HP ChemStation

Macro Programming Guide

HP Part No. G2070-90107

Second edition, 11/95

Printed in Germany

Warranty

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties or merchantability and fitness for a particular purpose.

Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard GmbH
Hewlett-Packard-Strasse 8
D-76337 Waldbronn
Germany

Macro Programming Guide

In This Book

This handbook describes how to work with the commands to customize your ChemStation and make its operation more flexible. It explains programming techniques, and uses frequent examples to show how these techniques work in actual applications.

Contents

1

From Commands to Macros

From Commands to Macros

This chapter describes the purpose and basic structure of a macro and how macros are written using commands.

A Simple Macro: Displaying a Message

Your ChemStation displays information on the **message line** at the bottom of the ChemStation window. You can write messages by typing ChemStation commands into the command line. The ChemStation command line appears at the bottom of the ChemStation Window, below the message line. A cursor is positioned at the beginning of the command line, where you can start typing. The command line can be switched on and off from the System menu or from the Window menu.

If you type: `Print "This is a message!"` on the command line and press ENTER, your ChemStation displays **`This is a message`** in the message line. `Print` is the command used in this example.

Commands are words which the ChemStation understands and acts upon when you type the command and press ENTER. Commands are not case sensitive — you can type upper or lowercase characters. The software that evaluates and executes each command is called the **command processor** (CP).

In the example above, the command processor reads the command line: `Print "This is a message!"` and recognizes Print as a command and displays **`This is a message!`** in the message line.

On the command line you can enter several commands, separating each command by a semicolon (;). You can recall commands you typed earlier using the up arrow and down arrow cursor keys on your keyboard.

The most effective way to enter a number of commands is to group the commands together and give the group a new name. Entering this name executes all the commands in the group automatically. This group of commands is called a **macro**. Macros allow you to customize and automate the operation of your ChemStation.

To learn more about macros we will describe how to write a simple macro. The macro will send a message to the message line.

Writing a Macro

Each macro must start with the command Name and the name of the macro, separated by a space, and must end with the command EndMacro.

The macro to print the message looks like this:

```
Name MyMessage
  Print "This is a message!"
EndMacro
```

For ease of reading, it is recommended that you indent the commands within the bracket of Name, and EndMacro. You can use any text editor to write a macro. In the following examples, it is assumed that Microsoft® Windows Notepad is used.

Saving a Macro

Choose Save As from the File menu of Notepad to save the macro.

A dialog box appears for selecting the directory and entering the file name for this macro. For example, select:

C:\HPCHEM\CORE as directory, and

mymacro.mac as the macro file name.

To save your macro with the same file name choose the Save command from the File menu. This overwrites the previous contents of the file.

When you save the macro it becomes a file on the computer hard disk.

This file must be a text file so that it can be correctly read by the ChemStation. If you use a text editor other than Notepad, you may need to specify that you want to save the file in text format rather than in the default format used by the editor.

A macro file can include more than one macro.

Loading a Macro

To load a macro type: `Macro “mymacro.mac”` on the ChemStation command line. When you press ENTER the macro is loaded into memory.

To access all macro file names, type: `Macro` and press ENTER. You can select the directory and the macro file name. Load the selected macro by choosing OK.

Starting a Macro

To start the macro type the name of the macro: `MyMessage` and press ENTER. The message line displays the message.

To load and start the macro automatically, type:
`Macro "mymacro.mac", go` and press ENTER. If the macro file contains more than one macro, the ChemStation starts the last macro in the file.

Modifying a Macro

Using variables and logic, macros can be made more useful than the individual commands. We will describe how to display the name of the person who is executing the macro by modifying the macro.

Use Notepad and modify the macro as shown below:

```
Name MyMessage
  Parameter name$
  Print "This is a message from ", name$, "!"
EndMacro
```

Save the macro in a file. Now load with `macro "mymacro.mac"` and start the macro by typing `MyMessage "YourName"` and press ENTER. The following text is displayed: **`This is a message from YourName!`**

The command Parameter allows you to give information to the macro for it to use or interpret, name$ in this example. The text name$ is a string variable. We will describe variables later in this chapter.

Executing a Macro During Loading

If you write the name of the macro after the EndMacro command, the ChemStation starts your macro automatically. You must save the macro in a file on the hard disk, otherwise the ChemStation loads the previous version of your macro.

For example:

```
Name MyMessage
  Parameter  name$
  Print  "This is a message from ", name$,  "!"
EndMacro

MyMessage "Your Name"
```

Any commands *not* written within the bracket of the Name and EndMacro commands are started automatically when you load the macro file. This may be useful for initialization purposes.

Complex Macros

You can write more than one macro in the same file. Add a second macro to your file:

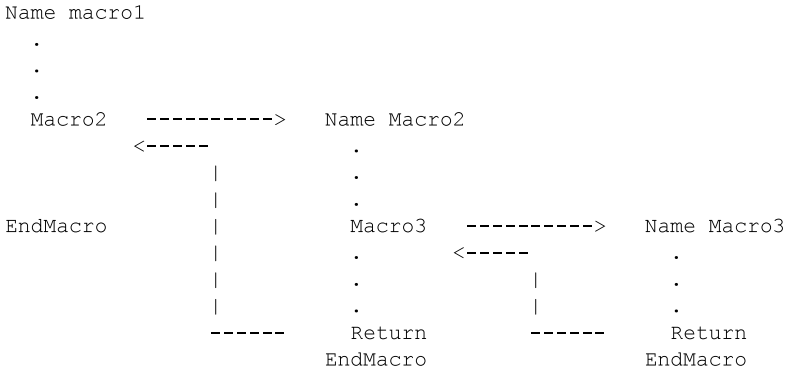
```
Name MyMessage
  Parameter  name$
  Print  "This is a message from" , name$,  "!"
  Return
EndMacro

Name Display_ok
  MyMessage "YourName"
  Sleep 2
  Print "ok!"
EndMacro
```

Save the macro in a file. Load the macros in the computer memory by typing: `Macro “mymacro.mac, go` and press ENTER
`This is a message from YourName!` appears in the message line and is overwritten two seconds later by **`ok!`**.

What happened?

- The ChemStation starts the macro Display_ok because:
  Display_ok is the last macro in the file.
  You typed the keyword *go* in the command line.
- The macro Display_ok starts the macro MyMessage and gives it information (YourName).
- The macro MyMessage displays
  `This is a message from YourName!`.
- The macro MyMessage has a return command which returns execution of command to the macro Display_ok. The macro Display_ok continues with the command Sleep 2, after MyMessage.
- The Sleep 2 command causes a two second pause in the execution of the macro.
- The ChemStation displays **`ok!`**.

The principle of starting one macro, starting a second macro, and returning to the first macro is called **nesting**. You can nest multiple macros, for example:



You can call the same macro using the same name. This is called **recursion**. For more information, see Chapter 5 “Writing Macros: A Customized Report”.

Using Function Macros

You can write macros that behave like functions. This type of macro can be used, for example, to solve mathematical problems. For example, we can write a macro to calculate the area of circles with different radii.

In the macro below, the mathematical operations are repeated for each circle.

```
Name AreaOfCircles
  Print "Area of circle radius 1cm is ", PI()* 1 * 1
  Print "Area of circle radius 4cm is ", PI() * 4 * 4
  Print "Area of circle radius 15.6cm is ", PI()* 15.6 * 15.6
  Print "Area of circle radius 179.5cm is ", PI() * 179.5 * 179.5
EndMacro
```

Instead of repeating the mathematical operation for each circle you can write a function macro to calculate area:

```
Name Area
  Parameter Radius
  Return (PI() * Radius * Radius)
EndMacro
```

The function PI() returns the value of pi. Using the area function macro, the original macro looks like this:

```
Name AreaOfCircles
  Print "Area of circle radius 1cm is ", Area(1)
  Print "Area of circle radius 4cm is ", Area(4)
  Print "Area of circle radius 15.6cm is ", Area(15.6)
  Print "Area of circle radius 179.5cm is ", Area(179.5)
EndMacro
```

Functions substitute the evaluation of the Return statement for the position in the formula or command from which they are called.

For example, the evaluation of the line below, extracted from the example above, would be:

```
Original:  Print "Area of circle radius 4cm is ", Area(4)
Step 1:    Print "Area of circle radius 4cm is ", 50.2654
Step 2:    Area of circle radius 4cm is 50.2654
```

Step 2 represents what appears on the message line.

Function macros can also return strings. Modify the MyMessage macro as follows:

```
Name MyMessage$
  Parameter name$
  Return  "This is a message from"   + name$ + "!"
EndMacro
```

Changing the name of the macro and introducing a Return statement with parameters makes the macro a function. Names of function macros that return strings must end in a $.

You cannot start a function macro in the same way as described in Starting a Macro. You must handle a function macro as a value which can be displayed, evaluated, or assigned to another variable.

To start the example, type:
`M$ = MyMessage$("YourName"); Print M$` in the command line and press ENTER.

This displays the same message but the macro is not called directly. Because this is a function macro you must assign it to a string variable M$. The function has a parameter enclosed in parentheses.

Instead of using a string variable and entering two lines, you can simplify the function macro by typing: `Print MyMessage$("YourName")` and press ENTER.

For more information about function macros and functions in general, see Chapter 5 “Writing Macros: A Customized Report”.

Automating Macros

By automating the loading and running of a macro you can make your ChemStation do operations unattended.

You have different choices for automating your macros:

Loading User Macros Automatically

You can define a macro file called `user.mac` in the ChemStation executable directory. The name of this directory depends on your installation; the default is: C:\HPCHEM\CORE. You can check this by typing: `Print _AutoPath$` on the ChemStation's command line.

The user.mac file in this directory will be automatically executed every time the ChemStation starts. You may enter commands to load and execute your own macros in this file. If, for example, user.mac is written as follows:

```
macro "mymenus.mac"
macro "mymaths.mac"
macro "mycalcs.mac"
LoadMyMenu
```

The macros in files mymenus.mac, mymaths.mac and mycalcs.mac will be automatically loaded and the user-written macro called LoadMyMenu will be automatically executed.

The commands in user.mac are written as direct execution, that is, they are not included within a Name – EndName bracket. This is because the ChemStation loads user.mac with the command: Macro "user.mac" without the *go* parameter.

Running User Macros Automatically in the Method

To make your macro part of a sample analysis, choose Runtime Checklist from the Method menu and enter the name of the macro. Specify the macro as either the prerun, custom data analysis, or postrun macro, and enable its execution by checking the appropriate box at the left. You will find details of the differences between these macros in the online help of the ChemStation.

Remember to save your method after the changes.

Save the macro in the AutoPath$ directory, (C:\HPCHEM\CORE). The macro must already have been loaded, for example, by user.mac.

Post-sequence Macro

After a sequence is run you can shut down the system automatically, turning off the lamp and the pump. You do this through a post-sequence command. Choose Edit Seq.Parameters from Sequence menu and the Sequence Parameters dialog box appears. In the Sequence Parameters dialog box you can enter your own shutdown macro which runs after the sequence is finished.

Macros in Menus

You can design the operation of the ChemStation to suit your own needs by building menus to start macros. The ChemStation includes a set of commands that allow you to build menu systems. For more details, see Chapter 5 “Writing Macros: A Customized Report”.

Loading and Deleting Macros

You load macros into Notepad to edit. You load the macros into the ChemStation to use them. When you load a macro into the ChemStation, it occupies memory which can only be released by removing the macro from the ChemStation memory. You can also remove macro files from the hard disk.

Loading a Macro File into Notepad

Macros are stored in files with the extension .mac on your computer hard disk. A macro file may contain one or more macros. To modify an existing macro file load it into a word processor. We recommend using Notepad.

To load a macro into Notepad, choose Open from the File menu of Notepad and enter *.mac in the file select box. Select the appropriate directory, from the directory list and all the *.mac files are displayed in the file list. Double-click the appropriate macro file to load it.

Save the edited file to a disk before loading the modified version into the ChemStation for testing.

Deleting Macro Files

To delete the macro file from the hard disk use the Windows File Manager or the ChemStation Delete command. Deleting the file from the disk removes your macro files. Always make backups of useful files before you delete them.

Loading a Macro File into the ChemStation

To load a macro file into the ChemStation type: `Macro "mymacro.mac"` on the command line and press ENTER.

To access all macro file names type: `Macro` and press ENTER. A list is displayed allowing you to select the directory and the macro file name. The macro is loaded automatically by choosing OK.

When loaded into memory the macros in the macro file are started by typing their name.

Removing a Macro from Memory

After starting the macro it remains in computer memory until a new macro of the same name is loaded or it is removed from memory by the Remove command. Type: `Remove MyMessage` and press ENTER.

If you have more than one macro in the macro file or you want to remove variables and macros, write a macro which deletes everything, including itself.

```
Name CleanUp
  Remove  MyMessages
  .
  .
  .
  Remove CleanUp
EndMacro
```

Removing macros from computer memory does not remove the macro file from the hard disk.

The macro file name is not necessarily the same as the name of the macro or macros it contains. To start or remove the macros from the ChemStation use the names defined for each macro by the Name command. To load the macro files into the word processor or ChemStation, or to delete them from the disk use the file name with the .mac extension.

Installation Verification for Customized ChemStation Applications

If you did extensive customization to your ChemStation software you can still take advantage of the installation verification program. This program uses a reference file against which your ChemStation files are compared using version information and a checksum for each file. The reference file is shipped with your ChemStation software. To use this program for a customized ChemStation installation you need to create your own reference file.

To create your own reference file for a LC ChemStation execute of the hpveri00.exe program from hpchem\sys directory with the following command line:

```
hpveri00.exe -f custom.ref -n . core sys lc drivers language
```

The files in the subdirectories core, sys, lc and drivers are checked and the result is stored in a reference file named custom.ref.

To check your customized installation against this reference file type:

```
hpveri00.exe -r custom.ref -n . core sys lc drivers language
```

To update IQ icon in the Program Manager to always check using the customized reference file:

1. Select the IQ icon and select Properties from the Program Manager File menu.
2. In the Command Line field, search for the -r parameter, and replace the reference file name with the new reference file name to make the current file list the reference for future installation qualifications.

   For further information, see the online help system of the installation verification program.

2

Using Variables to Get Results

Using Variables to Get Results

This chapter describes how to get results by using variables.

In Chapter 1 “From Commands to Macros” you used the variable Name$ to transfer information to a macro. In another example you assigned the result of a function to the variable M$.

Using String and Numeric Variables

The variables Name$ and M$ are called **string variables**. String variables always end with a $ sign. They contain text and are used for processing user-specific information. For example:

```
MyName$ = "Your Name"
```

Variables having a numeric value are called **numeric variables**. The names of numeric variables do not end with a $ sign. For example:

```
MyPeaks = 3
TooManyPeaks = 4
```

You can assign different values to string and numeric variables at different times:

```
MyPeaks = 3
IHave$ = "toomanypeaks"
MyPeaks = MyPeaks + 1
```

To use variables you do not need to define them specifically — using a string or numeric variable on the left side of the assignment statement defines the variable automatically and the variable exists until you remove it with the Remove command or you leave the ChemStation software. Implicitly generated variables are always global variables.

Using Global Variables

When you have loaded a macro file containing several macros and you assign a value to a variable in one of these macros, you can use the contents of the variable in another macro. These variables are called **global variables**. For example:

```
Name  Initialize
  Twenty = 20
EndMacro

Name  ShowVariable
  Print "Twenty = ", Twenty
EndMacro
```

Write and save these two macros in a macro file and load the macro file in the ChemStation. Start the first macro by typing: `Initialize` and press ENTER. This assigns the value 20 to the numeric variable Twenty.

Start the second macro by typing: `ShowVariable` and press ENTER. The ChemStation displays **`Twenty = 20`** on the message line.

These variables are called global variables because once defined they are recognized throughout the ChemStation software.

If you assign a value or a string to an existing variable, the original content of the variable is overwritten, for example:

```
Name  Initialize
Twenty = 20
EndMacro

Name  ShowVariable
Twenty = 100
Print "Twenty = ", Twenty
EndMacro
```

If you start the macro Initialize and then start ShowVariable, your ChemStation displays **`Twenty = 100`** in the message line. In this example the value in the variable Twenty is redefined by the macro ShowVariable.

Naming Variables

The names of variables:

- can be any length,
- can contain lower or uppercase letters, digits, and the underscore character (_) (no other characters are allowed), and
- must start with a letter or an underscore character (_).

Use initial capital letters in macro or variable names to improve readability.

For example: `PristanePhytaneRatio = Pristane / Phytane`

Uppercase and lowercase letters are not distinguished; for example Index and index refer to the same variable.

Commenting Macros

We recommend you put comments in your macros to remind yourself of the purpose of the macro. Begin a comment line with an exclamation mark (!). Text after the ! on the same line is ignored. For example:

```
! Macro function to return the nearest integer value
! Written by Your Name on 29/01/92

Name Nearest
  Parameter Number default 1 !equals 1 if no parameter given

  Number = Number + 0.5       ! Adjust value so it rounds down
                              ! to the correct result
  Return (floor(Number))      ! Rounds down to nearest integer
EndMacro
```

Using Local Variables

Local variables are variables that exist only within a specific macro. When the macro ends the local variables are removed, avoiding confusion or conflict with variables of similar names in other macros. Local variables may be string or numeric variables and are defined by the Local command within the macro. For example:

```
Name Initialize
  Local  Two
  Two = 2
EndMacro

Name ShowVariable
  Print "Two = ", Two
EndMacro
```

Loading this macro file and starting the macros Initialize and ShowVariable gives the error message **`Undefined symbol Two`**. The variable Two only exists within the macro Initialize. The macro ShowVariable does not recognize the variable Two and cannot print the contents.

If you define local variables and then need to access the contents in another macro, pass the variables to the other macro as a parameter. Variables defined with the Parameter command are also local variables. For example:

```
Name Initialize
  Local  Two
  Two = 2
  ShowVariable Two
EndMacro

Name ShowVariable
  Parameter  Number
  Print "Parameter = ", Number
EndMacro
```

Load this macro file and start the macro Initialize. It assigns a value of 2 to the local variable Two and then starts the macro ShowVariable macro which prints **`Parameter = 2`**.

The following example shows that different variables can have the same name if you declare at least one as a local variable. These variables do not interfere with each other. For example:

```
Name Initialize
  Local Two
  Two = 2                  ! Local Two
  ShowVariable Two
  Print "Two = ", Two
EndMacro

Name  ShowVariable
  Parameter  Number
  Two = 2 + 2              ! Global Two

  Message$ = "Number = "+val$(Number)+" and Two = "+val$(Two)
  Button = Alert (message$, 2)
EndMacro
```

Loading this macro file and starting the macro Initialize displays the message **`Number = 2 and Two = 4`** in the dialog box. The variable Number contains the value of the local variable Two from the Initialize macro. The global variable Two, defined in the ShowVariable macro, contains the value 4.

Choose OK to display **`Two = 2`** in the message line. This shows that the variable Two in the macro Initialize still contains the value 2.

String constructions such as Message$ =… in ShowVariable are described in Chapter 5 “Writing Macros: A Customized Report”.

Using System Variables

When you configured your ChemStation you may have given your instrument a name. For example type: `Print "My name is" , _InstName$` and press ENTER. The ChemStation displays the name of your instrument, for example: **`My name is Instrument1`**.

Instrument1 is the content of _InstName$ which is a string variable and also a **system variable**. System variables begin with an underscore character (_) and usually cannot be changed or removed. You will find a complete list of system variables in Chapter 9 “System Variables”. The following are commonly used system variables:

_DataFile$	Name of current acquisition data file.
_DataPath$	Directory path where current data file is located.
_DataSubDir$	Subdirectory path defined in Sample Info dialog box and is subdirectory where single analysis data files are stored.
_DataSeqSubDir$	Name of sequence subdirectory defined in Sequence Parameters dialog box and is subdirectory where all sequence data is stored.
_MethodOn	Set to 1 if a method is running or set to 0 if an analysis is not running.
_SequenceOn	Set to 1 if a sequence is running or set to 0 if a sequence is not running.
_AutoPath$	Directory path where system macros are located and where macros are loaded from by default.

To look at these variables use the Show command and select the system variable domain. Domain refers to types of ChemStation software definitions such as variables, commands, functions, and macros. If you select a variable name with the mouse or arrow keys, the current content of the variable is displayed.

3

Registers and Objects: An Introduction

Registers and Objects: An Introduction

This chapter describes how more complex data, for example a chromatogram, is stored and accessed by the ChemStation.

Registers

You can access general information using system variables. Because chromatographic and spectral information is too complex to be stored in simple variables, this type of data is stored in data structures called **registers**.

The ChemStation uses different pre-defined registers for different purposes. For example the ChromReg register is used to store the chromatographic data of the signals.

If your macros need to handle complex sets of data, you should create and store the data in registers you create yourself. These are called user-defined registers. You can define as many registers as you need.

A register comprises a contents list, a header, and one or more sections called objects. Each object has summary information followed by detailed information. Registers have names which you must use to access the data in the register. You will find a list of the system registers in Chapter 10 “System Registers”.

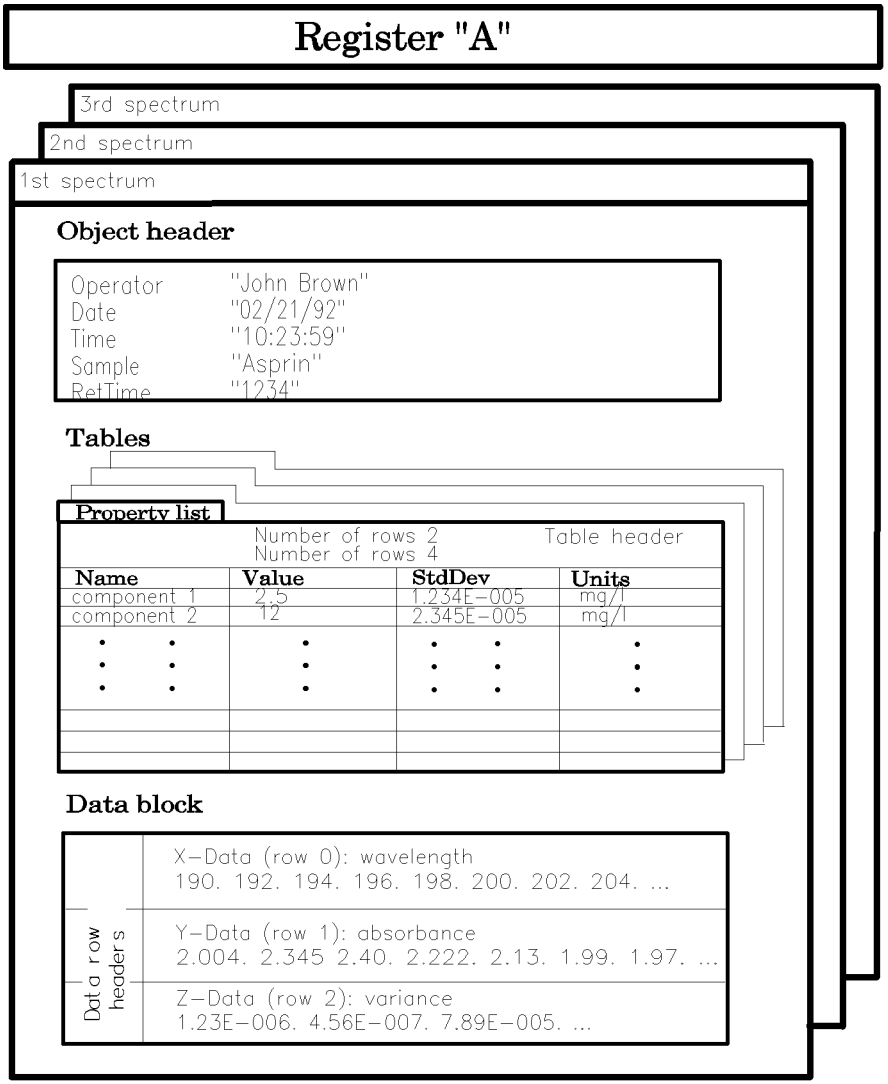
Full technical descriptions of the commands and functions used in this section can be found in the Commands section of the online help.

Figure 1 shows you the structure of a register.

You can use two command functions to examine the contents of a register:

regCont$(myReg	Gives information about the contents of MyReg. The information returned is a string describing the contents of the register.
RegSize(MyReg)	Gives the number of objects in the register MyRegister. Objects are the detailed sections of the register.

Figure 1 **Register Structure**



Objects

The system register for chromatographic data is called ChromReg and the system register for spectral data is called SpecReg.

Objects contain the data you work with. They have an object header that contains global information on the object. Objects can comprise a data block, tables and annotations. Objects exist only in registers and each object is identified by a number. For example, in a register called SpecReg, the third object in this register is called Specreg[3].

The ChemStation uses four different types of objects:

- chromatogram,
- spectrum (HP ^{3D}CE and HPLC^{3D} ChemStation only),
- matrix, and
- user-defined object.

When you load a data file, each chromatographic signal is loaded into a separate object. If you load a data file containing 5 different signals (from different detectors or collected at different wavelengths), the ChromReg register will contain 5 objects. To access the data for a specific signal, specify the number of the object in square brackets after the name of the register. For example, ChromReg[2] accesses the second object in the register ChromReg. If you do not specify the number of an object, the ChemStation assumes you want to access the first object or all objects depending on the command you use.

Object Headers

Each object has a set of **header** items which contain a description of the object. If the object is a chromatogram, there are object headers for the time and date of the injection, the operator name, vial number and more information, see description in Chapter 11 “Standard Objects”.

For example, load a data file by typing: `File “DEMODAD.D”` and press ENTER.

Now load the chromatographic signals into your register by typing: `LoadSignal,,, MyReg` and press ENTER.

You now have a register called MyReg that contains 3 objects. Each object contains one of the three chromatographic signals from the data file DEMODAD.D.

Check this using the command:

```
Print "MyReg register has ", RegSize(MyReg)," Objects."
```

Object header items are identified by a name which conforms to the syntax of the command processor variables. Each object header may be one of the following:

- Numeric, see NewObjHdrVal in the Commands section of the online help.
- String, see NewObjHdrText in the Commands section of the online help.
- Table, see NewTab in the Commands section of the online help.

In the following example you reference the vial number information in the object header using the name Vial:

```
VialNumber = ObjHdrVal(MyReg[1], "vial")
Print "The vial number is", VialNumber
```

Always enclose the name of object header items in quotation marks (" "), otherwise they may be mistaken for the names of numeric variables.

Chapter 10 "System Registers" gives you detailed information about the contents of specific registers and objects, including names of items needed to access this information.

To access the object header as described above you must know what object is loaded. If you do not know the type of information or the item name in the object header, there are 2 object functions which help you find out.

ObjHdrName$ (MyReg, Index) Retrieves the item name from the object. The index must be set to 1 for the first item, two for the second item, and so on.

ObjHdrType (MyReg, Item$) Determines if the object item is a string, number, or table (we describe tables later).

Macros written with loops and conditional statements may be used to list the header information from any object. Loops and conditional statements are explained in Chapter 5 "Writing Macros: A Customized Report".

The following commands print the second item from the first object in the MyReg register.

```
Item$ = ObjHdrName$ (MyReg[1],2)
Type  = ObjHdrType (MyReg[1], Item$)
Print "Item 2 in object 1 is called ", Item$," and is type ",Type
```

Object Data

Object data can represent one chromatographic signal or one spectrum. A data block is the x- and y-data points and additional information to display the x- and y-data points correctly. A data block distinguishes between a chromatogram, a spectrum, and a matrix:

- chromatograms comprise data points for the x- and y-axis,
- spectra comprise data points for the x- and y-axis and — depending on the detector — the variance (square of the standard deviation), and
- matrices comprise data points for the x- and y-axis, and the z-matrix.

Figure 2 shows a data point is defined in a matrix.

Figure 3 shows you the information you get from a matrix as you move along the columns and rows.

User-defined objects can contain data points for the x- and y-axis or they can contain no data points. A user-defined object is the only component which does not require data points — this means — it does not require a data block.

You access data using the data function. The data function extracts individual pieces of information from the data object. For example:

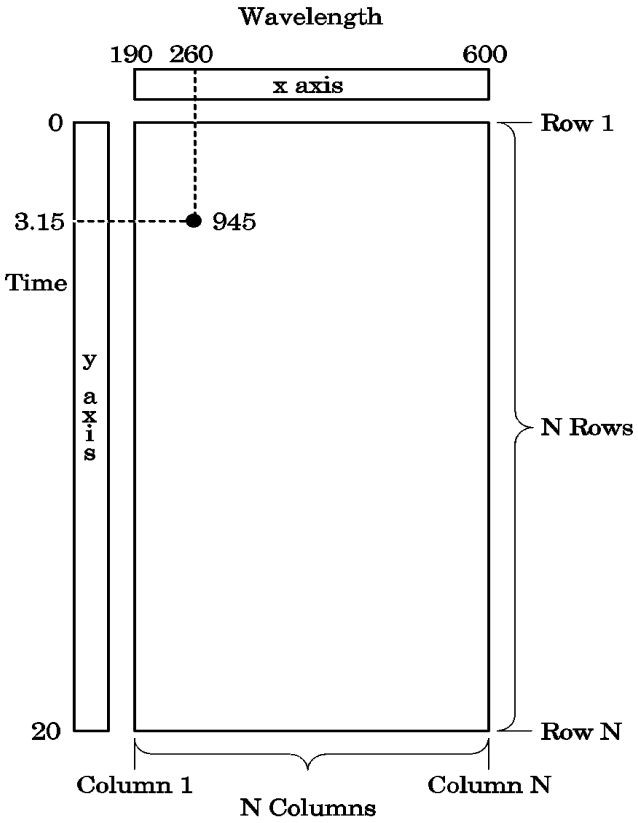
```
TimeOf3rdDataPoint = Data(MyReg[1], 0, 3)
AbsorbanceOf20thDataPoint = Data(MyReg[1], 1, 20)
```

Objects can store two dimensional data. For a chromatogram this is time and absorbance. For a spectrum, this is wavelength and absorbance (assuming that an absorbance detector was used).

To retrieve time or wavelength data (from the x-axis) set the second parameter to 0. To retrieve absorbance data (from the y-axis) set the parameter to 1.

To reference a time value in a chromatographic signal directly (for example, to find the absorbance at 5.0 minutes) use the DataIndex function. This gives the index number to use to retrieve the absorbance data at a given time.

Figure 2 **Data Point Definition in Matrix**



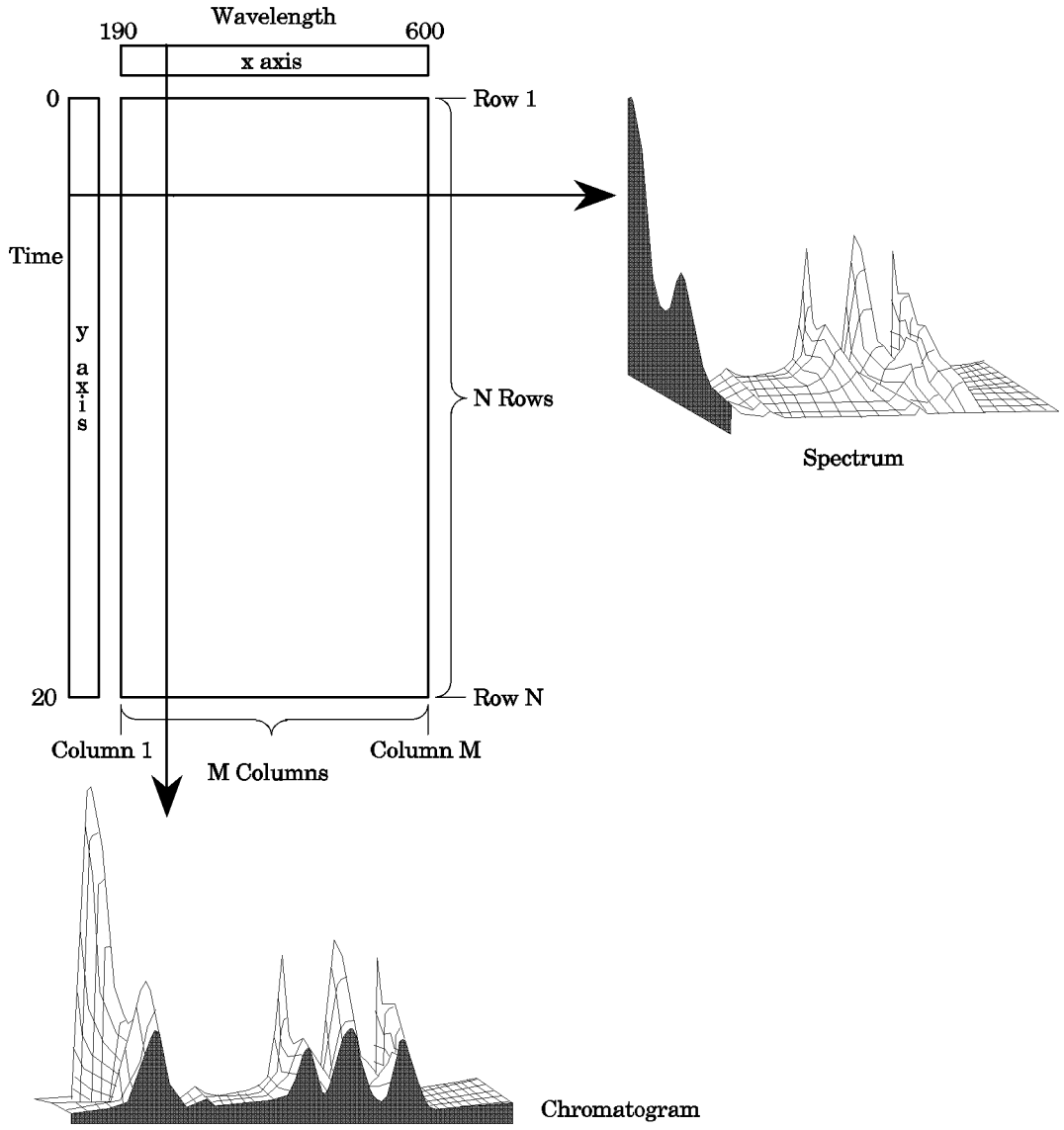

```
Index = DataIndex (MyReg[1], 5.0)  !find the index number near 5.0 minutes
Time = Data (MyReg[1],0,Index) !use the index to get the exact time
Absorbance = Data (MyReg[1],1,Index) !use the index to get the absorbance
```

If you are only interested in the y-axis value, you can combine both functions to obtain the absorbance at 5.0 minutes:

```
Absorbance = Data(MyReg[1], 1, DataIndex(MyReg[1], 5.0))
```

The principle for accessing header or data information is the same for all register objects.

Figure 3 **Information from Matrix**



Registers hold all but the simplest of the ChemStation's data. In addition to the **chromatographic** and **spectral** registers, there are **configuration** and many other registers, see Chapter 10 "System Registers".

You can modify, delete, or extend data and header information using the appropriate commands. See the Commands section of the online help for details of these commands.

User Registers

To handle complex sets of data your ChemStation allows you to define your own registers. These are called user-defined registers. As well as creating user-defined registers you can also edit the data in them. These types of registers are designed to process data sets like the time-absorbance arrays known as chromatograms.

You can store a two dimensional graphical representation in a register object. An example is the data that describes a titration curve:

```
NewObj Titration[1], 1, 1, 100
SetObjHdrText Titration[1], "title", "Titration results"
```

The command NewObj creates a user-defined register called Titration, with one object, which can hold up to 2 rows (0 and 1) of data each capable of holding up to 100 data points. The title is set according to the content of the register object.

You fill the register objects using the SetData command. The parameters of SetData specify the row, index, and value to be stored. Use the mathematical convention that has the independent variable on the x-axis (in a titration the amount of reagent) and the dependent variable (the pH value) on the y-axis.

To put the first pair of data in the register, reagent = 0.0 ml, pH = 3.5, enter:

```
SetData Titration[1], 0, 1, 0.0
SetData Titration[1], 1, 1, 3.5
```

Titration[1] defines the first object in the register Titration. The second parameter defines the x-axis (0) or the y-axis (1). The third parameter specifies the point number. The fourth parameter specifies the data value.

If adding 1 ml of reagent changes the pH value to 3.7, the second point is set as:

```
SetData Titration[1], 0, 2, 1.0
SetData Titration[1], 1, 2, 3.7
```

You can continue to enter data until the data set is complete. To save time you can enter the data automatically using a macro. Enter data automatically using loop and logic commands which are described in Chapter 5 “Writing

Macros: A Customized Report". When you have entered the data, you can draw, compare, plot, or annotate it.

Tables: An Introduction

In the previous sections, we described registers that contain objects comprising descriptive elements, object headers, and data blocks.

A register contains a set of objects. An object contains:

- header items (numbers, strings, and tables),
- data (chromatographic, spectral, or user-defined functional data), and
- annotations (described later in this section).

Object headers may contain numeric, string, and table data types. Tables differ from the objects data block in that they can be used to store dissimilar data types. For example, a mixture of strings and numbers are required to store a list of compound names and amounts.

A table is contained in an object. An object can contain more than one table. A table is typically used to hold information that relates to the object in which it is contained.

The table comprises a series of columns and rows. For example, a table could be a chromatogram or electropherogram. The rows would represent retention/migration time and the columns would represent the absorbance. Another example could be a collection a methods for different samples. The rows would represent the methods and the columns would represent the samples.

A table comprises table header items and table elements. Each table element is identified by a row number and a column name. The column names and the table header names conform to the syntax of the numeric command processor variables. Each column of a table may be a string table element or numeric table element.

String Table Elements

String table elements are text table elements with:

- an arbitrary length, up to 65535 characters, or
- a pre-defined maximum length.

See NewColText in the Commands section of the online help.

Numeric Table Elements

Numeric table elements are:

- rational numbers, including fractions, or
- integral number, *not* including fractions.

See NewColVal in the Commands section of the online help.

All table elements in the same column have the same type, which is independent of row number.

A row is a collection of table elements. Each table element may have a different type depending on how the table columns were defined.

You can customize operation of the ChemStation by changing parameters in the columns and rows of the table. Using loops and identifiers, you can link and access different tables.

Table Header Items

The table header contains administrative information which is valid for all cells in the table, for example the table dimension, method name and date. Table header items have the same type possibilities as table elements, these are:

- arbitrary length text,
- maximum length text,
- rational number, and
- integral number.

Annotations

Annotations are used to display information graphically. An annotation always comprises a position given by x- and y-coordinates, an angle, and a size. See the Commands section of the online help.

An annotation may be one of the following:

- text annotation — for retention/migration times or compound names,
- marker annotation — to mark levels of a calibration curve graphically, or
- polyline annotation — to construct baselines.

Table Dimensions and Change Information

The dimensions of a table, measured in the number of rows and columns, can be accessed by way of the header items NumberOfRows and NumberOfCol, for example:

```
NumberOfIntegratedPeaks \
= TabHdrVal(ChromReg[1], "IntResults", "NumberOfRows")
NumberOfCalibratedCompounds = \
TabHdrVal(_DAMethod, "Compound", "NumberOfRows")
```

In addition the header item Modified shows whether the content of the table has been changed, and the header item NumberOfHead shows the number of header items in the table.

System Tables

Here is an example of a table in which quantitative results are stored using a calibrated method. To prepare the results, first load a calibrated method, then choose Load Signal and select a data file that fits the loaded method. When the signals are displayed, open the Signal Options dialog box and make sure that the Compound Names option is checked. The main commands executed in this operation are:

File

LoadSignal

IntegrateObj

IdentifyPeaks

QuantifyPeaks

These commands load a chromatogram, integrate it, and then identify and quantify the peaks. When you load the data file it is stored in the ChemStation's system register ChromReg. Each signal is stored in a separate register object.

When each signal is integrated the integration results are stored in a table called IntResults. One IntResults table is created for each signal and the table is stored in the same object as the corresponding signal.

When you quantify the peaks the integration tables in ChromREG are identified and quantified. The resulting quantification lists are stored as tables in the ChromRES register. ChromRes holds the compound and quantification data for the entire analysis in the first object.

The main tables in the ChromRes register are:

Compound	Contains all the calibrated compounds. Holds parameters such as amount and name of each compound.
Peak	Contains all the peaks from the chromatogram. For each peak there is information including its area and retention/migration time.
Signal	Describes the signals in the chromatogram.

You access the numeric values in a table using the TabVal function. You access string values using the TabText$ function. The parameters for these functions specify the register, the object number, the name of the table, the row number, and the column name.

For example, to obtain the name of the 5th compound from the quantification results:

```
Name$ = TabText$(ChromRes, "compound", 5, "name")
```

Remember to enclose the names of tables, table header items, and columns in quotation marks (“ ”), so that they are not mistaken for the names of numeric variables.

To obtain the quantified amount for the compound use a different column name to access the same table.

```
Amount = TabVal(ChromRes, "compound", 5, "amount")
```

To access the compound’s retention/migration time, which is held in the separate peak table, you cannot use reference index 5 because it is likely that there are more identified peaks than calibrated compounds. You cannot assume that the two tables correspond with each other.

There is a special column in the Compound table to provide a link to the right entry in the Peak table. This link is given in the FirstPeak column:

```
FirstPeak = TabVal(ChromRes, "compound", 5, "FirstPeak")
```

When you have the peak index, you can use it to obtain the retention/migration time data using the MeasRetTime item:

```
RT = TabVal(ChromRes, "peak", FirstPeak, "MeasRetTime")
```

This can be written as one expression:

```
RT = TabVal(chromRes, "peak",\
TabVal(chromRes, "compound", 5, "FirstPeak"), "MeasRetTime")
```

Or shortened using the special link operator (tilde: ~):

```
RT = TabVal(chromRes, "compound", 5, "FirstPeak~MeasRetTime")
```

The tables we have described do not have headers. The tables holding the quantification parameters (as opposed to the results) do have headers. The _DAMethod register contains the current data analysis method loaded in the

ChemStation, including the quantification parameters. The quantification table is held in the first object of the _DAMethod register.

You access numeric values in table header information using the TabHdrVal function. You access string values using the TabHdrText$ function. For example, to obtain the calibrated units:

```
Unit$ = TabHdrText$(_DAMethod, "QuantParm", "units")
```

You will find a detailed description of system tables in Chapter 12 “System Tables”.

User Tables

You can define tables or modify and extend existing tables. You will find details of the table commands and functions in the Commands section of the online help.

As an example we will describe the steps to set up a table which organizes the backup of your data files on disks or tapes. The table is designed with the following items:

Table name: DataFiles

Register name: Backup

Object number: 1

Table 1 **Table Header Items**

Item Identifier	Type	Description
LastChangedBy	Text	Operator name who made last change
LastChangeTime	Text	Date and time of last change

Table 2 **Column Items**

Item Identifier	Type	Description
Name	Text	Name of the data file
Medium	Text	Medium where the files are stored
Directory	Text	Directory where to find the files
Date	Text	Time and date of storage
Description	Text	Further information about the data file

This example describes the first steps to making a backup system. For clarity, the steps below are described as individual commands. You can integrate this

example in an automated backup procedure based on the Copy command during a series of analyses.

Creating a Table

Create the table DataFiles in the BackUp Register:

```
NewTab BackUp, "DataFiles"
```

Defining the Header

Define the header items:

```
NewTabHdrText BackUp, "DataFiles", "LastChangeBy"
NewTabHdrText BackUp, "DataFiles", "LastChangeTime"
```

Defining the Columns

Define the table columns:

```
NewColText BackUp, "DataFiles", "Name"
NewColText BackUp, "DataFiles", "Medium"
NewColText BackUp, "DataFiles", "Directory"
NewColText BackUp, "DataFiles", "Date"
NewColText BackUp, "DataFiles", "Description"
```

The table definition is now complete.

Saving the Table

Save the definition to a file:

```
SaveObj BackUp, _DataPath$+ "backup.reg", new
```

The parameter *new* is necessary to save the table as a new file. An already existing file with the same name would be overwritten. If the *new* parameter is not given the table would be appended to an already existing file as new object.

Loading the Table

To archive data and store the new information in your table, load the table into memory:

```
DelReg BackUp  !delete old version of that register in memory
LoadObj _DataPath$+"backup.reg",,Backup
```

Updating the Table

When you backup an archived data file, you can update the table header with the new information:

```
SetTabHdrText BackUp, "DataFiles", "LastChangeBy", "YourName"
SetTabHdrText BackUp, "DataFiles", "LastChangeTime", Time$()
```

Extending the Table

You enter the details of the file as a new row in the table:

```
InsTabRow  BackUp, "DataFiles"
Row = TabHdrVal(Backup, "DataFiles", "NumberOfRows”)
SetTabText BackUp, "DataFiles", "Row" "Name", _DataFile$
SetTabText BackUp, "DataFiles", "Row" "Medium", "Disk"
SetTabText BackUp, "DataFiles", "Row" "Directory", "_DataPath$"
SetTabText BackUp, "DataFiles", "Row" "Date", Date$(+Time$)
SetTabText BackUp, "DataFiles", "Row" "Description", \
                              ObjHdrText (ChromRes, "SampleInfo")
```

Save the table to a disk using the SaveObj command.

Searching a Table

To search in the table for the file name of a particular sample, load the table from the disk and search for the sample by using its description:

```
DelReg Backup
LoadObj _DataPath$+ "backup.reg",, BackUp

Row = RowByText(BackUp,"DataFiles",,"Description","=","water soluble vitamins")
Print "File is ", TabText$(BackUp, "DataFiles", Row, "name")
```

The list entry that contains the description “water soluble vitamins” is found by the `RowByText` function. The last line will print the file name of that entry, extracted from the table with the `TabText$` function, on the ChemStation’s message line.

Type and Range Terminology

The types and ranges of the predefined object header items, table header items and table columns describe the internal format — and any restrictions — of the ChemStation system registers. The following terminology is used for the types and ranges.

