Macro Programming Guide

For RC .NET Drivers in OpenLAB CDS ChemStation Edition

2/9/2012
Agilent Technologies, Inc.

Table of Contents

Scope

OpenLAB CDS ChemStation Edition supports the new RapidControl .NET driver architecture (RC .NET), which is the architecture for all future Agilent chromatography devices.

This document describes how to programmatically access RC .NET drivers using the ChemStation macro language.

Getting Started

Introduction to RapidControl .NET

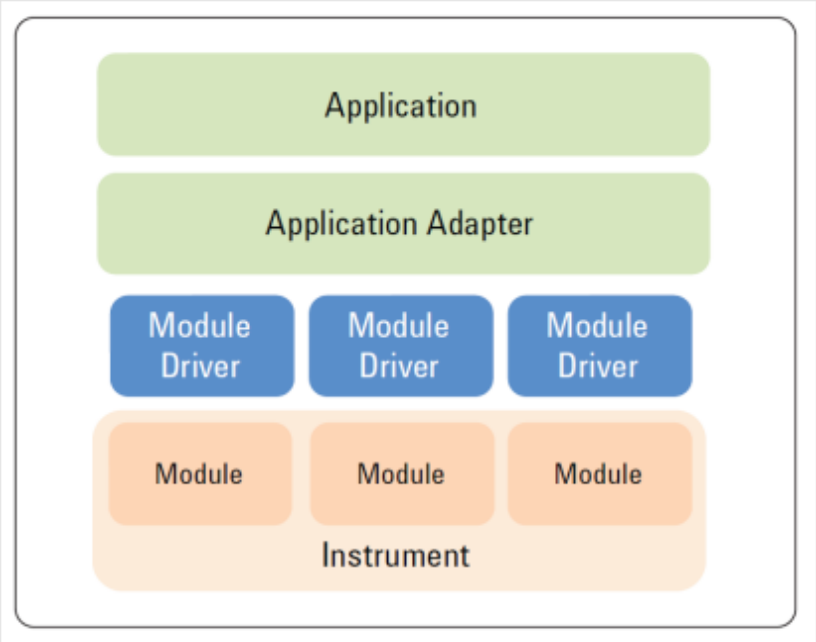
Before RC .NET was introduced, every analytical software platform (e.g. ChemStation, EZChrom, etc.) had to provide their own device drivers to control the same analytical devices (LC/GC or MS). The idea of RC .NET is to have one driver for all analytical software platforms. Customers benefit from faster device support, uniform user interface and higher driver quality across all software platforms.

In contrast to the classic drivers, which are deeply integrated into the ChemStation Data System, the new RC .NET drivers are almost decoupled from the data system.

The Instrument Control Architecture with RC .NET drivers

RC .NET drivers (Module Driver layer) communicate with the data system (Application layer) through a RapidControl Adapter (Application Adapter layer). The RapidControl Adapter is the link between the generic drivers and a data system. It translates RC .Net commands, functions and data structures into things the data systems can understand (e.g. in the ChemStation it is Macro, CP, DOP) and vice versa. The ChemStation CP commands/functions/variables or DOP Registers which belongs to RapidControl drivers are typically prefixed with "RC".

The following picture shows the instrument control architecture for RC .NET drivers.



