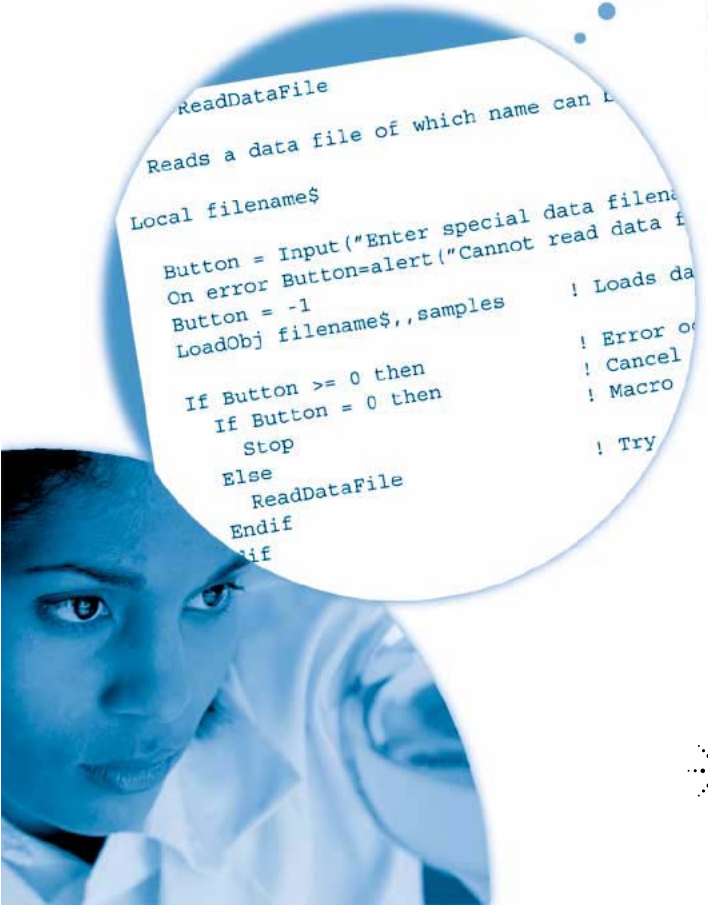
Agilent ChemStation for UV-visible Spectroscopy



Macro Programming Guide

Agilent Technologies

Notices

Manual Part Number

G1116-90006

Edition

10/2003

Printed in Germany

Agilent Technologies Deutschland GmbH
Hewlett-Packard-Strasse 8
76377 Waldbronn

Warranty

The material contained in this document is provided "as is," and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Agilent Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

Safety Notices

CAUTION

A **CAUTION** notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a **CAUTION** notice until the indicated conditions are fully understood and met.

WARNING

A WARNING notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a WARNING notice until the indicated conditions are fully understood and met.

In This Guide...

This book describes how you work with commands to customize your Agilent ChemStation for UV-visible spectroscopy, making its operation more flexible. It explains programming techniques and uses frequent examples to show how these techniques work in actual applications.

1 Internal Structure

This chapter describes the structure of ChemStation variables and how you enter commands.

2 Using Commands

This chapter explains how you use variables—with the Show command as an example of a simple command.

3 From Commands to Macros

This chapter explains the purpose and basic structure of a macro and how you write macros using commands.

4 Entering Data into a Macro

In this chapter three ways to enter data into a macro are shown.

5 Permanent Data

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

6 Registers and Tables

This chapter explains how the ChemStation uses registers and tables to handle data.

7 Windows and Display Description Tables

This chapter explains how the ChemStation uses windows to display data.

8 Spectral Data Register

This chapter explains how the ChamStation handles spectral data.

9 Reports and Print Description Tables

This chapter explains how the ChemStation prints reports.

10 Exchanging Information Between Windows Applications by DDE

This chapter contains dynamic data exchange and example macros.

11 Communicating Through the RS232 Serial Interface

This chapter explains how to set up communication channels.

12 Variables

This chapter contains system, string, scalar, and other predefined command processor variables.

13 Windows

This chapter contains a table of all parameters in the predefined ChemStation windows.

14 Registers

This chapter contains the registers used by the ChemStation.

Contents

1
Internal Structure

Entering Commands

The command processor (CP) is the part of the ChemStation software that interprets and executes commands. To enter commands you use the command line or macro hooks.

Using the Command Line

You use the command line to execute commands interactively. The command line appears at the bottom of the ChemStation Window, below the message line. The System menu allows you to turn the command line on or off. A cursor at the beginning of the command line shows where you start typing.

If you type:

```
Print "This happens, when you use the command PRINT"
```

on the command line and press ENTER, your ChemStation displays the sentence on the message line. In the example above, the command processor reads the command line, interprets *Print* as a command, and displays the sentence in double quotes on the message line.

Using Macro Hooks

The ChemStation has four additional places where it can execute commands during a measurement cycle and within an automated run.

- The method checklist has two edit lines that can hold a command or macro.
- The ChemStation executes the Pre-Measure Macro before making spectral measurements. You can use this macro, for example, for customized sample preparation and introduction.
- The ChemStation executes the Post-Measure Macro after making spectral measurements. You can use this macro, for example, to modify or add additional information to a spectrum or to remove the sample from the cell.
- The automation table contains the item User Macro. The ChemStation executes an entry for this item during the run of the automation table.

- A function is a macro that returns a value. You can use functions in the equation parameter box.
- A macro with the macro name AfterDataAnalysis is automatically executed, if available, after all data analysis. To use that macro hook, a macro with that particular name must be loaded. Every new method removes that macro automatically.

System Variables

The command processor uses predefined variables to hold information used frequently by the ChemStation, for example, data paths or the latest error information. The variables can hold strings or numeric values. The ChemStation annotates string variables by a dollar sign at the end of the variable name. System variables begin with an underscore character. You cannot remove system variables and some of them cannot be changed.

For example, the command Print _Operator$ displays the current operator name – a string – on the message line and Print _OffLine displays 0 or 1 on the command line (0 if you started the ChemStation in online mode or 1 if you started the ChemStation in offline mode).

Chapter 12, “Variables” gives a list of system variables and describes how you use them. To look at these variables you use the Show command explained in Chapter 2, “Using Commands”.

Register and Objects

Chapter 3, “From Commands to Macros” described how you access general information using system variables. Spectral data is too complex to be stored in simple variables and so the ChemStation stores this type of data in registers. The ChemStation uses a set of predefined registers for different purposes. You can define your own registers to handle complex data in your macros.

A register comprises a contents list and one or more sections called objects. Each object has a summary followed by detailed information. Chapter 6, “Registers and Tables” describes how to access registers.

The following list summarizes the commonly-used ChemStation registers. Chapter 12, “Variables” gives a complete list of all registers. The ChemStation uses four registers to hold raw data. These registers are the destination for the measurements done by the spectrophotometer, or for data loaded or imported from disk. You can load or save the sample and standard registers using the File menu.

Blank	Contains baseline spectrum taken by the latest Blank measurement.
Samples_Append	Buffers last measurement of *raw* sample spectra.
Samples	Contains *raw* spectra measured or loaded as Samples. All these spectra are used as the source for the action Analyse.
Standards_Append	Buffers last measurement of *raw* standard spectra.
Standards	Contains *raw* spectra measured or loaded as Standards . All these spectra are used as the source for the action Calibrate.
Auxiliary	Contains spectra measured or loaded as Auxiliary. One of these spectra can be used as the second spectrum in a binary mathematical operation such as add, subtract, multiply, and divide.

A further set of registers hold the spectra and data generated by the actions Analyze (for samples) and Calibrate (for standards). The parameters for data analysis are common to both samples and standards to make sure the ChemStation treats both data identically. There are four of each of theses corresponding to the four data analyses you can specify.

The suffixes xxx_1, xxx_2, xxx_3, and xxx_4 specify the corresponding data analysis (for example, ProcessedSamples_1 for data analysis 1 and ProcessedSamples_3 for data analysis 3). If you use confirmation analysis, xxx_2 corresponds to confirmation analysis 1, xxx_3 to confirmation analysis 2, and xxx_4 to confirmation analysis 3.

If you specify spectral processing in data analysis, the following register holds the result of all specified steps. If you specify no spectral processing, the content is a copy of the corresponding raw spectra.

ProcessedSamples_x	Holds the processed sample spectra after finishing all specified processing steps activated by the action Analyze.
ProcessedStandards_x	Holds the processed standard spectra after finishing all specified processing steps activated by the action Calibrate .

In the next step, Use wavelength, a set of data can be specified. These data values are copied to a further set of registers. If you specify None as Use Wavelength, nothing is copied to the registers.

WLResult_SMP_x	Holds the calculated data specified in the Used Wavelength step after the action Analyze. The sources are the ProcessedSamples_x registers.
WLResult_STD_x	Holds the calculated data specified in the Used Wavelength step after the action Calibrate. The sources are the ProcessedStandards_x registers .

The last step in data analysis is evaluation. Evaluations are using the data in the WLResult_XXX_x registers. If you select None, nothing is copied to the result registers. If you select Equation, SCA, or MCA, the calculated result is copied to the registers.

Eval_Results_x	Holds the calculated result data of all Samples.
Eval_Results_STD_x	Holds the evaluated data of the Standards.

Figure 1 shows the data flow and corresponding registers.

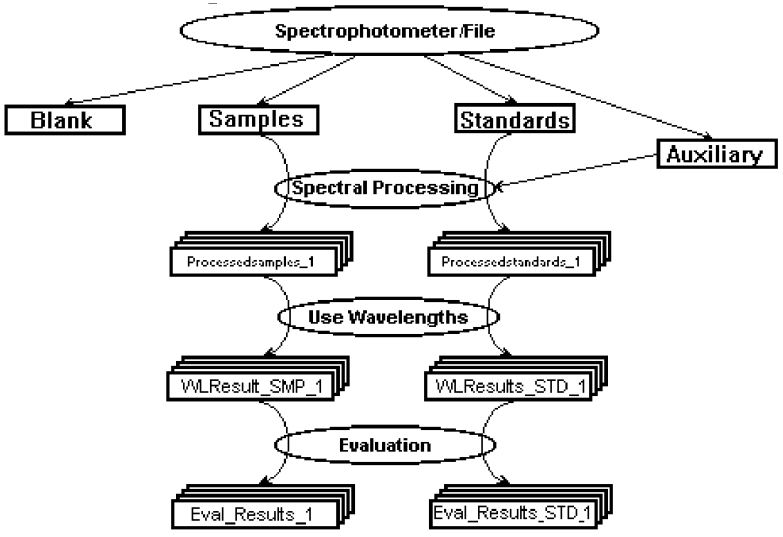


Figure 1 Data flow and Registers

The ChemStation copies the spectral results of interactive mathematical operations to a register called Arithm_Results. You must select the operands for these interactive operations before the ChemStation performs the calculations.

Another important register is the _Config register. This register contains eight objects.

User Interface	Contains information about axis styles, colors, windows, and views.
Table Templates	These templates can be used to create description tables by copying one of the templates.
Display Description Table	These tables contain information about the appearance of the data tables on the screen. Chapter 7, "Windows and Display Description Tables" describes these tables in detail.
System	Contains a translation table for commonly used messages.

Automation	This object is empty.
Acquisition	Contains information about path length, sampling systems, and spectrophotometer parameters.
Data Analysis	Contains parameters used by tasks in the Task menu.
Report	Contains print parameters, such as left margin and static text, used in the printed report.

2
Using Commands

Using the Show Command

The Show command displays a list of all commands, functions, labels, local variables, macros, open devices, standard variables, system variables, and DDE (dynamic data exchange) hot-linked variables that the ChemStation has currently available. If you call Show from the command line and not from within a macro, the section label and local variables are empty. For commands, functions, and macros you will also get a list of parameters for this action, if you select it in the list box at the right. If you leave the box after a selection using OK , the selection will be copied to the command line and the message line will show the syntax.

As an example, type `Show` on the command line and press ENTER. Select Commands and the list on the right shows you all the commands. Select ListMessages and the text field at the bottom of the box shows the parameters, in this example, ON|OFF[,ON|OFF(font)]. Choose OK to close the box. The ChemStation copies the command ListMessages to the command line and displays the parameter ON|OFF[,ON|OFF(font)] on the message line. Move the cursor to the end of the command line, type `ON` and press ENTER. This command opens a list box showing the recent messages from the ChemStation.

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

Using Strings and Scalar Variables

Variables are handled by variable names. Two types of variables are available: scalar and string variables. The last character of the name of a string variable is always a dollar sign. String variables contain text and you use them for processing user-specific information. For example:

```
A$="";For I = 1 to 100;A$=A$+".";Print A$;Next I
```

This command first defines the string variable A$ and assigns an empty string to the variable. The command then uses a For...Next loop to execute the bracketed commands repeatedly, in this example, 100 times. Each time the ChemStation goes through the loop, it appends a period (.) to the string of variable A$ and then displays the string on the message line. The variable I is the counter for the loop and is called a scalar variable because it has a numeric value. In contrast to string variables, scalar variables do not end with a dollar sign.

For example:

```
My_Value1 = 12
My_Value2 = 4
My_Value2 = My_Value2 + 2

Print My_Value1, My_Value2, My_Value1 + My_Value2
```

This displays the values 12, 6, and 18 on the message line.

To use variables you do not need to define them specifically. Using a string or scalar variable on the left side of the assignment defines the variable automatically as a global variable and it exists until you remove it with the Remove command or until you quit the ChemStation software. You can access the content of global variables from any macro in the ChemStation because, once defined, the ChemStation recognizes them throughout the software.

Naming Variables

The names of Variables:

- must be less than 98 characters length, and
- may contain lower or uppercase alphabetic characters (the ChemStation does not differentiate between lower and uppercase characters).

Using lower and uppercase letters in names helps readability. For example:

```
ConcOfUnknown = 12.3
```

3
From Commands to Macros

The most effective way to enter a number of commands is to group the commands together and give this group a 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. You can edit this subtitle or delete its content.

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

To write a macro you need to use a word processor. We recommend using Notepad. 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 a message looks like this:

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

Saving a Macro

Choose Save As from the File menu of Notepad. A dialog box appears to select a directory and to type a filename for the macro. For this example select C:\HPCHEM\UVEXE\SYSMACRO (assuming the ChemStation software was loaded with the default path settings) as the directory and type MYMACRO.MAC as the macro filename. When you choose OK the macro becomes a file on the hard disk.

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

A macro file can include more than one macro.

Loading a Macro

To load a macro, go to the ChemStation window and type:

```
Macro "MyMacro.mac"
```

When you press ENTER the macro is loaded into computer memory.

To access all macro filenames, type:

```
Macro
```

and press ENTER. You can select the directory and the macro filename. Choose OK to load the selected macro.

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

You can make macros more useful than the individual commands by using variables and logic. This section describes how you use a variable to print your name,

Go to 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. Load the macro by typing:

```
Macro "MyMacro.mac"
```

and then start the macro by typing:

```
MyMessage "YourName"
```

and press ENTER.

The ChemStation displays the following text:

```
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.

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 after the Name and before the EndMacro commands are executed immediately when you load the macro file.

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$,  ”!”
EndMacro

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

Save the macros in a file. Load the macros by typing:

```
Macro ”myMacro.mac”, go
```

The ChemStation displays:

```
This is a message from YourName
```

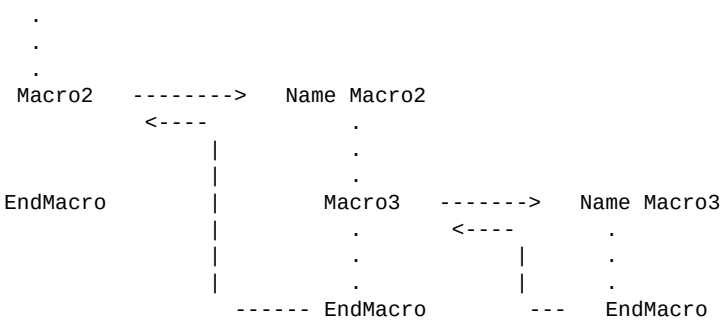
Two seconds later the ChemStation overwrites this message with:

```
ok!
```

What happened?

- The ChemStation starts the macro Display_ok because:
  - Display_ok is the last macro in the file.
  - You typed the word *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 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**.

Using Function Macros

You can use functions in macros to make the macro useful, for example, for solving 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 number π.

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 command 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 command with parameters makes the macro a function. Names of function macros that return strings must end in a dollar sign.

You cannot start a function macro in the same way as described in "Starting a Macro" on page 28. 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$("YourName")
Print M$
```

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")
```

For more information about function macros and functions in general, see Chapter 4, "Entering Data into a Macro".

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:

Executing Your Macro using a Method

To execute your macro part of a sample analysis, choose Method Checklist from the Method menu and enter the name of the macro. Specify the macro as either the pre-measure or post-measure macro.

Remember to save your method after any changes.

If your macro has the name AfterDataAnalysis it will be automatically called by the system after all samples are analyzed. This allows you for example to perform additional calculations based on the ChemStation results.

NOTE The macro is not part of the method and must be loaded manually by the user or automatically during start of the ChemStation. The AfterDataAnalysis macro is always removed if a method is loaded.

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. See Chapter 4, “Entering Data into a Macro” for more details.

Loading User Macros Automatically

You can define a macro file called UMACINIT.MAC in the ChemStation user macro directory. The name of this directory depends on your installation; the default is: C:\HPCHEM\UVEXE\USERMAC for the ChemStation.

The UMACINIT.MAC file in this directory will be automatically executed every time the ChemStation starts. This file should contain macros only and not direct executable commands.

There is a file UMACINIT.MA_ already in this directory. Rename the file or copy it to UMACINIT.MAC and read the important information in this file about how to use it to load and execute your macros.

If a macro called UM_AutoStart is part of your UMACINIT.MAC file, the ChemStation will execute this macro automatically when you start the software. This allows You the automatic execution of a general initialization of your macros.

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 type `*.MAC` in the file select box. Select the appropriate directory, from the directory list and all the files with the extension .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 File Manager or the ChemStation Delete command. Deleting the file from the disk removes your macro files permanently.

Removing a Macro from Memory

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

```
Remove MyMessage
```

If you have more than one macro in the macro file and 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 memory does not remove the macro file from the hard disk.

NOTE The macro filename 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 filename with the .mac extension.

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
```

This assigns the value 20 to the scalar variable Twenty.