Type/Range	Description
string	Text string of unspecified and arbitrary length.
string path name	Text string — file or directory name, including full path name.
string macro	Text string — macro name, conforming to macro name syntax.
string n characters	Text string with a maximum of n characters.
string date	String — date and time of day.
string reg	String — legal register name, conforming to register name syntax.
string regobj	String — legal register name and object number, conforming to regobj specification syntax.
string regobjrange	String — legal register name and object number or object number range, conforming to regobjrange specification syntax.
numeric	Number of unspecified range and precision.
integer	Integral number of unspecified range.
boolean	Integral number that can have only the values 0 (for false, off, or no) or 1 (for true, on, or yes). When displaying such a value, it could be converted into a string for a combination list. Translation into local language is then possible.
enumeration	Integral number that can have only predefined values. When displaying such a value, it could be converted into a string for a combination list. Translation into local language is then possible.

bitfield n

Integral number that consists of a OR combination of n individual bits. Each of the individual bits is explained and given with its numeric value.

rgb

Integral number that is an RGB (Red, Green, Blue) color value. Allowed values are from 0 to $2^{24}-1$. Example: Red = 255 = 0xFF

link

Integral number that is an index into another table in the same object. The links are corrected automatically after all operations which change the order of rows in the related tables. For any such link there must exist a string table header item with the same name as the link but with a "_" appended to the name. This table header item contains the name of the table that the link refers to.

There are several peculiarities when dealing with links:

The allowed range of a link changes dynamically. It is always either –1 (a link pointing to nowhere) or from 1 to the number of rows of the table into which the link is pointing.

A link changes its value automatically if any of the operations insert, delete, copy, move, sort, exchange is done on the table into which the link is pointing.

A link allows an indirect access to the elements of another table. This is done with the "~" operator in any of these commands/functions: SetTabHdrVal, SetTabHdrText, TabHdrVal, TabHdrText$, SetTabVal, SetTabText, TabVal, TabText$, RowByText, RowByVal.

Links generate structures between tables, for example, between Compound and Peaks of calibration data. These structures can be manipulated through commands and therefore can be destroyed through commands. This may make calibration or result data invalid and inaccessible for the special chromatography commands.

Tables that contain links may not be copied or deleted on their own, only through copying/deleting the

complete object.

time since 1970 Integral number that is the number of seconds elapsed since 1 Jan 1970 00:00:00.

The CP environment does not distinguish between numeric data types. The types Numeric, Integer, Boolean, Enumeration, Bitfield, RGB, and Link correspond to the same CP variable type. When writing into the tables a conversion to internal type (Numeric, Integer, Boolean, or Link) is done if possible.

4

Permanent Data

Permanent Data

This chapter describes how the ChemStation saves data permanently in files on the computer hard disk.

Files

In “User Tables” on page 55 we described how to load and save register objects. The LoadObj and SaveObj commands load the complete register from a disk, or save the complete register to a disk. The data saved on a disk is called **permanent** or **non-volatile** data, because the data is not lost when you turn off the computer. Data held in computer memory is called **temporary** or **volatile** data, because the data is lost when you turn off the computer.

Remember to save important data (for example, a method) every time you change.

Saving a register is one example of saving data permanently. Register data is saved to a file with the extension .REG. Register files have a special structure understood by the ChemStation.

File Access Commands

The Open and Close commands allow you to access ASCII text files on the disk. When you have opened a file you can read information from it using the Input command. You can write information to an open file using the Print command in the same way as you wrote information to the message line in Chapter 1 “From Commands to Macros”.

For example, to save the area of a peak to a file:

```
Open "areadata.txt" for output as #3
Print #3, TabVal(ChromRes, "peak", 1, "area")
Close #3
```

File Identifiers

Using **file identifiers** with the Input and Print commands, you can have several files open at the same time. You use the file identifier to specify which file you want to read from or write to.

```
Open "areadata.txt" for input as #3
Open "copydata.txt" for output as #4
Input #3, PeakArea
Print #4, PeakArea
Close #3
Close #4
```

File Extensions

Files have a three character extension that describes the type of file. For example, files with the extension .EXE are executable programs and files with .TXT contain text that you can read using an ASCII text editor such as Notepad.

Open the file areadata.txt to see what it contains.

To read the reference area that has been written to areadata.txt:

```
Open "areadata.txt" for input as #3
Input #3, PeakArea
Close #3
Print "The area in the file is ", PeakArea
```

Data Format

To load a set of ChemStation data into another program, you may have to write the data file in a format required by the other program.

The Print Using command allows you to specify a format for the data you print with the command. For example:

```
Number = 12.897761
Print #1 using "#####.##", number
```

This prints the value of 12.90 in the variable Number. For further details, see the Commands section of the online help.

5

Writing Macros: A Customized Report

Writing Macros: A Customized Report

As well as grouping ChemStation commands to form a macro, you can also use decision-making logic. This chapter describes the logic that is available and how you can use it in the design of a macro application.

We introduce the concepts behind decision-making logic using a practical example which involves producing a customized report. We have chosen this example because the objective of the macro is easy to understand and it enables us to introduce the more complex concepts gradually and logically.

The ChemStation includes a report layout program which has many more capabilities than the example described here. If you want to design your own reports, we *strongly* recommend you use the ChemStation report designer. You will find a description of the report designer in the online help.

You must have created a calibration table in the data analysis part of your ChemStation software for the data files which you will use for the example macros in this chapter.

Turning the Idea into a Report

Always start your design by writing down your ideas on paper. This should be a description of what you want your macro to do.

In this example, the objective is to write a macro that will print the following report:

```
                Data file name: result1.d

#   Ret.Time   Concentration   %MaxAmount
        [min]       [ng/ul]
---------------------------------------
 1      1.23         143.56         4.12
 2      2.35         189.45         5.44
 3 *    3.89        3482.29       100.00
 4      6.21           2.95         0.08
 5      6.73          37.21         1.07
```

The fourth column of this report shows the amount of each peak expressed as a percentage relative to the largest peak in the chromatogram.

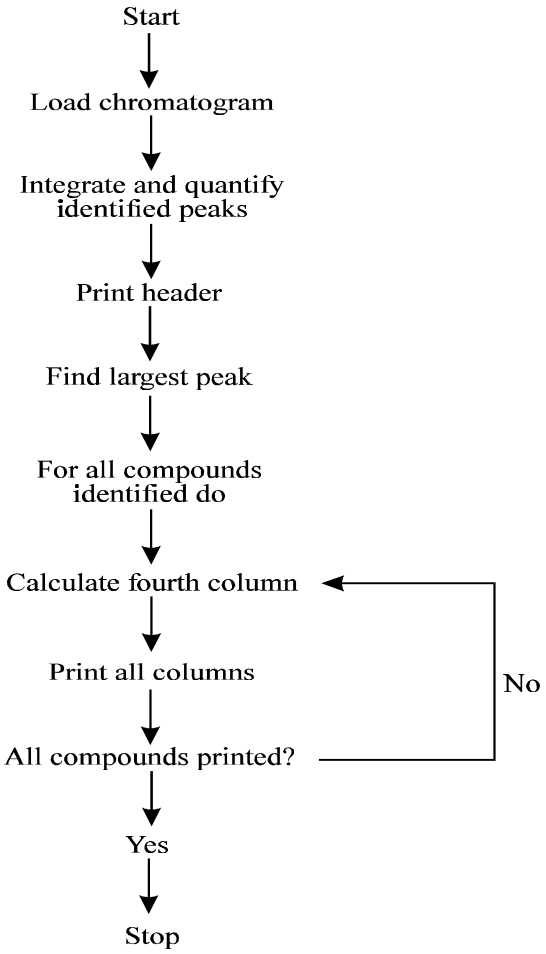
Defining the Structure of the Macro

To define the structure of the macro you must identify all the steps in the generation of the report.

In this example the starting point is a data file and a suitable method for quantifying the data. You must now consider the next steps which eventually lead to the finished report. Some of these steps are simple and simulate the operation of ChemStation software. For example, loading, integrating, and quantifying the data. Other steps are more complex — such as tabulating each peak as a percentage of the largest peak in the chromatogram. To do this you must determine the largest peak before printing any data.

A good way to visualize the steps in the generation of the report is to construct a flow diagram.

Figure 4 **Flow Diagram**



Defining the Contents of the Macro

Turn the flow diagram you drew in “Defining the Structure of the Macro” on page 67 into a list and substitute the task descriptions with command name.

```
Name RelativeAmounts
  LoadChromatogram
  PrepareResults
  PrintHeader
  FindBiggestPeak
  for all Compounds do
    Calculate4thColumn
    PrintRow
EndMacro
```

The *Name* command defines the start and the EndMacro command defines the end of the macro.

You now have a list of tasks that you have to do. These tasks represent individual parts of the overall project. What is important about them is that they are small independent blocks that you can write individually as sub-projects.

When you are designing and writing macros it is good practice to use simple blocks in creating a complex macro. This approach helps you to concentrate on small manageable pieces and makes it easier for you to understand changes to the macro.

Go to Notepad and type in the macro according to your definition.

Style Guidelines for Writing Macros

You can write a macro by typing the commands one after another, but you may have problems later understanding what the macro is doing. We recommend you structure your macro like a book with chapters and sections. You can also add comments to the macro, describing what each part of the macro is doing.

“Using Variables to Get Results” on page 29 describes how to put comments in macros. Remember to begin a comment with an exclamation point command (!). The command processor ignores any text after an exclamation point.

Your comments should help you identify what each part of the macro is doing. Compare the following examples.

```
a = 12
b = 3.141592
print b * a * a

! Print the area of the circle
    Radius = 12  ! Formula = PI * R^2
    Print PI() * Radius * Radius
```

When you are writing macros we recommend that you:

- group sections of the macro which belong together and separate the sections from each other by adding empty lines,
- indent sections which are part of a higher structure,
- write short commands on one line separating them with a semicolon (;), and
- name the macros by using a prefix which characterizes these macros according to the user or task, for example, USR_.

You can make complex commands more readable by writing them on several lines. You can write a single command over more than one line by ending each incomplete line with a backslash (\).

```
Return (TabVal(chromRes[1], compound, CurrentIndex, amount) * \
    100 / BigAmount)
```

Using these guidelines, modify the first draft of your macro as follows:

```
Name USR_RelativeAmounts

! Prints a report with amounts relative to the biggest peak
! Last modified by Your Name at 28-May-92

  USR_LoadChromatogram      ! into register
  USR_PrepareResults        ! integration, identification
                             ! and quantification of report
  USR_PreparePrinter        ! for output
  USR_FindBiggestPeak       ! in chromatogram

  For all compounds do  ! print report columns

    USR_Calculate4thColumn
    USR_PrintRow         ! print each row of the table

EndMacro ! RelativeAmounts
```

You can use macros as subroutines, each macro calling another macro. You can repeat this as many times as you need as described in Chapter 1 “From Commands to Macros”. Remember that each macro called must have a Return command to get back to the calling macro.

When using macros as subroutines we recommend you record the purpose of the macro as comments at the beginning so it is clear what the macro does. It is also important to record the parameters and the variables that the macro uses.

In the report macro example the statements LoadChromatogram and PrepareResults are macro subroutines. LoadChromatogram has commands to load the chromatogram so that the next step, PrepareResults, can be done.

Using Mathematical Operations and Functions

The numeric operators + (addition), – (subtraction), * (multiplication), and / (division) do the appropriate calculations. Your ChemStation evaluates mathematical expressions from left to right. The multiplication and division operators take priority over addition and subtraction, but parentheses — (and) — override this priority. All the operators are governed by standard algebraic rules.

Your ChemStation has special functions available for operations like square root or logarithm. For a complete list of these functions, see Chapter 14 “Summary of Commands”.

For example, this formula gives the standard deviation of a set of data:

$$\sigma = \sqrt{\frac{\sum (x - jx)^2}{n - 1}}$$

which would be changed into a form that your ChemStation can understand, assuming you have already calculated the square root of the sum of the square of the deviations $\sum (x - jx)^2$ (SumOfDeviations^2) and the number of data points n, as follows:

```
Sigma = sqrt(SumOfDeviations^2/(n - 1))
```

To calculate the amount of each peak relative to the amount of the largest peak, you can use the RowByVal function to determine the index of the largest peak, see the Commands section of the online help for more information.

The statement Calculate4thColumn is a separate macro:

```
Name USR_Calculate4thColumn

! Calculate the relative amount in the 4th report column
  Parameter CurrentIndex   ! index of compound in table

  Local BigPeak  ! index of biggest peak
  Local BigAmount  ! amount of biggest peak

  BigPeak = RowByVal(chromRes[1], "compound",, "amount", "max")
```

```
  BigAmount = TabVal(chromRes[1], "compound", BigPeak, "amount")

  Amount = TabVal(chromRes[1], "compound", CurrentIndex, "amount")
  AmountPercent = Amount * 100 / BigAmount
  Return

EndMacro ! USR_Calculate4thColumn
```

The variables BigAmount and BigIndex are local variables. The variables Amount and AmountPercent are used for printing and are global variables because they are also required in the PrintRow macro used by the RelativeAm the main macro, make this calculation a function that can be used directly by the PrintRow macro. You then do not need the macro Calculate4thColumn, the global variable AmountPercent, or the FindBiggestPeak macro because this can be found with the RowByVal function with the keyword Max.

```
Name USR_AmountPercent

! Function to calculate the relative amount in the 4th report column

Parameter CompoundNum   ! Index of compound in Compound table

  Local BigPeak         ! Index of biggest peak

  Local BigAmount       ! Amount of biggest peak

  Local Amount          ! Amount of current compound


  BigPeak = RowByVal(ChromRes[1], "Compound",, "Amount", "Max")

  BigAmount = TabVal(ChromRes[1], "Compound", BigPeak, "Amount")

  Amount = TabVal(ChromRes[1], "Compound", CompoundNum, "Amount")

  Return (Amount * 100 / BigAmount)


EndMacro ! USR_AmountPercent
```

Putting Text in the Macro

You can put text in a macro as comments or as printed messages and change or process text as ChemStation variables.

Assigning Text to String Variables

You assign text to a string variable as follows:

```
MyCompanyIs$ = "Hewlett-Packard Company"
```

The next example shows you how to use quotes within quotes when you want quotation marks to appear in the text string.

```
Instruction$ = "Print ""This is my message!"""
```

If you now type `Print instruction$` and press ENTER, the message line will display **`Print "This is my message!"`**

This string contains a command and you can evaluate the string directly using the Evaluate command. The Evaluate command executes the content of a string. In this example the Evaluate command executes the Print command:

Type `Evaluate instruction$` and press ENTER to display
`This is my message!`

Combining String Variables

You can combine string variables using the addition (+) operator. The most common use of this in the ChemStation software is the combining of system variables. For example to retrieve the complete file specification from the path components:

```
Print "File name = ", _DataPath$ + _DataFile$
```

Using String Functions and Indexing

The ChemStation has special string functions allowing you to manipulate the strings. For a complete list of these functions, see Chapter 14 “Summary of Commands”.

When you combine string functions you can create some powerful functions. A common string operation is to search and extract a substring from a larger string. For example, a line in a file contains the name of a compound, its retention/migration time and amount, separated by commas. This line is assigned to a variable called Line$. If you want to extract the compound name, use the following commands:

```
Line$ = "Phenol, 1.23, 10.0"   ! assigns the example

PosN = InStr (Line$,  "," )    ! finds position of first comma
Name$ = Line$ [1:PosN-1]       ! extracts the name "Phenol"
```

The InStr() function returns the starting position in the first string of the first occurrence of the second parameter. In this example it finds the first occurrence of a comma in the string variable Line$, which is the character after the compound name.

When you have a reference to the character in the string you can use the string index to extract the information you want. A string index refers to individual characters or a range of characters in the string. You specify indices in square brackets and separated by a colon.

For example, to print the first four characters of the Line$ variable, type: `Print Line$ [1:4]`.

The indices can also be numeric variables—the following two commands give the same results:

```
Start = 1; Stop = 4
Print Line$ [Start:Stop]
```

Process the Line$ string to extract the retention/migration times and the amount. This will delete the text you processed earlier.

```
Line$ = Line$ [PosN + 1: Len(Line$)]
```

The function PosN+1 refers to the position after the comma. The function Len() returns the length of the string. The command resets the variable Line$ to the contents of the original Line$ after the first comma.

Repeat this procedure for the next items in the string to extract the retention/migration time and amount:

```
PosN = InStr(Line$, ",")     ! finds the comma
RT =   Val(Line$[1:PosN - 1])
Amount = Val(Line$[PosN + 1: Len(Line$)]
```

The Val() function converts the string representation of the number into a numeric format.

Using Logic and Decision-Making Statements

You can make your macro intelligent by using logic and decision-making statements.

Using Logic Expressions

The quotation "To be, or not to be" is a logical expression and can be written mathematically like this:

```
(To_be = yes) OR NOT (to_be = no)
```

The result of a logical expression can be *true* or *false*. You can use more than one logical expression together. The ChemStation evaluates the expressions from left to right. Table 3 shows the comparison operators.

Table 3 **Comparison Operators**

Operator	Description	Allowed Operands
<	Less than	Both numeric
< =	Less than or equal to	Both numeric
=	Equal to	Both numeric or both string
> =	Greater than or equal to	Both numeric
>	Greater than	Both numeric
< >	Not equal to	Both numeric or both string

Only the operators equal to (=) and not equal to (< >) can be used with string expressions.

Remember that string comparison is case-sensitive, for example, “YES” < > “yes”

You can combine these comparison operators in Table 3 with the operators in Table 4 which are called Boolean operators.

Table 4 **Boolean Operators**

Operator	Description
AND	Logical and
OR	Logical or
NOT	Logical negation

Two logic expressions linked by the AND operator are true when both the first and the second expression are true. If either one or both operands is false, the result is also false. For example:

```
((Five = 5) and (Four = 4))
```

Expressions linked by the OR operator are true when either one or both operands are true.

The NOT operator inverses the result: true becomes false, and false becomes true.

You use the operators AND, OR, and NOT with logic expressions to form larger logic expressions. These keywords are not case sensitive — you can also use: and, or, not.

For example: (a=1) and (b=2)

is true only when both expressions are true.

In contrast: (a=1) or (b=2) is true when either expression is true.

Operators like AND, OR, and NOT have lowest priority compared to other operators. To improve readability, it is recommended that you use parentheses.

Using Decision-Making Statements

By using decision-making statements in a macro you can control the order in which the ChemStation executes the commands within the macro.

Using Conditional Statements

You make the decision to continue with a set of commands using the If, Then, Else, and EndIf commands. Using these commands together makes a conditional statement. You can use logical expressions with conditional

statements to test whether or not a set of commands is executed. You can omit the Else part of the conditional statement when there is no alternative. If the condition is not fulfilled, the ChemStation continues with the commands directly following the EndIf command.

The following shows you a generic version of a conditional statement with an example to the right of each line.

```
IF <LogicalExpression> THEN     !if <sequence not running> then

    execute these commands      !load current data file

  ELSE                          !else

    execute these commands      !ask the user for a file name

ENDIF                           !endif
```

Use this example to control loading a chromatogram. If a sequence is running, the ChemStation loads the current chromatogram. If a sequence is not running, the ChemStation asks you for the name of a data file. To apply the above example to the report macro, determine when a sequence is running and which data file to use.

Write the LoadChromatogram macro as follows:

```
Name USR_LoadChromatogram

! Load the chromatogram by selecting a data file

  If _SequenceOn = 1 then                   ! sequence running
    File _DataPath$ + _DataFile$             ! load current data file
    LoadSignal ,"A",, SignalReg
  Else                                      ! sequence not running
    _Button = Input("Enter data file name:", FileName$) ! ask for file
    File _DataPath$ + FileName$
    LoadSignal ,"A",, SignalReg
  EndIf
  Return

EndMacro ! USR_LoadChromatogram
```

Using Recursive Macros

In Chapter 1 “From Commands to Macros” we described macros that call themselves — we called this process **recursion**. The macro calls itself until all memory is used up, unless you program it to stop. Use the IF command to define programming criteria.

For example, you can write a macro to calculate the factorial value of a number:

```
Name USR_Factorial

! Calculates the factorial of a given number

  Parameter Number              ! of which the factorial is calculated

  If Number > 1 then           ! call factorial macro
                               ! with decremented number
    Return Number * Factorial(Number-1)
  Else                         ! stop recursive call
    Return Number              ! return current result

  EndIf

EndMacro ! USR_Factorial
```

Repeating Parts of the Macro

In Using decision-making statements, you learned how to program a macro to make decisions depending on the result of a logical expression. In this section you will learn how to return to an earlier part of the same macro, so that the part of the macro is repeated again and again. We call this type of control structure a **loop**.

Constructing Loops

We designed the report macro to print one row of data for each compound quantified. You could write a macro for a particular method with a fixed number of calibrated compounds. But this macro could not be used on another method unless the methods have the same number of calibrated compounds. It is better to repeat processing the individual peaks until all peaks in the quantification list have been processed. By writing the macro in this way you can use it on any data file quantified by any calibration.

Constructing loops simplifies the process of repeating methods. There are three types of loop which are distinguished by conditional statements:

WHILE

REPEAT

FOR

Using the WHILE–ENDWHILE Loop

Use the While command to decide whether the ChemStation should execute the commands within a loop. If the condition defined by the While command is satisfied, the ChemStation executes the loop. If the condition is not satisfied, the ChemStation goes to the commands following the EndWhile command.

In the example below a string is tested, character-by-character, from the last character forward until the first non-blank character is found.

```
Name RightTrim$        ! remove tailing blanks
  Parameter String$     ! text to be trimmed
  local i
  i = len (String$)
  if i=0 then          ! string is empty
    return " "
  endif
  while string[i] = " "
    if i=1 then        ! string has only blanks
      return " "
    endif
    i=i-1
  endwhile
  return string [1:i]
EndMacro
```

The parameter #1 is the identifier of the file which is open. You will find details of file identifiers in “File Identifiers” on page 63.

We recommend you indent the commands within a loop. This makes the loops easier to understand.

Using the REPEAT–UNTIL Loop

The second type of loop construction does an action and then checks for a condition. The ChemStation executes the commands within the loop at least once. For this type of loop you use the Repeat and Until commands.

The following example shows you how to calculate the sum of squares until the sum exceeds a specified limit:

```
Sum = 0; Step = 1               ! initializes variables
Repeat
  Sum = Sum + Step * Step       ! calculates sum of squares
  Step = Step + 1
Until Sum > 100
Print "The sum exceeds 100 after ", Step," steps."
```

Using the FOR–NEXT Loop

You use the For and Next commands to construct loops when you know the conditions at the beginning and at the end of the loop.

The example below shows you how to calculate the sum of the integers 1 through 10:

```
Sum = 0                              ! initialize variable Sum
For Counter = 1 to 10
  Sum = Sum + Counter
Next Counter
Print "The sum of the first 10 figures is ", Sum
```

When using the For and Next commands you set a variable as a counter which defines the number of times the ChemStation executes the commands within the loop.

In our report macro use the For and Next commands to determine the exact number of compounds in the quantification list. You will find more details about the quantification parameters table in Chapter 12 “System Tables”.

Write the loop as follows:

```
For Compound = 1 to TabHdrVal(ChromRes[1], "Compound", "NumberOfRows")
  PrintRow Compound
Next Compound
```

The variable Compound counts from 1 to the total number of rows in the compound table and the ChemStation executes the command within the loop for all identified compounds. The Next Compound statement increases the counter by 1 so that the next repetition of the loop takes the next compound in the compound list.

You can also put loops within loops. We call these nested loops.

For example, to fill a matrix having 5 rows and 7 columns in a user object with the row number multiplied by the column number product:

```
NewObj Matrix, 2, 5, 7       ! Makes a matrix object 5 x 7
For row = 1 to 5
  For column = 1 to 7
    SetData Matrix, row, column, row * column
  Next Column
Next Row
```

Preparing the Results

After loading the chromatogram the raw data are in the *ChromReg* register and the *ChromRes* register which will hold the quantitative results got just created. To fill it with the results you are expecting (integration and quantification results) more commands have to be executed. Basically these are the *IntegrateObj*, the *IdentifyPeaks* and the *QuantifyPeaks* command as shown in the example below:

```
Name USR_PrepareResults

! Prepare results for the report
! It is assumed that the method consists of a valid calibration table

  ! First integrate the chromatogram in SignalReg register

  IntegrateObj SignalReg

! Identify peaks and prepare the ChromRes register with all tables
! using the tables in the _DAMethod method register

  IdentifyPeaks SignalReg,, _DAMethod[1], ChromRes[1]

  ! Do quantification using the calibration table in _DAMethod register

  QuantifyPeaks ChromRes[1], _DAMethod

  ! Label peak with biggest amount in chromatogram

  USR_LabelChromatogram

  Return

EndMacro ! USR_PrepareResults
```

Before the results are printed the chromatogram will be labeled such that the peak reflecting the biggest amount is marked as such. This will be done with the macro *USR_LabelChromatogram* described in the next section. It also serves as an example for using the annotation commands.

Labeling the Chromatogram

When a chromatogram is loaded its data will be linked to an object in the *ChromReg* register (see Chapter 3 “Registers and Objects: An Introduction”). As long there is only one signal the first object will be used for the chromatographic data. If there are more signals avaialable or pressure and/or flow data were also loaded the object number must be identified using the signal description given in teh signal table of the *ChromRes* register:

```
ObjNum = GetObjNum(TabText$(ChromRes[1], “Signal”, SigNum, “Desc”)
```

where *GetObjNum()* is a macro function which matches the signal description between results and raw data and *SigNum* is an index relfection the signal number. In our example only one signal and no pressure and flow data are loaded. Therefore *SIgNum* is set to one and the signal will be in the first object of the *ChromReg* register.

The peak reflecting the biggest amount will be labeled with the following text:

```
Amount set to 100%
```

The top of the peak itself will be circled and a line will be drawn from the top of the peak to the text label. Basically three command functions are used to fulfill these tasks:

NewMarkerAn() which will create the circle mark around the top of the peak,

NewTextAnn() which will create the text annotation, and

NewPolyAnn() which will draw the connecting line.

Latter command function uses the *SetPolyAnnVal* command to set the data points for the polyline annotation. Our example uses local variables as constants to define the parameters of the annotation functions. This makes the functions in the macro more readable:

```
Name USR_LabelChromatogram

! Label chromatogram by maring the peak with biggest amount
```

```
Local Index                        ! of peak with bigggest amount
Local RetTime                      ! Retention time of biggest peak
Local Height                       ! Height of biggest peak
Local i                            ! Index for annotation

! Define annotation parameters

Local AttachedTo; AttachedTo = 4 ! attached to data
Local LineStyle; LineStyle = 0   ! solid line
Local LineWidth; LineWidth = 3   ! thickness of line
Local LineSize; LineSize = 1     ! enlarging line annoation
Local LineTopic; LineTopic = 9   ! polyline annotation
Local Mark; Mark = 2             ! mark with a circle
Local MarkSize; MarkSize = 4     ! size factor of marker
Local MarkStyle; MarkStyle = 1   ! hollow mark
Local MarkTopic; MarkTopic = 10  ! marker annotation
Local TextTopic; TextTopic = 8   ! text annotation
Local CharSize; CharSize = 12    ! character size factor of text
Local Color; Color = 65280       ! green color
Local Angle; Angle = 0           ! horizontal 0 degree
Local FitInto; FitInto = 3       ! which adjusts text into the window
Local Justify; Justify = 2       ! justifying text at middle left
Local Font$; Font$ = "Helvetica"

Index = RowByVal(ChromRes[1], "Compound",, "Amount", "Max")
RetTime = TabVal(ChromRes[1],"Compound",Index,"FirstPeak~MeasRetTime")
Height = TabVal(ChromRes[1], "Compound", Index, "FirstPeak~Height")

! Label peak with biggest amount using a circle around top of peak
i = NewMarkerAnn(SignalReg[1], Mark, RetTime, Height, \
    MarkSize, MarkTopic, MarkSize, Color, MarkStyle)

! Label this peak
i = NewTextAnn(SignalReg[1],"Amount set to 100%",RetTime+0.5,Height, \

AttachedTo,TextTopic,CharSize,Color,Angle,FitInto,Justify,Font$)

! Draw a connecting line between top of peak and text label
i = NewPolyAnn(SignalReg[1], 0, RetTime, Height, \
    AttachedTo, LineTopic, LineSize, Color,,, LineWidth)

SetPolyAnnVal SignalReg[1], i, 1, 0, 0         ! Start of line
SetPolyAnnVal SignalReg[1], i, 2, 0.5, 0       ! End of line
Return

EndMacro !  USR_LabelChromatogram
```

Printing the Results

We have described how to print messages on the ChemStation message line. To get a hardcopy of your results you use your printer. First, set up your printer to accept the text or graphics and print it in the appropriate format.

To send a simple message to the printer, open the printer in a similar way as you open a file, for example:

```
OpenDevice "Printer" as #5
Print #5, "Hello"
Close #5
```

The ChemStation prints through the Windows system which allows you to treat the printer generically. The type of printer you have and how it is connected is not important.

The information to be printed is collected and passed to the Windows system in a special file called a Windows **metafile**. You control the collection and passing of these files by the OpenDevice and Close commands.

In our report macro we will print the report on the printer with a specific header on each page and number each page in the “x of y” format, which helps users to see the report is complete.

A macro called *USR_PreparePrinter* is included in the report macro. This macro prepares the printer by opening the device, setting up the header and specifying the page numbering scheme.

```
Name USR_PreparePrinter

! prepares printer for printing a complete report

Local Header$               ! keeps header for each page
Local pWidth                ! page width

OpenDevice "PRINTER" as #3
pWidth = xMax()             ! gets characters per line

Header$ = "Company Confidential"
```

```
! Header should be placed at top center of each page
SetHeader Header$, pWidth / 2 - Len(Header$) / 2

! Set margins to have an intendation of 4 characters and two lines
! at the top and at the bottom
SetMargin 4, 2, 2

! Paging should be printed at bottom center of each page

BeginJob ! will give automatic paging e.g. Page 1 of 2 etc.
Return

EndMacro ! USR_PreparePrinter
```

After the Next command in the main report, the ChemStation closes the printer device which in turn closes the metafile and sends the output to the printer.

```
Name USR_RelativeAmounts

! Prints a report with amounts relative to the biggest peak

! Last modified by Your Name at 6-Oct-95

Local CompoundNum            ! Index of compound
Local NoOfCompounds          ! Total number of compounds
Local MyReport$              ! For EndJob command to display report
                              ! in report viewer

MyReport$ = "My Customized Report"

USR_LoadChromatogram         ! into register
USR_PrepareResults           ! Integration, identification
                              ! and quantification of report
USR_PreparePrinter  ! Prepares printer for output (StartJob)
USR_PrintHeader MyReport$ ! Consisting of chromatogram and
                          ! Report table header

NoOfCompounds = TabHdrVal(ChromRes[1], "Compound", "NumberOfRows")
For CompoundNum = 1 to NoOfCompounds    ! Loop through all compounds

    ! 4th column calculation done by function :  USR_AmountPercent()
    USR_PrintRow CompoundNum

Next CompoundNum
USR_ClosePrinter

EndMacro ! USR_RelativeAmounts
```

Printing the Results

To complete printing of our report macro examples, the three missing macros are given below. The second macro is a good example of how to use formatted output. The third macro *USR_ClosePrinter* also resets some settings which were made in the *USR_PreparePrinter* macro. This avoids surprises when performing standard outputs afterwards.

```
Name USR_PrintHeader

! Prints header of report consisting of chromatogram plot
! and report table header

Parameter Title$

Local yPos        ! vertical y position of report page
Local Margin

SetPageLoc 4, 3 ! start printing title at 3rd line,
                ! 4th character intended
Print #3, Center$(76, Title$)

! First print chromatogram using window 28 allowing all annotations

SetTabVal _Config[1], "Window", 28, "Topic", 0xFFFFFFFF

! Define size of plot: Height = 0.8% of page height, 0.3% of page width

SetWinPrintParams 28, "PRINTER", 0.8, 0.3

SetPageLoc 4, 5 ! start printing at 3rd line, 4th character intended

GetDataMinMax SignalReg[1], 1 ! to get minimum and maximum of response
Margin = 0.05 * (obj_max - obj_min) ! define a 5% margin
Draw 28, SignalReg[1],, obj_min-Margin:obj_max+Margin
Print using #3, "//"   ! Print two more lines

yPos = yLoc()          ! assign current line position to yPos
SetPageLoc 20, yPos    ! start report printing in current line
                         ! 20 characters intended

Print using #3, "%/", "  #  Ret.Time  Concentration  %Max. Amount"
Print using #3, "%/", "       [min]       [ng/ul]"
Print using #3, "%/", "========================================="
Return

EndMacro ! USR_PrintHeader

Name USR_PrintRow

! Prints all column for each compound
```

```
Parameter CompoundNum

Local Format$         ! Format for printing each line
Local Rt              ! Retention time of main peak
Local Amount          ! Amount of compound

! The format string specifies how the line will be printed
Format$ = " ##  4#.3#  7#.2#       4#.2#/"

! Get retention time and amount from compound table
Rt=TabVal(ChromRes[1],"Compound",CompoundNum,"FirstPeak~MeasRetTime")
Amount = TabVal(ChromRes[1], "Compound", CompoundNum, "Amount")

! Do the printing
Print using
#3,Format$,CompoundNum,Rt,Amount,USR_AmountPercent(CompoundNum)
Return

EndMacro ! USR_PrintRow
```

AmountPercent is the macro function we wrote earlier to replace the macro Calculate4thColumn.

```
Name USR_ClosePrinter

! Closes print jop to show the report first in the report viewer
! before it can be printed from there to the printer

EndJob; Close #3

! Reset printing parameters
SetHeader ""; SetFooter ""
SetMargin 0, 0, 0
Return

EndMacro ! USR_ClosePrinter
```

Displaying the Results

To display a single line on the screen you use the print command to send the text to the message line. If you want to display more lines of text lines — for example, a report — you can use the report viewer.

This can be controlled with the *EndJob* command. Adding a string variable as parameter to the EndJob command redirects the printout to the report viewer. Some additional commands are necessary to display the report viewer. The *USR_ClosePrinter* macro has to be modified and extended to reflect this new approach. Don't forget to set the *MyReport$* variable as parameter for the call of *USR_ClosePrinter* in the main macro *USR_RelativeAmounts*:

```
        :
        :
Next CompoundNum
USR_ClosePrinter MyReport$

EndMacro ! USR_RelativeAmounts


Name USR_ClosePrinter

! Closes print jop to show the report first in the report viewer
! before it can be printed from there to the printer

Parameter MyReport$

EndJob MyReport$ ! redirects report to viewer

SetWinTitle 69, MyReport$

ShowReport 69,MyReport$, "PrintJob", "DelJob" ! displays report viewer
SetActiveWindow 69
Close #3

! Reset printing parameters
SetHeader ""; SetFooter ""
SetMargin 0, 0, 0
Return

EndMacro ! USR_ClosePrinter
```

Printing Results to a Text File

The automatic header, footer, and page settings you used earlier are only applicable when you send information to the printer. It is not possible to output to a file at a specific position as you did using the SetPageLoc command. You position the information using carriage return, linefeed and blank characters. The following example shows a solution for the SetFileLoc command:

```
Name USR_SetFileLoc

! Puts an internal text pointer to the specific position in file

! Note: can only position forward (NOT backwards) and needs to know
!       the actual x- and y-position as global variables since it is
!       not possible to go back up lines in a file.

  Parameter NewX, NewY        ! position

  Local Counter               ! counter for lines and blank characters

! Check for range errors

  If (NewY < ActualY) return   ! Ignore if we have been asked to go up
!
  If ((NewY = ActualY) and (NewX < ActualX)) ! Or back across the same
 line

! Move to new Y

  For counter = 1 to (NewY - ActualY)  ! difference of lines
    Print using #7, "/"                ! print one newline

    ActualX = 1       ! If we print a newline the X-position becomes 1
  Next counter

! Move to requested X position

  For counter = 1 to (NewX - ActualX)  ! difference of columns
    Print using #7, " "                ! blank character
  Next counter

  ActualY = NewY; ActualX = NewX       ! Set new cursor position
  Return

  EndMacro ! USR_SetFileLoc
```

For our report macro the following example in PrintTitle shows the modifications required to print this information to a file:

```
Name USR_PrintTitleToFile

! Print the header part of the report to a file

  ActualX = 1; ActualY = 1      ! initialize printing position
  USR_SetFileLoc 20, 3          ! start printing line 3 column 20
  Print #7", File name:," , _dataPath$ + _dataFile$

  USR_SetFileLoc 10, 5          ! positioning of table titles
  Print #7, "           #  Ret.Time  Concentration  %MaxAmount"
  Print #7, "                 [min]      [ng/ul]               "
  Print #7, "          ----------------------------------------"
  Return

  EndMacro ! USR_PrintTitleToFile
```

The file is opened and closed like the printer:

```
Open "report.txt" as #7 for output   ! Opens text file
PrintTitleToFile                     ! Outputs the titles
.
.                                    ! Other code goes here
.
Close #7                             ! closes it
```

Entering Data into a Macro

You have a number of different ways to input or enter data into a macro.

- menus,
- single-item dialog boxes, or
- multiple-item dialog boxes.

Using Menus

Menus are a typical way for Windows applications to obtain instructions or information from a user through the mouse or other pointing device. You load a menu using the MenuRead command:

```
MenuRead "mymenu.mac"
```

The command reads the macro file mymenu.mac that contains a menu definition. The parameter Switch immediately loads and displays the menu defined in the macro file mymenu.mac.

The macro file mymenu.mac contains a series of MenuAdd commands that define the individual items in the menu:

```
MenuAdd "&My","Run my macro...","MyThing",,"Executes my own macro"
MenuAdd "&My",,"SEPARATOR"
MenuAdd "&My","Programs","SUBMENU"
MenuAdd "&My|Programs","Run Excel...","ExecNoWait" "Excel.exe"",1"
MenuAdd "&My|Programs","Run Notepad...","ExecNoWait "Notepad.exe"",1"
```

Menu definition macros do not have Name or EndMacro commands. The MenuAdd is only valid for the current view.

This example creates a menu bar with one menu item called My. Under this item is a drop down menu with two items separated by a line, the SEPARATOR. The first item, called Run My Macro, starts a macro called MyThing. The second item is called Programs and has a submenu comprising two items, Run Excel and Run Notepad. These items start the appropriate applications when selected using the ExecNoWait() function.

You will find a complete list of menu commands in the Commands section of the online help.

Using Single-Item Dialog Boxes

You can write macros to ask for input and to assign given information to the specified variable. The Input function displays a single-item input box on the screen, allowing you to prompt the user for a value or text.

In the report macro, we used the Input function to enter the file name. To display a default answer for the user, set the variable that will receive the input to your desired default value before you execute the Input function. For example:

```
Day$ = "Monday"
Button = Input ("What is the day of the week?", Day$)
```

A dialog box appears and displays Monday. Either accept Monday by choosing OK, or type a new day into the input field. Choosing Cancel leaves the variable Day$ as Monday. The variable _Button is set according to which key—OK or Cancel—you select.

Rewrite the LoadChromatogram macro to stop printing the report by choosing Cancel.

```
Name LoadChromatogram

! Load the chromatogram by selecting a data file

  Local OK; OK = 1                                ! for better readabilit
y
  If _SequenceOn = 0 then                         ! sequence is not runni
ng
    File _dataPath$+_dataFile$                    ! load current data fil
e
    LoadSignal ,"A",, signalReg
  Else                                            ! sequence is running
    Button = Input ("Enter data file name:", fileName$)   ! ask for fi
le
    If _Button = OK then
      File fileName$
      LoadSignal ,"A",, signalReg
    Else; Stop                                    ! which stops complete
macro
    EndIf
  EndIf
Return

EndMacro ! LoadChromatogram
```

The Input function only works with string variables. To input a number use the val() and val$() functions to convert numbers to strings. For example:

```
Number=5
Number$=Val$ (Number)                          ! convert number to string
Button = Input ("Enter Number:", Number$)
Number=Val (number$)                           ! convert string to number
```

Using Dialog Boxes

The Input function is sufficient when you want to enter one item. When you want to enter several items you can use a dialog box containing several input fields. Using dialog boxes helps you to create an intelligent interface.

The following macro defines the dialog box which allows you to enter the name of a data file and select a destination for the report. If you select the screen as the destination, you must enter the name of a text file or use the default file Report.txt, because output to the screen uses a text file.

```
Name DefineMyRepParms

  ! Define dialog box for customized report parameter

  BeginDialog "MyRepParms", 93, 47, 187, 119, "Customized report param
eters"
  StaticText 19, 19, 66, 9, "Data file name :"
  EditBox 77, 18, 91, 11, FileName$
  GroupBox 19, 39, 149, 50, "Report destination :"

  ToPrinter = 1
  CheckBox 35, 49, 66, 10, "Printer", ToPrinter

  ToFile = 0
  CheckBox 35, 61, 23, 10, "File", ToFile

  ToScreen = 0
  CheckBox 35, 73, 34, 10, Screen, ToScreen

  StaticText 81, 62, 66, 9, "Destination file name :"
  DestinationFile$ = "Report.txt"
  EditBox 96, 71, 66, 11, DestinationFile$

  OkButton 39, 97, 36, 11, "OK"
  CancelButton 112, 97, 36, 11, "Cancel"

  EndDialog
  Return

EndMacro ! defineMyRepParms
```

To use this dialog box in the report macro, modify the LoadChromatogram macro:

```
Name LoadChromatogram

! Load the chromatogram by selecting a data file

  Local OK; OK = 1                              ! for better readability
  If _SequenceOn = 0 then                       ! sequence is not running
    File _dataPath$+_dataFile$                  ! load current data file
    LoadSignal ,A,,
  Else                                          ! sequence is running
    DefineMyParms                               ! defines dialog box
    If showDialog("MyRepParms") = OK then
      removeDialog "MyRepParms"                 ! to clear memory
      File fileName$
      LoadSignal ,A,
    Else
      removeDialog "myRepParms"; Stop           ! which stops complete macro
    EndIf
  EndIf
Return

EndMacro ! LoadChromatogram
```

It is more complicated to write a dialog box for a macro — you must first define the dialog box in its own macro and then use a separate command to show and remove the dialog box.

The definition and function of the OK and Cancel buttons are the same as with the Input function. The variables defined in the dialog box are automatically filled with the user's input.

What To Do If Something Goes Wrong

As your macros become more complex, the chance that something will go wrong becomes greater, especially when you are trying a macro for the first time. When this happens you want to know what went wrong, why and where it went wrong. The procedure for removing errors from your macro is called **debugging**. It will usually take you a number of attempts before your macro is free of errors or **bugs**.

Handling Errors

Most errors are caused by typing mistakes or incorrect logic. You can find typing errors easily. When you have misspelled a command or the ChemStation cannot evaluate variables, the macro is stopped and the ChemStation displays the type of error on the message line.

Logic errors are more difficult to find and you will find it helpful to use special tools and techniques.

Reading Error Messages

To find out where the macro has stopped, enter: `ListMessages on` and press ENTER. A box appears in the ChemStation window where multiple messages are displayed.

- If an error occurs when you load a macro — for example, misspelled commands that lead to syntax errors — the line where the error occurred is displayed.
- If an error occurs when the macro is running, the line and name of the incorrect command statement are displayed.

When you are using the macro development tools the messages box is switched on automatically, see Chapter 1 “From Commands to Macros”. You can start the development tools by typing: `macro "mactools.mac"` on the command line and pressing ENTER.

Recording the Steps of a Macro

You can use the SetVerbose command to display the commands as they are executed. This helps to determine the actual flow of execution in your macro.

Using the Logging Command to Record the Steps of a Macro

A better way for you to look for logic errors is to record or *log* the execution of a macro in a file. You switch on logging with the following command:

```
Logging 7, "C:\HPCHEM\CORE\DEBUG.LOG"
```

The file Debug.Log gives you a record of what the macro did at each command and is a complete step-by-step execution history of the macro. This file can become large, particularly when you have many loops in your macro. When the macro has finished, load the log file into your word processor and see which statement was executed last. You can include the logging command in your macro to record all critical paths of the macro.

You can increase the level of detail of this logfile with the following command:

```
setverbose 3
```

Debugging Individual Commands

If you are having trouble debugging a macro it is often useful to see what happens when each individual command is executed. You can copy individual commands from a text editor, such as Notepad, to the Windows clipboard and paste them into the ChemStations command line by typing CNTL+V . This allows you to test and modify each command separately. You can also copy the debugged line back to the editor clipboard by typing CNTL+C when the correct version of the command is displayed at the ChemStation command line.

Using Messages to Record the Steps of a Macro

There are two more ways to record the execution flow of your macro.

You can include Print commands into your macro with information about what the macro is doing.

The Print command displays the text in the message line, but each Print command overwrites the previous text. It is, therefore, recommended that you use the ListMessages or Logging commands in combination with the Print command.

For example:

```
Name MyMacro
.
.
.
For Index = 1 to Max
  If ThisWay = Yes then
    Print, " ***** go this way *****" ! debug
.
.
.
  Else
    Print, " ----- go that way -----" ! debug
.
.
.
  EndIf
Next Index
```

The second way to record the execution flow of the macro is to use the Alert command in the macro. Include information on the contents of variables and on what the macro is doing. For example:

```
Name MyMacro
.
.
.
  a = log(sqrt(value/(k+exp(1-n))))
  Button = Alert ("Calculation of a = " + val$(a), 1)  ! debug
.
.
.
```

The advantage of using the Alert command is that it stops execution and displays the parameters in the input dialog box. You can check the result of a calculation and continue the macro by choosing OK.

After the Print or Alert commands have done their job, delete them from the macro or comment them out using the ! character. If you add comments such as:

```
! debug after each Alert or Print, it makes them easier to find.
```

Another way to record your commands is to execute the debug statements based on a debugging flag, for example:

```
Name MyMacro

Debug = 1    ! Debug flag
.
.
.
  For Index = 1 to Max
    If ThisWay = Yes then
      If (Debug=1); Button = Alert (" ***** going this way *****", 1);
 endIf
      .
      .
      .
    Else
      If (Debug=1); Button = Alert (" ----- going that way -----
", 1); endIf
      .
      .
      .
    EndIf
  Next Index
```

To switch off your debugging messages set Debug=0. You can remove the debugging conditional statements when the macro is working as they are always evaluated and slow the execution of your macro.