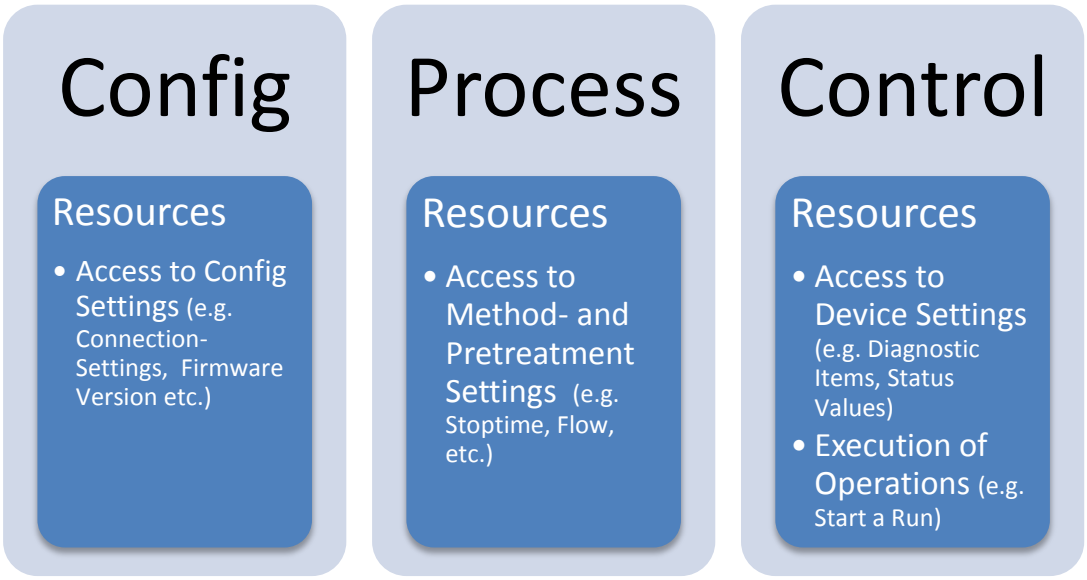
Resource Items - the door to the driver's data

RC .NET provides a functionality called **Resource Items**, which enables RapidControl .NET drivers to expose driver specific data structures and functions to the data systems.

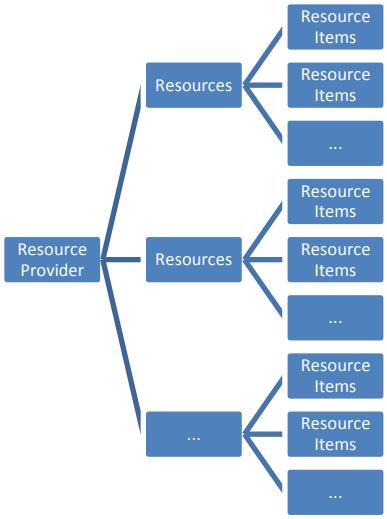
Resource items are optional and driver specific. Agilent developed drivers typically supports resource items. But it is not guaranteed that every driver exposes resource items. Furthermore it is driver specific which items are exposed. For a list of available resource items please see the manufacturer's documentation.

Resource items are accessible in the ChemStation via CP commands and functions. Furthermore resource items are propagated to DOP Registers.

A resource item belongs to a logical element of a RC .Net driver (Config, Process, Control).



The type of a resource item can be a string, number or collection. Resource items are logically grouped by resource providers.



Resource items are accessible via identifiers. In order to access a resource item, the data source (Config, Process, Control), the ID of the resource provider and the ID of the resource item must be specified.

RC .NET vs. Classic Driver Model

RC .NET drivers are available for the most of the existing Agilent chromatography devices. For a small set of devices, however, there only exist classic drivers. New devices will only be supported as RC .NET drivers.

A list of modules and their driver models (RC .NET and/or Classic) can be found on the ***Supported Instruments and Firmware Guide*** at the ***OpenLAB CDS Installation Disk*** (Disk1: \Docs\ENU\CDS_SupportedInstFirmware.pdf).

Please note that RC .NET drivers and Classic Drivers generally cannot run in the same ChemStation instrument. Combinations of modules in one instrument stack need to have one common driver model.
However, a small set of Classic drivers, they are called Classic Plus drivers, work in combination with both driver models (e.g. ADC, MS).

Summary of changes

A side effect to the independent and generic architecture is that some driver specific CP commands/functions and ChemStation Registers which were available with classic drivers cannot be supported in the same way anymore. But there is replacement of functionality available.