Start the second macro by typing:

```
ShowVariable
```

The ChemStation displays

```
Twenty = 20
```

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.

Commenting Macros

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

```
! Macro function to return the nearest integer value
! Written by Your Name on 11/11/93

Name Nearest
  Parameter Number default 1 ! sets Number to 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 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 scalar 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**.

This example showed that different variables can have the same name if you declared them as local variables. 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.

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.

"Commenting Macros" on page 39 describes how to put comments in macros. Remember to begin a comment with an exclamation mark (!). The command processor ignores any text after the exclamation mark.

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

Example 1

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

Example 2

```
! 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 (;).

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 (\).

```
Analyte_Name$= TabText$(Eval_Results_1[1],AnalyteTable,\
MyCounter,AnalyteName)
```

You can use macros as subroutines, each macro calling another macro. You can repeat this as many times as you need as described in "Complex Macros" on page 30.

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. Meaningful parameter names helps You a lot in macro programming. The parameter names appear with the show command in the macro domain. A parameter name clearly identifying its purpose helps using this macro successfully.

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. You will find a complete list of these functions in your *Commands* handbook.

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$ = ”Agilent Technologies”
```

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$
```

The ChemStation displays:

```
Print ”This is a 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$
```

The ChemStation displays:

```
This is a 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. You will find a complete list of these functions in your *Commands* handbook.

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 concentration, and temperature, 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, 21.5”  ! 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 scalar variables — the following two commands give the same results:

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

Process the Line$ string to extract the concentration and the temperature. 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 time and amount:

```
PosN = InStr(Line$, ”,”)    ! finds the comma
Conc =   Val(Line$[1:PosN - 1])
Temp = Val(Line$[PosN + 1: Len(Line$)]
```

The Val() function returns 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 Statements

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 and works only on scalar expressions. Table 1 shows additional operators. Only the operators *equal to* (=) and *not equal to* (<>) can be used with string expressions.

Table 1 Scalar operators

Operator	Description
<	Less than
<=	Less than or equal to
=	Equal to
>=	Greater than or equal to
>	Greater than
<>	Not equal to

You can combine these basic operators in Table 1 with the operators in Table 2 which are called Boolean operators.

Table 2 Boolean operators

Operator	Description
AND	Logical sum
OR	Logical or
NOT	Logical negation

Two 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 conditions are *true*.

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

You use the operators AND, OR, and NOT with numeric expressions to form larger expressions. These operators are not case sensitive — you can use lower or uppercase characters.

For example:

```
a equals 1 AND b equals 2
```

is true only when both expressions are true.

In contrast:

```
a equals 1 OR b equals 2
```

is true when either expression is true.

Operators like AND, OR, and NOT have lowest priority compared to other operators. You can use parentheses to override this priority.

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 statements with conditional statements to test whether or not a set of commands is executed. You can omit the Else part of the conditional command 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.

```
IF <logical expression> THEN
    execute these commands
  ELSE
    execute these commands
ENDIF
```

Below is an example of decision making in a macro for the calculation of vitamin A content. Due to the absorbance at three wavelengths the content has to be calculated in different ways. The macro is a function and has to be used as an equation in the Method Data Analysis Equation dialog box. To pass the absorbance values, call it with the variables R1, R2, and R3 as parameters. The setup of the Data Analysis is as follows:

Table 3 Data analysis setup

Spectra processing:	Spline, Step 2	
Used wavelengths:	List: 310, 325, 334	
Equation:	Normalize pathlength: yes	
Name	Equation	Unit
Vitamin A	vitamina(R1,R2,R3)/A1	mg
A1	weight	

```
name vitamina

! Calculate Vitamin A based on USP method

  parameter a,b,c                        ! Absorbance values R1,R2,R3
  local res,corr

  corr = 6815*b-2.555*a-4.26*c           ! Calculate corrected absorbance
  If (b > corr/1.03) AND (b %< corr/0.97)
    res = 0.549*b                        ! Use single absorbance
  EndIf
  If corr %< b/1.03
    res = 0.549*corr                     ! Use corrected absorbance
  Else
    res = 0                              ! Invalid result
  EndIf

return res

endmacro
```

Using Recursive Macros

In "Complex Macros" on page 30 we described macros that call themselves — we called this process **recursion**. The macro calls itself, unless you program it to stop, until all memory is used. Use the If command to define programming criteria.

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

```
Name Factorial

! Calculates the factorial of a given number

  Parameter Number                       ! number of factorial
                                         ! to be calculated

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

EndMacro ! Factorial
```

Repeating Parts of the Macro

In "Using Decision-Making Statements" on page 50 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 analyte quantified. You could write a macro for a particular method with a fixed number of calibrated analytes. But this macro could not be used on another method unless the methods have the same number of calibrated analytes. It is better to repeat processing the individual analytes until all in the method have been processed. By writing the macro in this way you can use it any data file quantified by any calibration.

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

- While
- Repeat
- For

Using the While and EndWhile Commands

You 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 the variable _Error is set by the Input command and equals 0 (for no error). As long as _Error remains equal to 0 the lines of the file will be read and displayed. As soon as the end of the file is reached or if the file is empty, _Error becomes unequal to 1 and the loop ends at the EndWhile command.

```
Input #1, Line$          ! tries to read first string from file
While _Error = 0         ! _Error is set by Input command
  Print Line$            ! displays string
  Input #1, Line$        ! read next string from file
EndWhile
```

The parameter #1 is the identifier of the file which is open. You will find details of file identifiers in "File Identifiers" on page 71.

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

Using the Repeat Command

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 and Next Commands

You use the For and Next commands to construct loops when you run an index from a defined start index to a defined end index with increments of 1.

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.

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
```

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 the same way as you open a file, for example:

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

The ChemStation prints through Windows allowing 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.

NOTE See note in "Using the While and EndWhile Commands" on page 52.

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 a macro stops, type:

```
ListMessages on
```

A box appears in the ChemStation window where multiple messages can be 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 incorrect command is displayed.

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\UVEXE\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 command was executed last. You can include the logging command in your macro to record all critical paths of the macro.

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 Clipboard and paste them to the ChemStation command line by holding down CTRL and pressing V . This allows you to test and modify each command separately. You can also copy the debugged line back to the Clipboard by holding down CTRL and pressing 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 or Print #1 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.

The Print #1 command prints the text on the printer. Remember you must open and close the printer device, see “Printing the Results” on page 55.

For example:

```
Name MyMacro
  .
  .
  .
For Index = 1 to Max
  If ThisWay = Yes then
    Print #1, ” ***** go this way *****” ! debug
      .
      .
      .
  Else
    Print #1, ” ----- 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 an exclamation mark. 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_Flag = 1   ! Debug flag
  .
  .
  .
  For Index = 1 to Max
    If ThisWay = Yes then
      If (Debug_Flag = 1); Button = Alert (” ***** going this way *****”,
 1); endIf
      .
      .
      .
    Else
      If (Debug_Flag = 1); Button = Alert (” ----- going that way -----”,
 1); endIf
      .
      .
      .
    EndIf
  Next Index
```

To turn 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 spectra 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 ReadDataFile

! Reads a data file of which name can be entered

Local filename$

  Button = Input("Enter special data filename:", filename$)
  On error Button=alert("Cannot read data file! Try it again!",1)
  Button = -1
  LoadObj filename$,,samples     ! Loads data file in samples register

  If Button >= 0 then            ! Error occured
    If Button = 0 then           ! Cancel pressed
      Stop                       ! Macro execution
    Else
      ReadDataFile               ! Try it again
    Endif
  Endif

endmacro                         ! ReadDataFile
```

The macro ReadDataFile allows 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 #1
  Button = Alert ("Closing first opened file!", 2)
  Stop
EndMacro ! CleanUpLevel1

Name CleanUpLevel2
  Close #1; close #2
  Button = Alert ("Closing all opened files!", 2)
  Stop
```

```
EndMacro ! CleanUpLevel1

Name ProcessSpectra

! Process spectra and make some special calculations

  On error cleanUpLevel1
  Open ”factors.txt” as #1 for input

  On error cleanUpLevel2
  Open ”results.txt” as #2 for output
  .
  .
  .
  If numPeaks <= 1 then; generate error; endIf
  .
  .
  .
```

If opening the first file fails, the macro CleanUpLevel1 is started. If opening the second file fails, the macro CleanUpLevel2 is started. When there are not enough spectra 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.

4
Entering Data into a Macro

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)"
```

NOTE Menu definition macros do not have Name or EndMacro commands.

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 your *Commands* handbook.

NOTE Items which are added to original existing menu items of the ChemStation will be deleted during the initialization of the menu due to some actions within the ChemStation.

Using Single-line Dialog Boxes

You can write macros to ask for input and to assign given information to the specified variables. The Input function displays a single-line input box on the screen, allowing you to prompt the user for values and 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.

The macro ReadDataFile is an example for a simple load of *.SD files.

```
Name ReadDataFile

! Reads a spectral data file which name can be entered

  Local filename$
  Local OK; OK=1                 ! For better readability

    Button = Input(”Enter speactra data filename:”, fileName$)
    If Button = OK
      LoadObj filename$,,samples ! Loads data file in samples register
  else
    Stop                         ! Macro execution
  EndIf
  Return

endmacro                         ! ReadDataFile
```

Using Multiple-line Dialog Boxes

When you want to enter several items you can use a dialog box containing several input fields. Using dialog boxes helps you to create a more friendly user interface. The advantage of using multiple-line dialog boxes is that all of the information is present at the same time and only selected items can be changed individually.

As an example the following macro imports multiple *.WAV files at a time. The files have to have the same root name and an ascending index number. Valid sequences of files are:

dye001.wav through dye023.wav, or

dye1.wav through dye23.wav.

The leading zeros are identified by the entry in the start index field 001 or 1.

The macro defines a dialog box where you enter the name of a data file, the start and end digits of the file name and the destination register. For the input of the destination register a combo box is used.

```
Name DefineMultiWavDialog
 BeginDialog ”MultiWavDialog”, 93, 47, 187, 119,\
             ”Load *.WAV with ascending numbers”

   StaticText 10,19,66,9, ”Filename with path:”
   EditBox 77,18,101,12, rootname$

   StaticText 10,31,66,9, ”Startvalue:”
   EditBox 77,30,20,11,start$

   StaticText 10,43,66,9, ”Endvalue:”
   EditBox 77,42,20,11,end$

   StaticText 10,55,66,9, ”Register be loaded:”
   ComboBox 77,54,60,30,RegListVar$,Load_reg$,64

   OKButton 39,90,50,20,”Load them”
   CANCELButton 112,90,50,20, ”Cancel”
  EndDialog

EndMacro
```

To use this dialog box in a macro you have to call it as shown in the macro MultiWav:

```
Name MultiWav

! This macro loads *.WAV with ascending numbers
! e.g. DYE001.WAV to DYE018.WAV

  Local OK; OK=1

  start$="1"                          ! Defaults for dialog box
  end$="2"
  RegListVar$="Samples|Standards"
  Load_reg$="Samples"

  DefineMultiWavDialog
  If ShowDialog("MultiWavDialog") = OK Then
     RemoveDialog "MultiWavDialog"    ! Clear memory
     xxstart=VAL(start$)
     xxend=VAL(end$)
     sbdigits=LEN(start$)
     loadmultiwav rootname$,xxstart,xxend,Load_reg$,sbdigits
  Else
     RemoveDialog "MultiWavDialog"    ! Clear memory
  EndIf
Return
EndMacro

Name number$

! Generates a number as a string with leading zeros e.g. 008

Parameter value,length DEFAULT 3

 number$=VAL$(value)
 While Len(number$)<length           ! check for required length
   number$="0"+number$                 ! adds leading zeros to string
 EndWhile
 Return number$
EndMacro

Name loadmultiwav

! Subroutine to load the *.WAV files in the
! specified register (Samples or Standards)

Parameter rootname$,xxstart,xxend,ToReg$,sbdigits
```

```
Local i
 For i = xxstart to xxend
   filename$=rootname$+number$(i,sbdigits) ! build up the filename
   filename$=filename$+".WAV"              ! and add extension
   Print filename$                    ! Info on message line
   Evaluate "ImportSpectrum filename$,WAV,"+ToReg$
                                      !Evaluate instruction string
 Next i
EndMacro
```

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 seperate command to show and remove the dialog box.

The function of the OK and Cancel buttons are the same as with the Input function, but you can define the position, size and text of the buttons. The variables defined in the dialog box are automatically filled with the user's input.

5
Permanent Data

Files

In Chapter 3, “From Commands to Macros” we described how to load register objects. The LoadObj and SaveObj commands load 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 or quit the ChemStation software.

NOTE Remember to save important data (for example, a method) every time you make a 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 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 3, “From Commands to Macros”.

For example, to save the sample name of first spectrum to a file:

```
Open ”Smpname.txt” for output as #3
Print #3, ObjHdrText$(Samples[1],Samplename)
Close #3
```

For the Open command you need to specify a number as an identifier for a file, in this example 3. This file identifier must be unique. To check for a free file identifier you can use the Show command and look at the open devices, this means the already used file identifiers.

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 "Smpname.txt" for input as #3
Open "CopyData.txt" for output as #4
Input #3, Samplename$
Print #4, Samplename$
Close #3
Close #4
```

File Extensions

File names 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 Smpname.TXT to see what it contains.

To read the sample name that has been written to Smpname.TXT:

```
Open "Smpname.txt" for input as #3
Input #3, Samplename$
Close #3
Print "The sample name in the file ", Samplename$
```

ASCII text files are useful to export data to other applications. A common format for example is the *.CSV format, this means the values are comma separated. The example Store_Temp writes the actual temperature value of the external sensor of the 89090A Peltier temperature control unit every *wait* seconds *loop* times in the file filename$.

```
name Store_Temp

! Stores the Temperature of the external sensor in a ASCII file

parameter loop,wait,filename$

local z,time,starttime,temperature

open filename$ for output as #4
starttime=mtime()              ! Offset value for time scale
```

```
for z = 1 to loop                ! loop gives the number of data points
   time=mtime()-starttime     ! calculate relative time
   temperature = objhdrval(Temco_Status[1],ExtTemp)\
                              ! read external sensor
   time=floor(time/1000)      ! convert ms in seconds
   print time,"s  ",temperature       ! show on message line
   print #4,time,",",temperature      ! and write comma-separated to file
   sleep wait                 ! sleep the specified wait time
next z
close #4                         ! close file

endmacro
```

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 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 using #1, "#####.##", number
```

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

6
Registers and Tables

The ChemStation uses constructions called objects to handle data records such as measurement data and data matrices. This data includes numeric and string information as well as tables. The ChemStation uses registers to store and access these objects. The registers are global and the ChemStation accesses them using register names. The register names must be unique. The ChemStation accesses an object in a register using the register name and an object index. Chapter 12, “Variables” lists the predefined registers and your *Commands* handbook gives you the naming conventions for registers.

A register and its contents are only available when the application is running. If you want to use the contents of a register in several sessions, you must save and load them from disk. The configuration register and its contents is the only register saved automatically to disk. All other registers used by the ChemStation are created at the beginning but most of them are empty. Registers are created implicitly using commands which require a destination parameter. These registers are referred to as ToReg in the description of the commands syntax in your commands online help.

Register Commands

To avoid naming conflicts you use the RegName$ function to find the names of all existing registers. This function allows you to access the internal register list using an index.

The following macro example displays on the message line all register names of the current application. To get a better overview, the macro uses the ListMessages command to see more than one name on the display and it uses the Sleep command to slow the appearance of the register names on the command line.

