The very important function intended to manage the device state. I.e. the PRERUN, RUN, POSTRUN and IDLE conditions. The **dsXXXX** constants are used as AState. The driver must confirm or decline state changing.

Example: UniChrom makes the instrument go into prerun –

`drvSetState(hIns,dsPrerun)`

drvGetState function

```
function drvGetState(HInstrument:IHandle):integer;stdcall;
```

The function must return the actual state of instrument. States are described by **dsXXXX** constants.

Example: UniChrom checks the postrun conditions before switching to next sample

`if (drvGetState(hins)=dsPostRun) then …` do some actions

fcbSetRecEx function

```
function fcbSetRecEx(HInstrument:IHandle;
      AZone,AZoneObject:integer;PRec:PFlowRec):BOOL;stdcall;
```

Function for setting flow gradient ramps in HPLC system.

The zone model for HPLC system is consisting of the following.

AZone – always 0 for now

AzoneObject – index of program ramp item being loaded 0-based

PRec - pointer to record of the following structure:

```
TPumpRamp=packed record
      Time,Flow,Flow1,Flow2,Flow3,Flow4:double;
end;
```

Where: Time – time in minutes of flow program segment,

Flow – total flow of all pumps in ml/min;

Flow1,Flow2,Flow3,Flow4 – gradient values of flow in parts of 1.

When the PRec is **nil** and AZoneObject is less than 0, that is the check for ramp count. Absolute value of AZoneObject is the total gradient ramp count.

drvUpdate function

```
function drvUpdate(HInstrument:IHandle;AFlag:integer):boolean;stdcall;
```

The function set the method locking. It is very similar to transaction control. Because all method parameters are loaded by UniChrom application step by step, we have to know when the method upload is complete. There is two ways of determining this:

1. Start counting time, when any of the **xxxSet** function called, then wait for 3 seconds and the pass the method to instrument.
2. Increase the method lock count after calling **drvUpdate(dlBeginUpdate)**, and pass the method to instrument after subsequent calls of **drvUpdate(dlEndUpdate)** will decrease the lock count to 0.

Cancel the method upload and clear the clock count after **drvUpdate(dlCancelUpdate)**

The UniChrom does the following actions when loading the method:

drvUpdate(dlBeginUpdate); // increase lock count to 1

drvUpdate(dlBeginUpdate); // lock count is 2

drvSetZoneValueEx(…..)

drvUpdate(dlEndUpdate); // lock count is 1

drvUpdate(dlBeginUpdate); // lock count is 2

drvSetZoneValueEx(…..)

drvUpdate(dlEndUpdate); // lock count is 1

drvUpdate(dlEndUpdate); // lock count is 0 – upload the method to instrument;

drvGetZoneInfo function

```
function drvGetZoneInfo(HInstrument:IHandle;AZone,AZoneObject:integer;lpBuf:PChar;var
cb:Integer):boolean;stdcall;
```

The unction is intended for getting textual information from specified zone. The **cb** parameter should contain the exact buffer size. Data is returned as null-terminated string. UniChrom is calling this entry point for text information, which actions the instrument does. If AZone and AZoneObject both equal to -1, the driver can report the entire instrument state.

smpSetParam function

```
function smpSetParam(HInstrument:IHandle;
      ATower,AParamIndex,AParam:integer):bool;stdcall;
```

The function sets sampling system parameters. The parameters like number of washes in ALS are set here.

Please remember:

Samples are 1-based

Towers are 1-based

The injection source is a critical parameter and it is set by UniChrom before loading the sequence. The following indices are used for injection source selection

```
smpiInjectionSource = 1;

  smpManual = 0;
  smpALS            = 1;
  smpValve   = 2;
```

The object on which the function is execute is determined by ATower parameter. This paramenter is not exactly the tower but the number of injection unit. For gas sampling it may be the valve number. The prefix **twiXXXX** means tower info, **ssXXXX** means sampling system.

```
  twiAny = -1;
  ssAny  = twiAny;
```