Using the On Error Command

Normally when an error occurs the ChemStation stops executing the commands in the macro. For example, if you load the chromatogram from a disk and forgot to insert the disk in the drive, the macro stops and an error message is displayed. You can avoid this using the On Error command.

```
Name DefineDataFile

! Defines data file of which name can be entered

  Button = Input ("Enter data file name:", fileName$)
  On error Button=alert ("Cannot read data file! Try it again!", 1)
  Button = -
1  ! initialize it so that it can be checked if no error occurs
  File fileName$   ! Opens data file which might fail

  If Button >= 0 then ! error has occurred
    If Button = 0 then ! cancel key pressed
      Stop ! macro execution
    Else
       defineDataFile ! try it again
     EndIf
```

```
  EndIf
Return

EndMacro ! defineDataFile


Name loadSignalmacro

! Load a signal A from entered data file

  DefineDataFile ! enter file name
  LoadSignal ,"A",,signalReg
  Return

EndMacro ! loadSignalmacro
```

The first macro DefineDataFile is a subroutine allowing you to enter and define the name of a data file. If this fails, a dialog box appears telling you something went wrong and to try it again. Choose Cancel to stop, or solve the problem and choose OK. The macro asks again for the file name.

It is not only the system that produces error messages. Based on your macro you can generate error messages, for example:

```
Name CleanUpLevel1
  Close #3
  Button = Alert ("Closing first opened file!", 2)
  Stop
EndMacro ! CleanUpLevel1

Name CleanUpLevel2
  Close #3; close #4
  Button = Alert ("Closing all opened files!", 2)
  Stop
EndMacro ! CleanUpLevel1

Name ProcessPeaks

! Process peaks and make some special calculations

  Open "factors.txt" as #3 for input
  On error cleanUpLevel1

  Open "results.txt" as #4 for output
  On error cleanUpLevel2
  .
  .
```

```
.
If numPeaks <= 1 then; generate error; endIf
.
.
.
```

If opening the second file fails, the macro CleanUpLevel1 is started. When there are not enough peaks available to process (< = 1), an error is generated. The error starts the macro CleanUpLevel2 because it is the last On Error setup. When an error is generated using the Generate Error command the On Error routine is activated.

Error Conditions

The following situations will cause a command processor error:

- attempting to read from a non-existing register, object, object header item, table header item, or table,
- attempting to write outside the dimensions of a data block, or
- attempting to remove a non-existing register, object, object header item, table, table header item, or table column will result in non-operation.

Error Variables

If an error occurs, the following variables are set:

_ERROR	error code (=0 if no error occurred).
_ERRORMSG$	error text.
_ERRCMD$	name of command or function that failed.
_ERRMACRO$	name of macro that called the failing command.
_ERRLINE$	line number in the file where the error occurred. Note that the reported line number will refer to the previous line if the error occurred during the evaluation of a function whose return value is used as a parameter.
_ERRFILE$	name of the file from which the macro was read.

6

User Defined Hooks

User Defined Hooks

This chapter describes user defined hooks, how to use it, and gives you some macro examples.

Dynamic Data Exchange

Situation: The user wants to modify the operation of the standard ChemStation by adding own macros that should be executed in certain places, so-called *hooks*.

Problem: It is not desired to force the user to modify the standard macros other than in those files that are delivered as .mac files anyhow. And even in .mac files there is no complete freedom to change at will.

Goal: It should be possible that the standard ChemStation can be updated to a new version without disrupting the user macro hooks, provided that the user macros are still compatible.

The Hook Concept

At certain points in the standard ChemStation a (named) hook is defined. For each hook it is described what data is available at this point and what the hook macro might do here.

If in the following it is mentioned that a hook is called during Run Method, this also implies Run Sequence. If the called hook macro needs to differentiate between the two, the variable _SequenceOn can be checked.

Some of the hooks are called both during Run Method (and Run Sequence) and also interactively from various menu items. If the called hook macro needs to differentiate between automatic and interactive mode, the variable _MethodOn can be checked.

The Hooks

PreViewMenu

Called during setting up the menu immediately before the View menu is added. Can be used to define a new menu in the current view. Note that the current view is stored as table header item CurrentView in the Window table. This can be used to find more info, esp. the MenuFile, in the MenusForViews table. By comparison of the MenuFile table header item with the ShortMenuFile and FullMenuFile column names in there the short/full mode can be determined.

LoadMethod

Called at the end of loading a method, either interactively from the menu item or automatically during a sequence. At this point _MethPath$ and _MethFile$ are updated to the new method, and all parameter files within that .m directory are loaded. Can be used to load additional files from the .m directory.

SaveMethod

Called at the end of interactively saving a method, either from the Save or Save As menu items. At this point _MethPath$ and _MethFile$ are updated to the new method, and all parameter files are saved to that .m directory. Can be used to save additional files to the .m directory.

EditMethod

Called at the end of Edit Entire Method, after all other method sections are displayed. Can be used to edit additional parameters. Note that the hook is called even if (through Cancel) the rest of the originally selected method sections are skipped.

PrintMethod

Called at the end of printing a method, either interactively from the menu item or automatically as part of a sequence summary report. At this point the output destination port #5 is still open. With the DevNum command it can be checked whether this port is connected to the printer or to a .txt file. Can be used to print additional parameters.

PreRun

Called during Run Method immediately before the PreRun macro is done, regardless whether this is enabled by the user or not. Can be used to modify the Runtime Checklist's system variables.

AcqRes

Called during Run Method immediately after data acquisition is finished in an online system, while the acqres.reg file is being generated. At this time the register AcqRes exists. This will afterwards be saved to the file acqres.reg. The hook can be used to add information to the Data File. The first object of acqres.reg will become the basis of the ChromRes register in Data Analysis. Therefore the names of object header items and tables must not coincide

with those added by Data Analysis later on. To avoid naming conflicts it is strongly suggested that names generated by the user, e.g. for info generated in the PreRun macro, start with User, Usr, or Cust. Names generated by HP should use prefixes like Acq, Mod, Inst, or Dev.

PostAcq — Called during Run Method immediately after data acquisition is finished in an online system. At this time all raw data files exist. Can be used to do instrument-specific control.

PreLoadFile — Called during Run Method before standard Data Analysis loads the rawdata file. Can be used to change the file name and several other variables that control the standard Data Analysis.

PreInteg — Called during Run Method before standard Data Analysis does the integration. Also called during interactive Load Signal or Overlay Signal or Subtract Blank Run from the menu in the Data Analysis view. At all these times all required signals in the ChromReg register are loaded, ChromRes register is setup. Can be used to modify the loaded signals, e.g., by scaling or shifting, subtracting a blank run, etc. It is advisable to put some kind of flag, for example, a new object header item, into each modified object. This enables the hook macro to check before the modification whether the modification has already been done, especially in the case of Overlay Signal or Subtract Blank Run where the ChromReg register is merely enhanced by several objects, but not generated completely anew.

PostInteg — Called during Run Method after standard Data Analysis has done the integration, but before peak identification. Also called during interactive Integrate or Auto Integrate from the menu in the Data Analysis view. At all these times the signals are integrated. Can be used to modify the integration results, for example, through manual integration, re-integration with optimized parameters, or similar things.

PreQuant — Called during Run Method before standard Data Analysis does the quantification or automatic recalibration (in case of calibration runs). Also called during interactive Print Report from the menu in the Data Analysis view. At both these times the peak identification is finished. Can be used to modify the identification results in ChromRes.

PostQuant — Called during Run Method after standard Data Analysis does the quantification or automatic recalibration (in case of calibration runs). Also called during interactive Print Report from the menu in the Data Analysis view. At both these times the quantification results in ChromRes are fully available. Can be used to modify the Amount values in ChromRes through special calculation. Also any additional result calculation should be done

here and added to ChromRes because this is the latest point before report formatting starts.

PostDA — Called during Run Method at the end of processing the data file but before the PostRun macro. At this time both standard and custom Data Analysis is finished, all results have been calculated and printed, all result files have been generated and the Save GLP Data has been done (that is, the glpsave.reg file has been generated). Can be used to transfer the result files from the .D directory to some other place. The difference between this hook and the PostRun macro is that this hook is done on a data file basis, that is, it might be delayed in case of bracketing, whereas the PostRun macro is done on an instrument-run basis, that is, in bracketing it might be done before the final quantification of the data file takes place. Another difference is that in the GC with two injectors the PostDA hook is executed once per data file, whereas the PostRun macro is called just once.

PostRun — Called during Run Method immediately after the PostRun macro is done, regardless whether this is enabled by the user or not. Can be used to wrap up the instrument run in connection with other external devices or a server. Note that this hook will not be called if a sequence is running which has Acquisition Only selected in its Sequence Parameter screen. Note also that in case of a bracketing sequence this hook might be called before the calculation and report of the sample run is done because that is delayed until the calibration run of the closing bracket is performed.

PreSeq — Called during Run Sequence before the actual sequence is started. Can be used for initialization, for example, opening a data base or establishing contact to a host computer for data transfers during a sequence.

PostSeq — Called during Run Sequence when the sequence table is finished and the user-specified Post-Sequence macro has been executed, and the sequence summary report macro is finished. Can be used to close a data base or instruct a host computer to start some processing before closing the communication links.

CloseApp — Called during the Exit or Close menu item. Can be used to close open files or save registers.

Initialization of a Hook

For each hook that the user wants to use, a macro has to be written that is executed when the hook point is reached. This macro can either be put into user.mac directly, or, especially if several such macros have to be written, put into a separate .mac file with the loading Macro command in user.mac.

Next, the hook macro has to be connected to the hook itself. For this a macro called SetHook is available. It has two string parameters: the name of the hook, as defined in the previous chapter, and the name of the macro to be executed.

Hook Example

In the following an example is shown which can be used as a structure for specific hooks. It shows the principle of the hook concept:

Example

```
name MyPreIntegHook
 ! this will be called as PreInteg hook
 ! As the hook will be called each time (it is not possible
 ! to remove the hook again) it should be possible to
 ! enable or disable the modifications done by the hook.
 ! For this it is necessary to check whether we really need
 ! to do something in this macro.
 ! This could be done by checking a global variable, like:
 !   if Check (Variable, USR_MyPreInteg) = 1 then
 !     if USR_MyPreInteg = 1 then
 !       < do the real work here >
 !     endif
 !   endif
 !
 ! or by checking a header item in the method, like:
 !   if ObjHdrType (_DAMethod[1], "USR_MyPreInteg") <> 99 then
 !     if ObjHdrVal (_DAMethod[1], "USR_MyPreInteg") = 1 then
 !       < do the real work here >
 !     endif
 !   endif
endmacro
SetHook "PreInteg", "MyPreIntegHook"
```

7

Exchanging Information Between Windows Applications by Dynamic Data Exchange

Exchanging Information Between Windows Applications by Dynamic Data Exchange

This chapter describes dynamic data exchange, how to use it, and gives you some macro examples.

Dynamic Data Exchange

Dynamic data exchange is for advanced users and is specific to the application you are using. This chapter describes dynamic data exchange for the Hewlett-Packard ChemStation. For other applications we recommend you use the relevant handbooks, for example, Microsoft Excel.

Dynamic data exchange (DDE) can be used by Microsoft Windows programs to communicate with each other.

DDE Commands

The ChemStation has a set of DDE commands to initiate and terminate communication with other programs, to retrieve or send data and to execute remote commands or macros.

Using DDE

DDE is used as one function and five commands in the ChemStation software, see Table 5.

Table 5 **Commands and Functions**

Type	Name	Description
Function	DDEInitiate	Initiates DDE link between two applications and selects topic for exchange
Command	DDETerminate	Terminates DDE link
Command	DDERequest	Retrieves data from other application
Command	DDEPoke	Sends data to other application
Command	DDEExecute	Executes command or macro in other application
Command	DDEAdvise	Sets up a hot link between variables in the two applications

The syntax of the above commands is explained in the Commands section of the online help.

DDE Terminology

This section explains the terms used when describing the DDE communication process.

All applications using DDE for exchanging information have three parts.

Application The first part of the communication structure used by DDE. This is the name of the program being addressed.

Topic The second part of the communication structure used by DDE. A topic sets the scope and behavior of the link.

Item The third part of the communication structure used by DDE. This is a single data object that can be transmitted using a DDE exchange and is a character string.

Other DDE terms that are used are:

Client	The application initiating the communication.
Server	The application providing services to the client.
Link	An active DDE conversation that is uniquely identified by a channel number set when the link was initiated.
Hot Link	A link in which the client has requested the server to provide updates on a particular item whenever that item changes.

DDE Sessions

DDE commands can be used in a macro or entered interactively. Each DDE session is divided into *three* sections:

- initialization section,
- conversation section, and
- termination section.

Initialization Section

The DDEInitiate function selects an application program and a conversation topic. The conversation topics are dependent on the server application, see “ChemStation Topics” on page 116. The topics determine the scope and behavior of the conversation section that follows.

The initialization function returns a channel number or identifier that is used in the subsequent sections. For example, using the offline ChemStation of Instrument 1 as a server from the online version of the ChemStation Instrument 1:

```
AppName$ = ProfileString$("PCS,1","_DDEOffline")
Channel = DDEInitiate ("AppName$", "SYSTEM")
```

Conversation Section

In this section items are used, within the limits of the topic selected during initialization, to exchange data, remotely execute commands, and set up hot links. For example:

```
DDEExecute Channel, "FILE ""DEMODAD.D"""
DDEExecute Channel, "DRAW 1,CHROMREG"
```

Do not start more than one task for the same server.

Termination Section

DDETerminate is used to close the link and free the associated resources, for example:

```
DDETerminate Channel
```

Application Names

The application names are the names of the program files *without* the .exe extension, for example:

EXCEL is the application name for the Microsoft EXCEL spreadsheet program.

The application is specified during initialization. In the ChemStation, a command processor variable called _DDENAME$ holds the application name. This name should be used by a client to initiate a link. As the name must be known by the client, the ChemStation writes that name to the [PCS,n] section of the win.ini file, where “n” is the number of the instrument. The item name in that section is either _DDEOnline or _DDEOffline, depending on the ChemStation.

ChemStation Topics

There are three topics that can be used in the ChemStation.

SYSTEM

The SYSTEM topic is used to return information about the status and capabilities of the ChemStation when it is acting as a server.

CPWAIT

The CPWAIT topic sets the behavior of the link. In this mode the ChemStation acts as a client and will wait for remote commands or transactions to be completed before continuing.

CPNOWAIT

The CPNOWAIT topic sets the behavior of the link. In this mode the ChemStation acts as a client and will continue to execute commands without waiting for the server to complete the command or function that has been initiated.

ChemStation Items

This section describes the ChemStation items.

Items with the SYSTEM Topic

Table 6 shows the items for the SYSTEM topic. The SYSTEM topic is used by a client to retrieve information from the ChemStation which is acting as the server.

Table 6 **Items and Values Returned**

Item	Value returned
SysItems	A list of all items you can use with the System topic
Topics	A list of the implemented topics—the current implementation is SYSTEM, CPWAIT, and CPNOWAIT
Formats	A list of the supported Clipboard formats—the current implementation is CF_TEXT
Status	The current status of the command processor—can be BUSY or READY
Returnmessage	Special return message

Items with the CPWAIT and CPNOWAIT Topics

You can specify any ChemStation command processor variable, command, or macro as an item.

DDE Macro Examples

This section describes some macros that use DDE commands to communicate between two applications:

- all examples use the Microsoft Excel program, and
- DDE communication is *only* possible with Microsoft Windows applications that support DDE.

What You Need To Do

- You must install and start the ChemStation software and Microsoft Excel.
- You should store all the macros in the directory defined by the variable _AUTOPATH$ (C:\HPCHEM\CORE as default).
- You must load the macros into the ChemStation using the MACRO command.
- You must start Microsoft Excel with a spreadsheet called SHEET1.XLS.

To turn on the command line in the ChemStation:

1 Choose the System menu box in the upper-left corner of the window.

2 Choose Cmdline On from the System Menu.

Example 1: Sending Data to Excel by DDE

This section describes the macro DDETest1:

- the macro DDETest1 opens a DDE channel with a Microsoft Excel spreadsheet called Sheet1.xls, and
- the macro sets the contents of row 1 column 2 in the spreadsheet to 3 and prints what it has done on the print line of the ChemStation.

Before you run these examples you *must* start Excel with a spreadsheet called SHEET1.XLS. It is possible to start Excel automatically with a particular spreadsheet by using the Exec(), ExecWait(), or ExecNoWait() commands. See the Commands section of the online help.

The ON ERROR command is used to ensure that the DDE channel is closed if an unexpected error occurs.

In this example:

- the client is the ChemStation,
- the server is EXCEL,
- the application is EXCEL,
- the topic is Sheet1.xls, and
- the items are R1C2 and R2C2.

```
name DDETest1

 local MyVar, MyString$
 !
 !!! Using an error trap is useful in order not to leave channels open
 !
 on error CloseDDE

 !
 !!! Initiation block
 !
 Chan = DDEInitiate("EXCEL","Sheet1")                          ! Open the channel

 !
 ! Conversation block
 !
 MyVar = 3
 MyString$ = val$(MyVar)

 DDEPoke Chan, "R1C2", MyString$                               ! Send the data
 Print "Sheet1.xls contains ",MyString$," at row 1 column2" ! Print the result
```

Example 1: Sending Data to Excel by DDE

```
!
!!! Termination section
!
DDETerminate Chan                                          ! Close the channel

Return

EndMacro
!
!!! Error handling macro
!
Name CloseDDE

Local Button
DDETerminate Chan                                          ! Close the channel
Button = Alert ("Stopped on error",3)                      ! Print a warning
Return
EndMacro
```

Example 2: Getting Data from Excel by DDE

This section describes the macro DDETest2:

- the macro DDETest2 opens a DDE channel with a Microsoft Excel spreadsheet called SHEET1.XLS, and
- the macro gets the contents of row 5 column 6 from the spreadsheet and prints the data on the print line of the ChemStation.

Before you run these examples you *must* start Excel with a spreadsheet called SHEET1.XLS. It is possible to start Excel automatically with a particular spreadsheet by using the Exec(), ExecWait(), or ExecNoWait() commands. See the Commands section of the online help.

For this second example you should enter a value into the SHEET1.XLS spreadsheet at row 5 column 6.

The ON ERROR command is used to ensure that the DDE channel is closed if an unexpected error occurs.

In this example:

- the client is the ChemStation,
- the server is EXCEL,
- the application is EXCEL,
- the topic is Sheet1.xls, and
- the item is R5C6.

```
Name DDETest2

 local MyVar,MyString$

 !
 !!! Using an error trap is useful in order not to leave channels open
 !
 on error CloseDDE

 !
 !!! Initiation block
 !
 Chan = DDEInitiate("EXCEL", "Sheet1")                          ! Open the channel

 !
 !!! Conversation block
 !!! N.B. put a value in Sheet1.xls row 5 column 6 before you run this macro
```

Example 2: Getting Data from Excel by DDE

```
!
DDERequest Chan, "R5C6",MyString$                          ! Get the data
Print "Sheet1.xls contains ",MyString$," at row 5 column 6" ! Print the result

!
!!! Termination block
!
DDETerminate Chan                                          ! Close the channel

Return

EndMacro

!
!!! Error handling macro
!
Name CloseDDE

Local Button
DDETerminate Chan                                          ! Close the channel
Button = Alert ("Stopped on error",3)                      ! Print a warning
Return
EndMacro
```

Example 3: Executing a Command in Excel Through DDE

This section describes the macro DDETest3:

- the macro DDETest3 opens a DDE channel with a Microsoft Excel spreadsheet called SHEET1.XLS, and
- the macro executes the Excel Hide() function to hide the Sheet1.xls spreadsheet from the user. You can use the Excel Unhide command to make the spreadsheet visible again.

Before you run these examples you *must* start Excel with a spreadsheet called SHEET1.XLS. It is possible to start Excel automatically with a particular spreadsheet by using the Exec(), ExecWait(), or ExecNoWait() commands. See the Commands section of the online help.

The ON ERROR command is used to ensure that the DDE channel is closed if an unexpected error occurs.

In this example:

- the client is the ChemStation,
- the server is EXCEL,
- the application is EXCEL,
- the topic is Sheet1.xls, and
- the item is the Excel Hide() function.

```
Name DDETest3

  local MyVar,MyString$

  !
  !!! Using an error trap is useful in order not to leave channels open
  !
  on error CloseDDE

  !
  !!! Initiation block
  !
  Chan = DDEInitiate("EXCEL", "Sheet1")               ! Open the channel

  !
  !!! Conversation block
  !
  DDEExecute Chan, "[Hide()]"                         ! Run the command
  !
```

Example 3: Executing a Command in Excel Through DDE

```
!!! Termination section
!
DDETerminate Chan                                    ! Close the channel

Return

EndMacro

!
!!! Error handling macro
!
Name Close DDE

Local Button
DDETerminate Chan                                    ! Close the channel
Button = Alert ("Stopped on error",3)                ! Print a warning
Return
EndMacro
```

Example 4: Setting Up a DDE Hotlink to Excel

This section describes the macro DDETest4:

- the macro DDETest4 opens a DDE channel with a Microsoft Excel spreadsheet called SHEET1.XLS, and
- the macro sets up a hotlink between row 5 column 6 in the spreadsheet and the ChemStation variable HotData$ and waits for the user to change the contents of row 5 column 6—when the data is changed the new data is automatically retrieved and printed by the ChemStation.

Before you run these examples you *must* start Excel with a spreadsheet called SHEET1.XLS. It is possible to start Excel automatically with a particular spreadsheet by using the Exec(), ExecWait(), or ExecNoWait() commands. See the Commands section of the online help.

The ON ERROR command is used to ensure that the DDE channel is closed if an unexpected error occurs.

In this example:

- the client is the ChemStation,
- the server is EXCEL,
- the application is EXCEL,
- the topic is Sheet1.xls, and
- the item is R5C6.

```
Name DDETest4

 local MyVar,MyString$

 !
 !!! Using an error trap is useful in order not to leave channels open
 !
 on error CloseDDE

 !
 !!! Initiation block
 !
 Chan = DDEInitiate("EXCEL", "Sheet1")              ! Open the channel

 !
 !!! Conversation block
 !
```

Example 4: Setting Up a DDE Hotlink to Excel

```
DDERequest Chan, "R5C6", OriginalData$              ! Get the original data

!
!!! Set the hot link variable to the original data so we can test for a change
!
HotData$ = OriginalData$

!
!!! Set up the hot link
!
DDEAdvise Chan, "R5C6", HotData$

!
!!! Loop waiting for the data to be changed by the user
!

While (HotData$ = OriginalData$) do

  Print "Waiting for row 5 column 6 to change"       ! Print a reminder
  Sleep 1                                           ! pause a second

EndWhile

Print "Hot link data changed from ",OriginalData$," to ",HotData$
!
!!! Termination section
!
DDETerminate Chan                                   ! Close the channel

Return

EndMacro

!
!!! Error handling macro
!
Name Close DDE

Local Button
DDETerminate Chan                                   ! Close the channel
Button = Alert ("Stopped on error",3)               ! Print a warning
Return
EndMacro
```

Summary

This section summarizes the main DDE points for the ChemStation.

DDE Levels

The previous examples show three levels of DDE communication:

Application	For example, EXCEL.
Topics	For example, SYSTEM, CPWAIT and CPNOWAIT in the ChemStation, and others such as the sheet1.xls in the Microsoft EXCEL spreadsheet.
Items	For example, the variables, macros and SYSTEM items for the ChemStation.

Summary

8

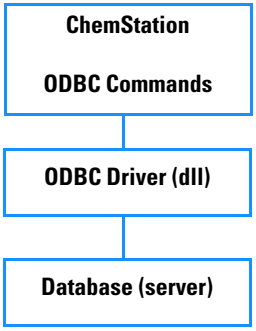
Open Database Connectivity (ODBC)

Open DataBase Connectivity (ODBC)

Open DataBase Connectivity (ODBC) is an interface defined by X/Open and SQL Access Group for the access to relational or non-relational databases. This access is independent of the provider of the database. It is based on the standards of the SQL Access Group, of X/Open, and of ANSI (American National Standard Institution). It uses a superset of the SQL (Structured Query Language). ODBC not only allows you to save data, open and query databases, it also offers functionality to provide these capabilities over local or global networks.

The advantage of using ODBC as a standard interface to database is that data can be exchanged between applications such as the database and a ChemStation without knowing how the data are managed on each side. It allows flexibility and independency of a particular type of database or vendors.

The following graph visualizes the principles of the ODBC interface between the ChemStation and a database which can be Microsoft® Access, DBase, or another database on a server such as Oracle:



Steps to Create Your Own Database

Set up Database

The first step is to set up the database according to your needs. You need to install the database and to create the database tables with all the field definitions and links which are needed.

Install ODBC Driver

The next step is to install the ODBC drivers. Your database product must offer an ODBC driver to work with the ChemStation. You will find an ODBC icon in the Control Panel of the Windows Program Manager. When selecting this ODBC icon the Data Sources dialog appears. Here you must select Add and choose the appropriate ODBC driver from the file system as instructed in your database documentation. A new dialog appears where you enter the name of the database and depending on the driver you must enter some more specific information.

Data Transfer

The transfer of data from the ChemStation to your database is done through the ODBC commands described in Chapter 14 “Summary of Commands”. This command set is installed into the ChemStation by using the InstallCmds() function (see the ChemStation commands). You will find some useful functions and variables in Chapter 14 “Summary of Commands”. The ODBC command to enter data into the database (DBNewRow) requires you to prepare a suitable table for the data using the macro programming language described in the ChemStation *Macro Programming Guide*. This manual also describes the ChemStation memory structure and variables that will allow you to find information you want to transfer. Look also at the example in the following section to get an idea how your program should look like.

Your macros must be stored in a file (for example, myodbc.mac) in the core directory of your ChemStation. To automate the data transfer your macro file has to be entered in the Post Run field of the Method Run Time Check List as

```
macro "myodbc.mac", go
```

User Examples

Receiving Data from ChemStore

This example will demonstrate how you can use the ODBC commands and macros to receive data from ChemStore by yourself with a postrun macro. In a similar way data can be retrieved from your own database which you might use instead of ChemStore. For this example parts of the sample table are transferred from ChemStore to the ChemStation.

This example comprises three parts:

- The initialization part, which loads the module containing the ODBC command set, opens the access to ChemStore and initializes some global variables. The prefix DBC_ (DataBaseCustomization) is used to distinguish between your macros and the standard ChemStation macros.

```
! Initializes access to database and prepares ResultReg register
Name DBC_InitChemStore
! Global DBC_ConNum  : Connectivity number
! Global DBC_DBPath$ : Path of database
  Local DatabasePath$, ODBCdll$
  Local Transaction$ ! Table name
  Local ODBC
  If Check(Function, DBOpen) = 0 then
    DatabasePath$ = ObjHdrText$(DBS_Config[1], "ChemStorePath")
    ODBCdll$ = ObjHdrText$(DBS_Config[1], "ODBCdll")
    ODBC = InstallCmds(DatabasePath$ + ODBCdll$)
  EndIf
  Transaction$ = ObjHdrText$(DBS_Config[1], "ODBCSource")
  DBC_DBPath$ = ObjHdrText$(DBS_Config[1], "DBPath")
  DBC_ConNum = DBOpen(Transaction$, "", "") ! Initialize access to ChemStore
  DelReg ResultReg; NewObj ResultReg, 1, 0, 0 ! to keep the results
  Return
EndMacro ! DBC_InitChemStore
```

- The macro which actually extracts sample information from ChemStore and transfers it into the results register of the ChemStation. This macro uses a parameter where the search pattern for the sample name can be entered.

```
   ! Returns sample table of database in Samples table of Results register
considering a given sample name content
   Name DBC_Samples
     Parameter Content$
     Local SQL$
     SQL$ = "SELECT Sample_Name, Sample_Type, Solvent, Sample_Info, Sample_Id
From Sample WHERE Sample_Name = '%"  Content$ + "%'"
     DBExec DBC_ConNum, SQL$, ResultReg, Samples ! Search and transfer results
     EdTab 1, ResultReg[1], "Samples" ! Display resulting sample table
   EndMacro ! DBC_Samples
```

- The command lines which call the initialization macro, allow to enter a sample name pattern and call the macro to perform the search and transfer. The result will be displayed with the table editor. The last command finally closes the connection to ChemStore

```
! Run the init macro immediatelly after loading
DBC_InitChemStore
ok = input("Please enter selection criteria for sample name:", Criteria$)
DBC_Samples Criteria$
DBClose DBC_ConNum
```

Writing Data to a Database

This example will demonstrate how you can use the ODBC commands and macros to write data from the ChemStation to a database. To keep it generic and simple a database in Access is used here, representing a compound table with compound name, amount, unit, and peak time. These specific compound results from the ChemStation are appended to this table.

This macro example consists of four parts.

- The initialization part which loads the module containing the ODBC command set, opens the access to your database, and initializes some global variables. The prefix DBC_ (DataBaseCustomization) is used to distinguish between your macros and the standard ChemStation macros.

```
! Initializes access to database and prepares ResultReg register
Name DBC_InitDatabase
! Global DBC_ConNum  : Connectivity number
! Global DBC_DBPath$ : Path of database
  Local DatabasePath$, ODBCdll$
  Local Transaction$ ! Table name
  Local ODBC
  If Check(Function, DBOpen) = 0 then
    DatabasePath$ = "c:\hpchem\database\"
    ODBCdll$ = "HPODBC00.dll"
    ODBC = InstallCmds(DatabasePath$ + ODBCdll$)
  EndIf
  Transaction$ = "Compound"
  DBC_DBPath$ = "c:\access\"
  DBC_ConNum = DBOpen(Transaction$, "", "")   ! Initialize access to compound
table in Access database
  DelReg ResultReg; NewObj ResultReg, 1, 0, 0 ! to keep the results
  Return
EndMacro ! DBC_InitDatabase
```

- The macro which is used for preparing the table and performing the transfer of this table to the database. The preparation of this table is done through selecting an empty table from the database. In this case just the table structure with the appropriate header items is returned and can be used for filling it up with your specific data, which finally are sent to the database.

```
   ! Prepares and fills the table used for the transfer
   Name DBC_PrepTable
     Local SQL$
     SQL$ = "SELECT * From Compound WHERE CompoundName = '@#%&?'" ! Provides
emty table
     DBExec DBC_ConNum, SQL$, ResultReg, Compounds ! Transfer empty compound
table
     DBC_FillCompoundsTable ! Fill compounds table with appropriate data
     Return
   EndMacro ! DBC_PrepTable
```

- The macro which just fills the pre-prepared table with the appropriate data.

```
   ! Prepares and fills the table used for the transfer
   Name DBC_FillCompoundsTable
     Local n, NoOfCompounds
     Local CompoundName$, Amount, Unit$, PeakTime
     NoOfCompounds = TabHdrVal(ChromRes[1], "Compound", "NumberOfRows")
     InsTabRow ResultReg, "Compounds", 1:NoOfCompounds
     For n = 1 to noOfCompounds
       CompoundName$ = TabText$(ChromRes[1], "Compound", n, "Name")
       SetTabText ResultReg, "Compounds", n, "CompoundName", CompoundName$
       Amount = TabVal(ChromRes[1], "Compound", n, "Amount")
       SetTabVal ResultReg, "Compounds", n, "Amount", Amount
       Unit$ = TabHdrText$(ChromRes[1], "QuantParm", "Units")
       SetTabText ResultReg, "Compounds", n, "AmountUnit", Unit$
       PeakTime = TabVal(ChromRes[1], "Compound", n, "FirstPeak~MeasRetTime")
       SetTabVal ResultReg, "Compounds", n, "PeakTime", PeakTime
     Next n
     Return
   EndMacro ! DBC_FillCompoundsTable
```

- The macro which appends the table from ChemStation to the appropriate table in the database.

```
   ! Prepares and fills the table used for the transfer
   Name DBC_AppendTable
     DBNewRow DBC_ConNum, ResultReg, "Compounds",, "Compound" ! Append table
   EndMacro ! DBC_AppendTable
```

- The command lines which call the initialization macro, the macro which prepares and fills the prepared table and the macro which appends the results to the compound table in the database and finally the command

which closes the connection to the database.

```
! Run the init macro immediatelly after loading
DBC_InitDatabase
DBC_PrepTable
DBC_AppendTable
DBClose DBC_ConNum
```

9

System Variables

System Variables

This chapter describes the ChemStation system string, scalar, and other predefined command processor variables.

String Variables

For some variables three copies of the same type exist. Example: _DATAFILE1$, _DATAFILE2$, _DATAFILE. This is information for the front (_DATAFILE1$) and rear (-DATAFILE2$) injectors. In LC/CE only _DATAFILE1$ is used. During execution of macros the current values are copied into the original variable (_DATAFILE$) for processing.

Table 7 **String Variables**

Variable	Description
_ABORT$	If present, it is assumed that this string variable contains a command or macro to be executed as the abort action. If not present, the standard command sequence "CLOSE; WINUPDATE;ABORTCLEANUP" is executed as the abort action.
_AUTOPATH$	Path where macros are located. Used as default in macro command. Copied from configuration database. Cannot be changed by CP.
_BARCODE$ _BARCODE$1 _BARCODE$2	Barcode read from a vial. Copied to rawdata file.
_CONFIGMETHPATH$	Configured method path. Copied from configuration database. Cannot be changed by CP.
_CONFIGSEQPATH$	Configured sequence path. Copied from configuration database. Cannot be changed by CP.
_DATAPATH$	Path of current instrument data directory.
_DATAFILE$ _DATAFILE1 _DATAFILE2	Name of current data file.
_DATANAME$ _DATANAME1$ _DATANAME2$	Variable used to set sample name for next run. Copied to raw data file.

Table 7 **String Variables, continued**

Variable	Description
_DATASUBDIR$	Subdirectory for data files during a simple method run. Current directory is _datapath$ + _datasubdir$.
_DATASEQSUBDIR$	Subdirectory for data files during a sequence. Current directory is _datapath$ + _dataseqsubdir$.
_DDENAME$	Name of application. Used to setup DDE communication. Cannot be changed by CP.
_DRIVERPATH$	Path of current instrument driver directory. Default c:\HPCHEM\DRIVERS
_ERRCMD$	Name of command. The variable is set if a command aborts with an error.
_ERRFILE$	Name of the macro file. The variable is set if a macro aborts with an error.
_ERRMACRO$	Name of the macro. The variable is set if a macro aborts with an error.
_ERRMSG$	Text of last error.
_EXEPATH$	Path where ChemStation software is located. Copied from configuration database. Cannot be changed by command processor (CP).
_FEATURES$	identifies installed software modules, for example LC3D
_INSTNAME$	Instrument name. Copied from configuration database. Cannot be changed by CP.
_INSTPATH$	Path of current instrument directory. Copied from configuration database. Cannot be changed by CP.
_LIBPATH$	Path of current library.
_LIBFILE$	Name of current library.
_License$	License number
_METHODINFO$	Method information of currently loaded method.
_METHPATH$	Path of current method.

Table 7 **String Variables, continued**

Variable	Description
_METHFILE$	Name of current method.
_OPERATOR$	Operator name. Used when operator name is copied to method and rawdata files.
_Pg_HeaderText$	Text which appears in the header of a report
_Pg_FooterText$	Text which appears in the footer of a report
_PRODUCTPATH$	Path were currently loaded software is located, for example C:\HPCHEM\LC
_PRODUCTTYPE$	Product Type Identifier (for example LC)
_SAMPINFO$ _SAMPINFO1$ _SAMPINFO2$	Sample information comment.
_SEQFILE$	Name of current sequence file.
_SEQPATH$	Path for current sequence file. Copied from the configuration database.
_SEQVERSION$	Version of sequence.
_TEMPPATH$	Path for temporary files. Copied from configuration database. Cannot be changed by CP.
_VERSION$	Version string.

Scalar Variables

For some variables three copies of the same type exist. Example: _RUNTYPE1, _RUNTYPE2, _RUNTYPE. This is information for the front (_RUNTYPE1) and rear (_RUNTYPE2) injectors. In LC/CE only _RUNTYPE1 is used. During execution of macros the current values are copied into the original variable (_RUNTYPE) for processing.

Table 8 **Scalar Variables**

Variable	Description
_ABAMULTISIGNAL	Integrator control variable. 0 Inhibits the signal line in the calibration table display.
_ABORT	Default is 0. When set to 1, aborting is triggered for all processes that are registered as Busy with the MIF. When no Off parameters are stacked for the SetAbort command and the _Abort system variable is set to 1 by DDE or a macro, the abort process is started. When the SetAbort command is Off, then the _Abort system variable is not set to 1 until the SetAbort command is set to On.
_Acquisition	0 Product uses acquisition 1 Product without acquisistion
_ALSBOTTLE _ALSBOTTLE1 _ALSBOTTLE2	–10 Blank run –200 Vial number

Table 8 **Scalar Variables, continued**

Variable	Description
_CALIBINJECT _CALIBINJECT 1 _CALIBINJECT 2	Set by sequence control during bracketing 1 Single calibration injection 2 First calibration injection of a series of calibration runs 3 Intermediate calibration injection 4 Last calibration injection Above numbers 1 and 2 are incremented by 10 if it is first calibration block of a new method with bracketing. Cannot be changed by CP.
_CALIBLEVEL _CALIBLEVEL1 _CALIBLEVEL2	0 Initial value >0 Calibration level Cannot be changed by CP
_CALIBLINE _CALIBLINE1 _CALIBLINE2	0 Initial value –1 Sample run >0 Calibration line during calibration run Cannot be changed by CP
_DILUTION _DILUTION1 _DILUTION2	Dilution factor specified
_ERRLINE	Line number in a macro file. Variable is set if a macro aborts with an error.
_ERROR	0 No error occurred 1 Error occurred
_ISTDAMT _ISTDAMT1 _ISTDAMT2	ISTD amount of current run, stored with rawdata
_INSTINSTANCE	Instance number of current instrument.
_INSTANCE	Instance number of application. Cannot be changed by CP

Table 8 **Scalar Variables, continued**

Variable	Description
_INSTRUMENT	1 Single instrument environment
	1–4 Multi-instrument environment
	Cannot be changed by CP
_MENU_MEMORY	Number of views the software keeps in memory. Default 4 for single instrument runs.
_METHODCHANGED	0 Method was not changed after load
	1 Method was changed after load
_METHODINJECTORS	2bit value. Defines number and position of available injectors
	1 Front injector or LC injector
	2 Rear injector (GC only)
	3 Front and rear injector (GC only)
_METHODON	0 Method not currently running
	1 Method currently running
_MultiInstrument	0 Single Instrument ChemStation
	1 Multi Instrument ChemStation
_MULTIPLIER _MULTIPLIER 1 _MULTIPLIER 2	Multiplier of current run, stored with rawdata
_OFFLINE	0 Application runs online
	1 Application runs offline
	Cannot be changed by CP
_Pg_BottomMargin	Margin at the bottom of a page given in number of page lines
_Pg_FooterLines	Number of lines used for the footer of the page
_Pg_HeaderLines	Number of lines used for the header of the page
_Pg_LeftMargin	Margin at the left of a page given in number of page columns
_Pg_TopMargin	Margin at the top of a page given in number of page lines

Table 8 **Scalar Variables, continued**

Variable	Description
_REPLICATE _REPLICATE1 _REPLICATE2	0 Initial value >0 Injection number Cannot be changed by CP
_RUNTYPE _RUNTYPE1 _RUNTYPE2	0 Initial value 1 Sample run 2 Blank run 3 Calibration run 4 Control Sample 5 Control Calibration Cannot be changed by CP
_SAMPAMT _SAMPAMT1 _SAMPAMT2	Sample amount of current run, stored with rawdata
_SEQINDEX _SEQINDEX1 _SEQINDEX2	0 Not in sequence >0 Sequence line during sequence Cannot be changed by CP
_SEQUENCE-CHANGED	0 Sequence was not changed after load 1 Sequence was changed after load
_SEQUENCEON	0 No sequence running 1 Sequence running Cannot be changed by CP
_SEQMODE	Part of Methods which will be processed in sequence 0 Runtime Checklist 1 Acquisition only 2 Data analysis only (using SAMPLE.MAC, ignoring Sample Table) 3 Data analysis only (using Sample Table)

Table 8 **Scalar Variables, continued**

Variable	Description	
_SEQRUNRESULT1	0	Sequence running
	1	Stop sequence, execute shutdown macro, output sequence summary report and execute post sequence macro.
_SEQRUNRESULT2	0	Sequence running
	1	Stop sequence, execute shutdown macro, output sequence summary report and execute postsequence macro.
_Spectrum	0	Spectrum module is not installed
	1	Spectrum module is installed
_UPDRF _UPDRF1 _UPDRF2	0	No update of response factor
	1	Average
	2	Replace
	3	Bracketing
	Cannot be changed by CP	
_UPDRT _UPDRT1 _UPDRT2	0	No update of retention time
	1	Average
	2	Replace
	Cannot be changed by CP	

Acquisition Variables

Table 9 **Acquisition Variables**

Variable	Description	
ACQSTATUS$	Variable contains the current instrument status. Valid strings are as follows:	
	Status	**Description**
	INITIALIZING	Set during initialization
	NOMODULE	No module configured
	OFFLINE	Currently in offline mode
	STANDBY	All modules in standby mode. All lamps and pumps are switched off
	PRERUN	Ready to start run
	INJECTING	Injecting
	PREPARING	Run is being prepared. For example, doing a balance
	RUN	Run is in progress
	POSTRUN	Postrun is in progress
	RAWDATA	Rawdata are still being processed following a run
	NOTREADY	Run cannot be started
	ERROR	Error occurred
	BREAK	Injection paused
ACQRUNTIME	Variable contains the current run time or stop time, depending on the state of the instrument.	
ACQSTOPTIME	Variable contains the longest stop time of all configured modules. See also ACQPOSTTIME.	

Table 9 **Acquisition Variables, continued**

Variable	Description
ACQPOSTTIME	Variable contains the post time that will *follow* the stop time (variable ACQSTOPTIME). The sum of ACQSTOPTIME + ACQPOSTTIME specifies the time of the whole instrument run.
ACQTIMEOUT	Variable contains the timeout value that specifies how long the modules will wait for a run before switching off their lamps and pumps. This timeout is set in all cases except when a sequence is running, (during a sequence the variable SEQTIMEOUT is used). Note: use the command SETACQTIMEOUT to set the variable ACQTIMEOUT, do not set the variable directly. Default timeout: 0.0 minutes.
ACQNOTREADYWAIT	Timeout variable specifies the maximum time the system waits for all modules to be ready, before a run is started. If the time elapses and at least one module is not ready, the run is aborted. Default: 10 minutes.
DEV1090PUMP	1 HP 1090 pump installed
	0 Pump not installed
DEV1090INJECT	1 HP 1090 injector installed
	0 Injector not installed
DEV1090OVEN	1 HP 1090 oven installed
	0 Oven not installed
DEV1090SAMPLER	1 HP 1090 autosampler installed
	0 Autosampler not installed
DEV1090DAD	1 HP 1090 DAD installed
	0 DAD not installed
DEV1090FPD	1 HP 1090 FPD installed
	0 FPD not installed
DEV1049ECD	1 HP 1049 ECD installed
	0 ECD not installed

Table 9 **Acquisition Variables, continued**

Variable	Description	
DEV1046FLD	1	HP 1046 FLD installed
	0	FLD not installed
DEV1050VWD	1	HP 1050 VWD installed
	0	VWD not installed
DEV1050DAD	1	HP 1050 DAD installed
	0	DAD not installed
DEV1050MWD	1	HP 1050 MWD installed
	0	MWD not installed
DEVADC	1	HP 35900C ADC installed
	0	ADC not installed
DEV1050SAMPLER	1	HP 1050 autosampler installed
	0	Autosampler not installed
DEVBARCODE	1	Barcode reader option for the HP 1050 Autosampler installed
	0	Barcode reader not installed
DEV1050HEATER	1	HP 1050 column heater installed
	0	Heater not installed
DEVHPCE	1	HP ^{3D}CE instrument installed
	0	HP ^{3D}CE not installed
_InjectionSource$	HP 7673 (Product name of GC autoinjector) or Manual or Front or Back or Dual. String used to identify GC injector.	
_GenieMethod	0	Use old integrator
	1	Use new Genie integrator

Table 9 **Acquisition Variables, continued**

Variable	Description
SAMPLER	This variable is set if an autosampler is part of the instrument. It reflects the highest vial number as follows: 100 HP 1090 autosampler 21 HP 1050 autosampler with 21 vial tray 34 HP 1050 autosampler with 34 vial tray 200 HP 1050 autosampler with 100 vial tray 48 Autosampler for HP ^{3D}CE instrument
_SlaveIdle	0 No ChemStation control from a different application 1 ChemStation is controled from another application. Certain menu items will be disabled to prevent user interaction on the ChemStation.
USEEXTERNALINJECTOR	If this variable is defined and set to 1, for example, within a prerun macro, the ChemStation does not start the HP 1090 or HP 1050 autosampler. In this case, the autosampler can be started by an external injector through a remote Start contact closure. If the variable is not defined or defined but set to 0, the HP 1090 or HP 1050 autosampler is started by the ChemStation.