```
ListMessages on              ! switch message display on
i = 1                        ! initialize register index
Repeat
  Register$ = RegName$(i)    ! get name
  Print Register$            ! display register name
  i = i + 1                  ! increment counter
  Sleep 1                    ! wait one second
Until (Register$ = "")         ! empty string is last element in list
```

You can find the number of objects in a register using the RegSize function. As an example the following macro determines the number of objects in the Samples register and deletes the last object.

```
x = RegSize(Samples)         ! get number of objects
If x > 0 then                ! any object ?
    DelObj Samples[x]        ! delete last
EndIf
```

You can also use the RegSize function to check for naming conflicts. This example checks whether a register called MyRegister exists.

```
x = RegSize(MyRegister)      ! check register
If x >= 0 then
  Print "Register MyRegister already exists"
Else
  Print "Register MyRegister does not exist"
EndIf
```

A register size of –1 indicates a register with the given name does not exist.

You use the DelReg command to delete a register. If you created a temporary register called Temp which you now want to remove, the command:

```
DelReg Temp                  ! remove register
```

deletes all objects in the register and removes the register from the register list.

You use the DelObj command to delete the objects in a register without removing the register from the register list. For example, the command:

```
DelObj Samples                      ! clear all samples
```

deletes all samples from the Samples register but *does not* remove the Samples register from the register list.

Object Commands

The ChemStation handles objects using object commands. When you want a macro to modify data, a good approach is to copy the objects to a working register. For example, if you want to manipulate the second data set in the Samples register, copy the data using the CopyObj command:

```
CopyObj Samples[2],Work          ! copy to working register
```

The command appends the second object of the Samples register to the objects in the Work register. The ChemStation creates the Work register automatically, if the register does not already exist.

You can also move objects between registers. For example, if you want to define the second and third sets of data in the Samples register as standards, you can move these objects using the MoveObj command:

```
MoveObj Samples[2:3],Standards    ! move spectra
```

This command removes the second and third objects from the Samples register and appendeds them to the Standards register. The MoveObj command is fast because it does not duplicate data, in contrast to the CopyObj command.

The Exchange command is in a similar category. This command allows you to swap the contents of two registers. As with the Move command its execution is fast, because data need not be copied. If the Samples_2 register contains a second set of samples and you want to use these samples in a calculation instead of the current samples, the command:

```
Exchange Samples, Samples_2       ! exchange register contents
```

swaps the two sets of sample data. To restore the original data execute the command a second time:

```
Exchange Samples, Samples_2       ! restore register
```

You use the SaveObj and LoadObj commands to save and load objects from a file. If you want to use the objects in the User register in several sessions of the ChemStation software, you must save them on the hard disk and then load them each time you start a new session. You save all objects of the User register in a file called MyData.Reg using the command:

```
SaveObj User,"MyData.Reg",New     ! store data to file
```

The mode keyword New makes sure the ChemStation overwrites the file if the file already exists. The default mode is Append. In the Append mode the ChemStation appends objects to existing files. Using the command:

```
SaveObj User,"MyData.Reg"          ! append data to file
```

could lead to unexpected file contents, if a data file with the same name already exists. In the example above where you have to load and save the contents of a register to the hard disk, the New mode is the correct choice. The ChemStation would overwrite an existing file with the latest data.

If you do not specify a path with the filename, the ChemStation uses the path saved with the system variable _ExePath$. If you want a different path, you must specify it with the filename. If the register contains spectral data, you can use the data path with the data extension. The system variable _DataPath$ contains the current data path. To save the data you would use the command:

```
SaveObj User,_DataPath$ + "MyData.sd",New     ! use system data path
```

Using the load file menu you can load data that you saved with the .SD extension. You use the LoadObj command to load register contents independent of naming conventions. You can load the contents of the User register using:

```
LoadObj "MyData.Reg",,User         ! get data from file
```

In this example you must clear the User register before you use the LoadObj command. If you do not clear the register, the ChemStation appends the objects to the register contents. The path handling is the same as with the SaveObj command.

You use the NewObj command to create new objects in a register. With this command you specify the object type and the dimensions. Your *Commands* handbook refers to the object type as ObjClass. You cannot change these characteristics afterwards. This means you can only use fixed data types and dimensions. Chapter 7, “Windows and Display Description Tables” gives examples and describes the internal object structure.

Objects

Objects contain data. The ChemStation has five predefined data types:

1	User
2	Matrix
3	GC chromatogram
4	HPLC chromatogram
5	UV spectrum

You can access register contents in terms of the data type using the RegCont$ function. The ChemStation gives you the number and the type of all the objects in a register. The abbreviations of the types are:

US	User Data
MAT	Matrices
GC	GC chromatograms
LC	HPLC chromatograms
UV	UV-visible spectra

The command:

```
Print RegCont$(Test)         ! display register contents
```

displays a list of all objects sorted in the different types.

Data Block Commands

The ChemStation structures primary data two-dimensionally in rows and columns. Object headers contain all additional information. You access these object headers using their item names. Object header items can contain three different data types: dynamic strings, numeric values, and tables. The first two data types are the same as the simple scalar and string variables of the command processor. The rules for naming object header items are similar to those for the command processor scalar variables except that the length is limited to 15 characters. You use access functions to access the data saved with an object.

Figure 2 on page 80 shows you an overview of the register structure.

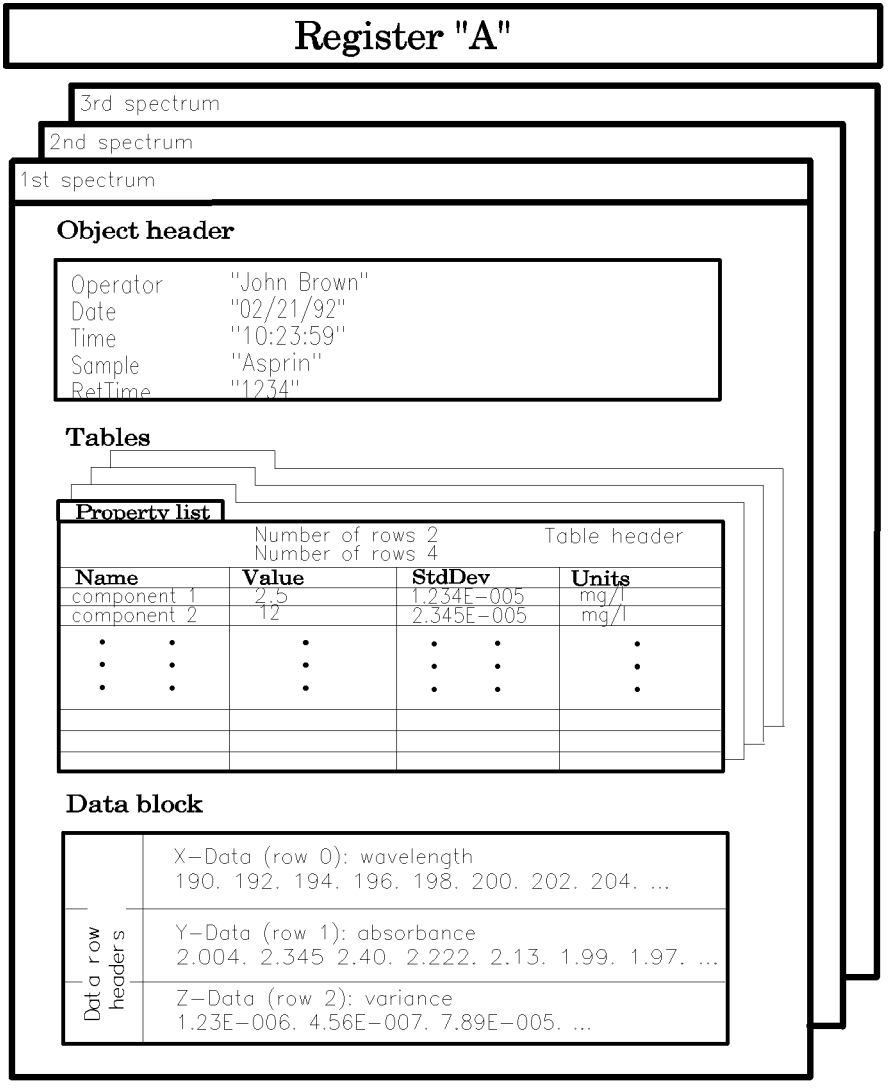


Figure 2 Register Structure

When you use the NewObj command to create a new data object, the ChemStation always creates two object headers with the predefined item names Title and ObjClass. The Title is a string item with no characters as default. The ObjClass is a scalar item containing the data type specification of the NewObj command used to create the object.

For a better overview it is good practice to assign meaningful titles to all objects. To set the title My Data to the object User[1], use the command:

```
SetObjHdrText User[1],"Title","My Data"      ! assign title to object
```

The ChemStation has functions which you use to get all the information included with an object. To access the item data correctly you must know their data type. The ObjHdrType function gives you the required information.

You use the data block commands to analyze the data section of an object. The functions DataCols and DataRows allow you to get the dimensions of the data matrix. You access the data using the Data function. To print all data of an unknown object User[2], use the following commands:

```
ListMessages on                                ! switch multi line display
on
rows = DataRows(User[2])                      ! get number of rows
cols = DataCols(User[2])                        ! get number of columns
for i = 1 to rows                             ! scan all rows
  for j = 1 to cols                           ! scan all columns
    Print Using "##.### ,", Data(User[2],i,j) ! print column data formatt
ed
  next j
    Print                                      ! display one line
next i
```

Depending on the structure of the data, a transposed form of the above example may give better results:

```
for j = 1 to cols                               ! scan columns first
  for i = 1 to rows                             ! scan all rows
    Print Using "##.### ,", Data(User[2],i,j) ! print row data formatted
  next i
Print                                           ! display one line
next j
```

You can set data points in objects in a similar way. The SetData command allows you to do this task. If you want to subtract a constant offset of 0.5 from all data points in row 1 of the object User[2], use the following commands:

```
cols = DataCols(User[2])             ! get number of columns
for i = 1 to cols                    ! scan all columns
  SetData User[2],1,i,Data(User[2],1,i) - 0.5  ! subtract offset
next i
```

If, as in the above example, you process measured data, the ChemStation sets a flag to indicate the modification. You use the DataModified function to get this information, for example:

```
x = DataModified(User[2])            ! get status
If x = 1 then                        ! check flag
  Print "Data User[2] are modified"
Else
  Print "Data User[2] are original"
EndIf
```

Additional commands allow you to handle data. If you want to use the same x-scale in several data objects, you can create this scale once and then copy it to the other data objects. For example, you create a quadratic scale in the data object User[1] and then copy it to the objects User[2] through User[4]:

```
For i = 1 to 4                       ! create all objects
  NewObj User[i],1,1,8               ! create all objects
next i
for i = 1 to 8                       ! 8 x-axis values
  SetData User[1],0,i,sqr(i)         ! set x-axis value
next i
for i = 2 to 4         ! set same x value to other data sets
  CopyDataRow User[1],0,,User[i],0   ! copy x-axis
next i
```

You use the GetDataMinMax command to get the minimum and maximum values of a data row. For the above scale the command:

```
GetDataMinMax User[3],0              ! get min/max values
```

gives these values in four global scalar variables called OBJ_MAX, OBJ_MIN, OBJ_X_MAX, and OBJ_X_MIN. These values are the minimum value (OBJ_MIN), the x-position of the minimum value (OBJ_X_MIN), and the maximum value (OBJ_MAX) with its x-position (OBJ_X_MAX). In the above example (the x-axis) the values and position are identical, but with measured data sets you get the corresponding x-axis values.

You use the DataIndex function to access data values in a data object using units. This allows you to, for example, access UV-visible data using wavelength instead of indices. If you want to access the absorbance value at 350 nm in a UV-Visible spectral data object, use the command:

```
Print Data (Samples[1],1,DataIndex(Samples[1],350))\
                                   ! print data at 350 nm
```

Object Header Commands

The data objects of the object header items contain additional information. The first step describes the two simple scalar and string items with their commands. Your *Commands* handbook summarizes these as object header commands.

You use the SetObjHdrVal command to save numeric values with an object. If you want to store a pH value with the data object User[3], use the command:

```
SetObjHdrVal User[3],"ph",2.5       ! store pH value
```

This command creates a new object header item with the item name pH and assigns the scalar value 2.5. The NewObjHdrVal command gives you a more sophisticated approach to creating a scalar object header item. With this command you can protect a header item and restrict the range of allowed values. In the example you can restrict pH values to the range 0-14 and protect the item against deletion. You create the item using the command:

```
NewObjHdrVal User[3],"ph",2.5,1,,0,14 ! create item to store pH value
```

You use the ObjHdrVal function to retrieve the numeric values saved with a data object. To access the pH values of four data objects User[1] through User[4] and print them on the message line, use the following commands:

```
for i = 1 to 4                      ! access four objects
  Print Using "User[#] : ",i        ! print index
  Print Using " ##.## pH ,",ObjHdrVal(User[i],"ph") ! print pH value
next i
Print                               ! display.
```

You can also use this function to read and display data from an instrument. If you want to read the current temperature of the external sensor of the 89090A Peltier temperature control accessory, use the command:

```
Print "Current temperature: ",ObjHdrVal(Temco_Status[1],"ExtTemp")
```

A prerequisite is that the temperature controller is online with the external sensor connected. The ChemStation updates this object header item continuously and you can therefore also monitor temperature changes.

You use text object header items to add text information to data objects. For example, to add the chemical formula of benzene to the third user object, use the command:

```
SetObjHdrText User[3],"Formula","C6H6"          ! store formula
```

If you want to restrict the length of the text to 20 characters and protect the formula against deletion, use the command:

```
NewObjHdrText User[3],"Formula","C6H6",1,20      ! restrict length
```

Limiting the length of the text is useful if you want to use the text in a table.

You use the string function ObjHdrText$ to get text information. To print the previously-saved chemical formula on the message line, use the command:

```
Print "Formula: ",ObjHdrText$(User[3],"Formula") ! display formula
```

You may delete object header items depending on their protection mode. The DelObjHdr command removes an item. If you do not protect your chemical formula against deletion, the command:

```
DelObjHdr User[3],"Formula"       ! remove formual item
```

deletes the item from the data set User[3].

As described above the ChemStation creates the two object header items Title and ObjClass automatically. You can access the ObjClass but you cannot remove or change it. The ChemStation also protects the Title item against deletion but allows you to set it. You use this possibility to identify data.

You use the ObjHdrName$ function to scan an unknown data object for existing object header items. To scan the object User[3]:

```
i = 1                             ! initialize start
repeat                            ! check all
  A$ = ObjHdrName$(User[3],i)     ! get name
  If x > 0 then                   ! available?
    Print "Item : ", A$           ! yes, display name
  EndIf
Until (x=0)
```

You use the ObjHdrType command to find out the data type of an unknown object. If the variable A$ contains the item name of interest in the object User[3], use the command:

```
Print ObjHdrType(User[3],A$)      ! get item type
```

to get the data type.

The ChemStation supports the types:

0 string

1 scalar

4 table

This allows you to access items of which you know only the name.

Tables

Tables are another type of object header items. In contrast to the previous simple data types, tables are more complex and can contain much data: text, as well as numeric values.

The ChemStation organizes tables in columns and rows. You can change dynamically both the number of columns and the number of rows. You access rows using an index. Indices range from 1 to the last row. The special index 0 allows you to access default values for the columns. The index –1 is equivalent to the last row. You access columns using a name. You must use unique names for the columns in a table and follow the same naming rules as for object header item names: you may use a maximum of 16 alphabetic or numeric characters. Columns can contain numeric values as well as strings. You can fix the length of a string element, with up to 255 characters, or it can be dynamic.

You can protect columns or the complete table and you can set defaults for each column. In addition, you can set lowest and highest allowed numbers for numeric values, together with the internal representation as double or single precision floating point or long integer values.

You can copy complete table columns to other tables and you can reorganize table rows in the same table.

You can read from or write to table elements providing you have not protected them. Special commands also allow key access to the elements of the specified column.

In addition to tabular data you can save information in so-called table header items. The ChemStation can handle two data types, scalar and string values, in the same way as object header items.

Table Commands

The ChemStation has several commands for you to handle tables. You use the NewTab command to create a new table in an existing object. You have two ways to create a table. The easy way is to use the structure of an existing table. Alternatively you can build a table from scratch. With the first approach you must specify the table you want to copy. For example, a table TabletWeights in the object User[1] exists and you want to create another table with Weights in the object User[2]. Use the command:

```
NewTab User[2],"Weights",User[1],"TabletWeights" ! use template
```

The object User[2] must already exist for the command to run without error.

To build a table from scratch, your first step is to create the table. To build a table called Results, as part of the object User[1], and containing sample names and corresponding concentration values, use the command:

```
NewTab User[1],"Resluts"              ! create table
```

If, as in the example, you misspelled the table name, you can correct your mistake using the command:

```
RenTab User[1],,"Resluts","Results"   ! rename table
```

You use the DelTab command to delete a table from an object. If you want to remove the Weights table from object User[2], use the command:

```
DelTab User[2],"Weights"              ! remove table
```

You can also copy a complete table. You use the CopyTab command to copy a table and its contents instead of creating a new table and only copying its structure. If you want to copy the table TabletWeights in the object User[2] to the table Weights, use the command:

```
CopyTab User[1],"TabletWeights",User[2],"Weights" ! copy table
```

Table Column Commands

The next step in creating a table from scratch is to specify its structure. You do this mainly in terms of columns. In the example you have to create two columns: one column for sample names and one column for a scalar result.

First you create the text column with the sample names. For example, to create a column called SampleNames, limit the sample names to 15 characters, set the default text to <no name>, and not restrict column access, use the command:

```
NewColText User[1],"Results","SampleName","<no name>",0,15\
                                          ! create column
```

If you now want to add a second numeric column named Concentration, allowing the range of concentration values of 0 through 200, use the command:

```
NewColVal User[1],"Results","Concentration",,,,0,200 ! create column
```

You can also create a column and its contents in a table by copying a complete table column from one table to another table. If, as in the previous example, you want to add a column with lot numbers of samples and a table including these lot numbers exists. You must copy the column containing the lot numbers of this table called TabletWeights in the same object to the Results table:

```
CopyTabCol User[1],"TabletWeights","LotNumber",User[1],"Results",\
           "LotNumber"
```

A fixed row relationship of the two tables must exist in terms of sample names. In other words every sample has to be in the same row in both tables. Otherwise a copy would not make sense.

You can also remove a complete column using the DelTabCol command. If, in the example, you no longer need the sample names, you can delete the sample names column using the command:

```
DelTabCol User[1],"Results","SampleNames" ! remove column
```

You use the two table functions TabColName$ and TabColType to scan the column structure of a table. For example, to display on the message line all column names of the Window table in the object _Config[1] use the commands:

```
i= 1                                              ! initialize counter

Repeat
  Column$ = TabColName$(_Config[1],"Window",i) ! get column name
  Print Column$                                ! display
  i = i + 1                                       ! increment counter
  Sleep 1                                      ! display 1 second
until (Column$ = "")                           ! exit if no name
```

The TabColType function returns the type of column to allow you to access the column content properly. In addition you can also use the TabColType function used to check for potential naming conflicts when you want to add a column to a table. If you want to add a new column with the name MyColumn to the Window table of the object _Config[1], the function TabColType must return a value of 99. The command:

```
Print TabColType(_Config[1],"Window","MyColumn") ! check column
```

displays 99 on the message line, indicating a column with the name MyColumn does not exist in the Window table.

Table Row Commands

The ChemStation has a series of commands to handle rows in tables. You use an index to access single rows. In addition you can use row ranges with all table row commands. This simplifies access to a table block. You can access all rows of a table using the range 1:–1. The range limit –1 represents the last table row.

If you have a table with many columns and only a few entries vary in a specific section, the quickest way to set these variable table elements at the macro level is to copy the specific section and modify the few entries afterwards.

For example, the rows 11 through 20 in the table ManyColumns are similar to rows 1 through 10. The quickest way to duplicate the information is to copy rows 1 through 10 to rows 11 through 20 using the command:

```
CopyTabRow User[3],"ManyColumns",1:10,11
```

To set the specific information in these rows you must set the column element in a loop.

A prerequisite to entering elements in a table as well as to copying rows is that the addressed rows already exist. To dimension a table in terms of rows you use the InsTabRow command. If you do not specify a range, this command adds one row at the end of the table and sets all column entries to their initial values. This use of the command is a good approach when you want to extend a table row-by-row.

You can add a new sample to a sample table by first inserting a row first and then making the appropriate entries. If you want to extend the sample table SampleTable in the first object of the User register, the command:

```
InsTabRow User[3],"SampleTable"   ! add new row
```

adds a new row at the end of the table with initial values for all columns.

Another application is a table of fixed size. Specifying a row range allows you to create the complete table block with a single command. For example, you can create a weight table with six rows using the command:

```
InsTabRow User[2],"Weights",1:6   ! create table block
```

In general, you can use the InsTabRow command to insert one or more rows in a table at a given row position. If this row is not the last row, the command changes the row indices of all rows that follow the inserted row. When you delete rows the ChemStation also changes row indices. You use the DelTabRow command to delete a range of rows. The rows you want to delete must exist.

You use the ExchangeTabRow command to swap sections of tables. This is useful when you want to reorganize a table. An application of this command is alphabetic sorting of table rows using one column as a key. "Table Element Access" on page 89 gives an example of this command.

Table Element Access

You use a row index and column name to access a table element directly. These two parameters identify a table element uniquely. Dependent on the column type, you can access text or numeric values. You use the function TabText$ to access text and the function TabVal to acces numeric values.

You can also access table rows using key elements. Here you can use numeric as well as string. When accessing elements you can specify a condition the function should test. These conditions allow you to access specified values, for example:

- minimum and maximum values
- smaller and greater values
- proximity conditions
- case sensitive and case insensitive string comparison
- substring operators.

You use the RowByVal function for numeric columns and the RowByText$ function for string columns. These functions return the row index of the first row where the condition is fulfilled. If you want to find the row with the sample lot number 12389, the command:

```
Print RowByVal(User[1],"Results",,"LotNumber","=",12389)
```

displays the row number of specified lot.

If the function finds the lot number in the table, the ChemStation displays the row index on the message line. If the function does not find the lot number, it returns –1. If the function found the lot number and returned an index of 198, you can use the TabText$ function to display the corresponding sample name:

```
Print TabText$(User[1],"Results",198) ! get sample name
```

A useful application in a results table is to find all samples above a certain limit. In the following example the ChemStation searches all samples for an upper concentration limit of 2.5. After successfully finding a sample, the ChemStation continues searching at the next row. The message line displays the samples using their lot numbers.

```
i = 1                              ! initialize start index

repeat
  x = RowByVal(User[1],"Results",i:-1,"Concentration",">",2.5)
                                   ! get row index
  If x > 0 then                    ! condition fulfilled?
    print using "########, ",TabVal(User[1],"Results",x,"LotNumber")
                                   ! prepare for display
    i = x + 1                      ! continue with next row
  EndIf
until (x = -1)                     ! repeat until nothing found

Print                              ! display all in one line
```

Using a similar technique you can search for a specific keyword in a text column. If the keyword is not unique, you can find all keywords by changing the row range for the next search. To find all samples with the name Benzene (irrespective of letter case) in the Results table and to display the row indices on the message line, use the commands:

```
i = 1                              ! initialize start index

repeat
  x = RowByText(User[1],"Results",i:-1,"SampleName","i=","Benzene")
                                   ! get row index
```

```
  If x > 0 then                    ! any row found?
    print using "###### ,",x       ! display row number
    i = x + 1                      ! set search start
  EndIf
until (x = -1)
Print
```

If you want to sort alphabetically the Results table in the object User[1] according to the sample names in the column SampleName and you assume the table length is 100 rows, use the commands:

```
i = 1                                    ! initialize row index
repeat
  A$ = TabText$(User[1],"Results",i,"SampleName")
                                         ! get sample name
  x = RowByText(User[1],"Results",i+1:-1,"SampleName","%<",A$)
                                         ! any name should be before?
  If x > 0 then                          ! check index
    ExchangeTabRow User[1],"Results",i,x ! yes, swap
  Else                                   ! no
    i = i + 1                            ! check next sample name
  EndIf
until (i = 100)                          ! continue until last row
```

Table Header Commands

In addition to the table elements, you can save information belonging to a table in the table header items. They are similar to object header items, but the ChemStation only supports two types, numeric and string table header items. The set of commands is similar to the set you use to handle object header items. One major difference is that you must specify a table in addition to the object when you want to address these header items.

When you create a new table, the ChemStation creates four predefined header items automatically. These items include information about the size of the table and its status. The table header item NumberOfRows is a numeric item containing the actual number of rows. The item NumberOfCol is the number of columns in the table. The item NumberOfHead is the total number of table header items. The ChemStation sets the item Modified to 1 the first time you put information in the table.

You use these header items to analyze the content of an unknown table. You display the names of all header items on the message line using the commands:

```
for i = 1 to TabHdrVal(User[1],"Results","NumberOfHead")
  print TabHdrName$(User[1],"Results",i) ! display name
  sleep 1
next i
```

You use the TabHdrType function to access the type of a column of known name. This allows you to use the correct access function for the column elements. A refinement of the above example is to display the items names with their contents. You can do this using the commands:

```
for i = 1 to TabHdrVal(User[1],"Results","NumberOfHead") ! scan all
                                                  ! table header items
  L$ = TabHdrName$(User[1],"Results",i)        ! get item name
  x = TabHdrType(User[1],"Results",L$)         ! get type of item
  If x > 0 then                                ! numeric?
    C$ = Val$(TabHdrVal(User[1],"Results",L$)) ! yes, get value
  Else
    C$ = TabHdrText$(User[1],"Results",L$)     ! no, get string
  EndIf
  Print L$ + " : " + C$                        ! display
  sleep 1                                      ! wait one second
next i
```

If you want to add global information to a table, the best way is to use the table header items. You can add an operator name and a time stamp to a table using the commands:

```
SetTabHdrText User[1],"Results","Operator","Thomas Klink" ! add name
SetTabHdrText User[1],"Results","LastUpDate",Time$()
                                          ! add time of last change
```

Arithmetic Commands

Arithmetic object commands allow you to do mathematical operations on complete objects. The operations include addition, subtraction, multiplication, and comparison of objects. The commands handle entire objects and not just single data points. This includes automatic interpolation of y-data, if the x-axis values do not match, as well as handling standard deviations for UV-visible spectra. If the result of the operation is a new object, the ChemStation automatically copies all object header items to the result object. You might use some of the arithmetic commands with registers or object ranges. The purpose of the arithmetic commands is to handle spectral data sets easily.

For example, you can use the SubObj command to subtract a constant spectral background from all spectra. If you want to correct all data objects in the Samples register with a background saved in the first object of the Background register, use the command:

```
SubObj Samples,Background[1] ! subtract spectrum
```

This command corrects all spectra for background and the result replaces the original data in the Samples register. If you want to keep the raw data in the Samples register, you can specify a destination register:

```
SubObj Samples,Background[1],Samples_corrected
                             ! subtract spectra, keep raw data
```

If you do not specify the destination, you can restore the original data by adding the background to the result data using the command:

```
AddObj Samples,Background[1] ! restore spectra
```

The only difference between this approach and specifying a destination register to keep the original data is that the ChemStation now marks the resaved data with a flag indicating you have modified the data.

You use the CompareObj to check the similarity between two data objects. This command calculates a linear regression between the two data objects and gives a match factor reflecting the spectral similarity of the two data objects. If you want to compare two sample spectra in data objects 3 and 5 and display the match factor on the message line, use the commands:

```
CompareObj Samples[3],Samples[5]  ! compare spectra
Print Match                       ! display result
```

As a rule of thumb a match factor greater than 990 indicates identical spectra, factors between 950 and 990 indicate similar spectra, and factors lower than 950 indicate different objects. You use the DerivObj command to calculate derivatives of spectral objects. If you want to calculate the derivative of all samples, use the command:

```
DerivObj Samples                    ! calculate derivative
```

The result is the derivative of all objects in the Samples register. You can also use this command to smooth data. Additional, optional parameters allow you to specify the filter length, the order, and the polynomial degree. The derivative calculation is based on a moving average. The number of points used in the average calculation must be odd and greater than the specified polynomial degree. The calculation of the smoothed and derivative data points is based on a least squares polynomial calculated in the filter interval. The default polynomial degree is 3. This is a good compromise to model spectral data. Polynomials of lower degree cannot model inflection points. Depending on the bandwidth of the absorbance bands, a too low polynomial degree or a too long filter length distorts the spectra. If you are unsure about the parameter settings for the derivative calculation, copy the spectra to a test register, calculate the smoothed data, and then compare the raw data with the smoothed data:

```
DelReg Test                         ! remove register
CopyObj Samples,Test                ! copy data
DerivObj Test,,0                    ! smooth data
for i = 1 to RegSize(Samples)       ! compare all spectra
  CompareObj Samples[i],Test[i]     ! compare smoothed/raw data
  print Match                       ! display result
next i
```

The match factor should indicate identical spectra (>990). You can repeat this sequence of commands using different filter lengths until you get acceptable match factors. You can then calculate derivatives using optimal parameters. In the above example you do not have to specify the order because the default order is one. When you calculate a derivative you must keep the original data (if needed) because the reverse operation (integration) is not possible.

You can calculate the ratio of spectral objects at each wavelength using the following commands:

```
DelReg Test                         ! remove register
DelReg Ratio                        ! remove register
CopyObj Samples[1],Test             ! copy first object
RecipObj Test[1]                    ! calculate reciprocal
MultObj Samples,Test[1],Ratio       ! ratio all samples
```

These commands calculate the absorbance ratios at each wavelength of all objects in the Samples register based on the first object. You can use spectral ratios for a set of standards of single components to find spectral ranges achieving good linearity. Ratioing uses the commands RecipObj for the reciprocal calculation RecipObj and MultObj for multiplication. The ChemStation saves the results in the Ratio register.

You use the cubic spline function SplineObj to interpolate data points. You use the step parameter to set the number of intervals between the data points. A step setting of 5 inserts 4 data points between two measured data points. To interpolate all sample data with a step of 5 use the command:

```
SplineObj Samples,5              ! interpolate
```

The command increases the total number of data points by a factor of 5. Another application of this command is to get closer to a true maximum in an absorbance spectrum. If the true maximum is between two measured data points the interpolation allows you to get a better approximation of the wavelength position as well as the true height.

The FindObjMinMax command creates two tables called MinTable and MaxTable in an object. These tables contain positions and heights of the peaks and valleys found. For absorbance spectra the command saves the wavelengths with the appropriate absorbance values in the tables. As with the DerivObj command, you can set a filter length parameter to adjust for sensitivity. To find the minima and maxima in the third samples spectrum, use the command:

```
FindObjMinMax Samples[3]         ! get min/max
```

In the third object of Samples, the command creates or updates the two tables (if they already exist). If the ChemStation does not find a minimum or maximum, the respective table is empty. The number of rows in the tables correspond to the number of minima and maxima found. You can display on the message line the number of absorbance maxima found using the command:

```
Print TabHdrVal(Samples[3],"MaxTable","NumberOfRows")
                                 ! get no. of peaks found
```

7 Windows and Display Description Tables

You use windows to display registers, objects, tables, and object header items. With the ChemStation you can display multiple windows. A set of windows is called a **view**. These views are a collection of windows relating to a certain context. The ChemStation has predefined views to display samples, standards, analysis, and calibration results. You can open additional windows to display further information.

The ChemStation has three types of windows:

- graphic windows
- table windows
- monitor windows.

You can display graphicly only data objects. Monitor windows have predefined window numbers and you cannot manipulate the contents. The only possibility you have for this type of window is to switch the window on and off. A table called Window in the first object of the _Config register contains the properties of the windows, for example:

- size
- position relative to the main menu
- graphic styles

- axis
- colors
- actions to be done on mouse clicks
- title
- command used to create the window

Chapter 14, “Registers” gives details of the structure of the Window table as well as for the two associated tables, the AxisStyle table and the Color table, in the same object. A row index relates these two tables to the Window table. Changes you make to these tables may cause to ChemStation to stop working correctly. It is therefore important you do the required changes carefully.

Window Commands

The ChemStation has window commands to allow you to handle windows easily. You access the windows using window numbers. The window number is the row index of the window table. Window numbers 1 to 10 are reserved for general purposes.

You use the FreeWin function to handle window numbers. The FreeWin function returns the next unused window number. To check a user window is available and display the result, use the commands:

```
x = FreeWin(1)                          ! get free window number
if x > 0 and x <10 then                 ! inside user window range
  Print "User window : ",x," available"
Else
  Print "No user window available"
EndIf
```

When you start a macro from the menu you use the ActiveWindow function to access the number of the window that is currently active. This is useful if general purpose macros, which can work on different windows, can find the window you want the macro to work on. To demonstrate this command you must make a menu entry otherwise the command line is always the active part of the screen. Therefore the following command includes a menu entry:

```
MenuAdd "User","Display active window number",\
"Print ""Active window : "",ActiveWindow()"
```

The ClearWin command is another general window command. If you do not specify a parameter, the command clears all windows. If you specify a parameter, the command clears the specified window. To delete the active window, use the following menu command:

```
MenuAdd "User","Delete active window",\
"x=ActiveWindow();if x >0 then;ClearWin x;EndIf"
```

You use the SetWinTitle command to set the titles in a window. If a user window with window number 2 exists, you display the title My Window in that window using the command:

```
SetWinTitle 2,"My Window"               ! set window title
```

Monitoring Windows

The ChemStation has predefined monitoring windows which allow you to display the status of the software and configured instruments. The available windows depend on the configuration of your system. You use the SystemStatus command to display the status of the system. You use the MonSpectrometerStatus and MonTemcoStatus commands to display status of the instruments. These commands allow you to switch the associated display on and off.

Graphics Commands

You use the Draw command to display data objects graphicly. To draw all the samples contained in window number 1, use the command:

```
Draw 1,Samples                          ! display spectra
```

This command uses automatic scaling for the minima and maxima of all spectra on the x- and y-axes. Alternatively you can specify a fixed scale using additional optional parameters. For spectra you can also display standard deviations and you can specify to separate or overlay multiple spectra. To display separated all samples and standard deviations on an absorbance scale from 0 to 1 and a wavelength range of 190 to 820 nm, use the command:

```
Draw 1,Samples,190:820,0:1,,1,1         ! display with fixed scale
```

In a window with overlaid spectra you use the mouse to select one of the spectra. If a window contains many spectra and you cannot see details because other spectra interfere with the spectrum you selected. You can create a menu item in the user menu to draw the selected spectrum or spectra in a separate window. This task requires several commands so you must create a macro called DispSelectedSpectra as follows:

```
Name DispSelectedSpectra

  Local h,x
  h = ActiveWindow()                    ! get window selection

  If h < 1 then                         ! any selected
    h = Alert("No graphics window selected") ! display warning
  Else
    x = SelectSize(h)       ! get number of selection in window
  If x = 0 then                         ! none?
    h = Alert("No graphic selection made") ! display warning
  Else
    DelReg Disp                         ! remove register
    for i = 1 to x
      GetSelect h,i                     ! get selection
      Evaluate "CopyObj " + Sel_RegObj$ + ",Disp"
                                        ! copy selected object
     Next i
     Draw 2,Disp                        ! display selected spectra
    EndIf
  EndIf

EndMacro
```

```
MenuAdd "User","Display selected spectra","DispSelectedSpectra"
                                          ! add menu item to user menu
```

This macro allows you to use the mouse in a graphic window to select one or more spectra. The user menu item Display Selected Spectra allows you to display the selected spectra full scale in a separate window. This works for overlaid and zoomed spectra where you see only a small section of the spectrum. In the example above you use the SelectSize function to access the size of the selection table. The ChemStation copies the selected objects to a new register called Disp and then displays the objects in a separate window.

The selection mechanism also allows you to add selections to a selection table. For spectra displayed in a graphic window this command allows you to use a macro to set selection indicators to one or more spectra. If you want to mark all samples in window 1 with the sample name Benzene, use the following commands:

```
ClearSelect 1                           ! remove all selections
for i = 1 to RegSize(Samples)           ! all samples
  If ObjHdrText$(Samples[i],"SampleName") = "Benzene" then
                                        ! benzene sample?
    AppendSelect 1,Samples[i],0         ! yes, set selction
  EndIf
next i
```

A window gives you a way to look at current data. The ChemStation has a mechanism to update the displayed data automatically, if you make changes to the data. For example, you manipulate a single data point in one of the objects displayed in window 1. If window 1 displays the samples spectra, you can add a spike to the spectrum of sample 1 using the command:

```
SetData Samples[1],1,20,Data(Samples[1],1,20) + 1
                                      ! change data value
```

The ChemStation updates the window displaying the samples data instantaneously. This example demonstrates well the manipulation of a single data point but it slows the execution of spectral manipulation in a loop because the ChemStation updates the graphic display each time you change a data point. You use the WinUpDate command to control this process. To switch off the update, use the command:

```
WinUpDate off                           ! switch automatic update off
```

If you now reset the manipulated spectrum using the command:

```
SetData Samples[1],1,20,Data(Samples[1],1,20) - 1
                                      ! reset data value
```

Nothing happens to the window displaying the data of the samples. You reactivate the update process using the command:

```
WinUpDate on                          ! switch automatic update on
```

After execution of the command the spike on the first sample spectrum disappears.

You use the Zoom command to rescale graphic windows. You can specify two ranges for the x- and y-axes. If you want to compare samples in a range of 0 to 0.5 absorbance units and in a wavelength range of 190 to 400 nm, use the command:

```
Zoom 1,190:400,0:0.5                  ! redraw with fixed scale
```

You can direct the result of a draw command to a file by changing the destination specification of the window used with the draw command. The ChemStation creates a windows meta file. You can use this file to export graphics to other windows applications such as a word processor, a spreadsheet, or graphics programs. To generate a meta file called Plot_1, use the following commands:

```
D$ = TabText$(_Config[1],"WINDOW",1,"Destination") ! get destination
SetTabText _Config[1],"WINDOW",1,"Destination",_ReportPath$ + "Plot_1"
                                      ! set file as destination
Draw 1,Samples                        ! generate graph to file
SetTabText _Config[1],"WINDOW",1,"Destination",D$
                                      ! restore destination
```

The first command gets the original destination and saves it in the string variable D$. The next command sets a new destination specification, the filename Plot_1 and the reports path, in the windows table for window number 1. The Draw command now generates the meta file output. The last command restores the original destination.

You use the MFPrint command to send a windows metafile to a printer. If you want to send the graphics generated previously to the printer specified in the printer control dialog, use the command:

```
MFPrint _ReportPath$ + "Plot_1"       ! print plot file
```

If you do not make any changes to the printer, the ChemStation uses the default printer configured in the windows operating environment.

You specify the size of a drawing using coordinates relative to the page size. The coordinates are called DefWXLow, DefWXHigh, DefWYLow, and DefWYHigh and the ChemStation saves their values in the windows table. The coordinates range from 0 to 1. A full page drawing has the coordinates 0,1,0,1.

You set the coordinates using the commands:

```
SetTabVal _Config[1],"WINDOW",1,"DefWXLow",0     ! set full page
SetTabVal _Config[1],"WINDOW",1,"DefWXHigh",1
SetTabVal _Config[1],"WINDOW",1,"DefWYLow",0
SetTabVal _Config[1],"WINDOW",1,"DefWYHigh",1
```

The following macro gives you an example of how to use the graphics export capability. By means of a user menu you can generate a windows meta file containing a full-page drawing of the selected graphics window.

```
Name GenerateMetafile

  Local h,x,t
  Local C$,D$,F$,P$

  h = ActiveWindow()                    ! get window with focus

  If h = 0 then
    h = Alert("No Selection made")      ! no focus set
  Else
    x = TabVal(_Config[1],"Window",h,"Type") ! graphics window?
    If x <> 1 then
      h = Alert("Not a graphics window")     ! no, warning
    Else
      P$ = _ReportPath$                   ! use report path as default
      If SelectFile(3,"Store Graphics to File ...","*.WMF",P$,F$,t)\
                  > 0 then              ! get path and file name
        If t <> 2 then                  ! ok?
          InsTabRow _Config[1],"WINDOW" ! create new window
          CopyTabRow _Config[1],"WINDOW",h,TabHdrVal(_Config[1],\
                      "WINDOW","NumberOfRows")
                                        ! copy window information
          C$ = TabText$(_Config[1],"WINDOW",h,"Command")
                                        ! get command to create
          SetTabText _Config[1],"WINDOW",h,"Destination",P$ + F$
                                        ! set file destination
          SetTabVal _Config[1],"WINDOW",h,"DefWXLow",0
                                        ! set full page
          SetTabVal _Config[1],"WINDOW",h,"DefWXHigh",1
          SetTabVal _Config[1],"WINDOW",h,"DefWYLow",0
          SetTabVal _Config[1],"WINDOW",h,"DefWYHigh",1
          Evaluate C$                   ! store to file
          CopyTabRow _Config[1],"WINDOW",TabHdrVal(_Config[1],\
             "WINDOW","NumberOfRows"),h  ! restore window information
          DelTabRow _Config[1],"WINDOW",TabHdrVal(_Config[1],\
            "WINDOW","NumberOfRows")    ! delete last window
          Evaluate C$                   ! restore window
          SetActiveWindow h             ! reset focus to window
        Else
          h = Alert("Not a file : "+ F$) ! warning, not a file
```

```
        EndIf
      EndIf
    EndIf
  EndIf

EndMacro

MenuAdd "User","Store graphics to file","GenerateMetafile"
                                        ! add to user menu
```

The following macro helps you to send windows meta files to a printer.

```
Name PrintMetafile

  Local h,x,t
  Local F$,P$

  P$ = _ReportPath$                      ! use report path as default
  If SelectFile(1,"Print Graphics Metafile ...","*.WMF",P$,F$,t)\
     > 0 then                            ! select path and file
    If t = 1 then                        ! selection ok?
      MFPrint P$ + F$                    ! send file to printer
    Else
      h = Alert("Not a file : "+ F$)     ! invalid selection
    EndIf
  EndIf

EndMacro

MenuAdd "User","Print Graphics Metafile","PrintMetafile"
                                        ! add to user menu
```

This macro allows you to select a meta file using a file selector box and sends the graphics saved with that file to a printer. Adding the user menu to the macro simplifies access to the macro.

Tabular Editors

Another type of window allows you to display and manipulate data in a tabular form. Three table editors allow you to access object header data across all objects in a register, data of a single object, and tables inside objects. These tabular windows are excellent tools for handling larger sets of data. As with graphic windows the ChemStation updates the display when you change the underlying data and if manipulation of data is allowed, you can change the data as well. This mechanism makes sure the ChemStation presents consistently all data on the screen.

You can use the same mechanism for the tabular represenatation of data as in the example where the SetData command manipulated a single data point and the ChemStation changed the graphic representation simultaneously. You can now display in window 1 the data of the first sample in a tabular form using the command:

```
EdDataTab 1, Samples[1]
```

The command displays the data with the x-axis data in column 0. In relation to the primary data, the ChemStation displays spectral data in a transposed form. This means that column 0 is row 0 of the primary data. Using the same manipulation as with the graphics display, the command:

```
SetData Samples[1],1,20,Data(Samples[1],1,20) + 1
```

adds a spike to the spectrum of sample 1 and updates the displayed value automatically. The absorbance data is in row 20 of the table.

You can select rows in a table by using the mouse. Move the pointer to the left of the table and the arrow pointer changes to a two-headed arrow pointer. You can select a single row, multiple rows, or a range of rows. You use the Edit menu to copy the selected table rows to the Windows Clipboard. This is the simplest way to transfer tabular data partially or totally to other applications such as word processors or spreadsheets.

You can also select table fields in a table. When you select a table field, its background color changes. You can select single fields only.

In the previous example the table window does not allow you to manipulate any data fields. To change data you must create a specification table called a **display description table** and adjust the control entries.

Display Description Tables

You use a display description table (DDT) to customize the look of a table. Chapter 14, “Registers” describes the structure of this table. The DDT is a tool for arranging tables on the screen. In addition you can set predefined table header items to create buttons in table windows and to react on events such as selections, mouse actions, and editing operations. This makes the tabular editors combined with DDTs a very powerful tool for customizing operation of your ChemStation. The first example showed that you must not use a DDT with tables. However, you must use DDTs if you want to change data, display only part of the data, or change column headers and field widths.

The relationship between data and the corresponding DDT is that you access every displayed column using a source string as descriptor. This is true for all types of table editors. DDTs are tables and they must be in the third object of the configuration register _Config. This allows you to use DDTs without the need to specify the register object that contains them.

The next example uses the EditSpectrum DDT to allow you to edit the samples data. To create the EditSpectrum DDT you copy the DDT template called DDT to the register object _Config[2]:

```
CopyTab _Config[2],”DDT”,_Config[3],”EditSpectrum”
```

This command copies the structure to the DDT object _Config[3] and allows you to specify the data you want to display.

To create a column with wavelength values and a second column with absorbance values, you must create two rows in the DDT using the command:

```
InsTabRow _Config[3],”EditSpectrum”,1:2
```

You use the index 0 as source descriptor to access the x-axis of the data and you use the index 1 to access the absorbance data. The first column of the displayed table shows the wavelength and the second column gives the values. Therefore the first row of the DDT describes the wavelength column. The columns of the DDT you must specify are:

- source
- column title
- column width
- format.

You arrange the first column with the wavelength using the commands:

```
SetTabText _Config[3],"EditSpectrum",1,"Source","0"
SetTabText _Config[3],"EditSpectrum",1,"Title","nm"
SetTabText _Config[3],"EditSpectrum",1,"Format","%3.0f"
SetTabVal  _Config[3],"EditSpectrum",1,"Width",6
```

You add the column with the absorbance values using the commands:

```
SetTabText _Config[3],"EditSpectrum",2,"Source","1"
SetTabText _Config[3],"EditSpectrum",2,"Title","Au"
SetTabText _Config[3],"EditSpectrum",2,"Format","%2.4f"
SetTabVal  _Config[3],"EditSpectrum",2,"Width",8
```

You display the table in window 2 using the command:

```
EdDataTab 2,Samples[1],"EditSpectrum"
```

If another window displays sample spectra, you can see in the graphic window the changes you made to the data using the table editor. To close the window choose Close from the Control menu. In addition to the field entries in the DDT you can use table header items to create an additional index column, which is not part of the table data, and buttons, which when active allow you to run associated macros.

An example is a series of small macros and commands which allow you to select individually sample and standard spectra using the object table editor. To simplify the access to the macro you make entries in the File menu:

```
MenuAdd "&File|Load (&selected)"
MenuAdd "&File|Load (&selected)","&Samples...","GetDataFile Samples",\
        ,"Load spectrum file to SAMPLES register"
MenuAdd "&File|Load (&selected)","&Standards...",\
         "GetDataFile Standards",,
         "Load spectrum file to STANDARDS register"
```

The macro GetDataFile uses a destination register name as parameter. The macro specifies as destinations the Samples and Standards registers with the appropriate menu entries.

```
Name GetDataFile

  Parameter RegName$                  ! get destination register name

  Local F$,P$
  Local type

  Register$ RegName$                ! assign to global variable
  P$ = _DataPath$                     ! get current data path
  If SelectFile(1,"Get Data Selected ...","*.sd",P$,F$,Type) = 2 then\
                                      ! display file selection box
    LoadObj P$ + F$,,TEMP             ! load all data
    SetTabText _Config[1], "WINDOW", 1, "WinTitle",\
```

```
            "Select Spectra ..."   ! set window title
   EdObjTab 1,TEMP,"DDT_DataSel"     ! display selection table
   SetTabText _Config[1], WINDOW", 1, "WinTitle", ""\
                                     ! reset window title
  EndIf

EndMacro
```

The GetDataFile macro uses a file selector box to select an existing data file to a temporary register called TEMP. The macro uses the DDT DDT_DataSel to create the selection table:

```
Name MakeDDT

  DelTab _config[3],"DDT_DataSel"
  NewTab _config[3],"DDT_DataSel",_config[2],"DDT"
  SetTabHdrVal  _config[3], "DDT_DataSel", "DispRowNum",  1
  SetTabHdrText _config[3], "DDT_DataSel", "RowNumTitle", "#"
  SetTabHdrVal  _config[3], "DDT_DataSel", "RowNumWidth", 4
  SetTabHdrText _config[3], "DDT_DataSel", "RowFormat",   "%d"
  SetTabHdrText _config[3], "DDT_DataSel", "TitleFont",   ""
  SetTabHdrText _config[3], "DDT_DataSel", "ElemFont",    ""
  SetTabHdrVal  _config[3], "DDT_DataSel", "FixedCols",   1
  SetTabHdrVal  _config[3], "DDT_DataSel", "WindowClass", 1
  SetTabHdrVal  _config[3], "DDT_DataSel", "DispXAxis",   0
  SetTabHdrText _config[3], "DDT_DataSel",\
               "Btn1",        "Load,0,0,0,0,DoGetSelectedData"
  SetTabHdrText _config[3], "DDT_DataSel",\
               "Btn2",        "Cancel,0,0,0,0,ClearSel"

  InsTabRow _config[3], "DDT_DataSel"
  SetTabText _config[3], "DDT_DataSel", 1, "Source",     "SampleName"
  SetTabText _config[3], "DDT_DataSel", 1, "Title",      "Spectrum"
  SetTabVal  _config[3], "DDT_DataSel", 1, "Control",    0
  SetTabText _config[3], "DDT_DataSel", 1, "EnumStrings",""
  SetTabVal  _config[3], "DDT_DataSel", 1, "Width",      16
  SetTabText _config[3], "DDT_DataSel", 1, "Format",     "%s"
  SetTabVal  _config[3], "DDT_DataSel", 1, "Justify",    0