The new RC .NET drivers still provide a register interface to access e.g. method parameters or status information. The major differences are:

- The register names are different from the one used by the classic drivers
- The content of the former module status register has been split into two registers, a status register and a configuration register
- Most of the header items and table names inside a register are different (see description of module registers at the end of this document)
- Some header items have been split into two parts. Example: the **StopTime** is now split into a **StopTime_Time** and a **StopTime_Mode**, where **StopTime_Mode** is either *Set* or *NoLimit*
- The injector program of a sampler is no longer part of the method register but resides now in a new **Pretreatment** register

For compatibility reasons, some of the commonly used commands like ***SendModule$*** are also available for RC .NET drivers, additionally new RC .NET specific commands can be used.

The diagnostic specific commands like **DiagRead<ModuleID>** are no longer available with RC .NET drivers.

Macro programming with RC .NET drivers

How to determine the driver model

The CP variable ***DevRCnet*** can be used to determine whether RC .NET drivers are used or not.

DevRCnet	>= 1	*RC .NET drivers are used (the value indicates the number of modules)*
	0	*RC .NET not used (Classic drivers are used instead see: DevLeoAccess)*

Note: Because classic drivers and RC .NET drivers cannot coexist, *DevLeoAccess* is zero in case of configured RC .NET drivers and vice versa.

Module Variables

Similar to the module variables for the classic drivers (e.g. DevLeoPmp, DevNucDad), there are new CP variables for each module type indicating the number of configured modules of a certain type. A module variable only exists if at least one module of this type is configured.

The module variables are named as follows:
DevRC<Module-Identifier>

Examples:
DevRCDAD for Diode Array Detectors, DevRCPMP for Pumps, DevRCALS for Autosamplers...

The next chapter shows the list of possible module identifiers.

Module Identifier

There are several places in the ChemStation where it is required to address a module uniquely. These are: CP commands or function, CP variables or ChemStation registers (DOP). To identify a RC .NET module in the ChemStation, a unique module identifier is used.

The format of a module identifier is: **<ModuleIdentifier><optional:ModuleNumber>**

Typically, a module identifier has a length of three characters, followed by an optional number which is required if more than one module of the same type is configured. The number is consecutive and starts with 1. For example, the identifiers "PMP" and "PMP1" have the same meaning and address both the first pump in the system. "PMP2" addresses the second pump in the system, "PMP3"the third pump, etc...

The following table shows a list of known module identifiers and the related modules:

Module Identifier	Related Modules
ALS	Agilent Autosampler
WLS	Agilent Wellplate Autosampler
AFC	Agilent Fraction Collectors
DAD	Agilent Diode Array Detectors
VWD	Agilent Variable Wavelength Detectors
MWD	Agilent Multiple Wavelength Detectors
FLD	Agilent Fluorescence Detectors
RID	Agilent Refractive Index Detectors
ELS	Agilent Evaporative Light Scattering Detector
PMP	Agilent Pumps (Binary, Quaternary, Isocratic, Low Flow, Prep)
VLV	Agilent Valves
THM	Agilent Column Thermostat
CCC	Agilent Column Compartment Cluster
PVC	Agilent Pump Valve Cluster
PPC	Agilent Prep. Pump Cluster
FLX	Agilent Flexible Cube
UIB	Agilent Universal Interface Box
CE	Agilent Capillary Electrophoresis Detector
GC	Agilent Gas Chromatograph
CTC	Agilent PAL Sampler
SFC	Aurora SFC and Fusion A5

Due to the fact that RC .NET drivers are decoupled from the data system, the list above is only a subset of all available module identifiers. For drivers which are not deployed with the OpenLAB ChemStation Installation (e.g. 3rd party modules), please contact the manufacturer of the RC .NET driver or use the RCListDevices$() CP function to get a list of configured RC .NET modules and their identifiers.

Sending commands to a module

To send any command to a module and read the response, the *SendModule$* command can also be used for RC .NET drivers. In addition a new command *RCSendDevice$* specific for RC .NET drivers is available.