The sampling system parameters that can be set by this function are determined by set of **smpiXXXX** (sample parameter index) constants. Each parameter represents one sampling system attribute:

```
// Sampling system parameters
  smpiSolventWash = -4; // number or time of washing
  smpiSamplePump    = -5; // number or time of sample pumping
  smpiInjSpeed             = -6; // speed or time of sample injection
  smpiLoadSpeed            = -7;  // speed or time of sample loading
  smpiDwellTime            = -8; // dwelling time in injector

// HP6890-added parameters
  smpiSolventAPreWash    = smpiSolventWash;
  smpiSolventBPreWash    = -9; //number or time of washing...
  smpiSolventAPostWash   = -10; // ...in two solvents...
  smpiSolventBPostWash   = -11; // ...before and after injection

  smpiSamplePreWash      = -12; // washing in sample before injection??
  //smpiSamplePump       = -5; // number or time of sample pumping
  //smpiInjSpeed         = -6; // speed or time of sample injection (0 or 1 for HP)
  smpiViscosityDelay     = smpiLoadSpeed;
  smpiPreInjectionDwell  = smpiDwellTime; // dwelling times in injector...
  smpiPostInjectionDwell = -13; // before and after injection, in 0.01 mins (0-100)
  smpiSampleSkimDepth    = -14; // skim depth
```

smpGetParam function

```
function smpGetParam(HInstrument:IHandle;ATower,AParamIndex:integer):integer;stdcall;
```

The function is intended for getting sampling system parameters.

ATower – number of sampling subsystem (Tower, Syringe, etc). The parameters of entire sampling system like number of towers are got with **ATower=ssAny**

AParamIndex – sample parameter index has the prefix **smpiXXXX**.

```
// getting parameters of chosen sample
smpGetParam(hInst,1,1,smpiPos) //- get the tray position of 1-st sample in sequence

smpiStatus   = -1;  //the number of installed towers
smpiTowerMax = -2; // the number of positions for tower
smpiTotal            = -3; // total number of samples

// these examples shows getting information on sampling system in total
// // get the number of installed towers
smpGetParam(hInst,twiAny,smpiTotal)
```

```
smpGetParam(hInst,2,smpiStatus) // - get the status of second tower
```

smpSetSample function

```
function smpSetSample(HInstrument:IHandle;
      ASamp,ATower,AVial,ACount,AVolume:integer):boolean;stdcall;
```

The sample sequence is loaded using the smpSetSample function.

Parameters:

HInstrument – instrument handle

ASamp – index of sample being asked (1-based)

ATower – this sample would be injected from ATower (1-based)

AVial – this sample would be injected from vial numbered AVial

ACount – this sample would be injected ACount times.

AVolume of this sample would be injected

After loading the sequence the UniChrom would call

```
smpSetParam(hInst,twiAny,smpiTotal,SequenceLength);
```

This is used to indicate the total number of samples in the sequence. This can be used as trigger for loading the sequence into instrument.

The Injection volume is passed in nanoliters (10^{-9} of liter). E.g. injection of 2 µL would pass to the driver value of 2000.

smpGetSample function

```
function smpGetSample(HInstrument:IHandle;ASamp,AParamIndex:integer):integer;stdcall;
```

The sequence can be loaded form driver using smpGetSample function.

Parameters:

HInstrument – instrument handle

ASamp – index of sample being asked (1-based)

AParamIndex – parameter of sample being asked. The parameter index have prefix **smpiXXXX** (sample parameter index)

This function must reply only to indices listed below:

```
// indices for getting sample parameters
smpiTower  = 0; // sample is injected by tower
smpiPos    = 1; // position of chosen sample in tray
smpiVolume = 2; // injection volume of chosen sample
smpiCount  = 3; // number of injections
```

```
// returns the vial number of selected sample in sequence
smpGetSample(hIns,2,smpiPos);
```

The Injection volume is returned in nanoliters (10^{-9} of liter).

evtSetParam function

```
function evtSetParam(HInstrument:IHandle;
      AEvent,AParamIndex,AParam:integer):bool;stdcall;
```

The function Parameters:

hInstrument – Instrument handle

AEvent – number of event being modified

AParamIndex – index of parameter for event being modified.

AParam – the value of parameter passed to the driver

The function is intended for passing timed event also as valve control event to driver. Timed event table event table is loaded into driver when UniChrom goes into PRERUN state. Look at the event table editor in UniChrom (Spectrum Window / Samples / Additional Parameters).