EndMacro

MakeDDT
```

Because a register saves spectra as individual objects, the macro uses the table editor EdObjTab which allows access to object header items (specified in a DDT) across objects. The macro uses the SampleName object header item of the spectra to make the selection. The macro uses buttons with a data table in a similar way to the object header editor. Therefore the setup of buttons within the DDT is identical. The macro MakeDDT creates the DDT DDT_DataSel.

The macro sets the table header item DispRowNum to 1 to switch on an index display of the objects in register TEMP. The next table header items specify the title of the index column, the column width, and number format.

In terms of customized operation of your ChemStation, buttons are the most flexible tools. These buttons allow you to execute macros within the displayed windows table environment. In the example the macro sets up two buttons, called Load and Cancel. The macro specifies these buttons as string table header items using fixed item names. The table header items Btn1 corresponds to the first button. The string of a button definition specifies the label of the button (for example, Load), the position of the button relative to the window, and the associated macro name (for example, DoGetSelectedData). If you want the ChemStation to position the buttons automatically, you must specify zeros for all position entries. The first row in the DDT specifies the first column in the selection table to be the sample names of the data objects. To run the buttons you must also define the two associated macros.

The macro DoGetSelectedData associated with the Load button uses the selection table of window 1. The SelectSize function returns the number of selected objects. If the size is greater than zero the GetSelect command gets the names of the objects in the global variable Sel_RegObj$ and the Evaluate command moves the selected objects to the destination register, specified with the global variable Register$:

```
Name DoGetSelectedData

  Parameter RegObj$        Default ""
  Parameter RowNumber      Default 0
  Parameter ColumnNumber   Default 0
  Parameter ColumnSource$  Default ""

  Local h,i

  h = SelectSize(1)
  If h > 0 then
    WinUpdate off
    for i = 1 to h
      GetSelect 1,i
      Evaluate "MoveObj " + Sel_RegObj$ + "," + Register$
    Next i
    WinUpdate on
  EndIf
  ClearWin 1
  DelReg TEMP
  Remove Register$
EndMacro
```

Macros associated with buttons have predefined parameters. You must use these parameters with the button macro definition. In the example the parameters are a register object (RegObj$), a row number (RowNumber), a column number (ColumnNumber), and a column name (ColumnSource$). The ChemStation always passes these parameters to button macros, even if the macro does not use them as in the above example. To run such a macro from the command line as well without the need to specify unnecessary parameters, defaults for these parameters can be used.

A second macro called ClearSel is associated with the Cancel button:

```
Name ClearSel

  Parameter RegObj$        Default ""
  Parameter RowNumber      Default 0
  Parameter ColumnNumber   Default 0
  Parameter ColumnSource$  Default ""

  ClearWin 1
  DelReg TEMP
  Remove Register$

EndMacro
```

This macro quits the table to select spectra without moving any spectrum into the destination register.

A macro example for generating and manipulating DDT's is shown in "Example of DDT Macro" on page 113. When you load the macro it creates a User menu with the two entries Edit/Create DDT and Get DDT of Window.

If you use the entry Edit/Create DDT, the macro prompts you for a DDT name. If you previously used the entry Get DDT of Window, the macro displays appropriate entries proposals in the entry field. This allows you to manipulate easily existing table windows. If you do not save the configuration when you exit the software, the changes you made are only temporary. If you specify the name of a DDT that does not exist, the macro creates a new DDT and displays an empty table.

You can create new columns by inserting rows in the DDT using the Insert Column and Append Column buttons in the Edit DDT window. You can use the Delete Column button only if rows exist in the DDT and you have selected one or more rows.

The Get Columns button allows you to get possible columns for the type of table you want to display. These possible columns are then available as selections in the Source Column specification field of the DDT. For a table, you

must specify the register object and the table name. For data you must enter the register object. For a table of object header items, you can specify a single object making the header items of this object available in the Source Column combo box.

You use the Display button to show the table. The ChemStation displays the table generated in a different window. The ChemStation chooses the window automatically from the user windows or you can set the window explicitly using the Window button. If you previously executed the Get DDT of Window menu item, the ChemStation uses the selected window. You can use the possibility of specifying the display window either to display different tables using the same DDT or to compare the display look for different DDTs using the same data.

You can use the Line # off button to switch off the additional index column and the Line # on button to switched it on again.

The Copy DDT button allows you to copy the current DDT to a DDT with a new name. If a DDT with the name specified already exists, the ChemStation gives you a warning and offers you the choice of overwriting the existing DDT or aborting the action.

The Store Table button allows you to save the table as ASCII text to a file. The Print Table button prints the table to the current printer. You use the Exit button to close the Edit DDT window.

In addition to the definition of buttons in table windows to execute user definable macros, macro hooks are also possible based on table events. These predefined events allow you to execute macros automatically, based on operations performed with the table. The mechanism to implement these event based macro hooks is similar to the button implementation: you use a predefined table header text item to specify the name of the macro you want to execut. As with button macros, the ChemStation sets a set of predefined parameters automatically. The event macro must read these parameters regardless of their usage.

“Example of DDT Macro” on page 113 shows an example of a table event based macro with the macro to handle DDTs. The macro UpDateSource is called based on changes to table fields. The macro uses the parameter

ColumnSource$ to catch entries you make in the source column fields. If you make these entries, the macro sets defaults in other fields in the same row automatically. This helps you with the setup of DDTs.

Example of DDT Macro

You can use the example macro DDT_COM.MAC to manipulate display description tables.

```
Name MakeDDT

 local h

 h = FreeWin(1)
 if h > 0 and h <10 then

   _WinDDT = h

   If check(Variable,_TabType) = 0 then
     _TabType = 0
   EndIf

   h = Input("Enter DDT name : ",_DDTName$)

     If h = 1 then
       SetWinTitle _WinDDT,"Edit DDT : " + _DDTName$
       On Error Goto NoTab
       h = TabColType(_Config[3],_DDTName$,"Source")
       EdTab _WinDDT,_Config[3],_DDTName$,"DDT_MakeDDT"
       Goto ExitMake
NoTab:
       On Error
       Print
       CopyTab _Config[2],"DDT",_Config[3],_DDTName$
       EdTab _WinDDT,_Config[3],_DDTName$,"DDT_MakeDDT"
ExitMake:
       MenuState "User","Edit/Create DDT",Insensitive
       MenuState "User","Get DDT of Window",Insensitive
     EndIf
   Else
   h = Alert("No user window available")
 EndIf

EndMacro
```

```
Name ExitDDT

  Parameter RegObj$        Default ""
  Parameter TableName$     Default ""
  Parameter RowNumber      Default 0
  Parameter ColumnNumber   Default 0
  Parameter ColumnSource$  Default ""

  If check(Variable,_DDTWin) = 1 then
    Remove _DDTWin
  EndIf
  If check(Variable,_WinDDT) = 1 then
    ClearWin _WinDDT
    Remove _WinDDT
  EndIf
  If check(Variable,_TabType) = 1 then
    Remove _TabType
  EndIf
  If check(Variable,_DDTName$) = 1 then
    Remove _DDTName$
  EndIf
  If check(Variable,_DispRegObj$) = 1 then
    Remove _DispRegObj$
  EndIf
  If check(Variable,_DispTabName$) = 1 then
    Remove _DispTabName$
  EndIf
  MenuState "User","Edit/Create DDT"
  MenuState "User","Get DDT of Window"

EndMacro

Name CopyDDT

  Parameter RegObj$        Default ""
  Parameter TableName$     Default ""
  Parameter RowNumber      Default 0
  Parameter ColumnNumber   Default 0
  Parameter ColumnSource$  Default ""

  Local h
  Local NewDDT$

  h = Input("Enter new DDT name : ",NewDDT$)
  If h = 1 then
    On Error Goto NoDDT
    h = TabColType(_Config[3],NewDDT$,"Source")
    h = Alert("Replace  " + NewDDT$,1)
    If h = 0 or NewDDT$ = _DDTName$ then
```

```
      Goto ExitCopy
    EndIf
    DelTab _Config[3],NewDDT$
NoDDT:
    On Error
    Print
    CopyTab _Config[3],_DDTName$,_Config[3],NewDDT$
    _DDTName$ = NewDDT$
    SetWinTitle _WinDDT,"Edit DDT : " + _DDTName$
ExitCopy:
  EndIf

EndMacro

Name LineOn

  Parameter RegObj$        Default ""
  Parameter TableName$     Default ""
  Parameter RowNumber      Default 0
  Parameter ColumnNumber   Default 0
  Parameter ColumnSource$  Default ""

  SetTabHdrVal  _config[3], _DDTName$, "DispRowNum",  1
  SetTabHdrText _config[3], "DDT_MakeDDT",\
                "Btn4", "Line # off,9,9,41,9,LineOff"
  EdTab _WinDDT,_Config[3],_DDTName$,"DDT_MakeDDT"

EndMacro

Name LineOff

  Parameter RegObj$        Default ""
  Parameter TableName$     Default ""
  Parameter RowNumber      Default 0
  Parameter ColumnNumber   Default 0
  Parameter ColumnSource$  Default ""

  SetTabHdrVal  _config[3], _DDTName$, "DispRowNum",  0
  SetTabHdrText _config[3], "DDT_MakeDDT",\
                "Btn4", "Line # on,9,9,41,9,LineOn"
  EdTab _WinDDT,_Config[3],_DDTName$,"DDT_MakeDDT"

EndMacro

Name WinNr

  Parameter RegObj$        Default ""
  Parameter TableName$     Default ""
  Parameter RowNumber      Default 0
```

```
  Parameter ColumnNumber   Default 0
  Parameter ColumnSource$  Default ""

  Local h

  h = Input("Enter Display Window number : ",_DDTWin)
  If h = 0 then
    If check(Variable,_DDTWin) = 1 then
      Remove _DDTWin
    EndIf
  EndIf

EndMacro

Name UpDateSource

  Parameter RegObj$        Default ""
  Parameter TableName$     Default ""
  Parameter RowNumber      Default 0
  Parameter ColumnNumber   Default 0
  Parameter ColumnSource$  Default ""

  Local h
  Local H$,F$

  If ColumnSource$ = "Source"
    H$ = TabText$(_Config[3],_DDTName$,RowNumber,"Title")
    If len(H$) = 0 then
      WinUpDate off
      F$ = TabText$(_Config[3],_DDTName$,RowNumber,"Source")
      SetTabText _Config[3],_DDTName$,RowNumber,"Title",F$
      SetTabVal _Config[3],_DDTName$,RowNumber,"Width",12
      If check(Variable,_TabType) = 1 then
        If _TabType = 1 then
          F$ = TabColDefFormat$(_DispRegObj$,_DispTabName$,"Source")
          SetTabText _Config[3],_DDTName$,RowNumber,"Format",F$
        EndIf
        If _TabType = 2 then
          SetTabText _Config[3],_DDTName$,RowNumber,"Format","%#12.5G"
        EndIf
        If TabType = 3 then
          Evaluate "h = ObjHdrType(" + _DispRegObj$ + ",F$)"
          If h = 0 then
            SetTabText _Config[3],_DDTName$,RowNumber,"Format","%s"
          Else
            SetTabText _Config[3],_DDTName$,RowNumber,"Format",\
                       "%#12.5G"
          EndIf
        EndIf
```

```
      EndIf
      WinUpDate on
    EndIf
  EndIf

EndMacro

Name GetColumns

  Parameter RegObj$        Default ""
  Parameter TableName$     Default ""
  Parameter RowNumber      Default 0
  Parameter ColumnNumber   Default 0
  Parameter ColumnSource$  Default ""

  Local h

  h =  showdialog("TabType")

  If h = 1 then
    If _TabType = 1 then
      h = Input("Enter Register Object and Table Name : ",\
                _DispRegObj$,_DispTabName$)
      If h = 1 then
        SetTabText _Config[3], "DDT_MakeDDT", 1, "EnumStrings",\
                   GetTabColHdr$(_DispRegObj$,_DispTabName$)
        EdTab _WinDDT,_Config[3],_DDTName$,"DDT_MakeDDT"
      EndIf
      Else
      If _TabType = 2 then
        h = Input("Enter Register Object Name: ",_DispRegObj$)
        If h = 1 then
          SetTabText _Config[3], "DDT_MakeDDT", 1, "EnumStrings",\
                     GetDataCols$(_DispRegObj$)
          EdTab _WinDDT,_Config[3],_DDTName$,"DDT_MakeDDT"
        EndIf
        Else
        h = Input("Enter Register Object Name: ",_DispRegObj$)
        If h = 1 then
          SetTabText _Config[3], "DDT_MakeDDT", 1, "EnumStrings",
                     GetObjHdr$(_DispRegObj$)
          EdTab _WinDDT,_Config[3],_DDTName$,"DDT_MakeDDT"
        EndIf
      EndIf
    EndIf
  EndIf

EndMacro
```

```
Name Display

  Parameter RegObj$        Default ""
  Parameter TableName$     Default ""
  Parameter RowNumber      Default 0
  Parameter ColumnNumber   Default 0
  Parameter ColumnSource$  Default ""

  Local h,x

  h = 0

  If _TabType = 0 then
    h =  showdialog("TabType")
  Else
    h = 1
  EndIf

  If check(Variable,_DDTWin) = 0 then
    x = FreeWin(1)
    If x > 0 and x %<10 then
      _DDTWin = x
    Else
    EndIf
  Else
    h = 1
  EndIf

  If h = 1 then
    If _TabType = 1 then
      h = Input("Enter Register Object and Table Name : ",\
                _DispRegObj$,_DispTabName$)
        If h = 1 then
          Evaluate "EdTab _DDTWin," + _DispRegObj$ + ",_DispTabName$,\
                   _DDTName$"
        EndIf
        Else
        If _TabType = 2 then
          h = Input("Enter Register Object Name: ",_DispRegObj$)
        If h = 1 then
          Evaluate "EdDataTab _DDTWin," + _DispRegObj$ + ",_DDTName$"
        EndIf
        Else
           h = Input("Enter Register Name: ",_DispRegObj$)
        If h = 1 then
          Evaluate "EdObjTab _DDTWin," + _DispRegObj$ + ",_DDTName$"
        EndIf
      EndIf
    EndIf
```

```
  EndIf

EndMacro

Name PrintButton

  Parameter RegObj$        Default ""
  Parameter TableName$     Default ""
  Parameter RowNumber      Default 0
  Parameter ColumnNumber   Default 0
  Parameter ColumnSource$  Default ""

  Local h
  Local H$

  If _TabType = 0 then
    h = Alert("Must Specify Data Source (Use Get Columns)",2)
  Else
    If _TabType = 1 then
      H$ = "Tab"
  Else
    If _TabType = 2 then
      H$ = "DataTab"
    Else
      H$ = "ObjTab"
    EndIf
  EndIf
  If check(Variable,_DispTabName$) = 0 then
    _DispTabName$ = ""
  EndIf
  InitReport "Edit DDT"
  PrintTab  _DispRegObj$,_DispTabName$,_DDTName$,,H$
  EndReport
  EndIf

EndMacro

Name StoreButton

  Parameter RegObj$        Default ""
  Parameter TableName$     Default ""
  Parameter RowNumber      Default 0
  Parameter ColumnNumber   Default 0
  Parameter ColumnSource$  Default ""

  Local h
  Local H$

  If _TabType = 0 then
```

```
    h = Alert("Must Specify Data Source (Use Get Columns)",2)
  Else
    If _TabType = 1 then
      H$ = "Tab"
    Else
      If _TabType = 2 then
        H$ = "DataTab"
      Else
        H$ = "ObjTab"
      EndIf
    EndIf
    If check(Variable,_DispTabName$) = 0 then
      _DispTabName$ = ""
    EndIf
    InitReport "Edit DDT",1
    StoreTab  _DispRegObj$,_DispTabName$,_DDTName$,,H$
    EndReport
  EndIf

EndMacro

NAME  GetObjHdr$

  Parameter RegObj$

  LOCAL i,k,x                          ! local variables
  LOCAL A$,H$,R$                       ! local strings

  i = 1                                ! initialize
  A$ =""                               ! initialize
  H$ =""                               ! initialize
  R$ =""                               ! initialize

  Repeat
  Evaluate "R$ = ObjHdrName$(" + RegObj$ + "," + Val$(i) + ")"\
                                       ! get item name
  x = Len(R$)
  If x > 0 then
    A$ = RegObj$ + ","""  +  R$ + """"
    Evaluate "k = ObjHdrType(" + A$ + ")" ! get item type
    If k = 0  or k = 1 then
      If H$ = "" then
        H$ = R$                        ! set first item
      Else
        H$ = H$ + "|"+ R$              ! add item
      EndIf
    EndIf
    i = i + 1                          ! increment counter
  EndIf
```