Syntax
SendModule$ (ModuleID$, Command$)

Description
Sends any command to the specified module, and reads the response. This command can be used for RC.NET drivers as well as for classic drivers.

Parameter

ModuleID$	Identifies the module to which the command will be sent. For valid module identifiers in case of RC.NET drivers see chapter Module Identifier **Note**: To be backwards compatible, the SendModule$ command also accepts the module identifiers used for the classic drivers.
Command$	Any valid command (see command set for each module)

Return value
Reply text from the module

Syntax
RCSendDevice$ (ModuleID$, Command$)

Description
Sends any command to the specified module, and reads the response. This command is only valid for RC.NET drivers and cannot be used for classic drivers.

Parameter

ModuleID$	Identifies the module to which the command will be sent. For valid module identifiers in case of RC.NET drivers see chapter Module Identifier
Command$	Any valid command (see command set for each module)

Return value
Reply text from the module
Example
print RCSendDevice$(wls1,”IDN?”)

Retrieving the instrument configuration

All configured modules of a LC ChemStation are listed in the table **ModuleInfo** in the first object of the **_Config** register. This table holds information about firmware revision and serial number of the configured modules plus a reference to the **Modules** table where further information like product number, module type or a descriptive module name can be found in case of classic drivers.
Modules controlled by a RC .NET driver are also listed in the **ModuleInfo** table with their serial number and firmware revision. Further information, however, must be obtained using new RC .NET specific commands:

Command	Parameter	Description
RCGetDeviceProductID$	ModuleID$ Identifies the module in the context of RC .NET drivers	Returns the product number of the specified module (e.g. G1315C)
RCGetDeviceFullModuleName$		Returns a descriptive name for the specified module (e.g. 1100/1200 Binary Pump)
RCGetDeviceSerialNumber$		Returns the serial number of the specified module
RCGetDeviceFirmwareRevision$		Returns the firmware revision of the specified module

The following macro example shows how to iterate through the <ModuleInfo> table and to list e.g. the product number, serial number, etc. for all RC .NET devices.

```
Name IterateRCNetDevices

    Local Row, NumberOfRows, NameRef, Number
    Local Var$, RCDeviceID$, ProductName$, ModuleName$, SerialNumber$

    NumberOfRows = TabHdrVal(_config[1], "ModuleInfo", "NumberOfRows")
    For Row = 1 to NumberOfRows
       NameRef = TabVal(_config[1], "ModuleInfo", Row, "NameRef")
       Number = TabVal (_config[1], "ModuleInfo", Row, "Number")
       Var$ = TabText$ (_config[1], "Modules", NameRef, "Var")
       If (Var$ = "GCI") then
          RCDeviceID$ = RCDeviceIDByNumber$(Number)
          If (RCDeviceID$ <> "") Then
             ProductName$ = RCGetDeviceProductID$(RCDeviceID$)
             ModuleName$ = RCGetDeviceFullModuleName$(RCDeviceID$)
             SerialNumber$ = RCGetDeviceSerialNumber$(RCDeviceID$)
             FirmwareRev$ = RCGetDeviceFirmwareRevision$(RCDeviceID$)
          Endif
       Endif
    Next Row
Endmacro
```

Access to Method- Status & Configuration Info

Like the existing classic LC module drivers, Method, Status and Configuration information of RC .NET drivers can still be accessed through a register interface. However, the name of the registers and item names inside the registers are different.

Method Parameter Register

The method parameters are stored in the first object as object header items of a register, which is related to the appropriate module. The second object of this register contains internal information about formats, value ranges, etc. Method registers are named as follows:

RC<Module-Identifier><Module-Number>Method

Valid module identifiers are e.g. ***PMP*** for pumps or ***VWD*** for a Variable Wavelength Detector.