Runtime Checklist Variables

Table 10 **Runtime Checklist Variables**

Variable	Description
LASTSEQRUN LASTSEQRUN1 LASTSEQRUN2	The last run was a sequence 0 False 1 True
NEWSEQMETHOD NEWSEQMETHOD1 NEWSEQMETHOD2	Is set for each run during a sequence. Can be used to check whether the run is the first after a method has been loaded. Note that a method is not loaded before the first run if the first sequence line does not have a method name defined, and after a Resume Sequence the method is reloaded according to the current sequence line. 1 Method has just been loaded 0 Method was not loaded
RUN_ACQ	Run data acquisition. 1 When checked in the method runtime checklist 0 When not checked in the method runtime checklist
RUN_CUSTDA	Run customized data analysis When checked in the method runtime checklist 0 When not checked in the method runtime checklist
RUN_CUSTDA$	Name of the command string for customized data analysis
RUN_DA	Run standard data analysis 1 When checked in the method runtime checklist 0 When not checked in the method runtime checklist
Run_Inj2Method$	Method name of second injector method in GC

Table 10 **Runtime Checklist Variables, continued**

Variable	Description
RUN_POSTRUN	Postrun a command or macro
	1 When checked in the method runtime checklist
	0 When not checked in the method runtime checklist
RUN_PRERUN	Prerun a command or macro
	1 When checked in the method runtime checklist
	0 When not checked in the method runtime checklist
Run_PostRun$	Name of postrun macro
Run_PreRun$	Name of prerun macro
RUN_SAVEGLPDATA	Save GLP data
	1 When checked in the method runtime checklist
	0 When not checked in the method runtime checklist
RUN_SAVEMETHOD	Save a copy of the method with data
	1 When checked in the method runtime checklist
	0 When not checked in the method runtime checklist

Other Predefined Command Processor Variables

Table 11 **Other Predefined Command Processor Variables**

Variable	Description
DADATAPATH$	Path variable used in data analysis to hold current data path name. Copied from _DATAPATH$, _DATASUBDIR$, _DATASEQSUBDIR$, or set by file I/O command set.
DADATAFILE$	File name variable used in data analysis. Copied from _DATAFILE$ or set by file I/O command set.
FREEDISKLIMIT	Disk limit set
MANAGERLEVELACCESS	0 Access level set to Operator
	1 Access level set to Manager
METHOD	Method status variable
	0 Method idle
	1 Method running
METHENDSTATUS	Method termination mode
	0 Normal termination
	1 Method stopped
	2 Method aborted
	3 Method aborted due to an error
NEWSEQMETHOD NEWSEQMETHOD1 NEWSEQMETHOD2	During the sequence this is method has just been loaded
	0 False
	1 True
SEQTIMEOUT	Sequence timeout value
SEQWAIT	Sequence wait time value

Table 11 Other Predefined Command Processor Variables, continued

Variable	Description
SEQENDSTATUS	Sequence termination mode
	0 Normal termination
	1 Sequence stopped
	2 Sequence aborted
	3 Sequence aborted due to an error

10

System Registers

System Registers

This chapter describes the ChemStation's system registers.

ChemStation System Registers

The ChemStation uses several predefined registers. Macros rely on the names of these registers and you must take care when accessing these registers through macros.

_Config Register

This register holds status information about windows, colors, axis styles, views, report styles. It also holds flags that can be set as preferences for the interactive session.

None of the information in this register belongs to the method, LoadMethod does not change this register. The _Config register will be loaded from the standard CONFIG.REG file (or CONF_OFF.REG if the application is the offline copy) upon startup. These two (binary) files exist once per instrument and are stored in the instrument directory for example C:\HPCHEM\1.

If at the time of startup the standard CONFIG.REG file does not exist, the _Config register is read from a default CONFIG.REG file. This CONFIG.REG file exists in the same directory as the other system macros that must not be changed by the user. C:\HPCHEM\CORE by default.

Table 12 shows the predefined header items in the first object of _Config[1].

Table 12 **Predefined Header Items in _Config[1]**

Item Name	Type/Range	Meaning
DADataPath	String	Path name of the data analysis directory — in which the current rawdata directory is located (set by File command) — which is used by the File command as directory when displaying the dialog box for file selection (initialized to _DATAPATH$ upon installation), and which is used by the File command as directory path if the given parameter in the File command is not a full path name, i.e. it does not start with a "\" or a drive selector, e.g. "C:" (set by RunDAMethod macro from _DATAPATH$ and _DATASUBDIR$ or _DATASEQSUBDIR$). The path always ends with a "\".
DADataFile	String	File name of current rawdata directory, e.g. DEMODAD.D — set by File command.
DefDet	String	Default detector for LoadChrom, LoadSpectrum, LoadSpectra commands — set by File command.
LoadAndInteg	Boolean	Value of checkbox “Integrate after signal load” in dialog box of File command.
LoadAndReport	Boolean	Value of checkbox "Print report after signal load" in dialog box of File command.

_Config[2] holds empty templates for most tables that have a predefined structure (for exceptions, see _Config[4] and _Config[5]). These empty tables have all predefined header items and columns with the correct names, types, access permission, error check information, default values, and so on. They do not have any rows except row 0.

The templates are used to create a predefined table from scratch with the NewTab command. The user may add other table templates for their own use.

_Config[3] holds the display description tables. These are used by the table editing commands. The display description tables specify the layout of a table when displayed for editing. The same object also holds the output description tables (ODT). These are an extension of the DDTs and are used

by the PipeTab command. The ODTs specify the layout of a table when printed.

_Config[4] holds empty templates for the 10 tables of the calibration data.

_Config[5] holds empty templates for the 7 tables of the chromatographic results.

_Config[6] holds the control tables for the PipeTab command. These tables cannot be edited or listed.

_DAMethod Register

This register holds the Data Analysis Method. For example, the integrator events tables, the calibration data and the report specifications.

SaveMethod saves this register to the binary DAMETHOD.REG file within the .M directory that the user specified for the complete method and saves the rest of the method. LoadMethod loads this register from the DAMETHOD.REG file and loads the rest of the method.

Tables in _DAMethod[1]

Integrator Events Comprises one table for each signal. The naming convention is “Event_” + Detector + SignalId for signal specific events and “_Event” for the default event table. When assocated with the enhanced integrator, each signal will also have method and detector personality tables which are described below.

Method Personality Comprises one table for each signal for the enhanced integrator. The naming convension is IntMethod_ + Instrument Type for the default method personality table.

Detector Personality Comprises one table for each signal for the enhanced integrator. The naming convension is IntDetect_ + Detector Class for the default detector personality table.

Calibration Data The calibration data is structured in tables as shown in Table 13.

Table 13 **Structure of Calibration Data in _DA Method[1]**

Data/Table Name	Description
Quantification parameters QuantParm	Default sample information, "method-wide" parameters and defaults
Compound groups CompGroup	List of compound groups
Compounds Compound	List of calibrated compounds
ISTD compounds ISTD	List of ISTD compounds
Expected peaks Peak	List of expected peaks
Signals Signal	List of used signals
Time references TimeRefGroup	List of time references (may contain several time reference groups)
Calibrated levels Level	List of level identifiers
Calibration points Calpoint	List of points on calibration curves
Calibration runs CalRun	List of calibration run results ("history", stored on request)
Error report ErrorReport	List of calibration errors

The tables in the calibration data contain references to each other. For example, a peak contains a reference (link) of the corresponding compound. Changes in a table, for example insert, automatically correct all references in all other tables.

Report Specification A table that contains settings for the report layout. Name: RptParm.

Automatic Library Search Parameters A table used for peak identification in the spectra library when the report style is Library Search. Name: AutoID

Noise and Statistic Parameters Stored in a table. Name: SysSuit

The second object of _DAMethod temporarily holds the calibration tables in the case of a sequence with bracketing. The tables are stored during the processing of calibration runs. They are copied to _DAMehtod[1] when the calibration runs, plus the preceding group of samples are finished, and the next group of samples is ready to be processed.

ChromRes Register

This register holds the chromatographic results of one data analysis run. The major part of the register consists of quantitative results,i.e. the output of the quantitative algorithms in the IdentifyPeaks and QuantifyPeaks commands. The quantitative results comprise the tables shown in Table 15. The ChromRes register is created when the signals for the raw data file are loaded. The results collected throughout data analysis are used by Print Report.

Table 14 shows the predefined header items in ChromRes[1].

Table 14 **Predefined Object Header Items in the Object ChromRes[1]**

Item Name	Type/Range	Meaning
Title	String	Results
AcqInstName	String	Instrument name during acquisition of raw data, (does not exist if the data file is old)
AcqVersion	String	Version of the software doing the acquisition of raw data
Version	String	Version of the software
MethodInfo	String	Method comment (copy of the comment in the Method Information box)
SeqLine	Integer	Sequence line during acquisition of raw data

Table 14 **Predefined Object Header Items in the Object ChromRes[1], continued**

Item Name	Type/Range	Meaning
Vial	Integer	Vial number during acquisition of raw data
Inj	Integer	Injection number during acquisition of raw data
BarCode	String	Barcode during acquisition of raw data
AcqMeth	String	Method during acquisition of raw data
AcqMethModTime	Time since 1970	Revision time (time of last save) of method used during acquisition of raw data. (Does not exist if data file or method is old.)
NewMethModBy	String	Operator that last saved the method used during acquisition of raw data. (Does not exist if data file or method is old.)
AcqMethModified	Boolean	Flag for whether the method used during acquisition of raw data was modified after loading. (Does not exist if data file or method is old.)
AcqOp	String	Operator during acquisition of raw data
InjDatTime	String	Injection date and time of raw data
SampleName	String	Sample name of raw data
AcqInstName	String	Instrument name during acquisition of raw data
SeqPathAndFile	String	Sequence file (including. path); only applicable if data analysis runs in a sequence
WrongSignals	Boolean	Flag; only exists if a signal description mismatch between raw data and calibration table was detected
AcqVialUnused	Boolean	0 Vial actually used
		1 Injector program did not use vial

Table 15 **Quantitative Result Tables in ChromRes Register**

Data/Table Name	Description
Quantification parameters QuantParm	Sample information and method wide quantification parameters.
Signals Signals	Signal information
Groups Groups	Compound group information
Compounds Compounds	Compound table (found and not found)
Peaks Peak	Peak table (identified and not identified)
ISTDs ISTD	Internal standards
Time References TimeRefGroup	Time references
Error report ErrorReport	Calibration errors

Each table row contains references (link) to corresponding rows in other tables. This allows navigation in the results. (for example, each peak contains a link to the corresponding compound.) The references across the tables are corrected automatically when the tables are manipulated (for example, rows are added or deleted).

LCDiag Register

The LCDiag register is not used in the HP ^{3D}CE ChemStation.

This register holds diagnostic information about the instrument. The register is generated by the device drivers of an HP 1090 and HP 1050 instruments at the end of an analysis. The contents of this register are stored in the file LCDIAG.REG, which is part of the data file.

ChromReg Register

This register is the work register for chromatograms, and is typically created with LoadSignal. The macros will usually expect to find the chromatograms they are working on in this register, and/or will deliver the resulting chromatograms to this register.

ChromReg2 Register

This register serves as a backup for ChromReg. The LoadFile macro will clear ChromReg2 and move all objects from ChromReg to ChromReg2.

SpecReg Register

This register is the work register for spectra. The macros will usually expect in here the spectra they are working on and/or will deliver the resulting spectra in here.

SpecReg2 Register

This register serves as a backup for SpecReg. The File command clears SpecReg2 and moves all objects from SpecReg to SpecReg2.

Deleted Register

This register serves as a store for deleted objects to enable an Undo Delete feature which can be used by the isoabsorbance plot dialog box.

Sweep Register

The Sweep Register is generated by the device driver of the HP 1049 Electrochemical Detector at the end of an analysis in Sweep mode. The voltamogram generated in sweep mode is stored in the file SWEEP.REG, which is part of the data file.

DADTest Register

The DADTest Register contains the results of a DAD test, with the lamp intensity and holmium spectrum stored in separate, *paired* objects. Each pair of objects is given the same *datetime* entry in the header item. Both objects in a pair are standard UV-spectra objects with additional object header items.

FLDScans Register

The FLDScans Register contains the results of a Fluorescence Detector excitation scan, emission scan, or both. The results of excitation and emission scans are stored in separate objects. If an excitation scan and an emission scan are saved as paired objects at the same time, they are given the same *DateTime* entry in the header item. Both objects are standard UV-spectra objects with additional object header items.

GLPSave Register

The GLPSave register contains the following information:

- chromatogram,
- integration results,
- quantification data,
- instrument performance data, and
- method details.

This information is saved to GLPSAVE.REG at the end of each analysis, if the appropriate checkbox in the Run Time Checklist is activated. A data file, protected by a binary checksum provides both security, and an audit trail for GLP purposes.

PeakPerfReg

This register holds information about a peak's performance, generated and used by system suitability.

SeqInfoReg

This register holds information generated during a sequence, and is then used in the sequence summary report.

Table 16 **Predefined Object Header Items in the Object SeqInfoReg[1]**

Item Name	Type/Range	Meaning
DateTime	String	Start date and time of sequence
ConfigPage	Integer	Page index for configuration listing
SequencePage	Integer	Page index for sequence table listing
LogbookPage	Integer	Page index for logbook listing
CalStatPage	Integer	Page index of statistic results listing of calibration runs
SmplStatPage	Integer	Page index of statistic results listing of sample runs

The object SeqInfoReg[1] also contains the tables shown in Table 17.

Table 17 **Tables in SeqInfoReg[1]**

Data/Table Name	Description
Methods	Keeps track of the methods used
ResultFiles	Keeps track of the data file names
"CompTab" + MethodNum	These tables keep track of compounds through the method runs

WorkReg

This register is used by the custom report generator. The object SeqInfoReg[1] contains the following tables:

Table 18 **Tables in SeqInfoReg[1]**

Data/Table Name	Description
Proposals	Automatic library search results
RepeatItems	Custom report repeat items
Totals	Custom report totals
SumTotal	Custom report sum totals
StatItems	Custom report limits
Template	Custom report template

_PartialSequence Register

This register is created from a template stored in object 2 of the _Sequence register when the Partial Sequence dialog is first displayed. This register contains one line of information for each sample run.

_SEQUENCE Register

All sequence settings are stored in one register the _SEQUENCE register. It contains two objects which comprises five tables.

Table 19 **Predefined Header Items in the First Object of _SEQUENCE**

Item Name	Type/Range	Meaning
SeqModTime	Integer	Time_t value when this sequence was last changed
SeqModBy	String	The operator who modifies the sequence (the value is set to value of _OPERATOR$).
SeqVersion	String	DLL version with which the sequence was created

Table 19 **Predefined Header Items in the First Object of _SEQUENCE, continued**

Item Name	Type/Range	Meaning
Producttype	String	Product type the sequence was created on
SeqReport	String	For sequence summary report
SeqStatistics	String	For sequence summary report.
SeqTable1	Table	Sequence table
SeqTable2	Table	Sequence table for second injector in GC
SeqParm	Table	Sequence parameters
PartialSequence	Table	Partial sequence table

HP 1100 Method Parameters Registers

The methods (acquisition part) as they are loaded from the HP 1100 modules (using specific Upload commands) are stored in the first object as object header items of a register which is related to the appropriate module. The module and its number is reflected in the name of the register. After modifying the method parameters in the register the register has to be downloaded to the appropriate module to make these modifications effective.

The second and third object of this register contains internal information related to the modules (control id's and format strings).

LeoAlsMethod<n>

(where <n> is the module number)

Table 20 **ALS Parameters in First Object as Object Header Items**

Name	Type	Description
DRAWSPEED	Integer	Speed to draw sample
EJECTSPEED	Integer	Speed to eject sample
STOPTIME	Float	0 means no limit 1 means as pump
POSTTIME	Float	0 means off
INJVOLUME	Float	Injection volume
DRAWPOSITION	Float	Height of needle above seat
INJECTMODE	Integer	1 standard injection 2 injection with needle wash 3 use injector program
WASHVIAL	Integer	Number of vial used for needle wash

Table 20 **ALS Parameters in First Object as Object Header Items, continued**

Name	Type	Description
PROGTABLE	Table	Injector Program
CONTACT_<x>	Boolean	TRUE if contact is closed <x> represents contact number 1 – 4
CONTACT_TT	Table	Time table for contacts

LeoThermostatMethod<n>

(where <n> is the module number)

Table 21 **Thermostat Method Parameters in First Object as Object Header Items**

Name	Type	Description
LeftTemp	Float	Temperature (left heat exchanger) <= -274 means ambient
RightTemp	Float	Temperature (right heat exchanger) <= -274 means ambient
LeftReady	Float	Ready range (left heat exchanger) 0 means off
RightReady	Float	Ready range (right heat exchanger) 0 means off
TwoTemperatures	Boolean	TRUE if 2 temperatures used (separated heat exchangers)
Stoptime	Float	Stoptime 0 means no Limit
Posttime	Float	Posttime 0 means off
StoptimeAsPump	Boolean	TRUE if stoptime as pump or injector
StoreLeftTemp	Boolean	TRUE if left temperature shall be stored in lcdiag.reg

Table 21 **Thermostat Method Parameters in First Object as Object Header Items, continued**

Name	Type	Description
StoreRighttTemp	Boolean	TRUE if right temperature shall be stored in lcdiag.reg
ColumnValve12	Boolean	Position of the column switching valve FALSE=Position 1 – 6 TRUE=Position 1 – 2
CONTACT_<x>	Boolean	TRUE if contact on <x>represents contact number 1 – 4
TimeTable	Table	Tiimetable of temeratures
Contact_TT	Table	Timetable of contacts

LeoPumpMethod<n>

(where <n> is the module number)

Table 22 **Pump Method Parameters in First Object as Object Header Items**

NDR Name	Type	Description
PRIM_CHANNEL	Integer	Primary channel of quat. pump
STROKE_A	Integer	Stroke of channel A (left) of binary pump
STROKE_B	Integer	Stroke of channel B (right) of binary pump
COMPRESS_A	Integer	Compressibility of channel A of binary pump
COMPRESS_B	Integer	Compressibility of channel B of binary pump
COMPESS_ISO	Integer	Compressibility of isocratic and quaternary pump
FLOW	Float	Flow
STOPTIME	Float	Stoptime
POSTTIME	Float	Postime
SOLV_RATIO_A	Float	%A composition

Table 22 **Pump Method Parameters in First Object as Object Header Items, continued**

NDR Name	Type	Description
SOLV_RATIO_B	Float	%B composition
SOLV_RATIO_C	Float	%C composition
SOLV_RATIO_D	Float	%D composition
SOLV_NAME_A	String	Solvent name in channel A
SOLV_NAME_B	String	Solvent name in channel B
SOLV_NAME_C	String	Solvent name in channel C
SOLV_NAME_D	String	Solvent name in channel D
PRESSURE_MIN	Integer	Minimum pressure limit
PRESSURE_MAX	Integer	Maximum pressure limit
FRAMP	Float	Flow ramp
SSV_A	Boolean	Solvent selection valve of channel A
SSV_B	Boolean	Solvent selection valve of channel B
STORE_RATIO_A	Boolean	Flag indicating whether gradient signal of %A will be stored
STORE_RATIO_B	Boolean	Flag indicating whether gradient signal of %B will be stored
STORE_RATIO_C	Boolean	Flag indicating whether gradient signal of %C will be stored
STORE_RATIO_D	Boolean	Flag indicating whether gradient signal of %D will be stored
STORE_FLOW	Boolean	Flag indicating whether flow signal will be stored
STORE_PRESS	Boolean	Flag indicating whether pressure signal will be stored

Table 22 **Pump Method Parameters in First Object as Object Header Items, continued**

NDR Name	Type	Description
TIMETABLE	Table	Time Table
Contact_TT	Table	Time table of contacts
CONTACT_<x>	Boolean	TRUE if contact on <x> represents contact number 1 – 4

LeoDADMethod<n>

(where <n> is the module number)

Table 23 **DAD Parameters in First Object as Object Header Items**

Name	Type	Description
Sample<x>Wl	Float	Sample wavelength of Signal x (x=A,B,C,D,E)
Sample<x>Bw	Float	Sample bandwidth of Signal x (x=A,B,C,D,E)
Reference<x>Wl	Float	Reference wavelength of Signal x (x=A,B,C,D,E) 0 means that the reference is off
Reference<x>Bw	Float	Reference bandwidth of Signal x (x=A,B,C,D,E)
Reference<x>On	Boolean	TRUE if the reference is on
StoreSignal<x>	Boolean	TRUE if Signal x shall be stored (x=A,B,C,D,E)
StoreSpectra	Enumeration	0 = None 1 = Apex + Baselines 2 = Apex + Slope + Baselines 3 = All in peak 4 = Every second 5 = All
RangeFrom	Float	Lower wavelenght limit for stored spectra
RangeTo	Float	Upper wavelenght limit for stored spectra
RangeStep	Float	Wavelength step for stored spectra
Threshold	Float	Threshold for stored spectra

Table 23 **DAD Parameters in First Object as Object Header Items, continued**

Name	Type	Description
Peakwidth	Enumeration	0 = < 0.01 min 1 = > 0.01 min 2 = > 0.03 min 3 = > 0.05 min 4 = > 0.1 min 5 = > 0.2 min 6 = > 0.4 min 7 = > 0.85 min
Slit	Enumeration	0 = 1 nm 1 = 2 nm 2 = 4 nm 3 = 8 nm 4 = 16 nm
Stoptime	Float	Stoptime 0 means no Limit
Posttime	Float	Posttime 0 means off
StoptimeAsPump	Boolean	TRUE if stoptime as pump or injector
UVLampRequired	Boolean	TRUE if UV lamp- is required
VisLampRequired	Boolean	TRUE if Vis lamp- is required
PrerunBalance	Boolean	TRUE if prerun balance is on
PostrunBalance	Boolean	TRUE if postrun balance is on
BalanceMode	Integer	0 = range - <Headroom> to max. Absorbance range 1 = full Absorbance range
Headroom	Integer	Used if BalanceMode = 0
Analog1Zero	Integer	Zero offset for analog output 1
Analog2Zero	Integer	Zero offset for analog output 2
Analog1Att	Enumeration	Attenuation for analog output 1 Full range output is 2 (attn-10) AU, i.e 11 = 2000 mAU, 10 = 1000 mAU, …, 0 = 0.98 mAU

Table 23 **DAD Parameters in First Object as Object Header Items, continued**

Name	Type	Description
Analog2Att	Enumeration	Attenuation for analog output 2 Same range as Analog1Att
CONTACT_<x>	Boolean	TRUE if contact on <x> represents contact number 1..4
Timetable	Table	Timetable for signals
Contact_TT	Table	Timetable for contacts

LeoVWDMethod<n>

(where <n> is the module number)

Table 24 **VWD Method Parameters in First Object as Object Header Items**

Name	Type	Description
Wavelength	Float	Wavelength
Peakwidth	Enumeration	0 = < 0.005 min 1 = > 0.005 min 2 = > 0.010 min 3 = > 0.025 min 4 = > 0.05 min 5 = > 0.1 min 6 = > 0.2 min 7 = > 0.4 min
Stoptime	Float	Stoptime 0 means no Limit
Posttime	Float	Posttime 0 means off
StoptimeAsPump	Boolean	TRUE if stoptime as pump or injector
PrerunBalance	Boolean	TRUE if prerun balance is on
PostrunBalance	Boolean	TRUE if postrun balance is on

Table 24 **VWD Method Parameters in First Object as Object Header Items, continued**

Name	Type	Description
BalanceMode	Integer	0 = range is - <headroom> to max. Absorbance range 1 = full Absorbance range
Headroom	Integer	used if BalanceMode = 0
Store	Integer	bit 0x02 = Store signal w/o reference, if TRUE bit 0x04 = Store reference only, if TRUE
AnalogZero	Integer	Zero offset for analog output
AnalogAtt	Enumeration	Attenuation for analog output Full range output is 2 (attn-10) AU 12 = 4000 mAU, 11 = 2000 mAU, ... , 0 = 0.98 mAU
CONTACT_<x>	Boolean	TRUE if contact on <x> represents contact number
Timetable	Table	Timetable of wavelength switching
Conatct_TT	Table	Timetable of contacts

11

Standard Objects

Standard Objects

This chapter describes the ChemStation's standard object header items in the various object classes.

Standard Object Header Items

When the ChemStation generates data objects by reading them from a rawdata file, it sets several standard object header items. Most of these are write-protected to ensure an audit trail for Good Laboratory Practice (GLP).

Header Items in Chromatogram Objects

Table 25 shows the predefined object header items in a chromatogram object as they are generated by, for example, the LoadSignal command.

Table 25 **Object Header Items in a Chromatogram Object**

Item Name	Type/Range	Access	Meaning
Title	String	r/w	Used e.g. in graphics, will be equal to SignalDesc plus the (path-stripped) RawDataFile upon creating the object
Operator	String	Read	Operator name, taken from rawdata
DateTime	String date	Read	Date and time of measurement, starting point of series of measurements, or date of injection taken from rawdata.
DataType	Enumeration	Read	Data type of signal 0 (UNK) Unknown 1 (ABS) Absorbance 2 (INT) Intensity 3 (TRANS) Transmission 4 (FLUOR) Fluorescence, phosphorescence, or chemiluminescence 12 (VOLT) Voltage 13 (CURR) Current 14 (POWER) Power

Table 25 **Object Header Items in a Chromatogram Object, continued**

Item Name	Type/Range	Access	Meaning	
DerivOrder	Integer ≥ 0	Read	Derivative order (0 if no derivative)	
ObjClass	Enumeration	Read	Class of object to identify it as chromatogram	
			3	GC chromatogram
			4	LC chromatogram
SampleName	String	Read	Sample name from sequence	
BarCode	String	Read	Barcode from injection	
MethodFile	String	Read	Method file name during acquisition	
SeqLine	Integer ≥ 0	Read	Sequence line (0 if not in sequence)	
Vial	Integer ≥ –1	Read	Vial number, –1 if unknown	
Inj	Integer ≥ 0	Read	Injection number (0 if not in sequence)	
Detector	String	Read	For example, DAD1, ADC2, MWD1	
SignalId	String 1 character	Read	For example, A, B, C	
SignalDesc	String	Read	For example, DAD1 A, Sig=254,4 Ref=450,80	
RawdataFile	String	Read	Rawdata directory name (.D), including full path	
SignalSource	Enumeration	Read	0	Signal is in a .CH file
			1	Signal is in a .UV file
			2	Signal is in LCDIAG.REG file

Table 25 **Object Header Items in a Chromatogram Object, continued**

Item Name	Type/Range	Access	Meaning
Start	Numeric	Read	Signal start time in minutes relative to start of run (DateTime)
End	Numeric	Read	Signal end time in minutes relative to start of run (DateTime)
BunchValue	Numeric	Read	Detector-dependent internal value used by the integrator
SolventSlope	Numeric	Read	Detector-dependent internal value used by the integrator
SolventSplit	Numeric	Read	Detector-dependent internal value used by the integrator

Header Items in Spectrum Objects

Table 26 shows the predefined object header items in a spectrum object as they are generated by, for example, the LoadSpectrum command.

Table 26 **Object Header Items in a Spectrum Object**

Item Name	Type/Range	Access	Meaning
Title	String	R/W	Used e.g. in graphics and will include Time plus (path-stripped) RawDataFile upon creating object
Operator	String	Read	Operator name, taken from rawdata
DateTime	String	Read	Date and time of measurement, starting point of series of measurements, or date of injection taken from rawdata
Time	Numeric	Read	Time of spectrum in minutes as offset to DateTime

Table 26 **Object Header Items in a Spectrum Object, continued**

Item Name	Type/Range	Access	Meaning	
DataType	Enumeration	Read	Data type of signal	
			0 (UNK)	Unknown
			1 (ABS)	Absorbance
			2 (INT)	Intensity
			3 (TRANS)	Transmission
			4 (FLUOR)	Fluorescence phosphorescence, or chemiluminescence
DerivOrder	Integer ≥ 0	Read	Derivative order (0 if no derivative)	
ObjClass	Enumeration	Read	Class of object to identify it as spectrum	
			5	UV spectrum
SampleName	String	Read	Sample name from sequence	
IntegrTime	Numeric > 0	Read	Integration time (exposure time) in ms	
BarCode	String	Read	Barcode from injection	
RawdataFile	String	Read	Rawdata directory name (.D), including full path	
MethodFile	String	Read	Method file name during acquisition	
SeqLine	Integer ≥ 0	Read	Sequence line (0 if not in sequence)	
Vial	Integer ≥ –1	Read	Vial number, –1 if unknown	
Inj	Integer ≥ 0	Read	Injection # (0 if not in sequence)	
Detector	String	Read	For example, DAD1, ADC2, MWD1	
Index	Integer > 0	Read	Spectrum index on rawdata file (starts with 1)	

Table 26 **Object Header Items in a Spectrum Object, continued**

Item Name	Type/Range	Access	Meaning
Attribute	Enumeration	Read	Detector-specific spectrum attribute
			If Detector = DAD, this is the peak detector code of spectrum:
			0x0034 Inflection point on upslope with rising curvature
			0x0035 Inflection point on upslope with falling curvature
			0x0036 Inflection point on downslope with rising curvature
			0x0037 Inflection point on downslope with falling curvature
			0x0041 Top of peak
			0x0043 Cancellation of peak
			0x0044 Valley
			0x0045 End of peak
			0x004d Manually taken
			0x0050 Periodically taken
			If Detector = FLD, this is the scan type of the spectrum:
			0x0001 Excitation scan
			0x0002 Emission scan

Object Header Items in User-defined Objects

Table 27 shows the predefined object header items in a user-defined object.

Table 27 Object Header Items in a User-defined Register

Item Name	Type/Range	Access	Meaning
Title	String	R/W	Used e.g. in graphics
ObjClass	Enumeration	Read	Class of object to identify it as user defined
			1 User defined

Object Header Items in Matrix Objects

Table 28 shows the predefined object header items in a matrix object.

Table 28 Object Header Items in a Matrix

Item name	Type/range	Access	Meaning
Title	String	R/W	Used e.g. in graphics
ObjClass	Enumeration	Read	Class of object to identify it as matrix
			2 Matrix

The LoadSpectra command generates object header items similar to those of a spectrum object, as far as they are meaningful.

Object Header Items in LCDiag Objects

The LCDiag objects are only used for LC ChemStations.

Table 29 **Object Header Items in Start/Stop Conditions Object**

Object Header Item	Text/Value
Title	"LC, Start/Stop Conditions" (HP 1090)
	"Pump, Start/Stop Conditions" (HP 1050)
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation
InjVolume	Only for 1090:
	Injection volume if >= 0, else:
	–1 If blank run
	–2 If manual injection
	–3 If injector program
StartPressure	High pressure at start run time
StartTemperature	Temperature at start run time
StopPressure	High pressure at stop run time
StopTemperature	Temperature at stop run time
StartFlow	Flow at start run time
StopFlow	Flow at stop run time

Start/Stop Conditions Object

Injection volume is measured in µl, pressure in bar, and temperature in °C. There is no data block.

Flow vs. Time Object

Table 30 **Object Header Items in Flow vs. Time object**

Object Header Item	Text/Value
Title	"LC, Flow" (HP 1090)
	"Pump, Flow" (HP 1050)
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation

Data block:

y-axis: Flow [ml/min]

x-axis: Time [min]

Pressure vs. Time Object

Table 31 **Object Header Items in Pressure vs. Time Objects**

Object Header Item	Text/Value
Title	"LC, Pressure" (HP 1090)
	"Pump, Pressure" (HP 1050)
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation

Data block:

y-axis: High Pressure [bar]

x-axis: Time [min]

Table 32 **Object Header Items in Low Pressure 1 vs. Time Object**

Object Header Item	Text/Value
Title	"LC, Low Pressure 1"
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation

Low Pressure 1 vs. Time Object (valid only for HP 1090)

Data block:

y-axis: Low Pressure 1 [counts]

x-axis: Time [min]

The object is generated when the pump diagnosis is active during the analysis.

Low Pressure 2 vs. Time Object (valid only for HP 1090)

Table 33 **Object Header Items in Low Pressure 2 vs. Time Object**

Object Header Item	Text / Value
Title	"LC, Low Pressure 2"
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation

Data block:

y-axis: Low Pressure 2 [counts]

x-axis: Time [min]

The object is generated when the pump diagnosis is active during the analysis.

Temperature vs. Time Object

Table 34 Object Header Items in Temperature vs. Time Object

Object Header Item	Text/Value
Title	"LC, Temperature" (HP 1090)
	"Pump, Temperature" (HP 1050)
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation

Data block:

y-axis: Temperature [°C]

x-axis: Time [min]

Solvent A vs. Time Object

Table 35 Object Header Items in Solvent A vs. Time Object

Object Header Item	Text/Value
Title	"LC, Solvent A" (HP 1090)
	"Pump, Solvent A" (HP 1050)
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation
Description	Solvent description, taken from the method

Data block:

y-axis: Solvent A [%]

x-axis: Time [min]

Solvent B vs. Time Object

Table 36 Object Header Items in Solvent B vs. Time Object

Object Header Item	Text/Value
Title	"LC, Solvent B" (HP 1090)
	"Pump, Solvent B" (HP 1050)
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation
Description	Solvent description, taken from the method

Data block:

y-axis: Solvent B [%]

x-axis: Time [min]

Solvent C vs. Time Object

Table 37 Object Header Items in Solvent C vs. Time Object

Object Header Item	Text/Value
Title	"LC, Solvent C" (HP 1090)
	"LC, Solvent C1" (PV5)
	"Pump, Solvent C" (HP 1050)
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation
Description	Solvent description, taken from the method

Data block:

y-axis: Solvent C [%]

x-axis: Time [min]

Solvent D vs. Time object

Table 38 Object Header Items in Solvent D vs. Time Object

Object Header Item	Text/Value
Title	"LC, Solvent D" (HP 1090)
	"LC, Solvent C2" (PV5)
	"Pump, Solvent D" (HP 1050)
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation
Description	Solvent description, taken from the method

Data block:

y-axis: Solvent D [%]

x-axis: Time [min]

Contact 1 vs. Time Object

Table 39 Object Header Items in Contact 1 vs. Time Object

Object Header Item	Text/Value
Title	"LC, Contact 1" (HP 1090)
	"Pump, Contact 1" (HP 1050)
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation

Data block:

y-axis: Contact 1 (0= "Contact off", 1= "Contact on")

x-axis: Time [min]

Contact 2 vs. Time Object

Table 40 Object Header Items in Contact 2 vs. Time Object

Object Header Item	Text/Value
Title	"LC, Contact 2"
	"Pump, Contact 2" (HP 1050)
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation

Data block:

y-axis: Contact 2 (0= "Contact off", 1= "Contact on")

x-axis: Time [min]

Contact 3 vs. Time Object (valid only for HP 1090)

Table 41 Object Header Items in Contact 3 vs. Time Object

Object Header Item	Text/Value
Title	"LC, Contact 3"
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation

Data block:

y-axis: Contact 3 (0= "Contact off", 1= "Contact on")

x-axis: Time [min]

Contact 4 vs. Time Object (valid only for HP 1090)

Table 42 Object Header Items in Contact 4 vs. Time Object

Object Header Item	Text/Value
Title	"LC, Contact 4"
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation

Data block:

y-axis: Contact 4 (0= "Contact off", 1= "Contact on")

x-axis: Time [min]

Column Switch vs. Time Object (valid only for HP 1090)

Table 43 Object Header Items in Column Switch vs. Time Object

Object Header Item	Text/Value
Title	"LC, Columnswitch"
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation

Data block:

y-axis: Columnswitch

0 = "Columnswitch deactivated"
1 = "Columnswitch activated"

x-axis: Time [min]

Object Header Items in Sweep Objects

Voltamogram Object

Table 44 **Object Header Items in Voltamogram Object**

Object header item	Text/value
Title	"ECD, Voltamogram"
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation
DataType	11
MethodFile	Method directory name including path
Vial	Vial number
RawdataFile	Rawdata directory name including path
SampleName	Sample name
Operator	Operator name

Data block:

y-axis: Current [µA or nA]

x-axis: Potential [V]

Potential vs. Time Object

Table 45 Object Header Items in Potential vs. Time Object

Object Header Item	Text/Value
Title	"ECD, Sweep, P1=%.3f, P2=%.3f"
ObjClass	1 (User defined object)
SignalSource	2
DateTime	ASCII date and time of object generation
DataType	12
MethodFile	Method directory name, including path
Vial	Vial number
RawdataFile	Rawdata directory name including path
SampleName	Sample name
Operator	Operator name

Data block:

y-axis: Potential [V]

x-axis: Time [min]

Object Header Items in DADTest Objects

Lamp Intensity Object

Table 46 Object Header Items in Lamp Intensity Object

Object Header Item	Text/Value
Title	"Lamp Intensity"
Min	Minimum of the measured counts
Max	Maximum of the measured counts

Holmium Spectrum Object

Table 47 Object Header Items in Holmium Spectrum Object

Object Header Item	Text/Value
Title	"Holmium Spectrum"
WCalls	Currently set wavelength calibration switch
WCalShouldBe	The wavelength calibration switch should be set to

Object Header Items in FLDScans Objects

Excitation Scan Object

Table 48 Object Header Items in Excitation Scan Object

Object Header Item	Text/Value
Title	"Excitation"
AtEmission	Complementary wavelength
Scanspeed	Scanspeed used
Pmtgain	Pmtgain used
Lamp	Lamp mode used
Responsetime	Response time used
Gate	Gate setting
Delay	Delay setting

Emission Scan Object

Table 49 Object Header Items in Emission Scan Objects

Object Header Item	Text/Value
Title	"Emission"
AtExcitation	Complementary wavelength
Scanspeed	Scanspeed used
Pmtgain	Pmtgain used
Lamp	Lamp mode used
Response time	Response time used
Gate	Gate setting
Delay	Delay setting

12

System Tables

System Tables

This chapter describes the ChemStation system tables with predefined structures in the system registers.

Standard Tables

Several tables in data analysis have been implemented using a *generic table* concept. This means that you can access these tables using generic table commands. In addition to the generic table commands there are some specific commands that operate on a specific table in a way that would not be possible with the generic table commands. To ensure the specific commands can work correctly, these tables are located in predefined registers and objects, and the tables are write-protected in their structure and sometimes in the individual values.

Table 50 **Contents of Standard Tables**

Description	RegObj	TabName
Window	_Config[1]	Window
Axis style	_Config[1]	AxisStyle
Color	_Config[1]	Color
Views	_Config[1]	View_<Name>
Factory defaults for views	_Config[1]	Def_<Name>
Report style	_Config[1]	ReportStyle
Currently used signals	_Config[1]	Signal
Spectra	_Config[1]	Spectra_<Detector>
Avail analytical columns	_Config[1]	Columns
Noise and statistic parameters	_Config[1]	SySuit
Table references	_Config[1]	TableRef
Available instrument modules	_Config[1]	Modules
Configured instrument modules	_Config[1]	ModuleInfo
Report styles for sequence summary	_Config[1]	SeqSumRptStyle
Custom report parameters	_Config[1]	CustReport

Table 50 **Contents of Standard Tables, continued**

Description	RegObj	TabName
Manual integration window state	_Config[1]	ManInt
Isoplot color scheme	_Config[1]	Iso_ <ColorSchemeName>
Table templates	_Config[2]	Same name as Config[1]
Display description table	_Config[3]	AnyName
Output description table	_Config[3]	AnyName
Calibration table templates	_Config[4]	Same name as DAMethod[1]
Result table templates	_Config[5]	Same name as ChromRes[1]
Integrator events	_DAMethod[1]	Event_<Detector><SignalId>
Default integrator events	_DAMethod[1]	Event
Detector personality	_DAMethod[1]	IntDetect_<DetectorClass>
Method personality	_DAMethod[1]	IntMethod_<InstrumentType>
Quantification parameters	_DAMethod[1]	QuantParm
Compound groups	_DAMethod[1]	CompGroup
Compounds	_DAMethod[1]	Compound
ISTD compounds	_DAMethod[1]	ISTD
Expected peaks	_DAMethod[1]	Peak
Signals used in method	_DAMethod[1]	Signal
Time references	_DAMethod[1]	TimeRefGroup
Calibrated levels	_DAMethod[1]	Level
Calibration points	_DAMethod[1]	CalPoint
Calibration runs	_DAMethod[1]	CalRun
Calibration errors	_DAMethod[1]	ErrorReport
Report specification	_DAMethod[1]	RptParm
Automatic library search	_DAMethod[1]	AutoID
Selected analytical columns	_DAMethod[1]	Columns

Table 50 **Contents of Standard Tables, continued**

Description	RegObj	TabName
Noise and statistic parameters	_DAMethod[1]	SysSuit
Integrator results	ChromReg[any]	IntResults
System suitability results	ChromReg[any]	IntResults (extensions)
Quantification parameters	ChromRes[1]	QuantParm
Compound groups	ChromRes[1]	CompGroup
Compounds	ChromRes[1]	Compound
ISTD compounds	ChromRes[1]	ISTD
Expected peaks	ChromRes[1]	Peak
Signals used in method	ChromRes[1]	Signal
Time references	ChromRes[1]	TimeRefGroup
Calibration errors	ChromRes[1]	ErrorReport
Automatic library search results	ChromRes[1]	AutoIDParms
Automatic library search results	ChromRes[1]	Compound (extensions)
Noise calculation results	ChromRes[1]	Noise
Noise calculation results	ChromRes[1]	Signal (extensions)
System suitability results	ChromRes[1]	Peak (extensions)
System suitability results	PeakPerfReg[1]	Results
Methods for sequence summary	SeqInfoReg[1]	Methods
Data files for sequence summary	SeqInfoReg[1]	ResultFiles
Compound for sequence summary	SeqInfoReg[1]	CompTab<MethodNum>
Automatic library search results	SeqInfoReg[1]	Proposals

Table 50 **Contents of Standard Tables, continued**

Description	RegObj	TabName
Custom report repeat items	WorkReg[1]	RepeatItems
Custom report totals	WorkReg[1]	Totals
Custom report sum totals	WorkReg[1]	SumTotal
Custom report limits	WorkReg[1]	StatItems
Custom report template	WorkReg[1]	Template

Window Table

Register name _Config

Object number 1

Table name Window

Table 51 shows the predefined header items in the window table.

Table 51 Predefined Header Items in Window Table

Item Name	Type/Range	Meaning
Escape	String macro	Mouse action macro for Escape key
DefLClick, DefLShiftClick, DefLCtrlClick, DefLDblClick, DefLEndDrag, DefLBeginDrag	String macro	Default values for mouse action, see LClick for explanation
LineStyle1	Enumeration	Line style of first objects in register (see LineStylesOn column) 0 (SOLID): solid line 1 (DASH): dashed line 2 (DOT): dotted line 3 (DASHDOT): line with dash and dot 4 (DASHDOTDOT): line with dash and two dots
LineStyle2, LineStyle3, LineStyle4, LineStyle5	Enumeration	Similar to LineStyle1 for next objects in register
CurrentView	String 9 characters	View name of current active view
XAxisStyle_	String 15 characters	Name of AxisStyle table
YAxisStyle_	String 15 characters	Name of AxisStyle table
ZAxisStyle_	String 15 characters	Name of AxisStyle table

Table 51 **Predefined Header Items in Window Table, continued**

Item Name	Type/Range	Meaning
Color_	String 15 characters	Name of Color table
Font_	String 15 characters	Name of font table
MaxWinNr	Integer	Maximum number of rows that table may reach
CRTxResolution	Integer	Horizontal screen resolution to use
DrawCursorDLL	String	File name (incl. path) of DLL used by Draw cmd for cursor bitmaps

Table 52 shows the predefined columns in the window table.

Table 52 **Predefined Columns in Window Table**

Column Name	Type/Range	Meaning
DefWXLow	Numeric 0 to 1	Left hand edge of window
DefWXHigh	Numeric 0 to 1	Right hand edge of window
DefWYLow	Numeric 0 to 1	Bottom edge of window
DefWYHigh	Numeric 0 to 1	Top edge of window

Table 52 **Predefined Columns in Window Table, continued**

Column Name	Type/Range	Meaning		
DefWinStyle	Bitfield 32	Window styles as "OR" combination of individual attributes:		
		0x0001	1	WS_CAPTION
		0x0002	2	WS_THICKFRAME
		0x0004	4	WS_MAXIMIZEBOX
		0x0008	8	WS_MINIMIZEBOX
		0x0010	16	WS_SYSMENU
		0x0020	32	WS_MAXIMIZE
		0x0040	64	WS_MINIMIZE
		0x0080	128	WS_BORDER
		0x0100	256	Must be 0
		0x0200	512	WS_HSCROLL
		For an explanation see Microsoft Windows Software Development Kit		

Table 52 **Predefined Columns in Window Table, continued**

Column Name	Type/Range	Meaning	
Destination	String	Destination device of window:	
		SCREEN	Window will be drawn to screen. DefWXLow through DeWYHigh refer to current size of application window.
		PRINTER	Window will be drawn to printer. DefWXLow through DefWYHigh refer to size of paper. The point WXLow=0, WYHigh=1 is current printing position.
		Any other	Metafile name prefix ≥ 6 characters, 00 to 99 plus. WMF will be added.

Table 52 **Predefined Columns in Window Table, continued**

Column Name	Type/Range	Meaning	
XAxisStyle	Link	Row index into AxisStyle table for x-axis	
YAxisStyle	Link	Row index into AxisStyle table for y-axis	
ZAxisStyle	Link	Row index into AxisStyle table for z-axis	
Color	Link	Row index into Color table	
LineStylesOn	Boolean	0	All objects in register will be drawn with same line style (the one given in table header item LineStyle1, if not specified otherwise in LineStyle parameter of object)
		1	Depending on number of colors available in destination device of window, the first objects in register are drawn with line style given in table header item LineStyle1. The next objects are drawn with line style given in LineStyle2, and so on. The object may individually specify its own line style in the LineStyle parameter.
Command	String	Command string for automatic creation of window. If empty, no automatic creation. Set by window when it is closed, to reflect current status of window.	