The event is consists of the following:

- The time when the event must occur. Relative from analysis start. User enters the time in currently selected units. The driver gets the time in milliseconds.
- The valve or the object number. 32-bit Integer value. Number them as you wish, but describe in manual.
- The valve or object state. 32-bit Integer value. Obviously the valves that can ON/OFF have the state of 0 or 1. When the valve is more complex object, the state can be from –MaxInt to +MaxInt;

Parameter indices are prefixed **vpiXXXX** (valve param index).

```
  vpiValve   = 1000;  // number of referenced valve
  vpiType    = 1001;  // type of referenced valve
  vpiInlet   = 1002;  // inlet to which referenced valve is connected
  vpiState   = 1003;  // state of referenced valve
  vpiTime    = 1004;  // time of valve event
  vpiVolume  = 1005;  // loop volume
  vpiLoadTime   = 1006;  // time for blowing sample loop with sample gas
  vpiInjectTime = 1007;  // time for connecting sample loop to inlet
  vpiCount    = -1;
```

Typically the UniChrom behaves in the following way:

```
// the event 0 manages valve numbered AValve
evtSetParam(hInst,0,vpiValve,AValve);
// the event 0 would occur at Atime milliseconds
evtSetParam(hInst,0,vpiTime,Atime);
// the event 0 would change state of Valve AValve to AState
evtSetParam(hInst,0,vpiState,AState);

// Set the total number of events
evtSetParam(hInst,0,vpiCount,ACount);
```

The UniChrom finishes loading of event table by settings the event count. This could be used as trigger for loading the event table into instrument.

evtGetParam function

```
function evtGetParam (HInstrument:IHandle;AEvent,AParamIndex:integer):integer;stdcall;
```

The function is complementary for evtSetParam (see above) and is intended for getting timed event table. Parameters:

hInstrument – instrument

AEvent – number of event being asked (from 0 to MaxInt)

AParamIndex – index of parameter for event being asked.

Typically the UniChrom behaves in the following way:

```
// the event 0 manages valve numbered AValve
AValve := evtGetParam(hInst,0,vpiValve);
// the event 0 would occur at Atime milliseconds
Atime := evtGetParam(hInst,0,vpiTime);
// the event 0 would change state of Valve AValve to AState
AState := evtGetParam(hInst,0,vpiState);

// get the total number of events
ACount := evtGetParam(hInst,0,vpiCount);
```

The UniChrom analyses the on function return. If the returned value is equal to **InvalidZoneValue** then the error occurred.

Sample driver

The following is the description of sample ADC driver for UniChrom written on MS VC.

First we must define exported symbols in “def” file

SampADC.def

```
EXPORTS
     drvOpenEx
     drvClose
     drvOpenInstrument
     drvCloseInstrument
     drvOpenInt32ChannelEx
     drvCloseChannelEx
     drvGetChannelInfo
     drvGetChannelFrequency
     drvSetChannelFrequency
```

and define them in “h” file

SampADC.h

```
#ifdef SAMPADC_EXPORTS
#define SAMPADC_API __declspec(dllexport) __stdcall
#else
#define SAMPADC_API __declspec(dllimport) __stdcall
#endif

BOOL SAMPADC_API drvOpenEx(TDebugWriteStr ADebugFunction,void*
AClient);
BOOL SAMPADC_API drvClose(void);
int SAMPADC_API drvOpenInstrument(int AProfile,TPageNotify
AnAction,void* AClient);
BOOL SAMPADC_API drvCloseInstrument(int HInstrument);
BOOL SAMPADC_API drvOpenInt32ChannelEx(IHandle
HInstrument,TInt32Callback AnAction,void* AClient,int AChannel);
```

```
BOOL SAMPADC_API drvCloseChannelEx(IHandle HInstrument,int
AChannel);
BOOL SAMPADC_API drvGetChannelInfo(IHandle HInstrument,int
AChannel,TDeviceInfo* DevInfo);
float SAMPADC_API drvGetChannelFrequency(IHandle HInstrument,int
AChannel);
float SAMPADC_API drvSetChannelFrequency(IHandle HInstrument,int
AChannel,float freq);
```

Then we have to realize these functions in “cpp” file

Every DLL must have DLLMain function

```
BOOL APIENTRY DllMain( HANDLE hModule,
   DWORD  ul_reason_for_call,
   LPVOID lpReserved
                         )
{
switch (ul_reason_for_call)
     {
          case DLL_PROCESS_ATTACH:
          case DLL_THREAD_ATTACH:
          case DLL_THREAD_DETACH:
          case DLL_PROCESS_DETACH:
               break;
}
return true;
}
```

Then drvOpenEx function accepts two parameters. It must store them for future use as debug info logging callback. Then this function could make some initializing actions that have to be done once.

```
BOOL SAMPADC_API drvOpenEx(TDebugWriteStr ADebugFunction,void*
AClient)
{
     //store ADebufFunction and AClient
     // do some initialization
```

```
    return true;
}
```