Examples:

RCDAD1Method *the method register of first Diode Array Detector*
RCTHM2Method *the method register of second Column Compartment*

In opposite to the ChemStation classic LC drivers, the injector program of a sampler (ALS or WLS) is no longer part of the method register. The Injector program is now located in a new register named RCALS*<n>Pretreatment* (for all auto samplers) or *RCWLS<n>Pretreatment* (for all Wellplate samplers).

Status & Configuration Register

The content of the former Status register for classic drivers is now split in two registers.
The first object of the Status register actual values like actual pressure or flow, not ready conditions, run state, error state, etc. The Configuration register contains information about module type, serial number, hardware capabilities, installed hardware options and other configuration parameters, e.g. installed trays or plates
The second object is for internal use and contains format strings, value ranges, etc.

Status registers are named as follows: ***RC<Module-Identifier><Module-Number>Status***
Config registers are named as follows: ***RC<Module-Identifier><Module-Number>Config***

Download/ upload methods to a module

After modifying a method register, you need to execute a Download command to send the register content to the module.

Syntax
DownloadRCMethod <ModuleID$>

Description
Sends the content of the method register to the module identified by <ModuleID$>

Parameter
ModuleID$ Identifies the module (see chapter Module Identifier)

Return value
None. The variable *RCDownloadError$* holds an error text if the register content could not be successfully transferred to the module. Otherwise the variable contains an empty string.

The command UploadRCMethod can be used to read the method from a module and write it to the module's method register.

Syntax
UploadRCMethod <ModuleID$>

Description
Reads the method from a module and writes the parameters to the associated method register

Parameter
ModuleID$ Identifies the module (see chapter Module Identifier)

Return value
None

Waiting for module events

The commands **ReadEvent<ModuleId>** (e.g. ReadEventLAIs) are fully compatible with RC .NET drivers. In addition there is a new RC .NET specific command available

Syntax
RCReadEvent ModuleID$, CpVar, Event, [EventParameter]

Description
Fills a cp variable with the time a specified instrument event occurs the next time

Parameter

ModuleID$	Identifies the module (see chapter Module Identifier)
CpVar	name of cp variable to get the time in seconds since 1970 when the event occurs
Event	Event string to wait for (e.g. *EE 0815*)
EventParameter	optional parameter of the event. If a parameter is specified the CP variable is not set if the event occurs but its parameter does not match the specified one.

Return value
None

Not supported functionality

The following commands are no longer supported in the context of RC .NET drivers:

- LvlvSetPos (command to set the position of an external valve)
- ReadDiag<ModuleID> (read out diagnostic info from a LC module)
- ControlLAIs
- DownloadPlates

Background Method Editing with RC .NET drivers

Overview

Background method editing is a feature to edit a method in the background without changing the running modules.

The commands are Agilent ChemStation CP commands, which can be used in macro programming. They can be used with the CP-Proxy interface in C# programming.

Load method

Use the following command to get access to a method in background:

DataSource$ = *RCBGEditStart*$(DeivceID$, Mode$, [ModeParam$], [Register$], [MethodConsistencyState$], [MethodConsistencyReport$])

Parameter	Description
DeviceID$	RapidControl Device ID, e. g. PMP1
Mode$	Interprets one of the following strings: DEFAULT loads the default method of the device driver UPLOAD loads the current method from the device LOAD loads the device method from the method folder as specified by ModeParam LOADEX loads the method file or migrates an already existing method
ModeParam$	In case of Mode is LOAD, this specifies the method folder, e. g. C:\Chem32\1\METHODS\TEST.M
Register$	Register name. If this parameter is specified, then the method will be transferred into object 1 of this register. When saving the method later with RCBGEdit, the method will be read from that register.
MethodConsistencyState$	CP string variable name, where the consistency state of the driver will be written. If the method is consistent, then the variable content will be “eConsistent”.
MethodConsistencyReport$	CP string variable name for the inconsistency report.