Table 52 **Predefined Columns in Window Table, continued**

Column Name	Type/Range	Meaning	
Type	Enumeration	Type of window.	
		-1	Window is protected agains being deleted or being cut out (Use copy and paste instead).
		0	Unspecified
		1	Graphic (Draw command)
		2	"Living" window with hard-coded WinNum, i.e. all modeless dialog boxes (isoplot, edit events, edit calib table).
		3	Signal monitor 1 (ShowSignal command)
		4	Signal monitor 2 (ShowSignal command)
		5	Spectrum monitor (ShowSpectra command)
		6	EdTab command
		7	EdObjTab command
		8	EdDataTab command
		9	EdRegTab command
		10	EdObjHdrTab command
		11	EdTabHdrTab command
ZXLow	Numeric	Lower limit in x-range parameter (backed-up from previous Zoom command)	
ZXHigh	Numeric	Upper limit in x-range parameter (backed-up from previous Zoom command)	
ZYLow	Numeric	Lower limit in y-range parameter (backed-up from previous Zoom command)	

Table 52 **Predefined Columns in Window Table, continued**

Column Name	Type/Range	Meaning	
ZYHigh	Numeric	Upper limit in y-range parameter (backed-up from previous Zoom command)	
LClick	String macro	Mouse action macro for left Click	
LShiftClick	String macro	Mouse action macro for left SHIFT+Click	
LCtrlClick	String macro	Mouse action macro for left CTRL+Click	
LDblClick	String macro	Mouse action macro for left double Click	
LEndDrag	String macro	Mouse action macro for left up Click after drag	
LBeginDrag	Enumeration	Visual feedback style for left drag, see also DragCursor.	
		0	No graphical response (default)
		1	No graphical response
		2	Draw rubber band line between start and actual coordinates
		3	Draw rectangular box between start and actual coordinates
		–n	Same as +n but display actual coordinates on message line numerically during drag.
DragCursor	Enumeration	Shape of cursor during drag (all mouse buttons)	
		0	Unchanged (same as without drag)
		1	Arrow pointing up and left
		2	Grabbing hand
		3	Cross hair
		4	Fat downwards pointing arrow
		5	Vertical line across client area
		6	Horizontal line across client area
		7	Sand clock

Table 52 **Predefined Columns in Window Table, continued**

Column Name	Type/Range	Meaning
MoveCursor	Enumeration	Shape of cursor during move across window client area (without pressing any mouse button); valid values same as for DragCursor
Topic	Bitfield 32	Set of bits that define a mask to draw (bit=1) / do not draw (bit=0) annotations with a specific topic. Initial setting is 0xFFFFFFFF. Defined topics are: **Mask Value / Number / Name** 0x00000001 / 1 / Title 0x00000002 / 2 / Baseline 0x00000004 / 3 / RetTime 0x00000008 / 4 / Compound 0x00000010 / 5 / PeakArea 0x00000020 / 6 / CalPoint 0x00000040 / 7 / WorkPoint 0x00000080 / 8 / Text 0x00000100 / 9 / PolyLine 0x00000200 / 10 / Marker 0x00000400 / 11 / Tickmark To specify that a topic shall be drawn, its mask value has to be OR'ed into the total Topic mask. The topic numbers conform to the Topic parameter in the SetAnnVal command.
HelpContext	Integer ≥ 0	Index of current Help Context to be used upon F1 help.
WinTitle	String	Title to be displayed in window's caption bar when window is created. Should be set by SetWinTitle command, not directly. If empty, the displayed title includes the window number and the register name.
Font	Link	Row index into font table

Table 52 **Predefined Columns in Window Table, continued**

Column Name	Type/Range	Meaning
CPName	String	CP identifier

Specific commands for the window table

The following is a list of specific commands for the Window table (the WinNum parameter of these commands is row index in the Window table):

ActiveWindow

AppendSelect

ClearSelect

ClearWin

Draw

EdDataTab

EdObjTab

EdObjHdrTab

EdRegTab

EdTab

EdTabHdrTab

EditAnn

FindNearestAnn

FindNearestObj

FreeWin

GetSelect

SetWinTitle

Zoom

ZoomOut

The following commands use a hard-coded window number:

DrawIsoPlot

Diagram

DrawIsoPlot

LogBook

EditEvents

MonADCStatus

MonFLDStatus

MonLCStatus

Mon1090DAD

Mon1050DAD

MonALSStatus

MonECDStatus

MonMWDStatus

MonPumpStatus

MonVWDStatus

RunStatus

ShowSignal

ShowSpectra

Notes about the window table

- Of the 32 available bits for annotation topics, the rightmost 24 bits (Topic24 = 0x00800000 to Topic1 = 0x00000001) are reserved for applications and macros supplied by Hewlett-Packard. The uppermost 8 bits (Topic32 = 0x80000000 to Topic25 = 0x01000000) may be used by the user to group annotations according to their purpose. Hewlett-Packard supplied macros will not use these.
- The user must not insert or delete any row within the first 70 rows of this table, as this will causes many macros and commands to crash. These macros know which window to access through a reserved row number. The window numbers are assigned as follows:

1 to 10 Free for use by user. Hewlett-Packard supplied macros will not use these window numbers. The windows will not take part in the View concept, see View table and SwitchView command.

11 to 16	Free for use by user. Hewlett-Packard supplied macros will not use these window numbers. The windows will take part in the View concept.
17 to 70	Reserved by Hewlett-Packard supplied macros and commands. The windows will take part in the View concept.
17	EdTab command
18	EdObjTab command
19	EdDataTab command
20	EdRegTab command
21	EdObjHdrTab command
22	EdTabHdrTab command
23	Print any graphic window from interactive menu item.
24	EdCalTable command.
25	EditEvents command.
26	Print chromatograms from chromatogram plot macro (Report).
27	Print calib curve from calibration curve plot macro (Report).
28	Print spectra from peak purity and library search plot macros (Report).
29	DrawIsoplot command.
30	Print signals from DrawIsoplot command.
31	Print spectra from DrawIsoplot command.
32	Print isoplot from DrawIsoplot command.
33	ShowSignal command for channel 1.
34	ShowSignal command for channel 2.
35	ShowSpectra command.
36	LogBook command.
37	RunStatus command.
38	Draw ChromReg register.

39	Draw SpecReg register.
40	Draw SweepReg register.
41	Draw SpecReg2 register.
42	Draw SPC_RefWinReg register.
43	Draw PurDiffReg register.
44	Draw PurSigReg register.
45	Draw PurRatReg register.
46	Draw LibDiffReg register.
47	Draw zoomed part of ChromReg register.
50 to 62	MonXXXStatus commands, where XXX is the device. The window numbers will be assigned upon startup of the application, according to the order of devices in the WIN.INI file. As long as the instrument configuration is not changed, the window numbers will be the same as they were in the previous application on the same instrument.
63	Diagram command to show the HPCE diagram.
64	LCPumpDiagnose command.
65	Display integration results.
66	Display quantification results.
67	Display quantification errors.
68	Display available analytical separation columns.
69	Display analytical separation columns of current method.

- The user may append rows at the end of this table, up to a total number of rows, that is set in the table header item MaxWinNr. These rows are intended for temporary windows that are created in some macro, used, and then cleared again. The appended rows must then be deleted again as well.
- When a window with Destination Screen is created or gets a message to reveal itself, it looks up its position and WinStyle in the Window table. If the window has a fixed size it takes the parameters for the upper left corner (DefWXLow, DefWYHigh) and adapts DefWXHigh, DefWYLow according to the fixed size.
- When a window with Destination Screen is interactively resized or moved,

this change is reflected automatically in the DefWXLow, DefWXHigh, DefWYLow, DefWYHigh columns of the Window table. When a window with Destination Screen is interactively maximized or minimized, this change is reflected automatically in the DefWinStyle column of the Window table.

- When a window with Destination Screen is deleted or gets a message to hide itself, this fact is noted internally. It is possible to check via macro whether a window exists or not, using the FreeWin function.
- When a window with Destination Screen is created, it generates a command string and stores this string into the Command column. This can be used later on, especially after bootup and reloading the Window table, to recreate the window with the same parameters it had upon the last time. This concerns the commands: Draw, DrawIsoPlot, Ed…Tab, ShowSignal, ShowSpectra, RunStatus, LogBook, Mon…Status, EditEvents, EdCalTbl.
- Upon installation the Color row index will be set to 1. This means that all windows have the same color scheme, i.e. the one that is defined through the WIN.INI file (see “Color Table” on page 221).
- Upon installation the LineStylesOn will be set to 1. The object will usually be created with its LineStyle, LineWidth, Marker parameters in the Data Block set to their *unspecified* values, see the SetDataVal command.
- The commands Draw, Zoom, ZoomOut will change the columns ZXLow. ZYHigh in a particular way, see those commands.
- The header items for default mouse actions are copied into the actual fields upon each mode reset, e.g. through the ESC key.

Axis Style Table

Register name _Config

Object number 1

Table name AxisStyle

The axis style table has no predefined header items. Table 53 shows the predefined columns in the axis style table.

Table 53 Predefined Columns in Axis Style Table

Column Name	Type/Range	Meaning	
AxisVisible	Boolean	1	Axis visible
		0	Axis not drawn, no real estate reserved
NumbersVisible	Boolean	1	Ticks and numbers visible
		0	Ticks, numbers not drawn, no real estate reserved
OrderReversed	Boolean	0	Smallest number at axis begin
		1	Highest number at axis begin
LogScale	Boolean	0	Linear scaling
		1	Logarithmic scaling
LogBase	Integer	Base of logarithm, used if LogScale=1	
FieldWidth	Integer	Field width of numbers for drawing	
Precision	Integer	Precision of numbers for drawing	
ScaleFactor	Numeric > 0	Numbers will be divided by ScaleFactor before drawing	
ScaleTitle	String 29 characters	Will be appended to title before drawing	
TitlePosition	Enumeration	Position of axis title relative to axis	
		0 (OFF)	Title not drawn, no real estate reserved
		1 (END)	Title drawn near end of axis
		2 (BEGIN)	Title drawn near beginning of axis
		3 (CENTER)	Title drawn near center of axis

Table 53 **Predefined Columns in Axis Style Table, continued**

Column Name	Type/Range	Meaning
TicksAbove	Boolean	Position of ticks, numbers and title relative to axis, as seen from beginning of axis
		1 Above / to right of axis
		0 Below / to left of axis
AxisXBegin	Numeric 0 to 1	X position of axis begin within window
		0 Is left edge
		1 Is right edge.
AxisYBegin	Numeric 0 to 1	Y position of axis begin within window
		0 Is bottom edge
		1 Is top edge.
NumberAngle	Integer	Angle between axis and numbers in degrees. Allowed values: 0, 90, 180, 270.
TitleAngle	Integer	Angle between axis and title in degrees. Allowed values: 0, 90, 180, 270.
AxisAngle	Integer	Axis angle counterclockwise from horizontal. Allowed values: 0, 90, 180, 270.

Specific Commands for the Axis Style Table

Draw

Notes About the Axis Style Table

- The user must not delete any row in this table, as this will cause an inconsistency between the Window table and this table. The row numbers are referenced in the Window table and are reserved in that sense. The row numbers are assigned as follows:

1 X axis for chromatogram windows, unless otherwise noted.

2 Y axis for chromatogram windows, unless otherwise noted.

3	X axis for spectrum windows, unless otherwise noted.
4	Y axis for spectrum windows, unless otherwise noted.
5	X axis for print chromatograms in the report.
6	Y axis for print chromatograms in the report.
7	X axis for print calibration curve in the report.
8	Y axis for print calibration curve in the report.
9	X axis for print spectrum in the report.
10	Y axis for print spectrum in the report.
11	X axis for chromatogram and spectrum subwindows in DrawIsoplot command.
12	Y axis for chromatogram and spectrum subwindows in DrawIsoplot command.
13	X axis for print chromatogram in DrawIsoplot command.
14	Y axis for print chromatogram in DrawIsoplot command.
15	X axis for print spectrum in DrawIsoplot command.
16	Y axis for print spectrum in DrawIsoplot command.

- The user may change elements in the table. This will have an effect on more than one window, if the row number is referenced in several rows of the Window table.
- The user may freely add rows to this table without any detrimental effect.
- The AxisXBegin and AxisYBegin define the start point of the line that is visualizing the axis. The numbers are in the range 0 to 1 and are relative to

the window. Table 54 shows you some examples.

Table 54 Examples of AxisXBegin and AxisYBegin Values

AxisXBegin	AxisYBegin	Start Point of Line
0.0	1.0	Top left corner
0.0	0.5	Left center
0.0	0.0	Bottom left corner
0.5	1.0	Top center
0.5	0.5	Center of window
0.5	0.0	Bottom center
1.0	1.0	Top right corner
1.0	0.5	Right center
1.0	0.0	Bottom right corner

If the position of ticks, numbers, and title (TicksAbove) is defined such that it would lie *outside* the window, for example, AxisXBegin=0.0, AxisYBegin=0.0, TicksAbove=0, AxisAngle=0 (usual x-axis), or AxisXBegin=0.0, AxisYBegin=0.0, TicksAbove=1, AxisAngle=90 (usual y-axis), then the axis will be shifted just enough to make room for ticks, numbers and title. The begin of the other axis is shifted accordingly, if the begin is defined to be the same for both axes.

- The FieldWidth and Precision values are used mainly as described in the standard C function printf(3S):

  If FieldWidth < 0, abs(FieldWidth) is used as field width.

  If FieldWidth = 0, a field width of 1 is used.

  If FieldWidth > 0, it is used as specified.

  If Precision ≥ 0, it is used as specified.

  If Precision < 0, abs(Precision) is used as precision.

The resulting types of conversion are shown in Table 55.

Table 55 Types of Conversion

	Precision < 0	Precision = 0	Precision > 0
FieldWidth < 0	%–G	%E	%E
FieldWidth = 0	See below	%–f	%–f
FieldWidth > 0	%#G	%f	%f

If FieldWidth = 0 and Precision < 0, the combination of FieldWidth and Precision is regarded as unspecified. When printing the numbers of the axis, a default conversion has to be used.

Color Table

Register name _Config

Object number 1

Table name Color

The color table has no predefined header items. Table 56 shows the predefined columns in the color table.

Table 56 Predefined Columns in Color Table

Column Name	Type/Range	Meaning
Window	RGB	Background color for window area
Data	RGB	Background color for data area
Axis	RGB	Axis color (including ticks and numbers)
AxisTitle	RGB	Axis title color
Obj1	RGB	Color of 1st object in register
Obj2	RGB	Color of 2nd object in register
Obj3	RGB	Color of 3rd object in register
Obj4	RGB	Color of 4th object in register
Obj5	RGB	Color of 5th object in register
Obj6	RGB	Color of 6th object in register
Obj7	RGB	Color of 7th object in register
Obj8	RGB	Color of 8th object in register
Topic1	RGB	Color of annotation with topic 1
Topic2	RGB	Color of annotation with topic 2
…	…	…
Topic32	RGB	Color of annotation with topic 32

Specific Commands for the Color Table

Draw

Notes About the Color Table

- Upon startup of the system, after loading the _Config register from the file, the row1 is overwritten with the appropriate values found in the WIN.INI file.
- Each valid color specification is a RGB number in the range 0 to 0x00FFFFFF. The number 0xFF000000 is an invalid color specification and is used as a special value that says: "This color is not specified here, take a default color instead."
- The colors are defined in a triple hierarchy of default values. The most overriding specification is the color that is defined within the data object (see the SetDataVal command). If the color is not specified there, the color definition of the current row (row number > 0) in this Color table is used. If the color is not specified in this row either, the row 1 of this Color table is used.
- Usually an object will be created with its curve color unspecified.
- The colors for Topic1 to Topic32 in this table serve as defaults if an annotation with that topic did not specify the color directly in the data object.
- Usually the annotations that have a non-zero Topic should be created with their color unspecified, see the SetAnnVal command. This enables the dynamic control of the annotation color without changing the object.
- The user must not delete any row in this table, as this will cause an inconsistency between the Window table and this table. The row numbers are referenced in the Window table and are reserved in that sense.
- The user may change elements in the table. This will have an effect on more than one window, if the row number is referenced in several rows of the Window table.
- The user may freely add rows to this table without any detrimental effect.
- The row 0 consists of all 0xFF000000 values. This means that a newly inserted row has all colors unspecified. If afterwards several fields are changed to valid colors, these will be taken upon drawing the window. For all other colors the values are taken—through the use of row 1—from the current WIN.INI file. If this tying to the current WIN.INI file is not wanted, the user should copy row 1 to the newly inserted row before changing

individual colors. This way later changes in the WIN.INI file have no more effect upon the new row.

View Table

Register name _Config

Object number 1

Table name "View_" + ViewName or "Def_" + ViewName

An arbitrary number of tables with this structure may exist within the register. The ViewName is used as parameter in the specific commands.

Table 57 shows the predefined header items in the view table.

Table 57 Predefined Header Items in View Table

Item Name	Type/Range	Meaning
Title	String	View title, to be displayed in a menu item
PreMacro	String macro	Name of a macro that will be executed during the SwitchView command, see there.
PostMacro	String macro	Name of a macro that will be executed during the SwitchView command, see there.
SaveLevel	Enumeration	Controls how much information is saved from the Window table to the current View table during the SwitchView command. 0 Information for all windows is saved, that have a row in the view table. If necessary new rows are added for visible windows that do not have a row in the view table yet. 1 Information for all windows is saved, that have a row in the view table. No new rows are added for visible windows that do not have a row in the view table. 2 Nothing is saved.

Table 58 shows the predefined columns in the view table.

Table 58 Predefined Columns in View Table

Column Name	Type/Range	Meaning
WinNum	Integer	Row index into Window table
WXLow	Numeric 0 to 1	Left-hand edge of window
WXHigh	Numeric 0 to 1	Right-hand edge of window
WYLow	Numeric 0 to 1	Bottom edge of window
WYHigh	Numeric 0 to 1	Top edge of window
WinStyle	Bitfield 32	Window styles as defined in Window table
Visible	Boolean	Window visible
		Window not visible

Specific Commands for the View Table

SwitchView

Notes About the View Table

- The View tables are used to control the display management. The major elements of a particular view are the graphic windows, status windows, modeless dialog boxes. Each of these windows must be represented by a row in the Window table.
- A view defines an arrangement of visible windows for one particular task or situation. Different views can be defined through different View tables. The name of the current View table is stored as table header item

CurrentView in _Config[1]. The defined views are shown in Table 59.

Table 59 Defined Views

Table Name	Title in Menu	Default Table Name
View_Top	Toplevel	Def_Top
View_DA	Data Analysis	Def_DA
View_SpecLib	Spectra Library	Def_SpecLib

- The predefined views will not be supplied as tables with that name on the installation media. Instead there will be a default table for each of them. Upon system startup, after loading the _Config register, it will be looked whether for each Def_xxx table there is a corresponding View_xxx table in _Config[1]. If not, the table is generated with a NewView xxx macro call. This avoids having to ship two identical tables for each view.
- For each window that shall be visible in a particular view there must be one row in the appropriate View table with the Visible column set to 1. See the notes under the Window table for the automatic interaction between the two tables upon creation, modification, deletion of a window.
- The user should not change anything in the current view table because this could then be inconsistent with the display status of the visible windows. Other view tables may freely be changed with generic table access commands.
- SwitchView will hide all windows that are currently visible and have no rows in the new view table or the Visible column there is set to 0, then set CurrentView to the new view table, then show and/or resize all windows that have rows in the now current view table with the Visible column set to 1.

  If CurrentView is empty upon entry into SwitchView this is interpreted as if no window is currently visible and the current view table is empty, i.e. contains no rows. This is the startup situation.

Report Style Table

Register name _Config

Object number 1

Table name ReportStyle

Table 60 shows the predefined header items in the report style table.

Table 60 Predefined Header Items in Report Style Table

Item Name	Type/Range	Meaning
FirstMacroCol	Integer	Contains index of first column with macro names–currently 3.
MaxUserStyle	Integer ≥ 0	Highest Style given by user
MinFactoryStyle	Integer < 0	Lowest Style reserved by factory

Table 61 shows the predefined columns in the report style table.

Table 61 Predefined Columns in the Report Style Table

Column Name	Type/Range	Meaning
Title	String	Name of report style in combination box
Style	Integer	Unique number as identifying key
M1	String macro	1st macro to be executed
M2	String macro	2nd macro to be executed
M3	String macro	3rd macro to be executed
M4	String macro	4th macro to be executed
Mn	String macro	nth macro to be executed

Specific Commands for the Report Style Table

EditRptParam

Notes About the Report Style Table

- This table is used to feed the Style combination box in the Specify Report dialog box. The Title column is (in that ordering) displayed in the combination box.
- The Title column can of course be translated into the local language.
- The Style is a number that must not have a duplicate in this table. The Style is the parameter that is stored as Report Style in the method (table header Style in table RptParm in object _DAMethod[1]). Through this feature the ordering of the table rows is irrelevant. The table rows can therefore be ordered such that the most used styles appear first in the combination box.
- By adding/inserting additional lines to the table, the user is able to add his/her customized report style. To ease the selection of a Style as unique key there is the header item MaxUserStyle. This item should be incremented by 1 and the resulting number used as the Style of the new table row.
- The Style numbers that are available from the factory are always negative numbers. MinFactoryStyle will be decremented any time a new style is made available from the factory and the resulting number is used as the Style of the new table row.
- This table is used during Report generation to determine what to do. The table is searched for a Style matching that from the Specify Report dialog box. The corresponding table row is searched for macro names. These macros are executed in the given order.
- The number of columns for macro names can be expanded if necessary, i.e. if a table row is added or changed that needs to call more macros than macro columns are available.
- The names of the columns for the macro names are not important, as there will be a generic macro to interpret the table when it comes to printing. This macro knows that all columns from the FirstMacroCol to the last contain macro names to execute. This means that if columns for other purposes are added, this has to be done before the macro columns. FirstMacroCol has then to be adapted.
- Customization is possible in several ways:

  Select a different combination out of the delivered standard macros.

  Write own macros that are then used instead of the delivered ones for certain blocks of information. For example, use a different macro for printing the file header.

Write a macro that sends all necessary information to a spreadsheet program for a fully customized report.

The macros called by the report formatting process and that are therefore stored in the Mx columns of the Report Style Table have to be defined using one string parameter defining the report destination (screen or printer).

For example:

```
PMyMacro
parameter destination$
...
```

_Config Register

Register name _Config

Object number 1

Table name Signal

The signal table has no predefined header items. Table 62 shows the predefined columns in the signal table.

Table 62 Predefined Columns in Signal Table of _Config Register

Column Name	Type/Range	Meaning
Source	Enumeration	Code for who generated this line 0 Other 1 File command 2 Edit IntEvents dialog (EditEvents command) 3 Dialog box for acquisition parameters, LoadMethod, or SaveMethod 4 Calibrate/Recalibrate dialog box (new calib or add signal) or Calibration Table dialog box 5 Method Signals Table dialog box (manually or through selecting from a combination box) 6 LoadChrom command

Specific Commands for the Signal Table

File

LoadChrom

LoadSignal

EditEvents

Notes About the Signal Table

- A table of same name and similar format is stored in _DAMethod[1], “Signal Table in _DAMethod Register” on page 232. It belongs to the method and holds all signals that will be evaluated during RunMethod.

- This table is used to store the currently available signals, either from the current rawdata file or from the current data acquisition parameters. The table holds all signals that are entered by:

  File or LoadSignal command,

  LoadChrom command,

  dialog boxes for acquisition parameters.

- The File command will delete all rows that have a source of 1 or 6, i.e. that were newly generated by the previous File command, or by a LoadChrom command issued since the last File command. It will then insert new rows for all signals that are available in the new rawdata file. For those new rows the source will be set to 1.

- The Data Acquisition will keep the table up-to-date with respect to the current signal parameters in the Data Acquisition method. This means that whenever there is a change in the number of signals acquired or in the wavelength specification of a signal, the appropriate row in the Signal table is updated or inserted. The Data Acquisition will update all rows that have a source of 3, i.e. that were originally generated by Data Acquisition. The signals entered here are those that will be acquired in the next run, either into .CH files or within the LCDIAG.REG file.

- The LoadChrom command will check whether a row with the signal description of the extracted signal is already present in the table. If not, it will be inserted, with a Source of 6.

- Upon bootup, after loading the _Config register but before loading the method, all rows within the Signal table of _Config[1] that have a source of 3 will be deleted. This avoids having old signals hanging around if the instrument configuration changed before bootup.

- For Data Acquisition, File command, LoadChrom command: When a new row is generated in the Signal table, it should be inserted at the beginning (row 1). Reason: The signal descriptions are used in a combination box within the dialog box for selecting signals to be evaluated during Run Method. For convenience the *newest* signals should be accessible without much scrolling.

Signal Table in _DAMethod Register

Register name _DAMethod

Object number 1

Table name Signal

Table 63 shows the predefined header items in the _DAMethod signal table.

Table 63 Predefined Header Items in Signal Table (_DAMethod Register)

Item Name	Type/Range	Meaning
FirstPeak_	String	Name of table referenced through FirstPeak

Table 64 shows the predefined columns in the _DAMethod signal table.

Table 64 Predefined Columns Items in Signal Table (_DAMethod Register)

Column Name	Type/Range	Meaning
FirstPeak	Link	Link to first peak of signal
Desc	String ≤47 characters	Signal description, for example, "DAD1 A, Sig=254,4 Ref=450,80"
Start	Numeric	Signal start time in minutes relative to start of run
End	Numeric	Signal end time in minutes relative to start of run
Delay	Numeric	Signal offset time in minutes, dead volume time

Specific Commands for the Signal Table

AddCalPeaks

CalibratePeaks

CorrectRetTime

EditEvents

IdentifyPeaks

QuantifyPeaks

Notes about the signal table

- A table of same name and similar format is stored in _Config[1]. It does not belong to the method and holds all signals that are currently available.
- This table here holds all signals that are to be evaluated during Run Method. These may either be signals that are to be integrated (and possibly are calibrated) or they may be signals (e.g. gradient or temperature profiles) that will merely be drawn together with the other signals.
- The EditEvents command may (through user interaction) create an events table for a new signal. In this case it will check that the signal, according to its Desc[1:instr(Desc,",")–1], i.e. everything before the first comma, which is Detector and SignalId, (not the complete Desc), is in the Signal table. If necessary, a new row will be generated, meaning that this signal will in future be evaluated during Run Method.
- The Calibrate/Recalibrate dialog box (new calib or add peaks) will check that, if a new signal is entered, this signal is, according to Desc, represented in the Signal table. If necessary, a new row will be generated, meaning that this signal will in future be evaluated during RunMethod.

  During this check it may happen that there is a row in the table with the same Detector and SignalId as the new signal but a different Desc. This means that the wavelength of the signal has been changed. The user must be warned about this fact and the clash has to be resolved, as it would surely produce an error when trying to run such a method.
- The Signal table is edited through the Signal Details dialog box that will be the first box shown in the Data Analysis part of edit entire method.
- In this dialog box the user has the possibility to add signals. The signal description of the new signal may be selected from a combination box. The contents of this combination box are exactly the Desc strings in the Signal table of _Config[1].

Spectra Table

(HP ^{3D}CE and HPLC^{3D} ChemStations only)

Register name _Config

Object number 1

Table name "Spectra_" + Detector

For each spectral detector, there exists one spectra table in the object. The table name reflects the file name within the rawdata directory. Example: the spectra of detector DAD1 are stored in the file DAD1.UV, and the spectra table is called Spectra_DAD1.

The spectra table has no predefined header items.

Table 65 shows the predefined columns in the spectra table.

Table 65 **Predefined Columns in Spectra Table**

Column Name	Type/Range	Meaning
Time	Numeric	Retention/migration time in minutes

Table 65 **Predefined Columns in Spectra Table, continued**

Column Name	Type/Range	Meaning
Attribute	Enumeration	Detector-specific spectrum attribute
		If Detector = DAD, this is the peak detector code of the spectrum:
		0x0034 inflection point on upslope with rising curvature
		0x0035 inflection point on upslope with falling curvature
		0x0036 inflection point on downslope with rising curvature
		0x0037 inflection point on downslope with falling curvature
		0x0041 top of peak
		0x0043 cancellation of peak
		0x0044 valley
		0x0045 end of peak
		0x004d manually taken
		0x0050 periodically taken
		If Detector = FLD this is the scan type of the spectrum:
		0x0001 Excitation scan
		0x0002 Emission scan
Offset	Integer	file offset of spectrum record

Specific Commands for the Spectra Table

File

LoadChrom

LoadSpectrum

LoadSpectra

Notes about the spectra table

- All columns within the table plus the table itself are write-protected. Via macro commands the items may only be read but not modified.
- The File command deletes all tables within _Config[1] that have a name Spectra_… . It will then build spectra tables for all .UV files that are in the rawdata directory. These tables may be empty, i.e. have no rows, if the .UV file exists but has no spectra in it.
- The spectra table may be used from the macro to decide which spectrum to load.
- The maximum number of entries in the spectra table is 6400. Any spectra with indices above 6400 are not entered in the table.

Analytical Column Parameters

This table lists the parameters for all available analytical columns.

Register name _Config

Object number 1

Table name Columns

Table 66 shows the predefined header items in the columns table.

Table 66 Predefined Header Items in Columns Table

Item Name	Type/Range	Meaning
LastChangeTime	String	Date and time of last change
LastChangeBy	String	Operator name

Table 67 shows the predefined columns in the columns table.

Table 67 Predefined Columns in Columns Table

Column Name	Type/Range	Meaning	Technique Used For
SerialNum	String	Serial number	LC, GC
ProductNum	String	Product number	LC, GC, CE
BatchNum	String	Batch number	LC, GC, CE
ColDescription	String	Description (filling material)	LC, GC, CE
ColLength	Numeric	Length	LC, GC, CE
ColDiameter	Numeric	Diameter	LC, GC, CE
ParticleSize	Numeric	Particle size	LC, GC

Table 67 **Predefined Columns in Columns Table, continued**

Column Name	Type/Range	Meaning	Technique Used For
ColDeadUnit	Enumeration	Type of unit used for dead void % for dead volume ml for dead volume min for dead time	LC, GC
ColDead	Numeric	Value of dead volume or dead timerelated to the unit defined above	LC, GC
ColInstalled	Enumeration	Indicates whether this column is part of the current method	LC, GC, CE
BubbleCapillary	Boolean	Flag for installed bubble capillary in HPCE	CE
EffLength	Numeric	Effective length of capillary in HPCE	CE

When the user quits the Edit Column Parameters dialog box using OK the entries of this table, for which ColInstalled is set to 1 are copied to the BuiltInColumns table.This table has the same structure as the Columns table.

System Suitability Table

This table lists the parameters used for noise calculation in the header, and all report items used for statistics and limit settings in the columns. The noise parameters are in _DAMethod[1], the report items in _config[1].

Register name _DAMethod, _Config

Object number 1

Table name SysSuit

Table 68 shows the predefined header items in the columns table.

Table 68 Predefined Header Items in System Suitability Table (_DAMethod Register)

Item Name	Type/Range	Meaning
LastChangeTime	String	Date and time of last change
LastChangeBy	String	Operator name
Noise1From	Numeric	Start ret.time of first noise range
Noise1To	Numeric	End ret.time of first noise range
Noise2From	Numeric	Start ret.time of second noise range
Noise2To	Numeric	End ret.time of second noise range
Noise3From	Numeric	Start ret.time of third noise range
Noise3To	Numeric	End ret.time of third noise range
Noise4From	Numeric	Start ret.time of fourth noise range
Noise4To	Numeric	End ret.time of fourth noise range
Noise5From	Numeric	Start ret.time of fifth noise range
Noise5To	Numeric	End ret.time of fifth noise range
Noise6From	Numeric	Start ret.time of sixth noise range
Noise6To	Numeric	End ret.time of sixth noise range
Noise7From	Numeric	Start ret.time of seventh noise range
Noise7To	Numeric	End ret.time of seventh noise range

Table 69 shows the predefined columns in the columns table.

Table 69 **Predefined Columns in System Suitability Table**

Column Name	Type/Range	Meaning
Title	String	Description of report item
Var	String	Variable identifier used to access report item in table
TableID	Integer	Table identifier (see TableRef in _Config[1])
LowLimit	Numeric	Low limit of a report item (available only in _Config[1])
HighLimit	Numeric	High limit of a report item (available only in _Config[1])

Table References Table

This table contains all references between tables and report items.

Register name _Config

Object number 1

Table name TableRef

Table 70 shows the predefined columns in the _Config table references table.

Table 70 Predefined Columns in Table References Table

Column Name	Type/Range	Meaning
Register	String	Name of the register containing the specified table
TableName	String	Name of the specified table
TableID	Integer	Defines the table identifier
IndexVar	String	Index variable used in customized report

Available Modules

This table contains references to all available modules.

Register name _Config

Object number 1

Table name Modules

Table 71 shows the predefined header items in the _Config modules table.

Table 71 Predefined Header Items in Modules Table (_Config Register)

Item Name	Type/Range	Meaning
LastChangeTime	String	Date and time of last change
LastChangeBy	String	Operator name

Table 72 shows the predefined columns in the columns table.

Table 72 Predefined Columns in Table Modules Table

Column Name	Type/Range	Meaning
Name	String	Name of the module
Var	String	Short string to identify the module

Configured Instrument Modules

This table contains references to all available modules.

Register name _Config

Object number 1

Table name Modules

Table 73 shows the predefined header items in the _Config modules table.

Table 73 Predefined Header Items in Instrument Modules Table (_Config Register)

Item Name	Type/Range	Meaning
Location	String	Location of instrument
Specials	String	Special additional remarks

Table 74 shows the predefined columns in the columns table.

Table 74 Predefined columns in instrument modules table

Column Name	Type/Range	Meaning
NameRef	String	Link between Modules and ModuleInfo table
Number	Integer	Number of the module
SerialNumber	String	Serial number of the module
FWRevision	String	Firmware revision number of the module

Report Styles for Sequence Summary

This table lists the names of the report templates which can be used in the sequence summary report. The names are displayed in the list boxes of the sequence summary dialog box.

Register name _Config

Object number 1

Table name SeqSumRptStyle

Table 75 shows the predefined columns in the _Config report styles for sequence summary table.

Table 75 Predefined Columns in Sequence Summary Report Style Table

Column Name	Type/Range	Meaning
Title	String	Title which appears in the dialog box
Template	String	File name of report template (.tpl file)
Category	Enumeration	Category of template in the sequence summary report being part of: Header Configuration listing Sequence/sample table listing Logbook listing Method listing Analysis report Statistic of runs Summary

Manual Integration Window State Table

Register name _Config

Object number 1

Table name ManInt

Table 76 shows the predefined header items in the _Config manual integration window state table.

Table 76 **Predefined Header Items in Manual Integration Window State Table (_Config register)**

Item Name	Type/Range	Meaning
Win	Integer	Row number in Window table
MoveCursor	Enumeration	Saved value of Window table´s MoveCursor
DragCursor	Enumeration	Saved value of Window table´s DragCursor
HelpContext	Integer	Saved value of Window table´s HelpContext
NegTan	Enumeration	IntegMode parameter to be used in ManInt command
Feedback	Enumeration	Type parameter to be used in AppendSelect command
WinTitle	string	Saved value of Window table´s WinTitle

Notes About the Manual Integration Window State Table

- This table holds information about graphics windows that are not in their default state with respect to the mouse macros, MoveCursor, DragCursor, HelpContext, WinTitle.

Isoplot Color Scheme Table

Register name _Config

Object number 1

Table name "Iso_"+ColorSchemeName

Table 77 shows the predefined header items in the _Config manual integration window state table.

Table 77 **Predefined Columns in Isoplot Color Scheme Table (_Config Register)**

Item Name	Type/Range	Meaning
Color	RGB	Color value

Specific Commands for the Isoplot Color Scheme Table

DrawIsoPlot

Notes About the Isoplot Color Scheme Table

- At startup, the object _Config[1] is searched for the table names starting with Iso_. The rest of the table name is then used as color scheme name and added to the combo box which is displayed in the Isoplot Preferences dialog box.
- The number of rows in the ColorScheme table is limited to the maximum number of colors available to the video display unit. Typically 16 colors are sufficient.
- The ColorScheme table is used to draw the middle window of the IsoPlot dialog box, i.e. for drawing the spectral matrix.
- The colors are used in the following way: the first color is assigned to the lowest absorbance value to be shown in the spectra matrix. The last color is assigned to the highest absorbance value to be shown. The remaining colors are then assigned to the absorbance values in-between.

Integrator Events Table

Register name _DAMethod

Object number 1

Table name "Event_" + Detector + SignalId

For each signal to be integrated one integrator events table may exist in the object. The table name reflects the signal description. Example: the signal A of detector DAD1 is integrated with the events table Event_DAD1A.

Table 78 shows the predefined header items in the integrator events table.

Table 78 Predefined Header Items in Integrator Events Table

Item Name	Type/Range	Meaning
InitReject	Numeric ≥ 0	Initial area reject value
InitThresh	Numeric	Initial threshold value
InitWidth	Numeric ≥ 0	Initial peak width value
ShouldersOn	Boolean	1 shoulder detection enabled
		0 shoulder detection disabled

Table 79 show the predefined columns in the integrator events table.

Table 79 Predefined Columns in Integrator Events Table

Column Name	Type/Range	Meaning
Time	Numeric ≥ 0	Event time in minutes

Table 79 **Predefined Columns in Integrator Events Table, continued**

Column Name	Type/Range	Meaning
EventCode	Enumeration	Event code
		0 Reset baseline now
		If Value = 0: baseline height at current signal height
		If Value ≠ 0: baseline height at Value
		1 Reset baseline at next valley
		–1 Cancel 1
		2 Reset baseline at all valleys
		–2 Cancel 2
		3 Begin tangent skimming (on trailing edge of next peak)
		–3 Cancel 3
		4 No automatic Solvent detection
		–4 Cancel 4
		5 Hold baseline
		–5 Cancel 5
		6 Stop integration
		–6 Start integration
		7 Start summing peak areas
		–7 Stop summing peak areas
		8 Start slices — Value is slice width
		–8 Stop slices, resume integration
		9 Recognize negative peaks
		–9 Cancel 9

Table 79 **Predefined Columns in Integrator Events Table, continued**

Column Name	Type/Range	Meaning
		10 Reset baseline backwards
		11 Enable shoulder detection
		–11 Disable shoulder detection
		12 Set threshold — Value is threshold
		13 Set area reject — Value is area reject
		14 Set peak width
		If Value > 0: absolute peak width, no auto width tracking
		If Value = 0: resume auto width tracking
		If Value = –1: double current peak width
		If Value = –2: halve current peak width
		17 Integrate shoulders as peaks (shoulder detection must be enabled)
		–17 Cancel 17
		103 Split peak now
		104 Split peak at next valley
Value	Numeric	Event value

Specific Commands for the Integrator Events Table

EditEvents

EventIndex

IntegrateObj

GetEvent (macro)

DelEvent (macro)

Notes About the Integrator Events Table

- If a signal has to be integrated and a corresponding event table does not

exist for this signal, the default event table named Event is used instead. If even this table does not exist, hard-coded defaults are used.

Enhanced Integrator Events Table

Register name _DAMethod

Object number 1

Table name "Event_" + Detector + SignalId

For each signal to be integrated by the enhanced HP integrator, an integrator events table may exist in the object. The table name reflects the detector and signal. Example: the Event_ table for an FID detector on a GC instrument is Event_FID.

Table 80 shows the predefined header items in the enhanced integrator events table.

Table 80 **Predefined Header Items in the Enhanced Integrator Events Table**

Item Name	Type/Range	Meaning
InitReject	double	Initial area reject value. A value of zero accepts all peaks (even negative ones).
InitThresh	double	Initial threshold value for slope sensitivity
InitWidth	double	Initial peak width value at half height
ShouldersOn	enumeration	0 No
		1 Yes, Tangent separation
		2 Treat all shoulders with drop line separation
InitHeight	double	Initial value for rejecting peaks with insufficient height. A value of zero will accept all peaks (even negative ones)
InitBaseDrift	double	Estimated value of baseline drift
SplitSolvThres	double	Split Solvent Detection Value (absolute value)
SolventThres	double	Minimum slope required to detect a peak as solvent. Applies mostly to GC and LC detectors. A value of zero or 1.0E38 disables this function.
StartIntegMode	enumeration	0 First data point is start of baseline
		1 Start integrate with baseline at Value
		2 Find best starting baseline

Table 80 **Predefined Header Items in the Enhanced Integrator Events Table, continued**

Item Name	Type/Range	Meaning
StartIntegValue	double	Value for StartIntegMode, if mode = 1
EndIntegMode	enumeration	0 Last data point is end baseline
		1 End integrate with baseline at Value
		2 Find best end of baseline
EndIntegValue	double	Value for EndIntegMode, if mode = 1
BaseLineClamp	double	Data points below this value will be set to this value. Set to eliminate the effect of negative baseline noise.
BaseResetThres	double	Height units above current baseline to limit baseline reset within a cluster. End of peak markings that exceed this value will not reset the baseline automatically, but the peak cluster will continue. A setting of zero disables baseline resets for upward drifting baselines only. If no upper limit is desired, a value of >= 1.0E38 will disable.
UseForInteg	Boolean	0 shoulder detection enabled
		1 shoulder detection disabled

Table 81 show the predefined columns in the enhanced integrator events table.

Table 81 **Predefined Columns in the Enhanced Integrator Events Table**

Column Name	Type/Range	Meaning
Time	double	Event time in minutes

Table 81 **Predefined Columns in the Enhanced Integrator Events Table, continued**

Column Name	Type/Range	Meaning
EventCode	Integer (-22 to +22)	Event code
		0 Reset baseline now
		1 Reset baseline at next valley
		–1 Cancel 1
		2 Reset baseline at all valleys
		–2 Cancel 2
		3 Tail tangent skim
		–3 Cancel 3
		4 No automatic Solvent detection
		–4 Cancel 4
		5 Hold baseline
		–5 Cancel 5
		6 Stop integration
		–6 Start integration
		7 Start summing peak areas
		–7 Stop summing peak areas
		8 Start baseline to Value
		–8 Reset baseline through Value
		9 Recognize negative peaks
		–9 Cancel 9

Table 81 **Predefined Columns in the Enhanced Integrator Events Table, continued**

Column Name	Type/Range	Meaning
		10 Reset baseline backwards
		11 Enable shoulder detection
		–11 Disable shoulder detection
		12 Set threshold — Value is threshold
		13 Set area reject — Value is area reject
		14 Fixed peak width at Time
		–14 Automatic peak width
		15 Set baseline slope bias to Value
		16 Front tangent skim next peak
		–16 Cancel 16
		17 Shoulders mode
		0 = Off
		1 = Tangent
		2 = Droplines
		–17 Cancel shoulders mode
		18 Clamp signal at Value
		–18 Clamp signal off
		19 Set baseline maximum to Value
		–19 Cancel baseline maximum
		20 Tangent skim mode
		0 = Straight
		1 = Exponential
		2 = Exponential and straight

Table 81 **Predefined Columns in the Enhanced Integrator Events Table, continued**

Column Name	Type/Range	Meaning
		–20 Cancel tangent skim
		21 Detect spikes
		–21 Cancel 21
		22 Split peak
Value	double	Event value

Specific commands for the Enhanced Integrator Events Table:

Notes about the Enhanced Integrator Events Table:

- The user defined specific signal tables are also located in _DAMethod.
- Events ±1, ±8, ±4, 10, –17, ±19, ±21 are not used by LC or CE instruments.

Enhanced Integrator/Method Personality Table

Register name _DAMethod

Object number 1

Table name "IntMethod_" + Instrument (? Lew)

For each signal to be integrated with the enhanced integrator, an integrator/method personality table may exist in the object. The table name reflects the instrument type. Example: the IntMethod_ table for a GC is named IntMethod_GC.

Table 82 shows the predefined header items in the Enhanced Integrator Method Personality Table.

Table 82 **Predefined Header Items in the Integrator Method Personality Table**

Item Name	Type/Range	Meaning
PeakStartSigma	double	Force peak start to be at least n * front sigma from top of peak. If start of peak found after peak top - n * front sigma peak start is set to peak top - n * front sigma
PeakEndSigma	double	Force peak end to be at least n * end sigma from top of peak. If end of peak found before peak top + n * end sigma peak end is set to peak top + n * end sigma
TailingFactor	double	Not Needed (factor to multiply peak sigma for end of peak)
BaseLineWidth	double	This number indicates the minimum number of peak widths the signal has to be on baseline for a new baseline level. A number of 0 disables this feature. Typical number 3-8
DeadVolTime	double	The integrator does not expect any real peaks before the dead volume time, and stays in a forced baseline state for this time, only collecting noise and drift information
MinimumWidth	double	This parameter is for rejecting peaks with insufficient width. The width count is in user units. A Width reject of 0 will accept all peaks.
Plates	double	Expected plate number. If set, a peak broadening is expected, starting at dead volume time

Table 82 **Predefined Header Items in the Integrator Method Personality Table, continued**

Item Name	Type/Range	Meaning
PeakWidthBroad	double	The Peak Width Broadening rate is an attempt to see if there is a linear broadening of peak widths other than what can be expressed by the theoretical plates calculation. It is expressed as a factor of theoretical plates broadening. i.e. 0 means no broadening, 1 means the same broadening as through theoretical plates, 0.5 means half the broadening of theoretical plates
PeakTopType	byte	0 the very peak top is used (highest data point) 1 parabolic interpolation 2 center of gravity 3 gauss fit
BaseDetectMode	byte	Mode for calculation of the baseline level over one peak width: 0 Highest value 1 Average value 2 Lowest value
TanSkimMode	byte	This is the initial setting for tangent skim treatment. It will be superseded by the timed event exp. tail (revert to normal tail) 0 straight 1 exponential 2 algorithm's choice (a la Genie)
WidthCalc	byte	0 no width calculation 1 Width at half height 2 Width at base (2 sigma front + 2 sigma rear) 3 Width at base (2 sigma front + 2 sigma rear) 4 Composite (a la GENIE)

Table 82 **Predefined Header Items in the Integrator Method Personality Table, continued**

Item Name	Type/Range	Meaning	
PlatesCalcMode	byte	0	do not calculate
		1	Plate Number (Retention Time / Peak Sigma)
		2	Effective Plate Number ((Retention Time - DeadVolumeTime) / Peak Sigma)
PeakSepCalc	byte	0	Do not calculate
		1	Calculate

The actual default values assigned to the IntMethod_ table head items are determined by the instrument type (GC, LC, CE) from the method. The specific values for each of these configurations is shown in Table 83.

Table 83 **IntMethod Table Headers with GC, LC, and CE Defaults**

Table Header	GC	LC	CE
PeakStartSigma	3	3	
PeakEndSigma	4	4	
Tailing factor	1	1	
BaseLineWidth	3	3	
DeadVolTime	Conflict with column table	Conflict with column table	
MinimumWidth	0		
Plates	Conflict with column table	Conflict with column table	
PeakWidthBroad	1	1	
PeakTopType	1	1	

Table 83 **IntMethod Table Headers with GC, LC, and CE Defaults, continued**

Table Header	GC	LC	CE
BaseDetectMode	1	1	
TanSkimMode	2	2	
WidthCalc	4	4	
PlatesCalcMode	1	1	
PeakSepCalc	1	1	

NOTE See note about other information required for enhanced integrator like Signal Detection and Instrument from the method and the data files.

Enhanced Integrator/Detector Personality Table

Register name _DAMethod

Object number 1

Table name "IntDetect_" + Detector Class

For each signal to be integrated with the enhanced integrator, an Integrator/Detector personality table may exist in the object. The table name reflects the detector class. Example: the IntDetect_ table for a FID (a GC detector) is named IntDetect_FID.

Table 84 shows the predefined header items in the Enhanced Integrator/Detector Personality Table.

Table 84 **Predefined Header Items in the Enhanced Integrator/Detector Table**

Item Name	Type/Range	Meaning
Max Data Limit	double	Floating point number indicating maximum valid value for data transfer. Data point exceeding this value will clamp to this value, and set over range in peak list.
Max Linear Limit	double	Floating point number indicating maximum valid value for linear data. Data point exceeding this value will set distorted in peak list.
Min Data Limit	double	Floating point number indicating minimum valid value for data transfer. Data point less than this value will clamp to this value, and set under range in peak list.
Min Linear Limit	double	Floating point number indicating minimum valid value for linear data. Data point less than this value will under range in peak list.

Integrator Results Table

Register name ChromReg or any other register that contains chromatogram objects

Object number Any object of class chromatogram

Table name IntResults

Table 85 shows the predefined header items for the integrator results table.

Table 85 Predefined Header Items in Integrator Results Table

Item Name	Type/Range	Meaning
DateTime	String date	Date and time of integration
IntegStart	Numeric	Start time in minutes in Integrate command
IntegEnd	Numeric	End time in minutes in Integrate command

Table 86 shows the predefined columns in the integrator results table.

Table 86 Predefined Columns in Integrator Results Table

Column Name	Type/Range	Meaning
Flags	bitfield 16	Flags of the peak. This is a combination of separation code and error flags that occurred on the peak. The error flags occupy bits 15 to 8. The separation codes occupy bits 7 to 0. The separation code can be obtained by (Flags mod 256). The start code can be obtained by (Flags div 16, mod 16). The stop code can be obtained by (Flags mod 16). Each start/stop code can have the values: 0 baseline 1 penetration 2 valley 3 horizontal 4 forced 5 manual The combined separation codes then are: 0 (BB) baseline baseline 1 (BP) baseline penetration 2 (BV) baseline valley 3 (BH) baseline horizontal 4 (BF) baseline forced 5 (BM) baseline manual 16 (PB) penetration baseline 17 (PP) penetration penetration 18 (PV) penetration valley 19 (PH) penetration horizontal 20 (PF) penetration forced 21 (PM) penetration manual

Table 86 Predefined Columns in Integrator Results Table, continued

Column Name	Type/Range	Meaning	
		32 (VB)	valley baseline
		33 (VP)	valley penetration
		34 (VV)	valley valley
		35 (VH)	valley horizontal
		36 (VF)	valley forced
		37 (VM)	valley manual
		48 (HB)	horizontal baseline
		49 (HP)	horizontal penetration
		50 (HV)	horizontal valley
		51 (HH)	horizontal horizontal
		52 (HF)	horizontal forced
		53 (HM)	horizontal manual
		64 (FB)	forced baseline
		65 (FP)	forced penetration
		66 (FV)	forced valley
		67 (FH)	forced horizontal
		68 (FF)	forced forced

Table 86 **Predefined Columns in Integrator Results Table, continued**

Column Name	Type/Range	Meaning	
		69 (FM)	forced manual
		80 (MB)	manual baseline
		81 (MP)	manual penetration
		82 (MV)	manual valley
		83 (MH)	manual horizontal
		84 (MF)	manual forced
		85 (MM)	manual manual
		The error flags are a set of bits for errors that occurred during integration of the peak. Any combination is valid. The individual bits are:	
		11	OverRange
		10	UnderRange
		9	Aborted
		8	Distorted

Table 86 **Predefined Columns in Integrator Results Table, continued**

Column Name	Type/Range	Meaning	
PeakType	Enumeration	peak type	
		4	Front shoulder (standard integrator)
		5	Rear shoulder (standard integrator)
		8	Normal peak
		9	Solvent peak
		10	Tangent skimmed peak
		11	Negative peak
		12	Area Summation
		16	Manual peak
		17	Manual negative peak
		18	Manual tangent skimmed peak
		20	Recalculate solvent peak
		25	Tangent skimmed exponential
		26	Front shoulder, tangent skimmed
		27	Rear shoulder, tangent skimmed
		28	Front shoulder, dropline
		29	Rear shoulder, dropline
RetTime	Numeric	Retention/migration time (top of peak) in minutes	
Area	Numeric	Peak area	
Height	Numeric	Peak height	
Width	Numeric	Peak width	
Symmetry	Numeric	Peak symmetry	
Baseline	Numeric	Baseline height at retention/migration time	
TimeStart	Numeric	Time of peak start in minutes	

Table 86 **Predefined Columns in Integrator Results Table, continued**

Column Name	Type/Range	Meaning
LevelStart	Numeric	Signal height at peak start time
BaselineStart	Numeric	Baseline height at peak start time
TimeEnd	Numeric	Time of peak end in minutes
LevelEnd	Numeric	Signal height at peak end time
BaselineEnd	Numeric	Baseline height at peak end time

Specific Commands for the Integrator Results Table

IntegrateObj

ManInt

SplitPeak

DelPeak

Notes About the Integration Results Table

- All fields in the IntResults table are write protected. The table is generated only through the IntegrateObj command and the commands for manual integration.
- The IntegrateObj command will erase an existing old IntResults table before generating it anew. The commands for manual integration will merely change the existing IntResults table.
- If the table row is a shoulder (PeakType 4 or 5) the 12 numeric fields are only partially used. The RetTime is the time of the additional inflection point. All other numeric fields are set to zero.

System Suitability Results

This is an extension of the integration result table in ChromReg to keep the peak width results of the system suitability evaluation for each peak. It is used by the peak tools from the user contributed library.

Register name ChromReg or any other register that contains chromatogram objects

Object number Any object of class chromatogram

Table name IntResults

There are no predefined header items for this extension.

Table 87 shows the predefined columns in the system suitability results table.

Table 87 Predefined Columns in System Suitability Results Table

Column Name	Type/Range	Meaning
HalfWidth	Numeric	Peak width at half height
SigmaWidth	Numeric	5-sigma peak width
TailWidth	Numeric	Peak width at tailing (5% peak height)
TangentWidth	Numeric	Peak width given by tangent method
Moment1	Numeric	1st statistical moment
Moment2	Numeric	2nd statistical moment

Report Parameters Table

Register name _DAMethod

Object number 1

Table name RptParm

The report parameters table contains the parameters for formatting reports.

Table 88 shows the predefined header items in the report parameters table.

Table 88 Predefined Header Items in Report Parameter Table

Item Name	Type/Range	Meaning
Title	String	Report title to be printed above report
DestPrinter	Boolean	0 No output to printer
		1 Output to printer
DestScreen	Boolean	0 No output to screen
		1 Output to screen, with Notepad
DestFile	Boolean	0 No output to file
		1 Output to file, see FileName
FileTypeTXT	Boolean	0 No output as .TXT file
		1 Text output as .TXT file, if DestFile=1
FileTypeWMF	Boolean	0 No output as .WMF file
		1 Graphic output as .WMF file, if DestFile=1
FileTypeDIF	Boolean	0 No output as .DIF file
		1 Text output as .DIF file, if DestFile=1
FileTypeCSV	Boolean	0 No output as .CSV file
		1 Text output as .CSV file, if DestFile=1
FileName	String 6 characters	File name prefix for report output, used if DestFile=1.
Style	Integer	Report style

Table 88 **Predefined Header Items in Report Parameter Table, continued**

Item Name	Type/Range	Meaning
SortedBy	Enumeration	Sorting of peaks in report, if peaks are from several signals: 0 Sorted by retention/migration time 1 Sorted by signal
Orientation	Enumeration	Orientation of graphic output: 0 Portrait 1 Landscape
TimeSize	Numeric 0.2–1.0	Part of page width (if Portrait) or part of page height (if Landscape) to be used for time axis of graphic output
TimeMin	Numeric	Low limit of time axis
TimeMinBlank	Boolean	0 TimeMin is to be used 1 Value 0 is to be used
TimeMax	Numeric	High limit of time axis
TimeMaxBlank	Boolean	0 TimeMax is to be used 1 Value 0 is to be used
RspSize	Numeric 0.2–1.0	Part of page height (if Portrait) or part of page width (if Landscape) to be used for response axis of graphic output
RspMin	Numeric	Low limit of response axis
RspMinBlank	Boolean	0 RspMin is to be used 1 Value 0 is to be used
RspMax	Numeric	High limit of response axis
RspMaxBlank	Boolean	0 RspMax is to be used 1 Value 0 is to be used
PlotAnn	Boolean	0 Draw no annotations 1 Draw annotations, but see PlotNames

Table 88 **Predefined Header Items in Report Parameter Table, continued**

Item Name	Type/Range	Meaning
PlotNames	Boolean	0 Draw retention/migration times but no compound name annotations
		1 Draw compound names but no retention/migration time annotations
ChromOverlaid	Boolean	0 Draw each signal separately in its own sub-window
		1 Draw all signals in same sub-window
ChromNormalized	Boolean	0 Draw all signals with same scaling
		1 Draw each signal with its own scaling
AxisVisible	Boolean	Controls whether x- and y-axes, including scales and unit strings, are to be drawn
		0 Axes visible
		1 Axes not drawn; no space reserved
Topic	Bitfield 32	Set of bits that defines to draw (bit=1), or not to draw (bit=0) annotations with a specific topic. For a description of defined topics, see the Topic column in the Window table.
UseRanges	Boolean	0 Time and response range items will be ignored and assumed to be blank.
		1 Time and response range items are used.
		2 Autoscale (time range items are used, response range items are calculated).
DrawSignal	Boolean	0 PChrom macro will not plot the signals.
		1 PChrom macro will plot the signals in the report.
LongHeader	Boolean	Contents of page header in addition to the usual one line, stating file name, and sample name.
		0 Nothing additional
		1 Additional lines for extended file header information.

Table 88 Predefined Header Items in Report Parameter Table, continued

Item Name	Type/Range	Meaning
MultiPage	Boolean	Switches between normal and continuous plot. 0 Normal plot; Orientation, TimeSize, RspSize are used. Pages is ignored. 1 Continuous plot. Pages is used. Orientation, TimeSize, RspSize are ignored.
Pages	Integer > 0	Number of pages for continuous landscape plot. Used only if MultiPage=1.

The report parameters table has no predefined columns.

Automatic Library Search Parameters

Register name _DAMethod

Object number 1

Table name AutoID

Table 89 shows the predefined header items in the automatic library search parameters.

Table 89 Predefined Header Items in Automatic Library Search Parameters

Item Name	Type/Range	Meaning
FromSearchTime	Numeric	Retention time where automatic library search starts
ToSearchTime	Numeric	Retention time where automatic library search ends
SearchMode	Enumeration	Search mode:
		1 Automatic library search
		2 Target compound analysis using calibration table
		3 Target compound analysis using spectral library
DoPurity	Boolean	Checking of peak purity during spectral library search:
		0 No peak purity check
		1 Peak purity is checked

Table 90 shows the predefined columns in the automatic library search parameters.

Table 90 Predefined Columns in Automatic Library Search Parameters

Column Name	Type/Range	Meaning
LibFile	String	Name of spectral library
Library	Boolean	Usage of library:
		0 Library will not be used
		1 Library will be used

Quantification Parameters Table in _DAMethod Register

Register name _DAMethod

Object number 1

Table name QuantParm

The quantification parameters table contains sample data and method-wide parameters and defaults.

Table 91 shows the predefined header items in the quantification parameters table.

Table 91 **Predefined Header Items in Quantification Parameters Table**

Item Name	Type/Range	Meaning
CalTitle	String	Calibration table title
CalNewBy	String	Operator name
CalNewDateTime	String date	Time of the original table
CalNewTime	Time since 1970	Time of the original table
CalModBy	String	Operator name
CalModDateTime	String date	Time of the last change
CalModTime	Time since 1970	Time of the last change
QuantRev	String 9 characters	Software revision of quantification
SampleInfo	String	Sample info
SampleAmount	Numeric	Sample amount
Multiplier	Numeric	Result multiplier
UseSampleVars	Boolean	Controls whether to overwrite header items SampleAmount and Multiplier in QuantParm of ChromRes[1], and column Amount in ISTD of ChromRes[1] with values set from sample table through file SAMPLE.MAC: 0 Do not overwrite 1 Overwrite

Table 91 **Predefined Header Items in Quantification Parameters Table, continued**

Item Name	Type/Range	Meaning
Level	Link	Default level identifier
Level_	String	Name of table referenced by Level
RunType	Enumeration	Sample purpose
		0 Unspecified
		1 Sample run
		2 Blank run
		3 Calibration run
		4 Control run
		5 Instrument check out sample
SortCalData	Enumeration	Sorting of the calibration data to be applied by the CalibratePeaks command
		0 No sorting
		1 Compounds sorted by retention/migration time of main peak
		2 Compounds sorted alphabetically
Units	String	Result units
CalcBase	Enumeration	0 Area calculations
		1 Height calculations
CalcType	Enumeration	0 %
		1 ESTD
		2 Norm%
		3 ISTD
		4 ESTD%
		5 ISTD%

Table 91 **Predefined Header Items in Quantification Parameters Table, continued**

Item Name	Type/Range	Meaning
DoIdent	Enumeration	Controls how to identify peaks
		0 No identification
		1 Retention/migration time based identification
IDMethod	Enumeration	Identification method
		0 Closest peak
		1 Largest peak (depending on CalcBase)
IDMethodRef	Enumeration	Identification method for reference peaks
		0 Closest peak
		1 Largest peak (depending on CalcBase)
UseQualifiers	Boolean	0 Qualifiers not used
		1 Qualifiers used
QualTolerance	Numeric	Qualifier tolerance (in %)
RTWinRefAbs	Numeric	Absolute reference window
RTWinRefRel	Numeric	Relative reference window (in %)
RTWinAbs	Numeric	Absolute non-reference window
RTWinRel	Numeric	Relative non-reference window (in %)
TimeCurve	Enumeration	Curve fit for time reference peaks
		0 Piecewise
		1 Linear fit

Table 91 **Predefined Header Items in Quantification Parameters Table, continued**

Item Name	Type/Range	Meaning
UnkCompCalc	Enumeration	Controls how to report uncalibrated peaks found in the integration results 0 Do not report unknown compounds; known compounds reported with their own calibration data. 1 Use fixed response factor UnkCompRF for unknown compounds; known compounds reported with their own calibration data. 2 Use response factor of compound UnkComp for unknown compounds; known compounds reported with their own calibration data. 3 No report for unknown compounds; known compounds reported with fixed response factor UnkCompRF. 4 No report for unknown compounds; known compounds reported with response factor of compound UnkComp. 5 Use fixed response factor UnkCompRF for all compounds. 6 Use response factor of compound UnkComp for all compounds.
UnkCompRF	Numeric	Response factor for unknowns
UnkComp	Link	Link to compound number for response factor
UnkComp_	string	Name of table referenced by UnkComp
UnkCompISTD	Link	Link to compound nbr for ISTD for unknowns
UnkCompISTD_	String	Name of table referenced by UnkCompISTD

Table 91 **Predefined Header Items in Quantification Parameters Table, continued**

Item Name	Type/Range	Meaning
UnkCompISTDRF	Enumeration	Interpretation of the UnkCompRF field in the case where UnkCompISTD is given 0 no ISTD correction 1 UnkCompRF is interpreted as absolute response factor (i.e. the response factor of the ISTD will be taken into calculation as well) 2 UnkCompRF is interpreted as relative response factor (i.e. it is assumed that UnkCompRF is the ratio of the response factor of the unknown divided by the response factor of the ISTD)
RTUpdate	Enumeration	Retention/migration time update method 0 None 1 Average 2 Floating average 3 Boxcar
RTReplace	Boolean	Retention/migration time replace method 0 None 1 Replace
RTUpdWeight	Numeric	Weight (in %) of new run for floating average
PartialCalib	Boolean	Controls how to recalibrate if not all calibrated peaks are identified 0 No recalibration, report error 1 Recalibrate only identified peaks
UpdateAllRT	Boolean	Controls how to update retention/migration times 0 Only identified peaks are updated 1 All peaks, even if not identified, are updated

Table 91 **Predefined Header Items in Quantification Parameters Table, continued**

Item Name	Type/Range	Meaning
RespUpdate	Enumeration	Response update method
		0 None
		1 Average
		2 Floating average
		3 Boxcar
RespReplace	Boolean	Response replace
		0 None
		1 Replace
RespUpdWeight	Numeric	Weight (in %) of new run for floating average
RespBoxCarLen	Integer	Boxcar filter length to be used for response recalibrations
RTBoxCarLen	Integer	Boxcar filter length to be used for retention/migration time recalibrations
BoxCarName	String	File name for the boxcar
CurveType	Enumeration	Calibration curve type
		–1 No curve
		0 Piecewise
		1 Linear
		2 Quadratic
		3 Cubic
		4 Exponential
		5 Logarithmic
		6 Power
		7 Average slope of response/amount

Table 91 **Predefined Header Items in Quantification Parameters Table, continued**

Item Name	Type/Range	Meaning
CurveOrigin	Enumeration	Origin handling 0 Ignore 1 Include 2 Force 3 Connect
CurveWeight	Enumeration	Flexible calibration point weighting. 0 No weighting (all points have the same weight). 1 Number of recalibrations of the point is used to weight the point. 2 Point is weighted with the reciprocal of the response (1/x). 3 Point is weighted with the squared reciprocal of the response ($1/x^2$). 4 User-defined weights; weight values are not updated by the calibration.
SignalCorrWin	Numeric	Signal correlation window in minutes, if set to 0.0, signal correlation is off

The quantification parameters table has no predefined columns.

Compound Group Table in _DAMethod Register

Register name _DAMethod

Object number 1

Table name CompGroup

The compound group table contains descriptions of the compound groups.

Table 92 shows the predefined header items in the compound group table.

Table 92 Predefined Header Items in Compound Group Table

Item Name	Type/Range	Meaning
FirstComp_	String	Name of table referenced by FirstComp

Table 93 shows the predefined columns in the compound group table.

Table 93 Predefined Columns In Compound Group Table

Column Name	Type/Range	Meaning
Name	String	Group name
ShortName	String 3 characters	Name shown in user interface
FirstComp	link	Link to first compound in the group
Multiplier	Integer	Individual multiplier for each compound (default = 1)
CalcMethod	Enummeration	0: no group calibration
		1: group calibration
ISTD	Link	Link to ISTD table
NextISTD	Link	Link to next ISTD group

Compound Group Amounts Table in _DAMethod Register

Register name _DAMethod

Object number 1

Table name LevelAmount

Table 94 shows the predefined header items in the level amount table.

Table 94 Predefined Header Items in Level Amount Table

Item Name	Type/Range	Meaning
NextOnLevel_	String	Name of table referenced by NextOnLevel
NextOnCurve_	String	Name of table referenced by NextOnCurve

Table 95 shows the predefined column items in the level amount table.

Table 95 Predefined Column Items in Level Amount Table

Item Name	Type/Range	Meaning
Amount	Numeric	Group amount
Level	Link	Link to level table
Group	Link	Link to group table
NextOnLevel	Link	Link to next level amount
NextOnCurve	Link	Link to next group amount

Calibrated Compounds Table in _DAMethod Register

Register name _DAMethod

Object number 1

Table name Compound

The calibrated compounds table contains compound attributes common to all expected peaks and levels.

Table 96 shows the predefined header items in the calibrated compounds table.

Table 96 **Predefined Header Items in Calibrated Compounds Table**

Item Name	Type/Range	Meaning
TimeRef_	String	Name of table referenced by TimeRef
NextTimeRef_	String	Name of table referenced by NextTimeRef
ISTD_	String	Name of table referenced by ISTD
NextISTD_	String	Name of table referenced by NextISTD
CompGroup_	String	Name of table referenced by CompGroup
NextCompGroup_	String	Name of table referenced by NextCompGroup
FirstPeak_	String	Name of table referenced by FirstPeak
Multiplier	Numeric	Individual, compound specific result multiplier; used in conjunction with the global multiplier in QuantParm

Table 97 shows the predefined columns in the calibrated compounds table.

Table 97 Predefined Columns in Calibrated Compounds Table

Column Name	Type/Range	Meaning
Name	String	Compound name
UseQualifiers	Boolean	Qualifier use
		0 Not used
		1 Used
AmtLimitLow	Numeric	Low amount limit
AmtLimitHigh	Numeric	Upper amount limit
IsTimeref	Boolean	Compound is a time reference
		0 Not time reference
		1 Time reference
TimeRef	Link	Link to time reference information of this compound
NextTimeRef	Link	Next time reference or next dependent compound
IsISTD	Boolean	Set for ISTD compound
		0 Not ISTD
		1 ISTD
ISTD	Link	Link to ISTD reference information
NextISTD	Link	Next ISTD dependent compound or next ISTD compound
CompGroup	Link	Link to the compound group
NextCompGroup	Link	Link to the next compound in the group
FirstPeak	Link	Link to the first expected peak
Multiplier	Numeric	Individual, compound specific result multiplier; used in conjunction with the global multiplier in QuantParm
ControlLimits	Links	Link to control sample

Control Limits Table in DAMethod Register

Register name _DAMethod

Object number 1

Table name ControlLimits

The control limits table contains the low and high amount limits for system suitability of a particular peak in the calibration table.

There are no predefined header items.

Table 98 shows the predefined columns.

Table 98 **Predefined Columns in the Control Limits Table**

Item Name	Type/Range	Meaning
CLowArea	numeric	low limit of area for comparison with system suitability result
CLowHeight	numeric	low limit of height for comparison with system suitability result
CLowAmount	numeric	low limit of amount for comparison with system suitability result
CLowPlates	numeric	low limit of plates for comparison with system suitability result
CLowSymmetry	numeric	low limit of symmetry for comparison with system suitability result
CLowKPrime	numeric	low limit of kprime for comparison with system suitability result
CLowResolution	numeric	low limit of resolution for comparison with system suitability result
CHighArea	numeric	low limit of area for comparison with system suitability result
CHighHeight	numeric	low limit of height for comparison with system suitability result

Table 98 **Predefined Columns in the Control Limits Table, continued**

Item Name	Type/Range	Meaning
CHighAmount	numeric	low limit of amount for comparison with system suitability result
CHighPlates	numeric	low limit of plates for comparison with system suitability result
CHighSymmetry	numeric	low limit of symmetry for comparison with system suitability result
CHighKPrime	numeric	low limit of kprime for comparison with system suitability result
CHighResolution	numeric	low limit of resolution for comparison with system suitability result

Expected Peaks Table in _DAMethod Register

Register name _DAMethod

Object number 1

Table name Peak

The expected peaks table contains the expected peaks for all calibrated compounds.

Table 99 shows the predefined header items in the expected peaks table.

Table 99 Predefined Header Items in Expected Peaks Table

Item Name	Type/Range	Meaning
Signal_	String	Name of table referenced by Signal
Compound_	String	Name of table referenced by Compound
NextPeakComp_	String	Name of table referenced by NextPeakComp
NextPeakSignal_	String	Name of table referenced by NextPeakSignal
FirstCalPoint_	String	Name of table referenced by FirstCalPoint

Table 99 shows the predefined columns in the expected peaks table.

Table 100 Predefined Columns in Expected Peaks Table

Column Name	Type/Range	Meaning
ExpRetTime	Numeric	Expected time of the peak
RTWinCount	Integer	Update counter for retention/migration time
CalPeakType	Enumeration	Peak use
		0 Calibrated main peak
		1 Calibrated qualifier
		2 Calibrated, ignore peak
		3 New main peak
		4 New peak, (ignore)

Internal Standards Table in _DAMethod Register

Register name _DAMethod

Object number 1

Table name ISTD

The internal standards table contains information for the ISTD compounds.

Table 101 shows the predefined header items in the _DAMethod internal standards table.

Table 101 **Predefined Header Items in Internal Standards Table of _DAMethod Register**

Item Name	Type/Range	Meaning
Compound_	String	Name of table referenced by Compound
FirstDependent_	String	Name of table referenced by FirstDependent

Table 102 shows the predefined columns in the internal standard table.

Table 102 **Predefined Columns in Internal Standard Table of _DAMethod Register**

Column Name	Type/Range	Meaning
ShortName	String 3 characters	Identifier if the ISTD (e.g. "1")
Compound	Link	Link to the ISTD compound
FirstDependent	Link	First dependent compound
Amount	Numeric	Default for known amount in the sample

Time Reference Table in _DAMethod Register

Register name _DAMethod

Object number 1

Table name TimeRefGroup

The time reference table contains time references *groups*, not individual peaks. There is only one time reference group in use (group id +).

Table 103 shows the predefined header items in the time reference table.

Table 103 Predefined Header Items in Time Reference Table of _DAMethod Register

Item Name	Type/Range	Meaning
FirstTimeRef_	String	Name of table referenced by FirstTimeRef
FirstDependent_	String	Name of table referenced by FirstDependent

Table 104 shows the predefined columns in the time reference table.

Table 104 Predefined Columns in Time Reference Table of _DAMethod Register

Column Name	Type/Range	Meaning
ShortName	String 3 characters	Id of the reference group (e.g. "+")
TimeCurve	Enumeration	Time correction curve type
		0 Piecewise
		1 Linear
FirstTimeRef	Link	First time reference compound
FirstDependent	Link	First dependent compound

Calibration Level Table

Register name _DAMethod

Object number 1

Table name Level

The calibration level table contains level descriptions.

Table 105 shows the predefined header items in the calibration level table.

Table 105 Predefined Header Items in Calibration Level Table

Item Name	Type/Range	Meaning
CalPoint_	String	Name of table referenced by CalPoint

Table 106 shows the predefined columns in the calibration level table.

Table 106 Predefined Columns in Calibration Level Table

Column Name	Type/Range	Meaning
Name	String	Level name
ShortName	String 3 characters	Level identifier (e.g. "1")
CalPoint	Link	Link to peak calibration point

Calibration Points Table

Register name _DAMethod

Object number 1

Table name CalPoint

The calibration points table contains the calibration curve points (level averages) for all expected peaks and levels.

Table 107 shows the predefined header items in the calibration points table.

Table 107 Predefined Header Items in Calibration Points Table

Item Name	Type/Range	Meaning
Level_	String	Name of table referenced by Level
Peak_	String	Name of table referenced by Peak
NextOnLevel_	String	Name of table referenced by NextOnLevel
NextOnCurve_	String	Name of table referenced by NextOnCurve

Table 108 shows the predefined columns in the calibration points table.

Table 108 Predefined Columns in Calibration Points Table

Column Name	Type/Range	Meaning
Counter	Integer	Number of updates
Amount	Numeric	Average of amounts
Area	Numeric	Average of areas
Height	Numeric	Average of heights
Weight	Numeric	Point weighting factor (depends on the method CurveWeight setting, see the note below)
Level	Link	Link to level description
Peak	Link	Link to expected peak
NextOnLevel	Link	Next point in this level

Table 108 **Predefined Columns in Calibration Points Table, continued**

Column Name	Type/Range	Meaning
NextOnCurve	Link	Link to next curve point
LastHistory	Integer	Offset information to last calibration history point
Valid	Boolean	Controls whether this calibration point shall be used for calibration: 0 Do not use this point 1 Use this point

If CurveWeight is set to 4, (user-defined weight), it is the responsibility of the user to ensure that the weight is correct following a recalibration.

Following is an example of a macro used to prepare calibration points for a weighted LSQ calculation:

```
Name: SetCalPointWeights

! Prepares calibration points for weighted LSQ calculations
! The point weights are set to the inverse of the estimated point variance
!
! weight = 1/Variance
!
! Variance is a function of the point Amount. In this example the weight
! is set to 1/Amount^Power. The standard software takes care of the
! weighting by number of calibrations, 1/Amount, and 1/Amount^2.
!
! Note: For correct calculations of the correlation coefficients, for
!  the weighted curves the quantification DLL: HPQUA100.DLL is needed.

  parameter Power

  local NOfPoints, IPoint, PtAmount, PtWeight

  NOfPoints = TabHdrVal(_DaMethod[1],"CalPoint","NumberOfRows")

  for Ipoint = 1 to NOfPoints

    ! Reads the calibration point amount
    PtAmount = TabVal(_DaMethod[1],"CalPoint",IPoint,"Amount")

    ! Calculates the weight
    if PtAmount > 0 then
      PtWeight= 1/EXP(Power*LN(PtAmount))
    else
      PtWeight = 1
      ! Infinite weight of the 0 amount. Origin forced.
      SetTabVal_DaMethod[1],"CalPoint",IPoint,"Peak-CurveOrigin",2
    endif
```

```
    ! Set the new weight
    SetTabVal_DaMethod[1],"CalPoint",IPoint,"Weight",PtWeight

  next IPoint

! Sets the "User Defined Weights" parameter in the calibration
! data header

  SetTabHdrVal_DaMethod[1],"QuantParm","CurveWeight",4

! The curves are recalculated

  calibratepeaks,,_damethod[1]

  return

endmacro
```

Calibration Runs Table

Register name _DAMethod

Object number 1

Table name CalRun

The calibration runs table is a temporary table containing calibration history and is used for boxcar calibration.

The calibration runs table has no predefined header items.

Table 109 shows the predefined columns in the calibration runs table.

Table 109 Predefined Columns in Calibration Runs Table

Column Name	Type/Range	Meaning
Amount	Numeric	Known amount
Area	Numeric	Measured area
Height	Numeric	Measured height
RetTime	Numeric	Measured retention/migration time
Valid	Boolean	Controls whether this calibration point shall be used for calibration: 0 Do not use this point 1 Use this point
PrevHistory	Integer	Offset information to previous calibration history point
CalPeak	Integer	Peak number

Error Report Table in _DAMethod Register

Register name _DAMethod

Object number 1

Table name ErrorReport

The error report table contains errors and warnings that occurred during calibration.

The error report table has no predefined header items.

Table 110 shows the predefined columns in the error report table.

Table 110 Predefined Columns in Error Report Table

Column Name	Type/Range	Meaning
ErrorStatus	Enumeration	Type of this message 0 Comment 1 Warning 2 Error
ErrorCode	Integer	Error number
ErrorLine	Integer	Row number in table TabLine
TabLine	String 15 characters	Name of table where the error occurred
Comment	String	Message text

Result Header Table in ChromRes Register

Register name ChromRes

Object number 1

Table name QuantPar

Table 111 shows the predefined header items in the result header table.

Table 111 Predefined Header Items in Result Header Table

Item Name	Type/Range	Meaning
ResTitle	String	Result title
ResNewBy	String	Operator name
ResNewDateTime	String date	Time of the original table
ResNewTime	Time since 1970	Time of the original table
ResModBy	String	Operator name
ResModDateTime	String date	Time of the last change
ResModTime	Time since 1970	Time of the last change

Notes About the Result Header Table

- The result header table contains more header items than those described above. For these other elements see Table 91: CalNewBy, CalNewDateTime, CalNewTime, CalModBy, CalModDateTime, CalModTime, QuantRev, SampleInfo, SampleAmount, Multiplier, UseSampleVars, RunType, Units, CalcBase, CalcType, DoIdent, IDMethod, IDMethodRef, UseQualifiers, UnkCompCalc, UnkCompRF, UnkComp, UnkComp_, RTUpdate, RTReplace, RTUpdWeight, RespUpdate, RespReplace, RespUpdWeight, RespBoxCarLen, RTBoxCarLen, BoxCarName, SignalCorrWin.
- The header items SampleInfo, SampleAmount, and Multiplier are set when executing the SAMPLE.MAC macro file.

Signal Table in ChromRes Register

Register name ChromRes

Object number 1

Table name Signal

The signal table contains the signals names and other signal attributes.

For details of the predefined header items in the signal table, see Table 63.

For details of the predefined column items in the signal table, see Table 64. Table 112 shows *additional* predefined columns in the signal table.

Table 112 **Predefined Column in Signal Table of ChromRes Register**

Column Name	Type/Range	Meaning
RawDataFile	String	Rawdata directory path name
AreaSum	Numeric	Signal peak area sum
HeightSum	Numeric	Signal peak height sum
AmtSum	Numeric	Amount sum for signal (main peaks)
AmtSumUnk	Numeric	Amount sum for unknowns
CalSignal	Integer	Row number of this signal in _DAMethod[1] signal table during identification
XUnits	String	Units of the x-axis, typically "min"
YUnits	String	Units of the y-axis, detector dependent
DataType	Enumeration	Data type of signal, detector dependent. For possible values, see Table 64.

Compound Group Table in ChromRes Register

Register name ChromRes

Object number 1

Table name CompGroup

The compound group table contains information about compound groups.

For details of the predefined header items in the ChromRes compound group table, see Table 92.

For details of the predefined columns in the compound group table, see Table 93. Table 113 shows *additional* predefined columns in the compound group table.

Table 113 **Predefined Columns in Compound Group Table of ChromRes Register**

Column Nam	Type/Range	Meaning
AreaSum	Numeric	Sum of areas in group (main peaks only)
HeightSum	Numeric	Sum of heights in group
AmtSum	Numeric	Sum of amounts for group (ISTD compounds not included)
CalCompGroup	Integer	Row number of this group in _DAMethod[1] CompGroup table during identification

Compound Results Table in ChromRes Register

Register name ChromRes

Object number 1

Table name Compound

The compound results table contains the results from peak identification and quantification.

For details of the predefined header items in the compound results table, see Table 92.

For details of the predefined columns in the compound results table, see Table 93. Table 114 shows *additional* predefined columns in the compound results table.

Table 114 **Predefined Columns in Compound Results Table**

Column Name	Type/Range	Meaning
Amount	Numeric	Calculated amount
IsQualified	Boolean	Set for qualifier hit
IsInLimits	Enumeration	Amount below low limit or above high limit
		–1 Below low limit
		0 Above high limit
		1 Within limits
CalCompound	Integer	Row number to calibrated compound in _DAMethod Compound table during identification

Peak Results Table in ChromRes Register

Register name ChromRes

Object number 1

Table name Peak

The peak results table contains the results for individual processed peaks.

For details of the predefined header items in the peak results table, see Table 99.

For details of the predefined columns in the peak results table, see Table 100. Table 115 shows *additional* predefined columns in the peak results table.

Table 115 **Predefined Columns in Peak Results Table**

Column Name	Type/Range	Meaning
MeasRetTime	Numeric	Measured retention/migration time
CorrExpRetTime	Numeric	Expected time corrected using time references
IntPeakType	String 5 characters	Evaluated by integrator (combines peak type and separation code)
Area	Numeric	Peak area
Height	Numeric	Peak height
CalPeak	Integer	Row number to calibrated peak in _DAMethod Peak table during identification

Internal Standards Table in ChromRes Register

Register name ChromRes

Object number 1

Table name ISTD

The internal standards table contains information for the ISTD compounds.

Table 116 shows the predefined header items in the internal standards table.

Table 116 **Predefined Header Items in Internal Standards Table of ChromRes Register**

Item Name	Type/Range	Meaning
Compound_	String	Name of table referenced by Compound
FirstDependent_	String	Name of table referenced by FirstDependent

Table 117 shows the predefined columns in the internal standards table.

Table 117 **Predefined Columns in Internal Standards Table of ChromRes Register**

Column Name	Type/Range	Meaning
ShortName	String 3 characters	Identifier if the ISTD (e.g. "1")
Compound	Link	Link to the ISTD compound
FirstDependent	Link	First dependent compound
Amount	Numeric	Default for known amount in the sample

Notes About the Internal Standards Table

The column Amount is set when executing the SAMPLE.MAC macro file. The row number is determined by comparing the ISTD number in SAMPLE.MAC with the ShortName column in Table 117.

Control Limits Table in ChromRes Register

Register name ChromRes

Object number 1

Table name ControlLimits

See “Control Limits Table in DAMethod Register” on page 284.

Time Reference Table in ChromRes Register

Register name ChromRes

Object number 1

Table name TimeRefGroup

The time reference table contains time reference groups (not individual peaks). Currently there is only one time reference group in use (group id +).

Table 118 shows the predefined header items in the time reference table.

Table 118 Predefined Header Items in Time Reference Table of ChromRes Register

Item Name	Type/Range	Meaning
FirstTimeRef_	String	Name of table referenced by FirstTimeRef
FirstDependent_	String	Name of table referenced by FirstDependent

For details of the predefined columns in the time reference table, see Table 104. Table 119 shows *additional* predefined columns in the time reference table.

Table 119 Predefined Columns in Time Reference Table of ChromRes Register

Column Name	Type/Range	Meaning
CurveParm1	Numeric	1st parameter (of retention/migration time correction curve)
CurveParm2	Numeric	2nd parameter (of retention/migration time correction curve)
CurveParm3	Numeric	3rd parameter (of retention/migration time correction curve)
CurveParm4	Numeric	4th parameter (of retention/migration time correction curve)
CalTimeRef	Integer	Row number to group in _DAMethod TimeRefGroup table during identification

Error Report Table in ChromRes Register

Register name ChromRes

Object number 1

Table name ErrorReport

The error report table contains errors and warnings that occurred during calibration.

The error report table has no predefined header items.

For details of the predefined columns in the error report table, see Table 110.

TRegister name ChromRes

Object number 1

Table name AutoIDParms

Table 120 shows the predefined header items in the automatic library search results table.

Table 120 **Predefined Header Items in Automatic Library Search Results Table of ChromRes Register**

Item Name	Type/Range	Meaning
Modus	String	Search mode of automatic library search
DoPurity	String	Flag for peak purity
		0 Peak purity check was not done
		1 Peak purity check was done

Table 121 shows the predefined columns in the automatic library search results table.

Table 121 Predefined Columns in Automatic Library Search Results Table of ChromRes Register

Column Name	Type/Range	Meaning
LibraryName	String	Name of spectral library
LibThreshold	Numeric	Search threshold
PurThreshold	Numeric	Purity threshold
AbsThreshold	Numeric	Absorbance threshold
LeftWindow	Numeric	Left search window
RightWindow	Numeric	Right search window
WaveShift	Numeric	Wavelength shift

Automatic Library Search Results in ChromRes Register

Register name ChromRes

Object number 1

Table name Compound

There are no predefined header items to this extension.

Table 122 shows the additional predefined columns in the automatic library search results table..

Table 122 **Predefined Columns in Automatic Library Search Results Compound Table of ChromRes Register**

Column Name	Type/Range	Meaning
LibraryNum	Integer	Number of library
Match	Numeric	Match factor of spectral comparison
Purity	Numeric	Purity factor of purity check
IDResult	String	Label to mark the result of identification in the report
PurResult	String	Label to mark the result of purity check in the report
RetTimeLib	Numeric	Retention/migration time of first peak as given in spectral library
Identified	Enumeration	Flag to second proposal
		0 Not second proposal
		2 Second proposal

Noise Calculation Results in ChromRes Register

This list contains the result of a noise calculation which will be printed in the report.

Register name ChromRes

Object number 1

Table name Noise

There are no predefined header items to this extension.

Table 123 shows the predefined columns in the noise calculation results table.

Table 123 Predefined Columns in Noise Calculation Results Table of ChromRes Register

Column Name	Type/Range	Meaning
Signal	Integer	Link to signal table
NextNoiseSignal	Integer	Link to next noise results of the same signal
TimeFrom	Numeric	Start ret.time of noise range
TimeTo	Numeric	End ret.time of noise range
CorrCoeff	Numeric	Correlation coefficient of linear regression
Noise	Numeric	Noise given by six times the standard deviation of regression
NoisePP	Numeric	Noise given by maximum peak-to-peak distance (drift corrected)
Drift	Numeric	Drift of the baseline
Coeff_A0	Numeric	Intercept of linear regression
Coeff_A1	Numeric	Slope of linear regression
Sd_A0	Numeric	Standard deviation of intercept
Rsd_A0	Numeric	Relative standard deviation of intercept

Table 123 **Predefined Columns in Noise Calculation Results Table of ChromRes Register, continued**

Column Name	Type/Range	Meaning
Tsd_A0	Numeric	Standard error of intercept
Sd_A1	Numeric	Standard deviation of slope
Rsd_A1	Numeric	Relative standard deviation of slope
Tsd_A1	Numeric	Standard error of slope
Dev	Numeric	Maximum deviation

Noise Calculation Results in ChromRes Register

This is an extension of the signal table.

Register name ChromRes

Object number 1

Table name Signal

There are no predefined header items to this extension.

Table 124 shows the predefined columns in the noise calculation results table.

Table 124 Predefined Columns in Noise Calculation Results Table of ChromRes Register

Column Name	Type/Range	Meaning
FirstNoise	Integer	Link to the first noise entry of this signal into the noise table

System Suitability Results in ChromRes (extension to peak table)

This is an extension of the Peak table in ChromRes to keep the results of the system suitability evaluation for each peak.

Register name ChromRes

Object number 1

Table name Peak

There are no predefined header items to this extension.

Table 125 shows the additional predefined columns in the system suitability results table..

Table 125 **Predefined Columns in System Suitability Results Table**

Column Name	Type/Range	Meaning
ObjNum	Integer	Object of signal in ChromReg
StartIndex	Integer	Index of data point of peak start
EndIndex	Integer	Index of data point of peak end
StartTime	Numeric	Retention time of peak start
EndTime	Numeric	Retention time of peak end
TimeIncrement	Numeric	Time increment between data points
DataPoints	Numeric	Number of data points belonging to this peak
kPrime	Numeric	K´ value
FrontSigma	Numeric	Retention time of start point of peak width based on 5-sigma= method
SigmaWidth	Numeric	5-sigma peak width
SigmaHeight	Numeric	Height for 5-sigma peak width
FrontTail	Numeric	Retention time of start point of peak width given at tailing
TailWidth	Numeric	Peak width at tailing (5% peak height)

Table 125 **Predefined Columns in System Suitability Results Table, continued**

Column Name	Type/Range	Meaning
FrontTangent	Numeric	Retention time of start point of peak width based on tangent method
TangentWidth	Numeric	Peak width given by tangent method
FrontTangEnd	Numeric	Ret.time of end point of front tangent
FrontTangentY	Numeric	Y-value of end point of front tangent
BackTangEnd	Numeric	Ret.time of end point of back tangent
BackTangentY	Numeric	Y-value of end point of back tangent
FrontHalfWidth	Numeric	Retention time of start point of peak width at half height
HalfWidth	Numeric	Peak width at half height
HalfWidthHeight	Numeric	Y-value of peak width at half height
Sigma	Numeric	Square root of 2^{nd} moment
Skew	Numeric	Skew
Excess	Numeric	Excess
Moment0	Numeric	0^{th} statistical moment
Moment1	Numeric	1^{st} statistical moment
Moment2	Numeric	2^{nd} statistical moment
Moment3	Numeric	3^{rd} statistical moment
Moment4	Numeric	4^{th} statistical moment
TailFact	Numeric	Tailing factor
TangentPlates	Numeric	Plate number based on tangent method
SigmaPlates	Numeric	Plate number based on 5-sigma method
HalfWidthPlates	Numeric	Plate number based on half width method
StatPlates	Numeric	Plate number based on moment calculation

Table 125 **Predefined Columns in System Suitability Results Table, continued**

Column Name	Type/Range	Meaning
TangentResol	Numeric	Resolution based on tangent method
SigmaResol	Numeric	Resolution based on 5-sigma method
HalfWidthResol	Numeric	Resolution based on half width method
StatResol	Numeric	Resolution based on moment calculation
Alpha	Numeric	Selectivity

Automatic Library Search Results in WorkReg Register

Register name WorkReg

Object number 1

Table name Proposals

There are no predefined header items.

Table 126 shows the predefined columns in the automatic library search results table of the WorkReg register.

Table 126 **Predefined Columns in Automatic Library Search Results Table**

Column Name	Type/Range	Meaning
ObjNum	Integer	Object number of signal from where the peak was taken.
PeakNum	Integer	Number of peak referring to the integration result table.
Num	Integer	Number of the first proposal.
Match	numeric	Match factor of the first proposal.
Name	string	Name of the compound given by the first proposal.
Num2	Integer	Number of the second proposal.
Name2	Integer	Name of the compound given by the second proposal.
KeepNum	Integer	Number of the proposal that was put aside as a better proposal for this compound was found.
KeepName	string	Name of the compound that was put aside as a better proposal for this compound was found.
P(i)	string	i^{th} proposal for this compound.
M(i)	numeric	Match of the i^{th} proposal
T(i)	numeric	Ret.time given in the library of i^{th} proposal.
L(i)	Integer	Number of the library used by the i^{th} proposal.

Results of System Suitability Evaluation in Peak Tools: Results Table

Register name PeakPerfReg

Object number 1

Table name Results

There are no predefined header items.

Table 127 shows the predefined columns in the results table.

Table 127 **Predefined Columns In System Suitability Evaluation In Peak Tools**

Column Name	Type/Range	Meaning
Original	Integer	Result value corresponding to description

Listing of Methods for Sequence Summary

Register name SeqInfoReg

Object number 1

Table name Methods

There are no predefined header items.

Table 128 shows the predefined columns in the listing of methods for the sequence summary table.

Table 128 **Predefined Columns in Listing of Methods for Sequence Summary**

Column Name	Type/Range	Meaning
Name	String	Name of the method
Path	String	Directory path in which method is saved
CompTabTable	String	Name of the calibration table related to this method
Warning	String	Warning
AmtUnits	String	Units for the amount
ResponseUnits	String	Units for the response
CalcType	Integer	Type of quantitative calculation
Statistics	Enumeration	Check whether data available for statistical evaluation in this method
		0 No data available
		1 Data available
Page	Integer	Page number where method listing is printed in sequence summary
FirstRun	Integer	Link to first run of this method
LastRun	Integer	Link to first run of this method in ResultFiles table
StartVial	Integer	Start vial
EndVial	Integer	Last vial
InjVial	Integer	Injections per vial

Listing of Data File Names for Sequence Summary

Register name SeqInfoReg

Object number 1

Table name ResultFiles

There are no predefined header items.

Table 129 shows the predefined columns in the listing of data file names for the sequence summary table.

Table 129 **Predefined Columns in Listing of Data File Names for Sequence Summary**

Column Name	Type/Range	Meaning
Name	String	Name of data file
Path	String	Directory path in which data file is saved
RunType	Enumeration	Type of run
		0 Calibration run
		1 Blank sample run
		2 Sample run
Method	Integer	Link to methods table
VialNum	Integer	Vial number
InjNum	Integer	Injection number
RunNum	Integer	Run number
NextRun	Integer	Link to row of next run in this table
Sample	String	Sample name
SampleAmount	Numeric	Sample amount
Multiplier	Numeric	Multiplier
NrOfCompounds	Integer	Number of compounds reported

Calibration Tables in Sequence Summary

Register name SeqInfoReg

Object number 1

Table name "CompTab"+val$(methodNumber)

There are no predefined header items.