Result string:
DataSource$ -> Identifier of the method for use with RC .NET commands

Save method

Use the following command to save a (modified) method in background, which you have received by *RCBGEditStart*:

RCBGEditCommit DeivceID$, DataSource$, Mode$, [ModeParam$], [Register$]

Parameters:

Parameter	Description
DeviceID$	RapidControl Device ID, e. g. PMP1
DataSource$	Identifier of the method as retrieved from RCBGEditStart$
Mode$	Interprets one of the following strings: ACT_APPLY transfers the device method to the device (device driver) SAVE saves the device method into the method folder as specified by ModeParam

	DISCARD Undo the changes to the method
ModeParam$	In case of Mode = SAVE, this specifies the method folder, e. g. C:\Chem32\1\METHODS\TEST.M
Register$	Register name. If this parameter is specified, then the method will be transferred into object 1 of this register. When saving the method later with RCBGEdit, the method will be read from that register.

Once you have saved a method, you should follow with a second call of RCBGEditCommit in $DISCARD mode for reset.

Clean-up memory

An instance of ProcessIF has been created and will be maintained for each created background method. This holds the logiocal link between method and DataSource.

You can manipulate, save and discard the method as often as you like. But if you are done with method changes and you do not need the DataSource connection any longer than you must clean-up the connection and the memory allocated with it. Therefore, you need to call:

RCBGEditEnd DeivceID$, DataSource$

Parameters:

Parameter	Description
DeviceID$	RapidControl Device ID, e. g. PMP1
DataSource$	Identifier of the method as retrieved from RCBGEditStart$

Manipulating the method

A loaded method in a register can be edited with the usual DOP commands. This is how I have done it.

Alternatively, you can edit the method with the following RC .NET commands referring the method by its DataSource$:

Parameters used for function / commands (see below):

Parameter	Description
DeviceID$	RapidControl Device ID, e. g. PMP1
DataSource$	Identifier of the method as retrieved from RCBGEditStart$
Provider$	An RC .NET Provider (driver specific, e. g. METHOD)
ResourceID$	An RC .NET Resource Item (driver specific, e. g. FLOW)

Available Functions:

Function	Parameters	Result
RCListDevices$	()	yields a list of configured RC .NET drivers. The single drivers are separated by |
RCGetResourceProviderText$	(DeviceID$, DataSource$, Provider$, ResourceID$)	reads a single Resource Item of type String
RCGetResourceProviderVal	(DeviceID$, DataSource$, Provider$, ResourceID$)	reads a single Resource Item of numeric type
RCGetResourceProvider-Collection	(DeviceID$, DataSource$, Provider$, ResourceID$, [RegisterName$])	Gets a Resource Item of type Collection into a Register. Uses register "RapidControlResources" if no register name is specified.

RCGetResourceProviderType$	(DeviceID$, DataSource$, Provider$, ResourceID$)	retrieves the type of a resource item: Bool, Double, Int, String or Collection

Available commands

Command	Parameters	Result
RCSetResourceProviderText	DeviceID$, DataSource$, Provider$, ResourceID$, Value$	Sets a single Resource Item of type String
RCSetResourceProviderVal	DeviceID$, DataSource$, Provider$, ResourceID$, Value, DataType	Sets a single Resource Item of numeric type
RCSetResourceProvider-Collection	DeviceID$, DataSource$, Provider$, ResourceID$, RegObj	Writes a Resource Item of type Collection from a register back (e. g. the register had been created by RCGetResourceProviderCollection)
RCGetResourceProvider-Content	DeviceID$, DataSource$, Provider$, Register$, [Detailed]	Retrieves the complete set of Resource Items of one provider into a register. If Detailed = 1, then extra information about the resource item will be written into the second object of the registers, e. g. Min, Max, type, etc. Detailed = 0 or 1
RCSetResourceProvider-Content	DeviceID$, DataSource$, Provider$, Register$	Writes the content of a register back to the provider. For example, use RCGetResourceProviderContent, then change values und finally write the changed content back with RCSetResourceProviderContent