Table 130 shows the predefined columns for the calibration tables in the sequence summary table.

Table 130 **Predefined Columns for Calibration Tables In Sequence Summary Table**

Column Name	Type/Range	Meaning
Name	String	Name of compound
FirstCalRun	Integer	Link to first calibration run
PrevCalRun	Integer	Link to previous calibration run
FirstCalRow	Integer	Link to row in the compound table of the ChromRes register
FirstSampleRun	Integer	Link to first sample run
PrevSampleRun	Integer	Link to previous sample run
FirstSampleRow	Integer	Link to row in the compound table of the ChromRes register
NextRun(i)	Integer	Link to next run of this run type
NextRow(i)	Integer	Link to row in the compound table of the ChromRes register for the next run

Font Table for Text Annotations

Register name _DAMethod, _Config

Object number 1

Table name Font

There are no predefined header items.

Table 131 shows the predefined columns for the font table.

Table 131 Predefined Columns for Font Table

Column Name	Type/Range	Meaning
FaceName	String 31 characters	Name of the font. Any installed font can be mentioned here. Examples are: Arial: a proportional, rotatable font Courier: a non-proportional font
FontStyle	Bitfield 4	Set of 4 bits that define additional effects of the font´s appearance. Any combination is valid Defined bits are: 0 Bold 1 Italic 2 Underscore 3 Strikeout
Size	Integer	Character height in points
Color	RGB	Color of text

Table 131 Predefined Columns for Font Table, continued

Column Name	Type/Range	Meaning
Justify	Enumeration	Defines how the text is justified relative to the text. Permitted values are: 1 Top left 2 Mid left 3 Bottom left 4 Top mid 5 Mid mid (center) 6 Bottom mid 7 Top right 8 Mid right 9 Bottom right
Angle	Integer	Angle of text rotation. Any number is valid as input. Upon drawing the text, the angle is taken modulo 360, i.e. -90 is equivalent to 270 degrees. Not all fonts can be rotated.

Notes About the Font Table

- The columns in the Font table correspond to the items that can be set with the commands SetAnnText and SetAnnVal for any specific Text Annotation.
- Row 2 in the Font table is used to describe the elements of the Font dialog box for retention/migration time, and the compound name annotations.
- Whenever the Font table changes, it is copied to the _Config[1], Font table. No additional rows should be added to the _Config[1], Font table as they will be lost when the table is next copied.

Instrument Data Curves Table

Register name _DAMethod

Object number 1

Table name InstCurv

Table 132 shows the predefined header items in the instrument data curves table.

Table 132 **Predefined Header Items in Instrument Data Curves Table**

Item Name	Type/Range	Meaning
A	Boolean	%A Solvent gradient according to method
		0 Don't show
		1 Show instrument data from LCDIAG.REG
B	Boolean	%B Solvent gradient according to method
		0 Don't show
		1 Show instrument data from LCDIAG.REG
C	Boolean	%C Solvent gradient according to method
		0 Don't show
		1 Show instrument data from LCDIAG.REG
D	Boolean	%D Solvent gradient according to method
		0 Don't show
		1 Show instrument data from LCDIAG.REG
Flow	Boolean	Flow according to method
		0 Don't show
		1 Show instrument data from LCDIAG.REG
Temp	Boolean	Measured temperature
		0 Don't show
		1 Show instrument data from LCDIAG.REG

Table 132 **Predefined Header Items in Instrument Data Curves Table, continued**

Item Name	Type/Range	Meaning
HighPress	Boolean	Measured high pressure
		0 Don't show
		1 Show instrument data from LCDIAG.REG

There are no predefined columns.

Customized Report Designer Parameters

This list contains the additional parameters that are used in the customized report designer to influence the printing of the report.

Register name _Config

Object number 1

Table name CustReport

Table 133 shows the predefined header items in the report designer parameters.

Table 133 Predefined Header Items in Report Designer Parameters Table

Item Name	Type/Range	Meaning
ViewAhead	Integer	Number of lines to view ahead when setting a new page.
SkipNegatives	Enumeration	0 All values will be printed
		1 Negative values will not be printed
Replacement	String	Text to be placed when a value is not printed
RepeatSame	Enumeration	Repeating of identical report items when printing table rows:
		0 No repetition
		1 Report item will be repeated
MaxPageNum	Integer	Limit for pages to be printed
OnePage	Enumeration	Flag to define usage of MaxPageNum
		0 Do not use MaxPageNum
		1 Use MaxPageNum
DateFormat	String	Default format for printing date
TimeFormat	String	Default format for printing time

Table 134 shows the predefined columns in the report designer parameters table.

Table 134 Predefined Columns in Report Designer Parameters Table

Column Name	Type/Range	Meaning
Month	String	Name of the month used to print date

Customized Report Designer: Repeat Report Items

This list contains information about items that might be repeated on a new page, when printing table rows after the reaching the end of the current page.

Register name WorkReg

Object number 1

Table name RepeatItems

There are no predefined header items.

Table 135 shows the predefined columns in the repeat report items of the report designer.

Table 135 Predefined Columns in Repeat Report Items of Report Designer

Column Name	Type/Range	Meaning
Item	String	Name of report item
TextFlag	Enumeration	Specifies this item
		0 Item is not text
		1 Item is text
Value	Numeric	Value of the item if it is not text
Format	String	Print of format
xPos	Integer	Horizontal (x) position of item
yPos	Integer	Vertical (y) position of item

Customized Report Designer: Totals

This list records information about items in which totals and averages are calculated when the sum up flag is set for that item, and contains the totals of one loop level.

Register name WorkReg

Object number 1

Table name Totals

There are no predefined header items.

Table 136 shows the predefined columns in the totals table of the report designer.

Table 136 Predefined Columns in Totals Table of Report Designer

Column Name	Type/Range	Meaning
Total	Numeric	Sum of values of a specific report item
NoOfValues	Integer	Number of value used in summation to obtain the value of the total
Mean	Numeric	Mean value
Sd	Numeric	Standard deviation of mean value
Rsd	Numeric	Relative standard deviation of mean value
StdErr	Numeric	Standard error of mean value

Customized Report Designer: Limits

This list contains the limits of statistical items and the general limits of the report items.

Register name WorkReg

Object number 1

Table name StatItems

There are no predefined header items.

Table 137 shows the predefined columns in the limits table of the report designer.

Table 137 Predefined Columns in Limits Table of Report Designer

Column Name	Type/Range	Meaning
Item	String	Description of the report item
Ref	Integer	Reference to the row in the template
LowLimit	Numeric	Low limit of report item
HighLimit	Numeric	High limit of report item

Customized Report Designer: Template

This list contains the definition of the report template.

Register name WorkReg

Object number 1

Table name Template

Table 138 shows the predefined header items in the report designer parameters.

Table 138 Predefined Header Items in Report Designer Parameters

Item Name	Type/Range	Meaning
HeadLeft	String	Left part of report header
HeadCenter	String	Center part of report header
HeadRight	String	Right part of report header part of report header
FootLeft	String	Left part of report footer
FootCenter	String	Center part of report header
FootRight	String	Right part of report header

Table 139 shows the predefined columns in the report designer template.

Table 139 Predefined Columns in Report Designer Template

Column Name	Type/Range	Meaning
xPos	Integer	Horizontal (x) position
yPos	Integer	Vertical (y) position
Width	Integer	Field width or width of plot area
Decimals	Integer	Number of decimals for a numeric item, or height of a plot area, or number of lines for a string.

Table 139 Predefined Columns in Report Designer Template, continued

Column Name	Type/Range	Meaning
Adjust	Enumeration	Alignment
		1 Right justified
		2 Left justified
		3 Centered
		4 Left and right justified
SumUp	Enumeration	Numeric value
		0 No sum up
		1 Sum up
		Plot macro
		0 Use width and height as in template
		1 Use width and height as in report parameters
		Text in loops
		0 Print text immediately
		1 Print text before first report item printed in a loop
ItemID	Enumeration	Identifier of report item
		1 Text
		2 String
		3 Printer control
		4 Numeric
		5 Macro
		6 Loop
		7 Condition
TableID	Integer	Identification of table to access report item content (see TableRef table in _Config[1])
ItemText	String	Description of report item

Display Description Table (DDT)

Register name _Config

Object number 3

Table name any table (specified as DispTabName in commands)

Table 140 shows the predefined header items in the display description table.

Table 140 Predefined Header Items in Display Description Table

Item Name	Type/Range	Meaning
DispRowNum	Boolean	0 No row numbers will be displayed; RowNumTitle, RowNumWidth, RowFormat are ignored 1 Row numbers will be displayed to the left of the table, according to RowNumTitle, RowNumWidth, RowFormat
RowNumTitle	String	Title of row numbers to be displayed centered above the row numbers
RowNumWidth	Integer	Number of chars to be reserved for the row numbers
RowFormat	String	Format string for the row numbers
TitleFont	String	Font specification for all column titles and the row numbers (if displayed). String has same syntax as in WIN.INI. If empty, application defined default "Table Title" font from WIN.INI is used.
ElemFont	String	Font specification for all table elements in all columns. String has same syntax as in WIN.INI. If empty, application defined default "Table Elem" font from WIN.INI is used.
FixedCols	Integer	Number of displayed columns (counted from the left side) that are not allowed to scroll horizontally; disregarding the row numbers which are never scrolled horizontally.
WindowClass	Integer	Window class

Table 141 shows the predefined columns in the display description table.

Table 141 Predefined Columns in Display Description Table

Column Name	Type/Range	Meaning
Title	String	Title of column (to be displayed centered above the column)
Control	Enumeration	Defines what kind of control to be used for the individual fields of the column. Allowed values are: 0 Table elements are read only, displayed elements will be truncated if Width (see below) is too small. 1 Table elements are read only, use Edit control because of scrolling feature for long strings. If the users make any changes in the field, they have to be warned that all changes will be ignored. 2 Table elements may be edited, use Edit control 3 Table elements are read only integers which are enumerated strings, i.e. they have to be transformed into strings before displaying. The enumeration strings are stored in enumstrings. Enumstrings is to be interpreted as an array of strings with the Integer as index into the array. Note that in this case Type (see below) refers to a numeric access, whereas Format (see below) refers to the display of a string. The displayed strings will be truncated if width (see below) is too small.

Table 141 **Predefined Columns in Display Description Table, continued**

Column Name	Type/Range	Meaning
		4 Table elements are integers, which may be edited. They are enumerated strings, i.e. they have to be transformed into strings before displaying. The enumeration strings are stored in enumstrings. Enumstrings is to be interpreted as an array of strings with the Integer as index into the array. Use a drop-down combination list. Note that in this case Type (see below) refers to a numeric access, whereas Format (see below) refers to the display of a string. The displayed strings will be truncated if width (see below) is too small.
		5 Table elements are strings which may be edited, use drop-down combination list with proposed strings. The proposed strings are stored in enumstrings. The displayed strings will be truncated if width (see below) is too small.
		6 Table elements are strings which may be edited, use drop-down combination box with proposed strings plus an editable empty field. The proposed strings are stored in enumstrings. The displayed strings in the drop-down combination box will be truncated if width (see below) is too small. The editable field has a horizontal scrolling feature.
		7 Table elements are check boxes: 0 for unchecked and 1 for checked.
		Note that the Control values 3 and 4 give the feature of translating enumerated codes into local language.
EnumStrings	String	Used only if Control in [3,4,5,6]. Holds the strings that are to be displayed in the table element. If enumerated, first string represents index "0", and so on. Strings are separated by a new line "\n" character.
Width	Integer	Number of chars to be reserved for the column

Table 141 **Predefined Columns in Display Description Table, continued**

Column Name	Type/Range	Meaning
Format	String	Format string for the column, according to the standard C function printf. See Help text for SPrinfF command for more details.
Justify	Enumeration	Justification code: 0 Left justify 1 Centered, fill with leading blanks 2 Right justify, fill with leading blanks
Source	String	Source specification for the column, see the specific commands for more information.
Convert	Bitfield 32	Used only if the displayed column contains strings. Controls whether and how the text is converted before assigning it to the internal table element. Defined bits are: 0 Convert from the ANSI character set to the OEM character set and then back to ANSI. This is useful for columns that contain file names. 1 Remove leading blanks (Left Trim) 2 Remove trailing blanks (Right Trim) 3 Convert all characters to uppercase 4 Convert all characters to lowercase

Specific Commands for the Display Description Table

EdTab

EdObjTab

EdDataTab

Notes About the Display Description Table

- A display description table is used to describe the contents and format of a tabular display which is presented by one of the above mentioned specific commands.

- The storage format of the items to be displayed must not necessarily be in a table. Only EdTab displays a table inside an object. The other commands display information about objects, object headers, and data blocks that can be presented and edited in tabular form.
- The display description table can be written in a way that it can be used for more than one table with similar structure. The structure does not even need to be the exactly the same, it is only necessary that each of the displayed tables contain all header items and columns that are requested in the display description.
- The display description is suitable for these classes of tabular display:

EdTab	display a table with selected columns
EdObjTab	display object headers across all objects in a register
EdDataTab	display a data block
EdRegTab	display list of registers
EdObjHdrTab	display list of object header and table names in an object
EdTabHdrTab	display list of table header and column names in a table

- Generally the tabular display allows these operations:
  horizontal and vertical scrolling,
  editing the displayed items,
  inserting and deleting rows,
  cutting, copying, pasting and paste-appending rows, involving the standard Windows Clipboard.
  Some of these operations are not available on all classes of tabular display, see the specific commands for more information.
- Format strings are, for example:

%s	string, all characters
%13s	string, first 13 characters, right justified
%–13s	string, first 13 characters, left justified
%5d	Integer number, 5 digits, right justified
%–7.2f	fixed-point number, 7 characters total, 2 places of decimals, left justified
%10.2E	floating-point number with exponent (E), 10 characters total, 2 places of decimals, right justified
%10.4g	number, shortest possible representation (fixed point,

floating point, Integer) depending on actual value, 10 characters total, up to 4 places of decimals, right justified

Output Description Table (ODT)

Register name _Config

Object number 3

Table name Any table (specified as FormatDescTabName in commands). The table format is an extension of the Display Description Table (DDT), i.e. any ODT is also usable as a DDT.

Table 142 shows the predefined header items in the output description table.

Table 142 Predefined Header Items in Output Description Table

Item Name	Type/Range	Meaning
DispRowNum	Boolean	Unused but compatible to DDT
RowNumTitle	String	Unused but compatible to DDT
RowNumWidth	Integer	Unused but compatible to DDT
RowFormat	String	Unused but compatible to DDT
TitleFont	String	Unused but compatible to DDT
ElemFont	String	Unused but compatible to DDT
FixedCols	Integer	Unused but compatible to DDT

Table 143 shows the predefined columns in the output description table.

Table 143 Predefined Columns in Output Description Table

Column Name	Type/Range	Meaning
ExtractorRowNr	Integer	Refers to row in extractor control tbl for mapping to ColumnInfo. ColumnInfo gives the rule to calculate the item.
Width	Integer	Number of characters to be reserved for the column.Defines the max. number of characters allowed in this field. Strings exceeding this max. length are truncated. Numbers exceeding this max. length are shown as ###. No CP command/function errors are reported. Width controls the layout of the table.
Format	String	Format string for the column, according to the standard C function: printf.
ExpChars	Integer	Defines the number of characters reserved for the exponential display defined by any format specification (Format, ExpFormat, NegFormat, etc.) 0 Use complete c-lib output which is e+001 1 to 3: Leading + and leading0 in the exponent are replaced by ending blanks until the number of ExpChars is reached.
ZeroFormat	String	A string that describes the format for values that are either zero or an empty string. It has the same syntax possibilities as the Format column. If ZeroFormat is empty, Format is used instead.
NegFormat	String	A string that describes the format for negative values. It has the same syntax possibilities as the Format column. If NegFormat is empty, Format is used instead. Only used for numeric fields.

Table 143 Predefined Columns in Output Description Table, continued

Column Name	Type/Range	Meaning
ExpFormat	String	A string that describes the format to be used for exponential formatting. It has the same syntax possibilities as the Format column. If ExpFormat is empty, MinPower and MaxPower are ignored, and Format is used instead. In this case the exponent of the value to be formatted is checked. If it is within the range MinPower:MaxPower, the Format column is used, otherwise the ExpFormat column is used for formatting. Only valid for numeric fields.
MinPower	Integer	An Integer that is the minimum value of the exponent where the value is still formatted as fixed point string.
MaxPower	Integer	An Integer that is the maximum value of the exponent where the value is still formatted as fixed point string.
RedundancyStr	String	The string to be written when the item is redundant.
ComplementForm	String	See Format. If the format is numeric, then the ComplementFormat should be a string and vice versa. This is required because the extractor allows conditional expressions where the type of the result can change to another data type. This field can be left blank.
NumericSum	Integer	Sum of all values in this call of the CP formatting command plus the value of this field at the start of the command.
CalcSum	Boolean	If TRUE, the NumericSum is calculated.
EnumStrings	String	As in Display Description Table. Holds the strings that are to be displayed in the table element. If enumerated, first string represents "0", and so on. Strings are separated by a newline "\n"

Specific Commands for the Output Description Table

PipeTab

Notes About the Output Description Table

- The Output Description Table contains many criteria concerning the

control of the output layout. The table is based on the DDT as its elements have proved to be useful. Possible callback macros, processing user input from a true DDT, are ignored by the formatter.

- These enhancements to the DDT are fully backward compatible. This means that commands for table editing, for example EdTab, need not be changed as they ignore the new columns. Therefore, the ODT could also be used for those commands, and the ODT can reside in the same object as all DDTs, namely _Config[3].

Sequence Parameter Table

The sequence parameters table contains the sequence parameters and other global sequence settings. This table contains no predefined columns. It contains header items only.

Register name: _Sequence

Object number: 1

Table name: SeqParm

Table 144 shows the predefined header items in the sequence parameter table.

Table 144 Predefined Header Items in Sequence Parameter Table

Item Name	Type/Range	Meaning
DataNaming	Enumeration	Specifies how to name files
		0 Auto Naming
		1 Prefix/Counter
DataSeqSubDir	String 12 characters	Name of the sequence subdirectory
Prefix1	String 7 characters	For prefix/counter naming
Prefix2	String 7 characters	For prefix/counter naming (GC only)
Counter1	String 7 characters	Initial counter values
Counter2	String 7 characters	Initial counter values (GC only)
UseBarcode	Boolean	0 Do not use barcode in sequence
		1 Do use barcode in sequence
BarcodeInject	Boolean	0 On mismatch do not inject
		1 On mismatch inject

Table 144 **Predefined Header Items in Sequence Parameter Table, continued**

Item Name	Type/Range	Meaning
SeqMode	Enumeration	Part of Method to run 0 According to Runtime Checklist 1 Acquisition only 2 Data analysis only (using Sample.mac,ignoring Sample Table) 3 Data analysis only (using Sample Table)
WaitTime	String 10 characters	Synchronized with CP variable SEQWAIT (LC/CE only)
TimeOut	String 10 characters	Synchronized with CP variable SEQTIMEOUT (LC/CE only)
DoShutdownMacro	Boolean	0 Do not run shutdown macro 1 Run shutdown macro
ShutdownMacro	String 255 characters	The name of the shutdown macro/command
Comment	String 4095 characters	Sequence Comment (multi line text field stored as a single text field, lines CR/LF separated)

"SeqTable1" – Injector 1 Sequence Table
"SeqTable2" – Injector 2 Sequence Table

These tables define the sequence settings for the injectors. SeqTable1 is the sequence table for the front injector, SeqTable2 (GC only) for the rear injector. Both of these tables will have the same layout.

Register name: _Sequence

Object number: 1

Table name: SeqTable1 or SeqTable2

The Sequence Table has no predefined header items. The row number in the table is the sequence "line" number. Table 145 shows the predefined columns in the sequence table.

Table 145 Predefined Columns in Sequence Table

Column Name	Type/Range	Meaning
Vial	Integer	The vial number of the run. -1 means blank run.
Method	String 8 characters	Method to use
CalLine	string 1 characters	Sequence recalibration table line
InjVial	Integer	Number of injections for this vial
InjVolume	string 10 characters	Injection volume (not used for CE)
SampleName	string 16 characters	Sample Name
Amount	string 12 characters	Sample Amount
Multiplier	string 12 characters	Multiplier
ISTDAmt	string 12 characters	ISTD Amount
Dilution	string 12 characters	Dilution factor

Table 145 **Predefined Columns in Sequence Table, continued**

Column Name	Type/Range	Meaning
SampleInfo	string 4095 characters	Sample Information (multi-line text field, stored as a single text field, CR/LF seperates lines).
SampleType	integer	Identifies what type of sample is being defined on this sequence table line. 1 Sample Run 3 Calibration Sample
CalLevel	integer	Calibration level to be updated. Valid only for SampleType=3
UpdateRF	integer	Defines how response factors will be updated in the calibration table when this calibration run occurs. Valid only for SampleType=3. 0 No Update 1 Average 2 Replace
UpdateRT	integer	Defines how response (or migration) times will be updated in the calibration table when this calibration run occurs. Valid only for SampleType=3. 0 No Update 1 Average 2 Replace 3 Bracket
Interval	integer	Defines the number of sample runs that will occur before this calibration is re-run. Leave blank for explicit (non-cyclic) calibrations. Valid only for SampleType=3.
DataFileName	string 8 characters	Overrides the automatically-generated data filename with the filename entered here.

Partial Sequence Table

This table is used as a template to create the _PartialSequence register.

Register name: _Sequence

Object number: 2

Table name: _PartialSequence

The partial sequence table has no predefined header items. Table 146 shows the predefined columns in the partial sequence table.

Table 146 Predefined Columns in the Partial Sequence Table

Column Name	Type/Range	Meaning
Sel	integer	Indicates that the run is selected.
Inj	integer	Injector number
SeqTabLine	integer	Sequence table line number corresponding to this.
Run	integer	Run number (row number). Runs are sequentially numbered from the start of the sequence.
Vial	integer	The vial number for the run. -1 indicates a blank run.
Method	string 12 characters	Name of method to be used for this run.
DataFile	string 12 characters	Name of the data file (.D directory) for this run.
SeqTab	string 10 characters	
CalibInfo	string 10 characters	For calibration runs, indicates how the calibration table will be updated.
SampleName	string 20 characters	Sample name for the sample used in this run.

HP 1100 Method Parameter Tables

The HP 1100 method parameters are stored in module specific registers. The injector program and time tables are part of these registers and will be described in the following.

HP 1100 Autosampler Injector Program

Register name: _LeoAlsMethod

Object number: 1

Table name: ProgTable

Table 147 shows the predefined column items of this table:

Table 147 **Injector Program Table**

Function	Param[0]	Param[1]	Param[2]	Param[3]	Param[4]
DRAW	Amount	Source	Vial	Speed	Offset
EJECT	Amount	Source	Vial	Speed	Offset
MIX	Amount	Source		Speed	Repeat
INJECT					
WAIT	Time				
VALVE	Function	Source	Vial	Time	
NEEDLE	Function	Source	Vial		Offset
SYRINGE	Amount	Function		Speed	Repeat
OUTPUT		Source	Vial		
REMOTE	Function				
CONTACT	Number	ON/OFF			
WAIT FOR	Function			Time	
MOVE		Source	Vial	Destination	Vial
REPEAT	Repeat				
ENDREPEAT					

HP 1100 Pumping System Time Table

Register name: _LeoPumpMethod

Object number: 1

Table name: TimeTable

Table 148 shows the predefined column items of this table

Table 148 Predefined Column Items in HP 1100 Pumping System Time Table

Name	Type	Range	Description
Time	float	0..99999.9min	Time
Solv_B	float	190..600 nm	Composition of channel A
Solv_C	float	0,1	Composition of channel B
Solv_D	float		Composition of channel C
Flow	flow	0,1	Flow
Pressure	integer		High pressure limit

HP 1100 Thermostat Time Table

Register name: _LeoAlsMethod

Object number: 1

Table name: TimeTable

Table 149 shows the predefined column items of this table

Table 149 **Predefined Column Items in HP 1100 Thermostat Time Table**

Name	Type	Range	Description
Time	Float	0 …99999.9 min	Time
LeftTemp	Float	-5 …80 °C	Temperature (left heat exchanger); 0 means no entry
RightTemp	Float	-5 …80 °C	Temperature (right heat exchanger); 0 means no entry
ColumnValve12	Integer	0, 1, 2	Position of the column switching valve: 0 = no entry 1 = Position 1->6 2 = Position 1->2

HP 1100 Diode Array Detector Time Table

Register name: _LeoDADMethod

Object number: 1

Table name: TimeTable

Table 150 shows the predefined column items of this table

Table 150 **Predefined Column Items in HP 1100 Diode Array Detector Time Table**

Name	Type	Range	Description
Time	float	0 ... 99999.9 min	Time
Signal	enum	0, 1, 2, 3, 4, 5	0 means unused, otherwise the number describes to which signal the wavelenght and bandwidth columns belong 1 = Signal A, 2=B, 3=C, 4=D, 5=E
SampleWl	float	191..950 nm	Sample wavelength if Signal <> 0; 0 means unused
SampleBw	float	2 ... 400 nm	Sample bandwidth if Signal <> 0; 0 means unused
ReferenceWl	float	191…950 nm or 0 for “off”	Reference wavelength if Signal <> 0 0 means that the reference is off
ReferenceBw	float	2…400 nm	or 0 for “off” 0 means that the reference is off
StoreSpectra	enum	0, 1, 2, 3, 4, 5	0 means unused 1 = None 2 = Apex + Baselines 3 = Apex + Slope + Baselines 4 = All in peak 5 = Every 2nd spectrum 6 = All
Threshold	float	0.1..1000 mAU	Threshold for stored spectra; 0 means unused

Table 150 **Predefined Column Items in HP 1100 Diode Array Detector Time Table, continued**

Name	Type	Range	Description
Peakwidth	enum	0, 1, …, 8	0 means unused
			1 = < 0.01 min
			2 = > 0.01 min
			3 = > 0.03 min
			4 = > 0.05 min
			5 = > 0.1 min
			6 = > 0.2 min
			7 = > 0.4 min
			8 = > 0.85 min
Balance	enum	0, 1	0 means unused
			1 = Balance

HP 1100 Variable Wavelength Detector Time Table

Register name: _LeoVWDMethod

Object number: 1

Table name: TimeTable

Table 151 shows the predefined column items of this table

Table 151 **Predefined Column Items in HP 1100 Variable Wavelength Detector**

Name	Type	Range	Description
Time	float	0..99999.9min	Time
Wavelength	float	190..600 nm	Wavelength; 0 means unused
Balance	enum	0,1	0 means unused
			1 = Balance

Table 151 **Predefined Column Items in HP 1100 Variable Wavelength Detector, continued**

Name	Type	Range	Description
Spectrum	enum	0,1	0 means unused 1 = Acquire Blank spectrum 2 = Acquire Sample spectrum
From	float	190..600 nm	Lower wavelength limit for acquired spectrum ; 0 means unused
Step	float	1..10 nm	Bandwidth for each spectral data point; 0 means unused
To	float	190..600 nm	Upper wavelength limit for acquired spectrum ; 0 means unused

HP 1100 Contact Time Table

Register name: _LeoALSMethod, _LeoThmMethod, _LeoPumpMethod, _LeoDADMethod, _LeoVWDMethod

Object number: 1

Table name: Contact_TT

Table 152 shows the predefined column items of the Contact_TT time table which is common to all method parameter registers:

Table 152 **Predefined Column Items in HP 1100 Contacts Time Table**

Name	Type	Range	Description
TIME	float	0 ... 99999.9 min	Time
CONTACT_<x>	enum	0, 1, 2	<x>=1..4: 0=unused, 1=contact on, 2=contact off

13

General Introduction to Commands

General Introduction to Commands

This chapter describes the command line, how to enter commands and the rules and conventions used for commands.

Command Line

You enter commands by typing the command, with parameters if required, in the command line and pressing ENTER.

The default setting for the command line is off.

To turn on the command line

There are two possibilities:

- choose Command Line On from the system menu, in the upper-left corner of the ChemStation window, or
- choose Command Line from the View menu.

When the command line is on, it appears at the bottom of the screen.

How are Commands Presented in the Online Help?

See the Commands part of the online help for a description of all of your ChemStation's commands. The commands are arranged alphabetically.

Read the following information carefully, it describes how commands are presented in the online help.

- Commands are printed in uppercase and lowercase for easy reading.
- Commands can be followed by parameters. Parameters are printed in uppercase and lowercase characters for easy reading.
- There are two types of parameters, required and optional.
  Required parameters are indicated without square brackets;
  Optional parameters are shown with square brackets [ ].
- If you can specify only one of several options, the options are separated by a vertical bar, |.
- When square brackets are nested, parameters in inner brackets can only be specified if the parameters in the outer square brackets are specified.
- All parameters within a command are separated by commas (,).

Entering Commands Correctly

Read the following information carefully, it is important that you know how to enter commands with the correct syntax.

- You can use uppercase or lowercase characters or a mixture of both.
- You must separate the command from the first parameter by at least one space.
- You may use any number of spaces elsewhere except within the names of commands, functions, variables or keywords.
- Do *not* type the square brackets that are documented in the command syntax to indicate optional parameters.
- Do *not* confuse the square brackets that indicate optional parameters with those used to specify substrings or objects within registers. You must type the square brackets to specify substrings or objects within registers.
- Optional parameters have default values. If you want the default value, enter the command only or enter a comma when the parameter is one of several parameters.
- All parameters within an instruction, even if they are not specified, *must* be separated by commas (,).
- You do not need to type the commas at the end of the command if there are only optional parameters following and you do not want to specify them.
- When you give new values to optional parameters, enter them in the specified order.
- When you enter a file name, use the complete path and file name unless the file is in the current directory.
- When entering text, use quotation marks, " ", if you want to use the symbols: ,( ) [ ] / + !
- When you have typed in the command and any parameters, press ENTER.

Listing Commands

You can obtain a complete alphabetical listing of all commands using the command line. This section shows you how to list all commands and select a command from the list.

To list all commands:

1. Position the cursor on the command line.
2. Click the left mouse button.
3. Type `SHOW` after the vertical cursor on the command line.
4. Press ENTER to display the commands list box.

To select a command:

1. Scroll through the command list to find the command you need.
2. Move the cursor to the command.
3. Click the button on the pointing device to select the command. The selected command appears on the command line.
4. Edit the command by supplying parameters if necessary.
5. Press ENTER to execute the command.

Default and Initial Values

Default values are used by your ChemStation when you do not enter values for optional parameters. The default values are shown in the description of the command in the online help.

Keywords

Keywords are command parameters that must be entered using the exact letters shown. Do not type keywords within double quotation marks; keywords are not string literals.

ON|OFF

The keyword parameter ON|OFF is used by many of the commands. Unless otherwise noted, you may use the keywords ON and OFF, YES and NO, or the digits 0 and 1 as listed below:

ON = 1 = YES

OFF = 0 = NO

YES and NO may be abbreviated to Y and N.

Quotation Marks for Strings

Quotation marks are not required when entering strings containing only letters, digits, or symbols not reserved for ChemStation use. You will need to use double quotes (" ") to delimit the string if you wish to include any of the following symbols:

`, + # $ ( ) [ ]''` command line and macros

`!` macros only

To include a double quote in a string, enter it twice, for example:

```
“This is a ““ quote””. ”
```

An unquoted string cannot begin with “ ” as this is interpreted as a empty string.

We recommend that you enclose strings within double quotes (“ ”), for example, INSTR(s1,“abc”). This is because an unquoted string may look like the name of a variable, and could be confused with a variable of the same name.

Functions

Functions are available in all ChemStation applications, they are expressions that return unique results when given valid parameters.

Function names are always followed by a parameter list enclosed in parentheses ( ).

Some functions do not require any parameters, but the parentheses must still be included with the function name, for example, DATE$().

Functions are used like variables to replace an expression. For example, Number=SQRT(96)+5 assigns the variable Number the value of the square root of 96 plus 5.

Naming Convention for Command Names

The name of a command follows the *verb-noun* convention. The first part of the name is the operation; for example, an action that says what the command does in the general sense. The second part specifies in more detail what the command does, or on what type of data the command acts, that is, the operand. The second part may be missing.

Examples:

```
GetDataMinMax
DelAnn
Draw
```

If the parameters are placed in parentheses, (), this is a function. The function name does not consist of a verb-noun combination, but only of the noun, because the verb is always Get.

Examples:

```
RegSize
TabText$
```

The command names consist of several parts, some of which are abbreviated to keep the names short. Often used part names are:

Operations

Add	Add.
Copy	Copy.
Del	Delete.
Edit	Display a dialog box for editing.
Find	Search for something in a register that is not readily available.
Get	Retrieve something from a register that is stored as is.
Ins	Insert.
Load	Load something from a file into a register.

Mult	Multiply.
New	Create new.
Ren	Rename.
Save	Save something from a register into a file.
Set	Store something into a register.
Sub	Subtract.

Operands

Ann	Annotate/annotation.
Cal	Calibration.
Chrom	Chromatogram.
Col	Column of a table or matrix.
Data	Data block inside an object.
Item	Single Item in an object or table or annotation.
Mat	Matrix.
Max	Maximum/maxima.
Min	Minimum/minima.
Name	Name of a table, column, table header item, object header item, register.
Obj	Object/objects.
Reg	Register/registers.
Row	Row of a table or matrix.
Spec	Spectrum.
Tab	Table.
Text	Textual value.
Val	Numeric value.
Win	Window.

Implicit Creation and Explicit Removal of Registers

User defined registers are always created implicitly by any command, if ToReg is specified with a name that does not already exist as user defined register.

Example:
Exchange, CopyObj, MoveObj, NewObj, AddObj, CenterMat, LoadSignal, may cause an implicit Create, if the specified ToReg does not exist.

In contrast a register is not removed automatically if the last object in that register is removed. The register has to be removed explicitly by the command DelReg.

Obtaining Data Using Commands

Commands that return data to the macro-programmer will:

- be implemented as functions if the return value is a single string or number, and
- generate *predefined* variables if several values at a time are to be returned. *Predefined* variables are standard variables with predefined names.

To avoid naming conflicts with user variables the following naming convention XXX_VariableName is used, where:

XXX	The prefix is chosen to reflect the context to which the data item belongs.
VariableName	Stands for any meaningful name to describe the information to be retrieved in that variable.

Examples:
Obj_Min, Obj_Max

Parameter Notation

This is a general description of some of the often used parameters.

RegName	A string that is the name of a register. A RegName must follow the naming rules of numeric variables.
RegObj	Register name plus optional object number in brackets. Examples of valid `RegObj` specifications are: hugo means the first object in hugo, and hugo[3] means the 3rd object in hugo.
RegObjRange	Register name plus optional object number or object number range in brackets. Examples of valid `RegObjRange` specifications are: hugo means all objects in hugo, hugo[3] means the 3rd object in hugo, hugo[3:7] means the 3rd, 4th, 5th, 6th, and 7th object in hugo, and hugo[3:0] means all objects from the 3rd to last object in hugo.
FromRegObj, FromRegObjRange	An object or range of objects that is used but not modified.
ToReg	A register to which an object is appended as last object. The register is created if it does not exist.
ToRegObj	An object that is created or modified. Several cases are possible:

- If ToRegObj specifies the register as well as the object number, and ToReg originally contains at least as many objects as specified in ToRegObj, the new object(s) will be inserted into ToReg. The rest of the already existing objects are pushed further along.
- If ToRegObj specifies only the register but no object number, the new objects will be appended at the end of ToReg. This is equivalent to specifying an object number of zero.
- If ToRegObj specifies the register as well as the object number, and ToReg

originally contains few objects so that the ToRegObj specification would yield a gap in the object numbering, the command will fail.

- An existing object is modified, as in the commands CopyDataRow, CopyTab, and CopyTabCol. In this case the object and the register must already exist.

ItemName A string that is the name of a string or numeric item. Items are stored in various places within objects:

Object Headers Items have user defined names. Since they are stored in the same list as tables within that object, the names must be unique.

Table Headers Items have user defined names. Since they are stored in the same list as columns within that table, the names must be unique.

Data Blocks Items have predefined names.

Annotations Items have predefined names.

An ItemName must follow the naming rules of numeric variables. When entering an ItemName as parameter, it may be given literally or as a string expression which yields the ItemName.

TabName A string that is the name of a table within an object. Tables have user defined names. Since they are stored in the same list as tables within that object, the names must be unique.

A TabName must follow the naming rules of numeric variables.

When entering a TabName as parameter, it may be given literally or as a string expression which yields the TabName.

FromTabName A table that is used but not modified.

ToTabName A table that is created or modified.

ColName A string that is the name of a column within a table. Columns have user defined names. Since they are stored in the same list as table headers within that table, the names must be unique.

A ColName must follow the naming rules of numeric variables.

	When entering a ColName as parameter, it may be given literally or as a string expression which yields the ColName.
FromColName	A table column that is used but not modified.
ToColName	A table column that is created or modified.
Index	A numeric value that is an index into a list. These lists are: the list of registers, the list of header items within an object, the list of tables within an object, the list of header items within a table. An Index always starts at 1. If an element of a list is deleted, the Index of the following elements changes automatically.
AnnIndex	A numeric value that is an index into the list of annotations within an object. An AnnIndex always starts at 1. If an annotation is deleted, the AnnIndex of the following annotations changes automatically.
RowIndex	A numeric value that is an index into the data block or a table within an object.

1 If the Row Index refers to a Data Block, the posible values are:

- From 1 to the number of rows within the Data Block.
- If it is a chromatogram or spectrum, then Row Index=1 refers to the measured data points, that is, absorbance or voltage.
- If it is a spectrum, then Row Index=2 refers to the variance of data points, if a variance has been measured.
- If it is a matrix, then Row Index refers to a row within the matrix. Each row of the matrix may contain for example, the data points of one spectrum.

  The Row Index=0 refers to the X-axis.

  If it is a chromatogram, these are time values.

  If it is a spectrum, these are wavelength values.

If it is a matrix, these may be wavelength values, if the matrix contains a spectrum in each row.

The Row Index=–1 refers to the Y-axis of a matrix.

If the matrix contain a spectrum in each row, then the values in the y-axis contain the time values of each spectrum.

The Row Index=–2 refers to the data points of a matrix as a whole. For example, this is used in the `SetDataVal` command to define general characteristics of the matrix data points.

2 If the Row Index refers to a Table, the possible values are:

From 1 to the number of rows within the table.

Greater than the number of rows in the table. This is allowed only in the InsTabRow command.

Row Index=0 refers to the row for initial values. These can be set in the NewColText, NewColVal, SetTabText, SetTabVal commands and are used in the InsTabRow command.

RowRange	A single RowIndex or range of row indices. For example, 7:9 means row indices 7, 8,and 9.
ColRange	A single ColIndex or range of column indices. For example, 1:20 means all columns from index 1 to index 20.
ColIndex	A numeric value that is an index into the data block within an object. A ColIndex always starts at 1.
TimeRange	Single time or range of times in minutes. Examples of valid TimeRange specifications are: 3.14 means 3.14 minutes. 3.14:5 means the time span from 3.14 to 5 minutes.
FileName	A string that is the name of a file on the disk. When entering a FileName as parameter, it may be given literally or as a string expression which yields the FileName.
FromFile	A file that is read but not modified.
ToFile	A file that is created or modified.

Invalid Data

Individual data points of a UV spectrum may be flagged as invalid by the spectrophotometer. When mathematical operations are done on such a spectrum, the invalid data points remain invalid. When this spectrum is drawn to a window the invalid data points are drawn linearly-interpolated between their left and right neighboring data points.

14

Summary of Commands

Summary of Commands

This chapter lists all available commands and functions together with the syntax and a short description. They are grouped according to the area where they can be used. See the commands part of the online help for detailed descriptions of each command.

The commands listed in this chapter are those available at the time of going to print. You will find that some commands can only be used with specific instruments; use the **Show** command to verify the type of instrument with which the command can be used. For the most recent information about the ChemStation commands, see the online help, or the .WRI file supplied with the software.

Annotation Commands

```
AnnIndex (RegObj, [AnnIndexRange], [AnnClass], [Topic])
```

Gets the smallest index of annotation that fits AnnClass and Topic.

```
AnnText$ (RegObj, AnnIndex, ItemName)
```

Gets one text item from the specified annotation.

```
AnnVal (RegObj, AnnIndex, ItemName)
```

Gets one numeric item from the specified annotation.

```
DelAnn RegObj, [AnnIndexRange], [Topic]
```

Deletes the annotation with the specified index from the object.

```
EditAnn RegObj, x, y, [Topic], [AnnIndex]
```

Displays a dialog box to add or edit text annotations.

```
GetPolyAnnVal RegObj, AnnIndex, PolylineIndex
```

Gets the x- and y-value from the polyline array of the specified annotation.

```
NewMarkerAnn (RegObj, Marker, [XPos], [YPos],
[AttachedTo], [Topic], [Size], [Color], [BrushStyle],
[Hatch]
```

Creates a new marker annotation with the specified contents.

```
NewPolyAnn (RegObj, LineStyle, [XPos], [YPos],
[AttachedTo], [Topic], [Size], [Color], [BrushStyle],
[Hatch], [LineWidth])
```

Creates a new polyline annotation with the specified contents.

```
NewTextAnn (RegObj, Text, [XPos], [YPos], [AttachedTo],
[Topic], [Size], [Color], [Angle], [FitInto], [Justify],
[FaceName], [FontStyle])
```

Creates a new text annotation with the specified contents.

```
SetAnnText RegObj, AnnIndex, ItemName, Text
```

Sets one text item in the specified annotation.

```
SetAnnVal RegObj, AnnIndex, ItemName, Value
```

Sets one numeric item in the specified annotation.

```
SetPolyAnnVal RegObj, AnnIndex, PolylineIndex, XVal,
YVal
```

Sets the x- and y-value in the polyline array of the specified annotation.

Application Control Commands

```
Bye
```

Quits the current application.

```
ExecNoWait Program_Specification, [Appearance],
[InputFileNum], [OutputFileNum], [ErrorFileNum]
```

Executes the specified program.

```
ExecNoWait (Program_Specification, [Appearance],
[InputFileNum], [OutputFileNum], [ErrorFileNum])
```

Executes the specified program.

```
ExecWait Program_Specification, [Appearance],
[InputFileNum], [OutputFileNum], [ErrorFileNum]
```

Executes the specified program.

```
ExecWait (Program_Specification, [Appearance],
[InputFileNum], [OutputFileNum], [ErrorFileNum])
```

Executes the specified program.

```
HPChemSched
```

Starts the HP ChemsStation Scheduler application

```
Number = IsAppLoaded (program_name)
```

Checks whether a program is running under Microsoft Windows.

```
WindowID (ApplicationTitle)
```

Finds out if an application with a given title is running.

Arithmetic Commands

```
AddObj FromRegObjRange1, [FromRegObjRange2], [ToReg]
```

Adds the objects of two registers into objects of a third register.

```
CompareObj RegObj1, RegObj2, [Threshold], [XShift]
```

Compares the object in RegObj1 with the (eventually shifted) one in RegObj2.

```
DerivObj RegObjRange, [FilterLength], [Order], [Degree]
```

Calculates the derivative of the data points.

```
ExpObj RegObjRange
```

Calculates the exponential function of the data points.

```
FindObjMinMax RegObjRange, [XRange], [Threshold],
[FilterLength]
```

Determines the minima and maxima of an object and generates two tables of them.

```
LogObj RegObjRange, [Threshold]
```

Calculates the natural logarithm of the data points.

```
MultObj FromRegObjRange1, FromRegObjRange2, [ToReg]
```

Multiplies the objects of two registers and puts the result into a third one.

```
RatioObj FromRegObjRange1, FromRegObjRange2, [ToReg],
[Threshold]
```

Calculates the ratio of the objects of two registers and puts the result into a third register.

```
RecipObj RegObjRange, [Threshold]
```

Calculates the reciprocal values (*1/n*) of the data points.

```
RegressObj RegObj1, [RowIndex1], [RegObj2], [RowIndex2],
[ToReg], [Type], [UseVariance], [tPercentage]
```

Calculates the regression coefficients of two or more objects.

```
SplineObj RegObjRange, Step
```

Constructs a cubic splined curve through the existing data points.

```
SubObj FromRegObjRange1, FromRegObjRange2, [ToReg]
```

Subtracts the objects of two registers and puts the result into a third register.

```
TransformObj RegObjRange, X0:X1, X2:X3,
[XAxis|YAxis|ZAxis]
```

Shifts and scales the data points.

Chromatography Commands

```
AddCalPeaks RegObj1, [TimeRange], RegObj2
```

Adds new peaks in the calibration data.

```
CalibratePeaks [RegObj1], [TimeRange], RegObj2
```

Recalibrates calibration data.

```
ClrManInt
```

Clears any manual integration events and parameters.

```
CorrectTime (RegObj1, [TimeRef], [Signal], Time)
```

Calculates corrected retention time using updated retention time of the identified time reference peaks.

```
DelPeak RegObj, TimeRange
```

Deletes the peaks in the specified time range of the objects defined by RegObj.

```
EdCalComp CalRegObj, [Compound]
```

Displays calibration data of one compound in table format.

```
EdCalCurve CalRegObj, [Peak]
```

Displays calibration curve for a single peak.

```
EdCalParms CalRegObj
```

Displays a dialog box with method wide calibration parameters.

```
EdCalSignals CalRegObj
```

Displays calibrated signals in table format.

```
EdCalTbl CalRegObj
```

Displays the Calibration Table.

```
EditEvents()
```

Displays the integration events dialog box.

```
EdReCalParms CalRegObj
```

Displays a dialog box for calibration data update parameters.

```
EdRptParms CalRegObj
```

Displays a dialog box with method wide report settings.

```
IdentifyPeaks RegObjRange, [TimeRange], RegObj2, RegObj3
```

Identifies and qualifies integrated peaks.

```
IntegrateObj RegObjRange, [TimeRange]
```

Integrates the specified chromatograms and generates integrated results in the objects.

```
ManInt RegObj, x1, y1, x2, y2, [IntegMode], [AllValleys]
```

Integrates a baseline from (x1,y1) to (x2,y2).

```
MergeCalData RegObj1, [Weight1], RegObj2, [Weight2]
```

Averages two calibration data objects.

```
NewCalCurve RegObj, Peak, ToReg, [Amount], [Points]
```

Prepares calibration curve of the selected peak for drawing.

```
ParseSigDesc SignalDesc
```

Parses a signal description into its individual parts.

```
PrepCalib CalRegObj
```

Displays a dialog box which allows interactive selection of calibration or recalibration mode and start a calibration.

```
QuantifyPeaks RegObj1, RegObj2
```

Calculates quantitative results.

```
SignalOptions (MthRegObj)
```

Displays a dialog box with settings for Signal Options.

```
SplitPeak RegObj, Time, [Valley]
```

Sets a SplitPeak or SplitPeak Valley manual integration event.

```
VerifyCalData RegObj
```

Verifies calibration data consistency.

Data Block Commands

```
CopyDataRow FromRegObj, [FromRowRange], [FromColRange],
ToRegObj, [ToRowIndex], [ToColIndex]
```

Copies (a part of) a (range of) data block row(s) from one object to another. The object classes of FromRegObj and ToRegObj may be different.

```
Data (RegObj, RowIndex, ColIndex)
```

Gets one single data point from the data block of the specified object.

```
DataCols (RegObj)
```

Gets the number of columns in the data block of the specified object.

```
DataIndex (RegObj, XValue)
```

Gets the ColIndex of the point whose x-value is closest to the specified one.

```
DataModified (RegObj)
```

Gets the modification information of the data block in the object.

```
DataRows (RegObj)
```

Gets the number of rows in the data block of the specified object.

```
DataText$ (RegObj, RowIndex, ItemName)
```

Gets one text item from the specified data block row.

```
DataVal (RegObj, RowIndex, ItemName)
```

Gets one numeric item from the specified data block row.

```
GetDataMinMax RegObj, RowIndex, [XRange]
```

Gets the minimum and maximum values of a (part of a) data block row in the object.

```
SetData RegObj, RowIndex, ColIndex, Value
```

Sets one single data point in the data block of the specified object.

```
SetDataText RegObj, RowIndex, ItemName, Text
```

Sets one text item in the specified data block row.

```
SetDataVal RegObj, RowIndex, ItemName, Value
```

Sets one numeric item in the specified data block row.

DDE Commands

```
DDEAdvise Channel, Item, VarName
```

Sets up a hot-link with a DDE server.

```
DDEExecute Channel, Command
```

Sends a command to a DDE server for execution.

```
DDEInitiate (AppName, Topic)
```

Initiates a DDE conversation.

```
DDEPoke Channel, Item, String
```

Sends unrequested data over a DDE channel.

```
DDERequest Channel, Item, VarName
```

Gets data from a DDE server.

```
DDETerminate Channel
```

Terminates a DDE conversation.

```
DDEUnadvise Channel, Item
```

Removes a hot link with a DDE server.

Dialog Box Commands

```
BeginDialog DlgBoxName, x, y, dx, dy, [Title],
[PointSize], [FontType]
```

Starts a dialog box declaration for a custom dialog box.

```
CancelButton x, y, dx, dy, [ButtonText]
```

Defines the position and size of the cancel button in a custom dialog box.

```
CancelButtonParm State
```

Establishes a connection between the Cancel Button and a state CP-variable.

```
CheckBox x, y, dx, dy, CheckText, CheckVar
```

Creates a check box with an accompanying text-field in a custom dialog box.

```
CheckBoxParm Field, CheckVar, [State]
```

Establishes a connection between a CheckBox resource and CP-variables.

```
ComboBox x, y, dx, dy, TextListVar, SelTextVar,
[Convert]
```

Creates a combination text box or list box in a custom dialog box.

```
ComboBoxParm FieldId, TextListVar, SetTextVar,
[Convert], [State]
```

Establishes a connection between a ComboBox resource in a DLL and CP-variables.

```
DialogExists (DlgBoxName)
```

Tests whether a custom dialog box exists or not.

```
DialogName$ (nNumber)
```

Gets the name of a custom dialog box.

```
EditBox x, y, dx, dy, EditText$, [Style], [Convert]
```

Creates a text box in a custom dialog box in which the user can enter and edit text.

```
EditBoxParm FieldId, EditTextVar, [Convert], State]
```

Establishes a connection between an EditText resource and CP-variables.

```
EndDialog
```

Ends the definition of the custom dialog box.

```
ExecPushButton
```

Defines the position and size of a push button in a custom dialog box. If the push button is selected, a CP command string is executed.

```
ExecPushButtonParm FieldId, CpIdentifier, CpCommand,
[State]
```

Establishes a connection between a push button control and CP-variables. If the push button is selected, a CP command string is executed.

```
GroupBox  x, y, dx, dy, [TitleText]
```

Creates a frame in a custom dialog box that can include a title in its top border.

```
GroupBoxParm FieldId, State
```

Establishes a connection between a GroupBox and a CP-variable.

```
ListBox x, y, dx, dy, TextListVar, SelTextVar
```

Creates a list box in a custom dialog box containing the strings from the TextListVar argument.

```
ListBoxParm FieldId, TextListVar, SetTextVar, [State]
```

Establishes a connection between a ListBox resource and CP-variables.

```
LoadDialog ResourceDLL, DialogBoxId, DialogBoxName,
[DialogTitleFormat]
```

Loads a dialog box from a resource DLL.

```
NumberBox x, y, dx, dy, NumberVar, [LimitLow],
[LimitHigh]
```

Creates a test box in a custom dialog box, in which numbers can be entered and edited.

```
NumberBoxParm FieldId, NumberVar, [LimitLow],
[LimitHigh], [Undefined], [Check], [State]
```

Sets the additional parameter for EDITTEXT resources needed for a number box.

```
OKButton x, y, dx, dy, [ButtonText]
```

Defines the position and size of the OK button in a custom dialog box.

```
OKButtonParm State
```

Establishes a connection between the OK button and a CP-variable.

```
OptionButton x, y, dx, dy, OptionText
```

Creates an option button with an accompanying text-field in a custom dialog box.

```
OptionGroup  SelectVar
```

Starts the definition of the option buttons in a custom dialog box.

```
OptionGroupParm FieldId, nSelect
```

Establishes a connection between a set of option buttons in a resource and a CP-variable.

```
PushButton, y, dx, dy, ButtonText, IDNumber
```

Defines the position and size of a push button in a custom dialog box.

```
PushButtonParm FieldId, State
```

Establishes a connection between a BUTTON resource and a state CP-variable.

```
RemoveDialog DlgBoxName, [bAll]
```

Removes custom dialog boxes.

```
ShowDialog (DlgBoxName, [HelpContext])
```

Displays a custom dialog box.

```
StaticText  x, y, dx, dy, Text, [Alignment]
```

Creates one or more lines of static text in a custom dialog box.

```
StaticTextParmFieldId, [Text], [State]
```

Establishes a connection between a static text resource in a DLL and CP-variables.

File Manipulation Commands

```
ChDir [Directory]
```

Changes the working directory to the specified directdory.

```
Close [#FileNumber], [...]
```

Removes the association of a file number with a file name or removes the association of a device number with a printer or plotter.

```
Copy From Name, To Name, [DontAsk]
```

Copies a file or directory (including all files and subdirectories).

```
DataDirExists (path)
```

Checks whether data directories (*.D) exist in a specified path.

```
Delete File_or_dir_name, [DONTASK]
```

Removes a file or directory (including all files and subdirectories).

```
EdUtil [SEARCH/BROWSE]
```

Displays the file browse or file search dialog box.

```
FILESTAT (Selector, Name)
```

Gets the status of the specified file or directory.

```
FindFile$ (FileName$[,Attribute$[,Separator$]])
```

Returns a list of files matching the pattern Filename$.

```
FreeDiskSpace (Drive, [Keyword])
```

Determines the amount of space on a specified disk that is free for use.

```
GetCWD$
```

Returns the name of the current working directory.

```
Input # Filenumber, Variable, [...]
```

Reads in values into variables from a sequential ASCII file.

```
Input Using # Filenumber, FormatImageString, Variable,
[...]
```

Reads in values into variables from a sequential ASCII file using a format string.

```
MkDir Argument
```

Makes a directory.

```
Open Filename [For Input/ For Output / For Append] As #
Filenumber
```

Associates a file number with a file name for use with Print #, Print Using #, Input #, or Input Using #.

```
Print # FileNumber, Expression, [Expression], [...]
```

Writes the results of string or scalar expressions to an ASCII file.

```
Print Using # FileNumber, FormatImageString,
[Expression], [...]
```

Writes formatted results of string or scalar expressions to an ASCII file.

```
Rename Oldname, Newname
```

Renames a file or directory and moves files.

```
RmDir Argument
```

Removes an empty directory.

```
SetCwd [Directory]
```

Changes the working directory to the specified directory.

Formatting Commands

```
AddPipeTabCol CtrlRegObj, Contents, [Module],
ColKeyWord, ColumnInfo, [Level], [Position],
[HideRowFlag]
```

Adds a new column to the table that controls the data access of the PipeTab command.

```
AddPipeTabLevel CtrlRegObj,Contents, [Module],
LevelKeyword, LevelParam, FirstParam, NextParam
```

Generates a table which controls the data access of the PipeTab command.

```
PipeTab CtrlRegObj, Contents, FormatRegObj,
FormatDescription, DataRegObj, Range
```

Extracts and formats tabular data from the tables and objects of the ChemStation.

```
RemovePipeTabCol CtrlRegObj, Contents, Position
```

Removes column data from the control table of the PipeTab command.

```
RemovePipeTabLevel CtrlRegObj, Contents
```

Removes level data from the control table of the PipeTab command.

Graphics Commands

```
DelFig WinNum, ObjNum, FigId
```

Deletes a figure from a graphics window.

```
Draw WinNum, RegObjRange, [XRange], [YRange],
[FullScale], [Separated], [DrawStdDev]
```

Draws the objects in the specified window.

```
DrawIsoPlot RegObj, [TimeRange], [WLRange],
[ValueRange], [ChromReg], [SpecReg], [DelReg], [Scale],
[IsoColor], [Print]
```

Displays the Isoplot dialog box or prints the isoplot.

```
NewFig (WinNum, ObjNum, FigType, [x1], [y1], [x2], [y2],
[BrushStyle], [Hatch])
```

Creates a figure in a graphics window.

```
ScrollWin WinNum, ObjNum
```

Scrolls a graphics window so that you can see a particular object.

```
Zoom WinNum, [XRange], [YRange], [ObjNum]
```

Zooms in on a portion of the specified window.

```
ZoomOut WinNum
```

Restores a zoomed window to its original, unzoomed state.

Instrument Control Commands

`ContInject`

Continues a run which has been stopped. The autosampler is in a status break or break no stop condition.

`Diagram`

Switches the HPCE Diagram screen for control of the HP ^{3D}CE ChemStation on and off.

`DownloadLAls [ModuleNumber]`

Sends the contents of the register LeoALSMethod to the HP 1100 Series Autosampler.

`DownloadLDad [ModuleNumber]`

Sends the contents of the register LeoDADMethod to the HP 1100 Series `Diode Array Detector`.

`DownloadLPmp [ModuleNumber]`

Sends the contents of the register LeoPMPMethod to the HP 1100 Series Pump.

`DownloadLThm [ModuleNumber]`

Sends the contents of the register LeoTHMMethod to the HP 1100 Series Column Thermostat.

`DownloadLVwd {moduleNumber]`

Sends the contents of the register LeoVWDMethod to the HP 1100 Series Variable Wavelength Detector.

`Ed1090DAD Sig|Ctrl|Test, [ModuleNumber]`

Displays the Edit HP 1090 DAD dialog box.

```
ED5890
[oven|inlet|inletpres|splitflow|packedflow|auxpres|purge
|valve|detector|signal]
```

Displays 5890 GC dialog box(s).

```
ED68GC [injector|valves|inlets|columns|oven|
detectors|signals|aux|runtime|options|quick setup]
```

Displays 6890 GC dialog box(s).

```
Ed7673 INJECTOR
```

Displays the 7673 autosampler dialog box.

```
EdAcquisition ()
```

Displays the Acquisition Method boxes, one after the other.

```
EdADC [ModuleNumber]
```

Displays the A/D Converter dialog box.

```
EdALS Vol|Prog|Cont|Ctrl
```

Displays a dialog to edit the HP 1050 Autosampler and injector parameters.

```
EdECD Meth|Ctrl, [ModuleNumber]
```

Displays a dialog box to edit the HP 1049 Electrochemical Detector parameters.

```
EdFLD Meth|Scan, [ModuleNumber]
```

Displays the Edit HP1046A FLD dialog box.

```
EdLCFPD Sig|Ctrl
```

Displays a dialog box to edit the HP 1090 FPD parameters.

```
EdLCInj Vol|Prog|Ctrl
```

Displays a dialog box to edit the HP 1090 Liquid Chromatograph injector/ autosampler parameters.

```
EdLCPump Solv|Oven|Cont|Ctrl
```

Displays a dialog box to edit the HP 1090 Liquid Chromatograph pump parameters.

```
EdPump Solve|Aux|Cont|Oven|Ctrl, [ModuleNumber]
```

Displays a dialog box to edit the HP 1050 Pump parameters.

```
EdSFC pressure|flow|modifier|column|fluid|density|
lowlevel|diagnostic|quickSFC|mode|module number
```

Displays a dialog box to edit the HP SFC Pump parameters.

```
EdSFCGC signal|oven|detector|value|detprog
```

Displays SFC GC 5890 GC dialog box(es).

```
EdVWD Sig|Ctrl
```

Displays a dialog box to edit HP 1050 Series VWD parameters.

```
GCdetStyle A|B SFC|GC
```

Sets the GC detector style.

```
GCKeyLock [on/off]
```

Locks or unlocks the GC keyboard.

```
LampAll ON|OFF, [NoWait|Wait]
```

Turns all detector lamps on or off.

```
LCInjReset
```

Resets the HP 1090 injector.

```
LCInjStep [Parameter]
```

Moves the HP 1090 injector/autosampler through certain steps.

```
LCPumpDiagnose [Mode]
```

Displays the HP 1090 pump diagnose window.

```
LCWash ON|OFF [Time] [NoWait|Wait]
```

Flushes the HP 1090 injector.

```
LoadSignal1050DAD
```

Loads a test chromatogram into the HP 1050 DAD.

```
MonFLDStatus [Mode], [Modulenumber]
```

Displays HP 1046A Fluorescence Detector.

```
PopApply {volt | curr | power | press | clean}
```

Displays dialog boxes to aplly voltage, current, power or pressure and to clean the tubing of the replenishment system.

```
PopBFlush
```

Displays the backflush capillary dialog box.

```
PopCassTemp
```

Displays the Set Capillary Temperature dialog box.

```
PopChngBottle
```

Displays the Change Bottles dialog box.

```
PopChngCass
```

Displays the Change Cassette dialog box.

```
PopFlush
```

Displays the Flush Capillary dialog box.

```
PopRepl { replenish | fill }
```

Displays the Replenish/Fill Vial dialog box.

```
PopVialRmv {inlet | outlet }
```

Displays the dialog boxes to remove the inlet or outlet vial.

```
PopVialSet {inlet | outlet | empty }
```

Displays the dialog boxes to set the inlet or outlet vial or to emtpy a vial.

```
PrepRun [Wait|NoWait]
```

Sets instrument parameters to the prerun state.

```
PumpAll ON|OFF, [NoWait|Wait]
```

Turns all pumps on or off.

```
ReadModule$ (ModuleId)
```

Reads data from a module connected to the ChemStation.

```
SendModule$ (ModuleID, Command)
```

Sends any command to a specified module.

```
SetAcqTimeout Value
```

Sets the not ready timeout for method runs.

```
SFCNoInjectRun [ON|OFF]
```

Starts current run without injection.

```
SFCPump [ON|OFF]
```

Switches HP SFC module on or off.

```
SFCPurge [ON|OFF]
```

Switches HP SFC purge module on or off.

```
Snapshot
```

Copies the currently created data file to SNAPSHOT.D in the current data directory.

```
StandBy [Wait|NoWait]
```

Sets the complete instrument to standby. Turns off all lamps and pumps.

```
UploadLAls [ModuleNumber]
```

Reads the method from the HP 1100 Series Autosampler and writes it into the register LeoALSMethod.

```
UploadLDad [ModuleNumber]
```

Reads the method from the HP 1100 Series Diode Array Detector and writes it into the register LeoDADMethod.

```
UploadLPmp [ModuleNumber]
```

Reads the method from the HP 1100 Series Pump and writes it into the register LeoPMPMethod.

```
UploadLThm [ModuleNumber]
```

Reads the method from the HP 1100 Series Column Thermostat and writes it into the register LeoTHMMethod.

```
UploadLVwd [ModuleNumber]
```

Reads the method from the HP 1100 Series Variable Wavelength Detector and writes it into the register LeoVWDMethod.

```
WriteModule ModuleId, Command
```

Writes a command to a module connected to the ChemStation, without expecting a response.

Instrument Monitoring Commands

```
ConfigSignalPlot WinNum
```

Opens a dialog box to configure the online signal plot window.

```
Mon1090DAD [MODE], [Module Number]
```

Displays HP 1090 Liquid Chromatograph DAD monitor window.

```
Mon1205 [ON|OFF]
```

Displays the HP 1205 pump status window.

```
Mon6890Status [ON|OFF]
```

Turns display of 6890 Status window on or off.

```
Mon5890 [ON|OFF]
```

Displays the HP 5890 Oven/Det status window.

```
Mon7673 [ON|OFF]
```

Displays the HP 7673 automatic sampler status window.

```
MonADCStatus [Mode], [Module Number]
```

Displays A/D converter monitor window.

```
MonALSStatus [ON|OFF]
```

Displays the HP 1050 Series Autosampler monitor window.

```
MonECDStatus [ON|OFF], [Module Number]
```

Displays the HP 1049 Electrochemical Detector monitor window.

```
MonLCStatus [ON|OFF], [Module Number]
```

Displays the Liquid Chromatograph status monitor window.

```
MonMWDStatus [ON|OFF], [Module Number]
```

Displays a monitor window for the HP 1050 Series MWD.

```
MonPumpStatus [ON|OFF], [Module Number]
```

Displays a monitor window for the HP 1050 Series Pumping System.

```
MonVWDStatus [ON|OFF], [Module Number]
```

Displays a monitor window for the HP 1050 Series VWD.

```
PopNrdyInfo
```

Displays the Not Ready Information dialogbox.

```
RunStatus
```

Displays the run status window.

```
ShowSignal [Channel], [Mode], [Detector], [SignalId],
[Attenuation], [Offset], [Time]
```

Displays the online signal plot.

```
ShowSpectra [Mode], [Detector], [YRange]
```

Displays the online spectra plot.

Logbook Commands

`ClearLogBook`

Clears the current online logbook file.

`Logbook Mode, [LogbookFile]`

Displays a logbook.

`PrintLogbook`

If no logbook is displayed, the current logbook is printed. If a logbook is displayed, it is printed.

`SaveAsLogbook LogbookFile`

Saves the current online logbook to a specified file.

`WriteToLogbook LogText`

Writes a line to the current logbook. Allows logbook entries from macros.

Macro Broadcast Server Commands

```
Event ([Reg], [On|Off])
```

Tests and sets the mode of the register, or sets the mode of the Macro Broadcast Server Process (mBSP).

```
GetEvent (Index)
```

Stores the parameters of a valid event found in the event table to several system variables.

```
OnEvent Reg, [Macro], [Priority]
```

Inserts or replaces an event in the event table, or removes an event from an event table.

```
PeekEventQueue ([Reg])
```

Searches an event for the register "Reg" in the event queue, and when found removes the event from the event queue.

Macro Control Commands

`Check (Selector, Identifier)`

Checks for existence of an identifier for a required format.

`CpName$(Index)`

Gets the name of the command processor with the specified index.

`CpRequest CpName, CommandString`

Executes a command using the given command processor.

`CpStart CpName`

Starts a new command processor.

`CpTerminate CpName`

Terminates a command processor.

`Else`

Causes parts of a macro to be selectively executed when the expression is false.

`EndIf`

Ends a conditional statement started with If command.

`EndMacro`

Specifies the physical end of the macro.

`EndWhile`

Ends a conditional statement started with While command.

```
Evaluate CommandString
```

Executes a command.

```
For ControlVariable = Initval To Limit
…
Next ControlVariable
```

Executes a set of commands while varying an index.

```
Generate Error [ErrNum], [ErrString], [ErrFile],
[ErrLine], ErrMacro], [ErrCmd]
```

Generates an error for an incorrect macro file by modifying several read-only system variables.

```
GetVerbose ()
```

Returns the current verbose level.

```
Goto Label
```

Does an unconditional branch to the statement with the specified label.

```
If Expression [Then] […] [Else […]] EndIf
```

Causes parts of a macro to be selectively executed depending on whether the expression is true or false.

```
InstallCmds ([FileName])
```

Installs new commands, functions and data types for the command processor.

```
Local Variable, […]
```

Defines one or more variables which are local to the macro.

```
Logging [SeverityLevel], [FileName], [ERASE]
```

Logs commands, messages, and errors to a file.

```
Macro [MacroFileName], [Go]
```

Loads macros from a file and optionally executes the last macro.

```
Name Id
```

Specifies the physical beginning of the macro.

```
On Error Trap
```

Used in a macro to specify the command to execute when an error occurs.

```
Parameter Variable1, [DEFAULT Value1, [Variable2
[DEFAULT Value2]], […]
```

Used in a macro to assign the parameters which are passed to the macro and then to the specified local indentifiers.

```
RemoveCmds IdCmds
```

Removes commands, functions and data types that were previously installed by the function InstallCmds().

```
REMOVE [MacroName/VariableName],[…]

REMOVE ([MacroName/VariableName], […])
```

Removes macros or variables from memory.

```
Repeat

_
Until Condition
```

Causes a set of commands to be repeated until a certain condition is set.

```
Return [Expression]
```

Used in a macro to return to the macro that caused the macro to run.

```
SendKey AppName, Key, [Repetition]
```

Sends a number of keystrokes to any application.

```
SetAbort [Value]
```

Defines whether an abort action is permitted.

```
SetSemaphore(VariableName)
```

Tests and sets a global variable.

```
SetVerbose VerboseLevel
```

Sets the verbose level.

```
Stop
```

Terminates the current macro and any other macros that called it.

```
While Condition [Do]
…
EndWhile
```

Causes the portion of a macro between the While and the EndWhile to be executed repeatedly until the condition is false.

Mathematical Functions

```
ABS (Argument)
```

Calculates the absolute value of a numerical expression.

```
Asc (Argument)
```

Returns the number representing the character according to the ASCII table.

```
Ceil (Argument)
```

Rounds the argument up to the next integer.

```
Exp (Argument)
```

Returns the value of *e* to the power of the argument.

```
Floor (Argument)
```

Rounds the expression down to the nearest integer.

```
LN (Argument)
```

Returns the natural logarithm (base *e*) of the argument.

```
LOG (Argument)
```

Returns the logarithm (base 10) of the argument.

```
Mod (Argument1, Argument2)
```

Returns the remainder of argument1 divided by argument2.

```
PI()
```

Returns the value of π.

```
SQR (Argument)
```

Calculates the square of a number.

```
SQRT (Argument)
```

Calculates the square root of a number.

```
TRUNC (Argument)
```

Returns the whole number part of the expression.

Matrix Commands

```
DecompMat FromRegObj, ToReg1, ToReg2, ToReg3
```

Decomposes a matrix object in a register (Single Value Decomposition).

```
InvertMat RegObj, [DataRank]
```

Inverts a matrix object in a register based on the single value decomposition.

```
TransposeMat RegObj
```

Transposes a matrix object in a register.

Menu Commands

```
MenuAdd Label, ,[[Item], [[Action], [[Switch],
[[Message], [Link], [HelpContext]]]]]
```

Adds or modifies a menu item, label or submenu.

```
MenuDelete [Label, [Item]]
```

Removes an entire menu, a menu label from the menu bar, a menu item, or a cascading menu.

```
MenuRead Filename, [option]
```

Displays a menu based on a menu file description.

```
MenuRelabel Label, [Item], NewName
```

Renames an existing menu label, submenu item, or menu item.

```
MenuState Label, Item,  [Switch]
```

Sets the display appearance of a menu item.

```
MenuStop Label, Item, [Message]
```

Aborts all runs, sequences, and macro operations.

Method Control Commands

`ColComp6890`

Starts a column compensation run on the 6890 GC.

`EdHpce ([MethodPart])`

Displays the Edit HPCE Method dialog box.

`EditMethod ([MethodPart])`

Displays the Edit Entire Method dialog box.

`EdMethodInfo()`

Displays Method Information dialog box to enter method comments.

`EdRunTimeChkLst()`

Displays Run Time Checklist dialog box.

`EdSeqRecalibTab()`

Displays Sequence Recalibration Table dialog box.

`LoadMethParams`

Loads the Method Information, Run Time Checklist, Sequence Recalibration Table and Acquisition part of method.

`PopTimeSet {stoptime | posttime}`

Displays the dialogboxes for setting the stoptime or the posttime.

`PrintMethParams`

Displays the Print Method dialog box which allows specific parts of the method to be printed.

`RunAcqMethod`

Starts and executes only the acquisition of a method.

`SaveMethParams`

Saves Method Information, Run Time Checklist, Sequence Recalibration Table and Acquisition part of method.

`Simulation`

Displays a dialog box to simulate the current method.

`StartMethod`

Starts and executes a method to run according to the Run Time Checklist.

`StopMethod`

Stops current method or sequence.

Object Commands

```
CopyObj FromRegObjRange, ToRegObj
```

Copies a range of objects from one register to another.

```
DelObj RegObjRange
```

Deletes a range of objects from a register.

```
LoadObj FromFile, [ObjRange], ToReg
```

Loads a range of objects from a file and appends them to a register.

```
MoveObj FromRegObjRange, ToRegObj
```

Moves a range of objects from one register to another.

```
NewObj RegObj, ObjClass, Rows, Cols
```

Creates a new, empty object of a specified class in a register.

```
SaveObj FromRegObjRange, ToFile, [Mode]
```

Saves a range of objects from a register to a file.

Object Header Commands

```
DelObjHdr RegObj, ItemName
```

Deletes a header item from the object.

```
NewObjHdrText RegObj, ItemName, [Text],
[ProtectionMode], [MaxLength]
```

Creates a new text item in the object header.

```
NewObjHdrVal RegObj, ItemName, [Value],
[ProtectionMode], [TypeCode], [LowBound], [HighBound]
```

Creates a new numeric item in the object header.

```
ObjHdrName$ (RegObj,  Index)
```

Gets the item name of the object header with the specified index.

```
ObjHdrText$ (RegObj, ItemName)
```

Gets a single text item from the object header.

```
ObjHdrType (RegObj, ItemName)
```

Gets the type of an object header item or table.

```
ObjHdrVal (RegObj, ItemName)
```

Gets a single numeric item from the object header.

```
SetObjHdrText RegObj, ItemName, Text
```

Sets a single text item in the object header.

```
SetObjHdrVal RegObj, ItemName, Value
```

Sets a single numeric item in the object header.

ODBC Commands

`DBOpen` (DataSource, UserID, Password [, Autocommit] [,Timeout]])

Function to create a connection to an ODBC data source.

`DBClose [ConNum [,CommitWork]]`

The connection to the open data source will be terminated.

`DBExec` ConNum, SQL Command, RegisterObject, Table

Executes a SQL statement on a given data management system.

`DBGetInfo` ConNum, RegisterObject, Table

The command reads information about the specified data source and writes it into a table structure.

`DBGetSources$()`

The function returns a string value that contains all available data sources by name.

`DBNewRow` ConNum, RegisterObject, Table, [RowRange], DatabaseTable

The command reads the specific range of source data rows and inserts them into the destination database table.

Printer or Plotter Commands

```
BeginJob [prefix], [infix], [x], [y]
```

Instructs the printer module to gather pages to print a job.

```
DelJob JobFileName
```

Deletes a job file.

```
Device [Port]
```

Selects the destination for printer or plotter output.

```
DevNum()
```

Gets a logical device number associated with actual printer or plotter.

```
EndJob
```

Indicates the end of a print job to the printer module.

```
FormFeed
```

Sends a new page instruction to the printer.

```
FPrint fname
```

Print an ASCII file.

```
MFPrint mfname
```

Print a metafile.

```
OpenDevice "Printer" AS #FileNum
```

Associates a logical device number with a printer or plotter.

```
PageNum()
```

Gets the page number of the actual page.

```
SetFooter [text], [x], [y]
```

Sets footer text for pages.

```
SetHeader [text], [x], [y]
```

Sets header text for pages.

```
SetJobName Name
```

Sets the name for a print job.

```
SetMargin [left], [top], [bottom]
```

Set new values for the logical margins of a page.

```
SetPageLoc [x], [y]
```

Sets a new starting location within a page.

```
SetPageNum [n], [x], [y]
```

Sets the page number for the actual page to print.

```
SPrint text
```

Prints a string.

```
XLOC()
```

Gets a horizontal coordinate for printing or drawing.

```
XLogMax()
```

Gets the width of a logical page.

```
XMAX()
```

Gets the width of the page.

`YLOC()`

Gets a vertical coordinate for printing or drawing.

`YLogMax()`

Gets the height of a logical page.

`YMAX()`

Gets the height of a page.

Raw Data File Commands

```
File [DataFileName]
```

Specifies the name of the raw data file to be loaded.

```
LoadAIA FromDirectory, [FileName], ToReg
```

Converts files form AIA standard to chromatograms.

```
LoadChrom [Detector], [TimeRange], [WLRange],
[RefWLRange], ToReg
```

Extracts a chromatogram from a spectra detector and loads it into a register.

```
LoadSignal [Detector], [SignalId], [TimeRange], ToReg
```

Loads one acquired or all selected signals from the .CH files into a register.

```
LoadSpectra [Detector], [TimeRange], [WLRange], ToReg
```

Loads many spectra as one matrix object into a register.

```
LoadSpectrum [Detector], TimeRange, [WLRange],
[RefRegObj1], [RefRegObj2], [Mode], To Reg
```

Loads one spectrum or several spectra into a register.

```
SaveAIA FromRegObjRange, ToDirectory, BaseName
```

Converts chromatograms to AIA standard.

Register Commands

```
DelReg Reg
```

Deletes all objects in the register plus the register itself.

```
Exchange [Reg1], [Reg2]
```

Exchanges the contents of two registers.

```
RegCont$ (Reg)
```

Gets the contents of the register.

```
RegName$ (Index)
```

Gets the name of the register with the specified index.

```
RegSize (Reg)
```

Gets the number of objects in the register. (–1 if register non-existent).

Sequence Control Commands

`EditandLogTable`

Displays the sequence Sample Log Table.

`EdSampleInfo`

Displays Sample Information dialog box.

`EdSampleTable`

Displays Sample Table dialog box.

`EdSequenceParms`

Displays Sequence Parameters dialog box.

`EdSequenceTable`

Displays Sequence Table dialog box.

`LoadSeqParams`

Loads the specified sequence.

`PauseSequence`

Pauses current sequence. Sequence is paused after the current method is completed.

`PrintSeqParamsPrint`

Sequence Dialog Box is displayed to check parts of sequence to be printed. Checked parts are printed.

`ResumeSequence`

Resumes a paused sequence.

`SaveSeqParams`

Saves a current sequence to a file.

`StartSequence`

Starts current sequence.

RS232 Commands

```
RS232GetTimeout (devicename)
```

Gets the RS232-timeout.

```
RS232Receive$ (devicename)
```

Receives a string from an RS232 device.

```
RS232Send (devicename), textstring
```

Sends a text string to an RS232 device.

```
RS232SetTimeout (devicename), textstring
```

Sets the RS232-timeout.

Spectra Processing Commands

```
PurityCompare RegObjRange, RegObjRange, [PlotType],
[MinThresSuc],[Purity_Thresh],[Match_Thresh],[Variance],

PurityReg, [ThreshReg]
```

Checks purity by fast comparing spectra.

```
PurityGetPeakSpectra [Detector], [TimeRange], [WLRange],
Reference, [Threshold], [DoZero], [AutomaticRef],
MatrixReg, [ [DoLoad], PeakTime, Width, Symmetry,
SpectraReg]
```

Gets peak spectra to check peak purity

```
SelectSpeclibEntries [FromReg],[ToRegObj],[ShowIDNames]
```

Allows to multiple select library entries.

Spectral Data Handling Commands

```
LoadSpectra [Detector], [TimeRange], [WLRange], ToReg
```

Loads many spectra as one matrix object into a register.

```
LoadSpectrum [Detector], TimeRange, [WLRange],
[RefRegObj1], [RefRegObj2], [Mode], ToReg
```

Loads one spectrum or several spectra into a register.

Spectral Library Commands

`LibAddEntry ([FromRegRange], [FromRegObj], [ToReg])`

Adds one or more entries to a library.

`LibCreateEntry ([FromReg], [FromRegObj], [ToReg])`

Displays the Library Add Entry dialog box.

`LibEditEntry ([FromReg], [FromRegObj1], [FromRegObj2])`

Calls the Library Edit Entry dialog box.

`LibEditHeader ([FromRegObj1], [FromRegObj2])`

Displays the Edit Library Header dialog box.

`LibEditWaveTable (FromRegObj, [ToRegObj])`

Displays the Library Edit Wave Table dialog box.

`LibManager (FromReg, [ToRegObj])`

Displays the Library Manager dialog box.

`LibSearchTemplate ([FromRegObj])`

Displays the Library Search Template dialog box.

`PMLibrary ([FromReg], [ToRegObj])`

Displays the Search Results Print Manager dialog box.

`PMSearchResults ([FromRegObj], [ToRegObj])`

Calls the Search Results Print Manager dialog box.

Spectrum Commands

```
DispPurity ChromReg, PeakNumber, [SpecReg],
[PurDiffReg], [PurSigReg], [PurRatReg], Action
```

When used within a macro, DispPurity displays a dialog box containing the results of a peak purity calculation. (HPLC^{3D} ChemStation Only).

```
ExportSpectrum FromRegObj, ToFile, FileType
```

Saves (exports) a spectrum as a .WAV or a .DX file.

```
ImportSpectrum FromFile, FileType, ToReg
```

Loads (imports) a spectrum from a .WAV or a .DX file.

String Handling Functions

```
Chr$ (Argument)
```

Returns the character represented by a number according to the ASCII table.

```
INSTR (s1, s2)
```

Returns the position of string s2 in string s1.

```
LEN (Argument)
```

Returns the number of characters present in the argument.

```
VAL (Argument)
```

Returns the numerical version of a string containing a numeric value.

```
VAL$ (Argument)
```

Returns the string version of a numeric value.

```
WrapText$(InputString, Width [, Height],
[TruncIndicator])
```

Wraps around the given text.

Table Commands

Creating and Removing Tables

```
CopyTab FromRegObj, FromTabName, ToRegObj, ToTabName,
[ProtectionMode]
```

Copies a table from one object to another.

```
DelTab RegObj, TabName
```

Deletes a table.

```
NewTab RegObj, TabName, [FromRegObj, FromTabName],
[ProtectionMode]
```

Creates a new, empty table in a object with the same structure as FromTable.

```
RenTab RegObj, TabName, NewTabName
```

Renames a table in an object.

Adding and Deleting Columns

```
CopyTabCol FromRegObj, FromTabName, FromColName,
ToRegObj, ToTabName, ToColName
```

Copies the contents of a column from one table to another.

```
DelTabCol RegObj, TabName, ColName
```

Deletes a column from a table.

```
NewColText RegObj, TabName, ColName, [InitText],
[ProtectionMode], [MaxLength]
```

Creates a new text column with initial value in a table.

```
NewColVal RegObj, TabName, ColName, [InitVal],
[ProtectionMode], [TypeCode], [LowBound], [HighBound]
```

Creates a new numeric column with initial value in a table.

```
ProtectTabCol RegObj, TabName, ColName, ProtectionMode
```

Changes the access protection of a table column.

```
RenTabCol RegObj, TabName, ColName, NewColName
```

Renames a column in a table.

```
TabColName$ (RegObj, TabName, Index)
```

Gets the name of the table column with specified index.

```
TabColType (RegObj, TabName, ColName)
```

Gets the type of a table column.

Adding, Deleting And Copying Rows

```
CopyTabRow RegObj, TabName, FromRowRange, ToRowIndex
```

Copies the contents of a (range of) row(s) in a table from one place to another.

```
DelTabRow RegObj, TabName, RowRange
```

Deletes a (range of) row(s) from a table.

```
ExchangeTabRow RegObj, TabName, RowRange, ToRowIndex
```

Exchanges two single rows or two blocks of rows.

```
InsTabRow RegObj, TabName, [RowRange]
```

Inserts a new row or range or new rows into a table.

```
MoveTabRow RegObj, TabName, RowRange, ToRowIndex
```

Moves a single row or a block of rows to a new location.

Accessing Table Elements

```
RowByText (RegObj, TabName, [RowRange], ColName,
Condition, Text)
```

Gets first row with specified column content in a table.

```
RowByVal (RegObj, TabName, [RowRange], ColName,
Condition, Value)
```

Gets first row with specified column content in a table.

```
SetTabText RegObj, TabName, RowIndex, ColName, Text
```

Sets a single text element in a table.

```
SetTabVal RegObj, TabName, RowIndex, ColName, Value
```

Sets a single numeric element in a table.

```
TabText$ (RegObj, TabName, RowIndex, ColName)
```

Gets a single text element from a table.

```
TabVal (RegObj, TabName, RowIndex, ColName)
```

Gets a single numeric element from a table.

Accessing Table Headers

```
DelTabHdr RegObj, TabName, ItemName
```

Deletes a header item from a table.

```
NewTabHdrText RegObj, TabName, ItemName, [Text],
[ProtectionMode], [MaxLength]
```

Creates a new text item in the table header.

```
NewTabHdrVal RegObj, TabName, ItemName, [Value],
[ProtectionMode], [TypeCode], [LowBound], [HighBound]
```

Creates a new numeric item in the table header.

```
SetTabHdrText RegObj, TabName, ItemName, Text
```

Sets a single text item in a table header.

```
SetTabHdrVal RegObj, TabName, ItemName, Value
```

Sets a single numeric item in a table header.

```
TabHdrName$ (RegObj, TabName, Index)
```

Gets the name of the table header with specified index.

```
TabHdrText$ (RegObj, TabName, ItemName)
```

Gets a single text item from a table header.

```
TabHdrType (RegObj, TabName, ItemName)
```

Gets the type of table header item.

```
TabHdrVal (RegObj, TabName, ItemName)
```

Gets a single numeric item from a table header.

Table Window Commands

```
EdDataTab WinNum, RegObj, [DispTabName], [FirstRow]
```

Displays a data block.

```
EdObjHdrTab WinNum, RegObj, [DispTabName], [FirstRow]
```

Displays a list of object header names in an object.

```
EdObjTab WinNum, Reg, [DispTabName], [FirstRow]
```

Displays object headers across all objects in a register.

```
EdTabHdrTab WinNum, RegObj, TabName, DispTabName,
[FirstRow]
```

Displays a list of table header and column names in an object.

```
EdTab WinNum, RegObj, TabName, [DispTabName], [FirstRow]
```

Displays a table with selected columns.

```
SelectTableCell WinNum, LineNumber , ColumnNumber
```

Selects a table cell of a table window.

```
SelectTableLine WinNum, SelectionMode, LineNumber ,
[Visible]
```

Select or deselects one or all table rows of a table window.

Timing Functions

```
DATE$ ([Mode])
```

Returns today's date in mm/dd/yy format.

```
Mtime([Mode])
```

Returns the time in milliseconds since the start of the system.

```
Sleep [Time]
```

Used in a macro to do nothing (sleep) for the specified number of seconds.

```
TIME ()
```

Returns the current time in seconds since January 1, 1970.

```
TIME$ ([Mode])
```

Returns the current time.

User interface Communication Commands

```
AboutBox
```

Dialog box displaying the current software revision and copyright statement.

```
ALERT (Message, [Type], [Title])
```

Displays a message box with buttons requiring user response.

```
AppTitle [title]
```

Shows an application title.

```
Background [BitmapFile]
```

Specified bitmap file is displayed as background.

```
CmdLine ON|OFF
```

Controls the display of the command line.

```
Help [HelpContext], [Delay]
```

Pops up the ChemStation help system.

```
HelpWin HelpContext, WinName
```

Displays a help topic in a secondary window.

```
INPUT (Prompt, Variable, [Variable], […])
```

Loads one or more variables using one or a sequence of dialog boxes that prompt the user for each value.

```
ListMessages ON|OFF
```

Controls the display of a listing window displaying the last messages from the software. Similar to a message logbook.

```
Message [Text]
```

Displays text in the user interface of the command processor.

```
Print Expression [Expression], […]
```

Writes the results of string or scalar expressions in free format to the message line of the current application.

```
Print Using FormatImageString, Expression, [Expression], […]
```

Writes formatted results of string or scalar expressions to the message line of the current application.

```
SELECTFILE, ([Style], [Title], [Extension], Path, File, [Type])
```

Dialog box that prompts the user to specify a file or directory including path.

```
SetFKey Key, Command, [Label]
```

Assigns a ChemStation command, function or macro to one of the PCs function keys (F1 to F12).

```
Show [Selector], […]
```

Displays currently available identifiers.

View Commands

```
ClearWin [WinNum]
```

Hides any graphic, table or other window.

```
SetRunStatus [SystemStatus], [MethodStatus]
```

Displays a status line(s) in the RunStatus window.

```
SetWinTitle WinNum, Title
```

Gives a title to a display window.

```
SwitchView ViewName
```

Switches the display from the current set of display windows. The set of display windows is defined in the corresponding view table.

Window Commands

```
ActiveWindow ()
```

Gets the window number of the active window.

```
AppendSelect  WinNum, RegObj, [Type], [Index], [XPos1],
[YPos1], [XPos2], [YPos2]
```

Appends one row to the selection table of the window.

```
ClearSelect WinNum, [Row Index]
```

Clears the selection table or row of the selection table of the window.

```
FindNearestAnn WinNum, XPos, YPos
```

Finds the annotation closest to the given XPos and YPos.

```
FindNearestObj WinNum, XPos, YPos [ObjNum]
```

Finds the object closest to the given XPos and YPos.

```
FreeWin (WinNum)
```

Find the next free window number, starting at a given one.

```
GetSelect WinNum, [RowIndex]
```

Obtains one row of the selection table of the window.

```
IsSelected (WinNum, RegObj, Type, Index, XPos1, YPos1,
XPos2, YPos2)
```

Checks whether a specified selection is defined for a window.

```
SelectSize (WinNum)
```

Gets the number of rows in the selection table of the window.

```
Setactivewindow WinNumber
```

Changes the focus to the window specified.

```
WinUpdate [ON/OFF]
```

Enables or disables update of window contents.

Index

Index

Index

Index

Your Comments Are Welcome

We welcome your evaluation of this book. Your comments and suggestions help us improve our publications. Please attach additional pages of comments if necessary.

1 Please circle Yes or No for each of the following:

Is it easy to find the information you need when you need it?	Yes No
Is the information technically accurate?	Yes No
Are the instructions clear and complete?	Yes No
Are there enough examples and illustrations?	Yes No
Are concepts explained clearly?	Yes No

2 Please rate the following features of the book for their usefulness.

1 = Inadequate. 2 = Adequate. 3 = Superior.

Table of contents	1	2	3
Index	1	2	3
Tabs	1	2	3
Glossary	1	2	3
Illustrations	1	2	3
Examples	1	2	3
Readability	1	2	3

Comments _____

_____

_____

_____

_____

Name _____

Title _____

Company _____

Address _____

City and State _____

Country _____ Postal Code _____ Phone_____

Please tear out and mail or fax.

Hewlett-Packard GmbH
Hewlett-Packard-Strasse 8
D-76337 Waldbronn
Germany

Fax (+49) 7243 602 501

HP ChemStation
Macro Programming Guide
G2070-90107 Second edition 11/95 Printed in Germany

Marketing Communications Group
Hewlett-Packard GmbH
Hewlett-Packard-Strasse 8
D-76337 Waldbronn
Germany

PUT
POSTAGE
HERE

Do not Staple - Please fold and tape

Your Comments Are Welcome

We welcome your evaluation of this book. Your comments and suggestions help us improve our publications. Please attach additional pages of comments if necessary.

1 Please circle Yes or No for each of the following:

Is it easy to find the information you need when you need it?	Yes No
Is the information technically accurate?	Yes No
Are the instructions clear and complete?	Yes No
Are there enough examples and illustrations?	Yes No
Are concepts explained clearly?	Yes No

2 Please rate the following features of the book for their usefulness.

1 = Inadequate. 2 = Adequate. 3 = Superior.

Table of contents	1	2	3
Index	1	2	3
Tabs	1	2	3
Glossary	1	2	3
Illustrations	1	2	3
Examples	1	2	3
Readability	1	2	3

Comments _____

_____

_____

_____

_____

Name _____

Title _____

Company _____

Address _____

City and State _____

Country _____ Postal Code _____ Phone_____

Please tear out and mail or fax.

Hewlett-Packard Company
Publications Department
2850 Centerville Road
Wilmington, DE 19808

Fax 302 633 8911

HP ChemStation
Macro Programming Guide
G2070-90107 Second edition 11/95 Printed in Germany

Hewlett-Packard Company
Publications Department
2850 Centerville Road
Wilmington, DE 19808

PUT
POSTAGE
HERE

Do not Staple - Please fold and tape

About This Edition

First edition, 10/94

Second edition, 11/95

This handbook is for A.03.xx revisions of the ChemStation software, where xx is a number from 00 through 99 and refers to minor revisions of the software that do not affect the technical accuracy of this handbook.

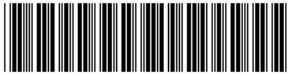
The handbook printing date and part number indicate the current edition. The printing date changes when a new edition is printed.

HEWLETT® PACKARD

In This Book

This handbook describes how to work with the commands to customize your ChemStation to make its operation more flexible. It explains programming techniques, and uses frequent examples to show how these techniques work in actual applications.

Printed in Germany 11/95
G2070-90107



G2070-90107