```
  Until (x = 0)                          ! until last

  Return H$

ENDMACRO

NAME  GetTabColHdr$

  Parameter RegObj$  DEFAULT ""
  Parameter TabName$  DEFAULT ""

  LOCAL i,MaxCol                         ! Item count
  LOCAL H$,X$,A$                         ! Item name

  Evaluate "MaxCol = TabHdrVal(" + RegObj$ + "," + TabName$ +\
           ",""NumberOfCols"")"
  H$ = ""
  If MaxCol > 0 then
    A$ = "X$ = TabColName$(" + RegObj$ + ",TabName$,Val$(i))"
    i = 1
    Evaluate A$
    H$ = X$                              ! initialize
    for i = 2 to MaxCol                  ! get rest of Headers
      Evaluate A$
      H$ = H$ + "|"+ X$                  ! add item
    Next i
  EndIf
  Return H$

ENDMACRO

NAME  CheckWin

  Parameter Win

  Local A$,B$,C$
  Local h

  L$ = TabText$(_CONFIG[1],"WINDOW",Win,"Command") ! get command
  If Len(L$) > 0 then                  ! not empty ?
    C$ = zzrpGetNextParam$("L$")       ! get command keyword
!-------------- Check command ----------------------------------------
    If C$="EdObjTab" OR C$="edobjtab" OR C$="EDOBJTAB" OR C$="EdTab"\
       OR C$="edtab" OR C$="EDTAB" OR C$="EdDataTab"\
       OR C$="eddatatab" OR C$="EDDATATAB" then
      A$ = zzrpGetNextParam$("L$") ! ignore first (WinNr)
      A$ = zzrpGetNextParam$("L$") ! get Reg/RegObj/RegObjRange
      repeat
       h = instr(A$," ")               ! leading blank
```

```
        If h = 1 then               ! yes
          A$ = A$[2:len(A$)]        ! remove
        EndIf
      until h <> 1                  ! all removed
      On Error Goto Try
        Evaluate "_DispRegObj$ = " + A$
        Goto TryEnd
Try:
        On Error
        Print
      Evaluate "_DispRegObj$ = """ + A$ + """"
TryEnd:
      On Error
      If len(L$) > 0 then
        A$ = zzrpGetNextParam$("L$") ! get additional parameter
      Else
        A$ = ""
      EndIf
      If len(L$) > 0 then
        B$ = zzrpGetNextParam$("L$") ! get additional parameter
      Else
        B$ = ""
      EndIf
!-------------- Table of Object Items -------------------------------
      If C$="EdObjTab" OR C$="edobjtab" OR C$="EDOBJTAB" then
        _TabType = 3
      EndIf
!-------------- Table -----------------------------------------------
      If C$="EdTab" OR C$="edtab" OR C$="EDTAB" then
        If len(A$) > 0 then
          On Error GoTo Try1
            Evaluate "_DispTabName$ = " + A$
          Goto Try1End
Try1:
          On Error
            Print
            Evaluate "_DispTabName$ = """ + A$ + """"
Try1End:
          On Error
        Else
          _DispTabName$ = ""
        EndIf
        _TabType = 1
        A$ = B$
      EndIf
!-------------- Data ------------------------------------------------
      If C$="EdDataTab" OR C$="eddatatab" OR C$="EDDATATAB" then
        _TabType = 2
      EndIf
```

```
!-------------- End command processing ------------------------------
      If len(A$) > 0 then
        On Error GoTo Try2
          Evaluate "_DDTName$ = " + A$
          Goto Try2End
Try2:
        On Error
          Print
          Evaluate "_DDTName$ = """ + A$ + """"
Try2End:
        On Error
          Evaluate "_DDTName$ = " + A$
      Else
        _DDTName$ = ""
        h = Alert("No DDT used in selected window")
      EndIf
    EndIf
  EndIf
  Remove L$

EndMacro

NAME  GetDataCols$

  Parameter RegObj$  DEFAULT ""

  LOCAL i,x
  LOCAL H$

  Evaluate "x = ObjHdrVal(" + RegObj$ +  ",""ObjClass"")"

  If x = 5 then
    Evaluate "x = DataRows(" + RegObj$ + ")"
  Else
    Evaluate "x = DataCols(" + RegObj$ + ")"
  EndIf

  H$ ="1"
  for i = 2 to x
    H$ = H$ + "|" + Val$(i)
  Next i
  Return H$

ENDMACRO

! DDT_MakeDDT

  DelTab _config[3],"DDT_MakeDDT"
  NewTab _config[3],"DDT_MakeDDT",_config[2],"DDT"
```

```
  SetTabHdrVal  _config[3], "DDT_MakeDDT", "DispRowNum",  1
  SetTabHdrText _config[3], "DDT_MakeDDT", "RowNumTitle", "Column"
  SetTabHdrVal  _config[3], "DDT_MakeDDT", "RowNumWidth", 7
  SetTabHdrText _config[3], "DDT_MakeDDT", "RowFormat",   "%d"
  SetTabHdrText _config[3], "DDT_MakeDDT", "TitleFont",   ""
  SetTabHdrText _config[3], "DDT_MakeDDT", "ElemFont",    ""
  SetTabHdrVal  _config[3], "DDT_MakeDDT", "FixedCols",   1
  SetTabHdrVal  _config[3], "DDT_MakeDDT", "WindowClass", 1
  SetTabHdrVal  _config[3], "DDT_MakeDDT", "DispXAxis",   0
  SetTabHdrText _config[3], "DDT_MakeDDT",\
                "Btn1",        "Get Columns,133,0,41,9,GetColumns"
  SetTabHdrText _config[3], "DDT_MakeDDT",\
                "Btn2",        "Display,175,0,41,9,Display"
  SetTabHdrText _config[3], "DDT_MakeDDT",\
                "Btn3",        "Exit,217,0,41,9,ExitDDT"
  SetTabHdrText _config[3], "DDT_MakeDDT",\
                "Btn4",        "Line # off,9,9,41,9,LineOff"
  SetTabHdrText _config[3], "DDT_MakeDDT",\
                "Btn5",        "Window,50,9,41,9,WinNr"
  SetTabHdrText _config[3], "DDT_MakeDDT",\
                "Btn6",        "Copy DDT,92,9,41,9,CopyDDT"
  SetTabHdrText _config[3], "DDT_MakeDDT",\
                 "Btn7",        "Store Table,133,9,41,9,StoreButton"
  SetTabHdrText _config[3], "DDT_MakeDDT",\
                 "Btn8",        "Print Table,175,9,41,9,PrintButton"
  SetTabHdrText _config[3], "DDT_MakeDDT", "BtnInsRow",\
                "Insert Column,9,0,41,9"
  SetTabHdrText _config[3], "DDT_MakeDDT", "BtnAppRow",\
                "Append Column,50,0,41,9"
  SetTabHdrText _config[3], "DDT_MakeDDT", "BtnDelRow",\
                "Delete Column,92,0,41,9"
  SetTabHdrText _config[3], "DDT_MakeDDT", "EvUpdCell",\
                "UpDateSource"
  SetTabHdrText _config[3], "DDT_MakeDDT", "TableRect",\
                "0,20,0,0"
    InsTabRow _config[3], "DDT_MakeDDT",1:6

!Source
  SetTabText _config[3], "DDT_MakeDDT", 1, "Source", "Source"
  SetTabText _config[3], "DDT_MakeDDT", 1, "Title", "Source Column"
  SetTabVal  _config[3], "DDT_MakeDDT", 1, "Control", 6
  SetTabText _config[3], "DDT_MakeDDT", 1, "EnumStrings","1|2|3"
  SetTabVal  _config[3], "DDT_MakeDDT", 1, "Width", 20
  SetTabText _config[3], "DDT_MakeDDT", 1, "Format", "%s"
  SetTabVal  _config[3], "DDT_MakeDDT", 1, "Justify", 0
!Title
  SetTabText _config[3], "DDT_MakeDDT", 2, "Source", "Title"
  SetTabText _config[3], "DDT_MakeDDT", 2, "Title", "Column Title"
  SetTabVal  _config[3], "DDT_MakeDDT", 2, "Control", 2
```

```
  SetTabText _config[3], "DDT_MakeDDT", 2, "EnumStrings",""
  SetTabVal  _config[3], "DDT_MakeDDT", 2, "Width", 20
  SetTabText _config[3], "DDT_MakeDDT", 2, "Format", "%s"
  SetTabVal  _config[3], "DDT_MakeDDT", 2, "Justify", 0
!Width
  SetTabText _config[3], "DDT_MakeDDT", 3, "Source", "Width"
  SetTabText _config[3], "DDT_MakeDDT", 3, "Title", "Width"
  SetTabVal  _config[3], "DDT_MakeDDT", 3, "Control", 2
  SetTabText _config[3], "DDT_MakeDDT", 3, "EnumStrings",""
  SetTabVal  _config[3], "DDT_MakeDDT", 3, "Width", 6
  SetTabText _config[3], "DDT_MakeDDT", 3, "Format", "%ld"
  SetTabVal  _config[3], "DDT_MakeDDT", 3, "Justify", 1
!Format
  SetTabText _config[3], "DDT_MakeDDT", 4, "Source", "Format"
  SetTabText _config[3], "DDT_MakeDDT", 4, "Title", "Format"
  SetTabVal  _config[3], "DDT_MakeDDT", 4, "Control", 6
  SetTabText _config[3], "DDT_MakeDDT", 4,"EnumStrings",\
"%s|%d|%f|%.2G|%.3G|%.4G|%.5G|%.6G|%.7G|%.2E|%.3E|%.4E|%.5E|%.6E|%.6E|%.7
E"
  SetTabVal  _config[3], "DDT_MakeDDT", 4, "Width", 8
  SetTabText _config[3], "DDT_MakeDDT", 4, "Format", "%s"
  SetTabVal  _config[3], "DDT_MakeDDT", 4, "Justify", 1
!Justify
  SetTabText _config[3], "DDT_MakeDDT", 5, "Source", "Justify"
  SetTabText _config[3], "DDT_MakeDDT", 5, "Title", "Justify"
  SetTabVal  _config[3], "DDT_MakeDDT", 5, "Control", 4
  SetTabText _config[3], "DDT_MakeDDT", 5, "EnumStrings",\
             "left|centered|right"
  SetTabVal  _config[3], "DDT_MakeDDT", 5, "Width", 8
  SetTabText _config[3], "DDT_MakeDDT", 5, "Format", "%s"
  SetTabVal  _config[3], "DDT_MakeDDT", 5, "Justify", 0
!Control
  SetTabText _config[3], "DDT_MakeDDT", 6, "Source", "Control"
  SetTabText _config[3], "DDT_MakeDDT", 6, "Title", "Control"
  SetTabVal  _config[3], "DDT_MakeDDT", 6, "Control", 4
  SetTabText _config[3], "DDT_MakeDDT", 6, "EnumStrings",\
             "read only|read only scroll|read/write"
  SetTabVal  _config[3], "DDT_MakeDDT", 6, "Width", 10
  SetTabText _config[3], "DDT_MakeDDT", 6, "Format","%s"
  SetTabVal  _config[3], "DDT_MakeDDT", 6, "Justify", 0

  RemoveDialog("TabType")

  BeginDialog "TabType", 80, 50,80,55,"Table Type"

    OptionGroup _TabType

    OptionButton  10,  5, 60, 10, "Table"
    OptionButton  10, 15, 60, 10, "Data Table"
```

```
  OptionButton  10, 25, 60, 10, "Object Header Table"

  okButton       5, 40, 30, 12, "OK"
  cancelButton  45, 40, 30, 12, "Cancel"

EndDialog

MenuAdd "User","Edit/Create DDT","MakeDDT"
MenuAdd "User","Get DDT of Window",\
        "_DDTWin = ActiveWindow();CheckWin _DDTWin"
```

8
Spectral Data Register

Spectral Data Register

When you measure data with the spectrophotmeter or load data from disk the ChemStation saves each data set as an object. The ChemStation puts these data objects in the Samples or Standards registers depending on the software and measurement process you used. The number of objects corresponds directly to the number of individual data sets. The acquisition process adds to the data header information such as the date and time, path length, instrument type, and the adjustable instrument number. If you add information such as dilution factors, this information becomes part of the data object. Chapter 14, “Registers” describes all predefined header items. The ChemStation structures the primary data so that the software can interpret the data easily and correctly. You can add concentration information using the AnalyteTable. If this information is available, you can use a data set as a standard in quantification.

You do data analysis on the objects in the samples register and standards register. Processing of data comprises three steps:

1 Spectral processing

2 Data access (use wavelengths)

3 Evaluation

The data analysis parameters defines the processing steps you want to perform. The ChemStation saves these parameters in the registers DataAnalysis_Param_1 through DataAnalysis_Param_4. You start the data analysis with the current parameters in memory by using the menu item Analyze or by using the Analysis Results view. You can activate up to four sets of data analysis parameter in parallel. This is useful when you want to optimize parameters or to compare results. The ChemStation saves the results of the data analysis in registers. The ChemStation uses an individual register for the results of each of the three processing steps. Because you can run more than one analysis at the same time, the ChemStation appends the data analysis index to the register name.

The ChemStation saves the results of the first processing step (spectral processing) in *processed* registers. If the data you process are samples, the ChemStation saves the results in the ProcessedSamples_1 register. For each data object in the Samples register the ChemStation creates a corresponding

data object in the ProcessedSamples_1 register according to the processing specification. All the header information from the original object is available and the ChemStation adds information about the processing.

If you specify the use of wavelengths for data access, the ChemStation saves the accessed values WLResult registers. For samples these registers are called WLResult_Smp_1 through WLResult_Smp_4. The ChemStation creates two matrix objects to access data. The first object contains the data and the second object contains the standard deviations. The ChemStation structures the matrix so that the object index in the Samples register corresponds to the row index in the matrix. In other words, the second row of the data matrix contains the data of sample 2. In addition, row 0 contains the corresponding wavelength information. You can access and display the first data point used with sample two using the command:

```
Print Data(WLResult_Smp_1[1],2,1)
```

Each data point you define with the access parameters corresponds to a column in the data matrix. The wavelength information for this data point is in row 0 of the same column. For the above example you can display the wavelength using the command:

```
Print Data(WLResult_Smp_1[1],0,1)
```

In addition the function result values used in SCA and equations are in column –1. No wavelength information is available for this column. You access the function result for sample 2 using the command:

```
Print Data(WLResult_Smp_1[1],2,-1)
```

You access the corresponding standard deviation of the function result using the second object of the access data register:

```
Print Data(WLResult_Smp_1[2],2,-1)
```

The ChemStation saves the results of an evaluation in the Eval_Results_1 through Eval_Results_4 registers, creating an individual object for each sample. Each object contains the results in a table called AnalyteTable. This table has eight columns:

- AnalyteName
- Value
- StdDev
- Unit
- ValueCorrPathl

- PathCorrStdDev
- PathValue
- PathStdDev

The AnalyteName column gives the equation property or the component name and the Value column gives the calculated result. The Unit column gives the units of the result values. The ValueCorrPathl column gives the calculated result corrected for dilution and path length. The PathValue column is the intermediate result only corrected for path length.. Three additional columns contain the estimated standard deviations: the StdDev column for the calculated result,the PathCorrStdDev column for the final result and the PathStdDev column for the path length corrected values.

TheChemStation Software has a similar structure for standards as described here for the samples. The result registers for spectral processing are ProcessedStandards_1 through ProcessedStandards_4, the data access results are in the registers WLResult_Std_1 through WLResult_Std_4, and the evaluation results in the registers Eval_Results_Std_1 through Eval_Results_Std_4.

9
Reports and Print Description Tables

Reports and Print Description Tables

The ChemStation offers you a maximum of three predefined report types. The Configure Report dialog of the Config Report menu item allows you to set the general report configuration. You can specify a page layout in terms of margins and the number of characters used for the indent. For graphics you can set an annotation limit and an additional line in the report page header can be specified.

The ChemStation handles report generation automatically when you use the Report menu items. This makes sure the ChemStation generates a report with valid results. Some of the macro commands used for generating reports are useful for programming customized reports. These report macros may include graphics and you can also direct them to file. When you direct a report including graphics to a file, the ChemStation saves the graphics in separate files, one file for each graphic. The ChemStation always overwrites these graphics files with the next report that includes graphics. To avoid overwriting of the old graphics files you must rename the files.

The macro commands useful in writing report macros and using the same style as the general report configuration are:

- InitReport
- EndReport
- PrintVal
- PrintText
- PrintTab
- HardCopy
- PDraw
- NewLine
- Indent

You use the InitReport command to start a report. With the InitReport command you can also specify a report name. If you want to generate the report My First, use the command:

```
InitReport "My First"
```

The InitReport command resets the page counter and sets a header with the report name, date, and time. You use the PrintVal and PrintText commands to print single numeric values and string data. These commands allow to set a text label infront of the information. In addition the ChemStation fixes the maximum length of the labels to simplify alignment of the information. The PrintTab command allows you to print all types of tables. As with the editing of tables, you can use a control table, called print description tables (PDT), for formatting. PDTs are similar to DDTs with the exception that you cannot use event hooks or buttons. You can also use existing DDTs to print a table in a formatted way, but the ChemStation ignores extra information.

If you create a PDT called PDT_Dilution with one column for sample names and another column for the dilution factors, the following commands print the table to the report you opened previously:

```
PrintVal ”Samples: ”,RegSize(Samples)
NewLine
PrintText ”Dilution Factors: ”,”Samples”
NewLine
PrintTab ”Samples”,,”PDT_Dilution”,,”ObjTab”
```

You print the report using:

```
EndReport ”My First”
```

The EndReport command adds an end label to the text and starts printing. The NewLine command add an empty line. You can also use the NewLine command to generate more than one empty line by specifying the number of lines as a parameter.

You can include graphics in the report. If you used first derivatives in the example above and you want to include these in the report, use the commands:

```
Draw 1, ProcessedSamples_1
HardCopy 1
```

You use the Indent command to move text or graphics to the right. The ChemStation moves to the right all print or graphics commands that following the Indent command. To reset the start position of the printout use the command:

```
Indent 0
```

You can also use staggered indents. You reset these indents stepwise.

Another approach is to use text templates combined with functions or commands to set actual results or graphics in a template. You can generate these templates as files using a text editor such as Notepad.

The text must be in ASCII format and not in a document file specific to the word processing application. You put the commands or macros in braces where the ChemStation should make replacements. The Report_Template command replaces these braces with actual text or executes the commands. The following is a simple example of a template file:

```
{InitReport "Spectrum"}


                  Normalized Spectrum Report


Sample Name    {ObjHdrText$(Samples[1],"SampleName")}
     Date      {ObjHdrText$(Samples[1],"Date")}
     Time      {ObjHdrText$(Samples[1],"TimeOfDay")}

Operator Name  {ObjHdrText$(Samples[1],"Operator")}


{Draw 1,Samples[1],190:500,0:1}
{HardCopy 1}

{EndReport "Spectrum"}
```

The filename is TEMPLATE.TXT and is saved in the reports directory. To print an actual report use the command:

```
Report_Template _ReportPath$ + "Template.txt".
```

10 Exchanging Information Between Windows Applications by DDE

Exchanging Information by DDE

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

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

Dynamic Data Exchange

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

DDE Commands

The ChemStation has a set of DDE commands to initiate and terminate communication with other applications, 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 4.

Table 4 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 your the *Commands* handbook.

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
- 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 139. 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 online ChemStation as a server from the offline version of the ChemStation:

```
Channel = DDEInitiate ("HPUV-VIS", "CPWAIT")
```

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, "Loadsamples"
DDEExecute Channel, "Print MeasureSample()"
```

NOTE *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
HPUV-VIS	is the application name for the UV-Visible ChemStation top level

The application is specified during initialization. In HPUV-VIS a command processor variable called _DDENAME$ holds the application name. This name should be used by a client to initiate a link.

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

Items with the SYSTEM Topic

Table 5 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 5 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 — it can be BUSY or IDLE
Returnmessage	Special return message

For example, if you want to get information about the status, you can request it through DDE.

```
Channel = DDEInitiate (”HPUV-VIS”,”SYSTEM”)
DDERequest Channel,”Status”,answer$
Print answer$
DDETerminate Channel
```

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.
- 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 must load the macros into the ChemStation using the MACRO command.
- You must start Microsoft Excel with a spreadsheet called SHEET1.XLS

NOTE To turn on the command line in the ChemStation, choose the System menu box in the upper-left corner of the window.

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.
- 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.

NOTE 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 your *Commands* handbook.

NOTE 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 HPUV-VIS.
- the server is Excel.
- the application is Excel.
- the topic is SHEET1.XLS.
- 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 = 0
  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

! Termination section

  DDETerminate Chan                     ! Close the channel

EndMacro


Name CloseDDE                           ! Error handling macro

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

EndMacro
```

The next macro sends the results of the equation, SCA or MCA quantification, to a Excel spreadsheet for further calculations. The macro first pokes the names of the samples in the first column of the spreadsheet and results of each analyte in a seperate column.

```
!*************************************************************
!                                                            *
!  This macro sends the results as columns to a Excel sheet  *
!  The results have to be calculated  prior by 'analyze'     *
!                                                            *
!*************************************************************

! Globals    Chan

name SendtoExcel

local i,j,k,cell$,value,analname$,name$
local endoftable,noanalyte,col$,row$

On Error CloseDDE
Chan = DDEInitiate("EXCEL","Sheet1")      ! Excel must already run


endoftable = regsize(eval_results_1)      ! number of samples
noanalyte = tabhdrval(eval_results_1[1],"Analytetable",\
                      "NumberofRows")     ! number of analytes
DDEPoke Chan,"R1C1","Sample Name"         ! Annotate first column
```

```
For i = 1 to endoftable          ! read samplename of each sample
  col$="C1"
  cell$="R"+val$(i+1)            ! Enter samplename in first column
  name$=ObjhdrText$(Eval_results_1[i],Samplename)
  DDEPoke Chan,cell$,name$
Next i

For i = 1 to noanalyte           ! each analyte gives one column
  col$="C"+val$(i+1)
  cell$="R1"+col$                ! Enter name of analyte in first row
  analname$= TabText$(Eval_Results_1[1],AnalyteTable,i,AnalyteName)
  DDEPoke Chan,cell$,analname$
  For j = 1 to endoftable        ! Loop to access all samples
     row$="R"+val$(j+1)
     cell$=row$+col$
     value = TabVal(Eval_Results_1[j],AnalyteTable,i,Value)
     DDEPoke Chan,cell$,val$(value)
  Next j
Next i
DDETerminate Chan
EndMacro

Name CloseDDE                    ! Closes DDE in case of an Error
Local Button

DDETerminate Chan
Button = Alert ("Stopped: DDE on error trap",3)
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.
- The macro gets the contents of row 5 column 6 from the spreadsheet and prints the data on the print line of the ChemStation.

NOTE 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 your *Commands* handbook.

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

NOTE 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 HPUV-VIS.
- the server is Excel.
- the application is Excel.
- the topic is SHEET1.XLS.
- 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 = 0
  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

  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

EndMacro

! Error handling macro

Name CloseDDE

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

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.
- 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.

NOTE 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 your *Commands* handbook.

NOTE 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 HPUV-VIS.
- the server is Excel.
- the application is Excel.
- the topic is SHEET1.XLS.
- 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 = 0
  Chan = DDEInitiate("EXCEL", "Sheet1")            ! Open the channel

! Conversation block
```

```
  DDEExecute Chan, ”[Hide()]”                    ! Run the command

! Termination section

  DDETerminate Chan                              ! Close the channel

  EndMacro

! Error handling macro

Name CloseDDE

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

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.
- 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.

NOTE 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 your *Commands* handbook.

NOTE 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 HPUV-VIS.
- the server is Excel.
- the application is Excel.
- the topic is SHEET1.XLS.
- 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 = 0
  Chan = DDEInitiate(”EXCEL”, ”Sheet1”)    ! Open the channel
```

```
! Conversation block

  DDERequest Chan, ”R5C6”, OriginalData$      ! Get the original data

! Set the hot link variable to the original data so we can
!     test for it to 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

EndMacro

! Error handling macro

Name CloseDDE

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

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, HPUV-VIS and 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.

11 Communicating Through the RS232 Serial Interface

This chapter describes how to set up specific ChemStation files to allow macro-based communication with other instruments through the serial interface. As an example a macro is described that reads the pH value from a Radiometer PHM 93 pH meter and controls a Gilson Dilutor 401.

Configuring an RS232 Device

Two modes of communication with devices connected via RS232 serial communication ports are desscribed. The first approach sets up a logical device and uses specific commands refferring to that device using a device name. The second approach shows how to use file access functions for serial communication.

The ChemStation has to be configured to provide the command for a serial communication. For this purpose several steps are necessary. These steps are done using a text editor like Notepad. To clarify how do this, we will assume that a pH meter has to be controlled by the ChemStation of Instrument 1 and no other RS232 device has been set up. The pH meter communicates through a serial link with 9600 baud, no parity, 8 data bits, and 1 stop bit and is connected to serial port COM1. To access the pH meter as a device, the default device name *SERIAL* is used.

The [PCS] Section of the WIN.INI File

Make a backup copy of your WIN.INI file in case something goes wrong, because this file contains important informations for all Windows applications. The WIN.INI file is located in the WINDOWS directory. Load WIN.INI in Notepad and search for [PCS]. The [PCS] section contains information about your ChemStations and is similar to the example below except for the characters printed in italics.

The [PCS] section also depends on how many applications other than your UV-Vis ChemStation are set up and which links and devices did already exist. To simplify the examples given, a single installation of your UV-Vis ChemStation using default setting is assumed.

```
[PCS]
Path=C:\HPCHEM
Applications=HP-UV
Instruments=1
Links=1,2
Link1=HPIB,1,7,0,3
Link2=RS232,COM1:9600,n,8,1   InitFile=c:\hpchem\rs232.ext,SERIAL
```

```
Devices=1,2
Device1=1,18,8452A
Device2=2,0,SERIAL
```

Identify the line with

```
Link2= RS232,COM1:9600,n,8,1 InitFile=c:\hpchem\rs232.ext,SERIAL
```

This line defines the basic communication parameters. Thes parameters are defining that serial port COM1: is used, the baud rate is set to 9600 baud, no parity is used, 8 data bits and one stop bit are used in the serial communication. Additional settings are specified with the parameters in file RS232.EXT descibed below.

The RS232 commands and functions are available in an instrument session only if the serial device is configured for that particular instrument. This is done in the [PCS,x] sections. To activate the serial communication for the first instrument, this section must include the device link:

```
[PCS,1]

Devices=2, ...
```

Device 2 refers to the SERIAL device.

Communications-Device Control Block

The file RS232.EXT contains information used by windows to control communication. The information defines the control setting for a serial communications device in the device control block (DCB). The following example of the RS232.EXT file contains default values. If you are an experienced user, you can use this example as a basis for modification.

Start Notepad and load the RS232.EXT file in ther C:\HPCHEM directory:

```
[SERIAL]

ReceiveEndofText = 0
SendEndofText = 0

SendStartofText = 0
ReturnEndofText = 0

RlsTimeout = 0
CtsTimeout = 0
DsrTimeout = 0
```

```
fBinary = FALSE
fRtsDisable = FALSE
fOutXCtsFlow = FALSE
fOutXDsrFlow = FALSE
fDtrDisable = FALSE
fOutX = FALSE
fInX = FALSE
fPeChar = FALSE
fNull = FALSE
fChEvt = FALSE
fRtsFlow = FALSE
fDtrFlow = FALSE
XonChar = 17
XoffChar = 12
XonLim = 500
XoffLim = 1000
PeChar = 0
EofChar = 0
EvtChar = 0
```

The RS232.EXT file contains two groups of information. The first group comprises four terms:

ReceiveEndofText

SendEndofText

SendStartofText

ReturnEndofText

You must specify all these terms. If you do not use the terms, specify them as 0 (zero). You specify the terms as a list of character values separated by commas. For example, if the SendEndofText is carriage return and line feed, specify it as:

```
SendEndofText=13,10,0
```

The character sequence is limited to four characters and the zero. The meanings of SendStartofText and SendEndofText are self-explanatory. ReceiveEndofText gives the character sequence that is expected to end the message of a instrument. These characters are removed from the message when received. ReturnEndofText specifies characters that can be added to the message before the message is passed to the application.

The second group of the RS232.EXT file is the DCB group. You use this group of fields to specify all values used in the DCB block (excluding information such as baud rate and parity). This group has three types of field:

Flags	Specified as TRUE or FALSE (for example, fRtsDisable = FALSE)
Numbers	A number (for example, XonLim = 500)
Characters	Specified by their ASCII value (for example, XonChar = 17)

For descriptions of these fields, see the *Software Development Kit* for the Mircosoft® Windows™ graphical environment.

The File HP-UV.INI

The file HP-UV.INI contains internal information necessary for the communication with devices. The file HP-UV.INI is in the C:\HPCHEM directory.

```
[Instruments]
DeviceTypes=8452A,89090A,SERIAL

[8452A]
Code=18
Protocol=OLD_HPIB
CU=2048

[89090A]
Code=20
Protocol=OLD_HPIB
CU=256

[SERIAL]
Code=1
Protocol=APG
CU=DEF:2000
```

Your ChemStation is now ready to communicate through RS232 using macros with a device called SERIAL.

RS232 Commands and Functions

The configuration of a RS232 device adds four additional commands to the ChemStation command set:

- **`RS232Send “Device Name”, string`**
  is a command to send a string.
- **`RS232Receive$( “Device Name”)`**
  is a function to receive a string.

These two commands are used for the communication itself and allow to send and receive strings. Also there are two miscellaneous commands to set or read the timeout setting of the RS232 device. This means the time the ChemStation waits for a response on receipt of a string, otherwise an error is generated.

- **`RS232SetTimeOut “Device Name”, value`**
  is a command to set the timeout in ms.
- **`RS232GetTimeOut( “Device Name”)`**
  is a function to get the timeout in ms.

Controlling a Radiometer PHM 93 pH Meter

The following macro demonstrates the use of these commands to read the pH value from a Radiometer PHM93 pH meter. The printer port of this pH meter can be used to send two-letter commands simulating keyboard input. In this example, the pH meter is set in the method screen (}M), then to the pH/mV readout screen (}F) and a printout of that data is started by the print command (}P). The pH meter sends a large block of text to the ChemStation, and the pH value has to be extracted from this block. This is done by a search for the keyword *pH* and the position of this keyword is then used to locate and extract the value.

```
!  Control of the Radiometer PHM93 pH meter using the RS232.DLL
!
!     These macros are not guaranteed or supported by Agilent

Name Read_pH_meter

! This Function reads one pH-Value from PHM 93

  Local buffer$,dummy$, position, pH,send$

  send$="}M"+chr$(13)
  RS232Send "SERIAL",send$           ! switch pH meter to Method
  Sleep 5                            ! give it time to do it
  buffer$ = RS232Receive$("SERIAL")  ! get response
  If buffer$[1:2] <> "{M" then       ! check for correct response
    Generate Error 42,"Communication Error }M"
  EndIf

  send$="}F"+chr$(13)
  RS232Send "SERIAL",send$           ! switch to pH/mV and measure
  Sleep 1                            ! give it time to do it
  buffer$ = RS232Receive$("SERIAL")  ! get response
  If buffer$[1:2] <> "{F" then       ! check for correct response
    Generate Error 42,"Communication Error }F"
  EndIf

  a$="}P"+chr$(13)
  RS232Send "SERIAL",a$              ! transmit actual pH to PC
  Sleep 1                            ! give it time to do it
  buffer$ = RS232Receive$("SERIAL")  ! get response
  If buffer$[1:2] <> "{P" then       ! check for correct response
    Generate Error 42,"Communication Error }P"
  EndIf
```

```
  If len(buffer$) %< 200 then          ! there must be more in buffer
    Sleep 1                          ! give it more time to do it
    dummy$ = RS232Receive$("SERIAL") ! get response
    buffer$=buffer$+dummy$           ! now we have all
  EndIf

  position = instr(buffer$[50:len(buffer$)],"pH")+49
                           ! search for keyword pH in response
  pH = val(buffer$[position-10:position-1])
                           ! and extract it
  Return pH                ! return the extracted pH value

EndMacro
```

The macro Read_pH_meter has also some error trapping in case the pH meter sends a wrong response. The second macro store_pH is an example of a macro called as a post-measurement macro. Post-measurement macros can be activated and specified in the method checklist of the ChemStation. The macro store_pH checks for the last measurement action which has be done and calls the macro Read_pH_meter in case of a sample or standard measurement. The pH value is added to the analytes of the sample or standard.

```
Name store_pH

! This macro can be used in the postmeasure macro hook
!    to store the pH value in the AnalyteTable at the specified
!    row (AnalyteNumber) of Sample or Standard

  Local pH, cmd$, Destination$, i, row

  i = currentmeasureactivity()

  If (i = 2) or (i = 3)   THEN
    pH = read_pH_meter()
    If i = 2 THEN
      InsTabRow Samples[Regsize(Samples)],"AnalyteTable"
      row = tabhdrval(Samples[Regsize(Samples)],"AnalyteTable",\
                      "NumberOfRows")
      Settabtext Samples[Regsize(Samples)],"AnalyteTable",row,\
                 "AnalyteName","pH"
      SettabVal Samples[Regsize(Samples)],"AnalyteTable",row,\
                "Value",pH
    Else
      InsTabRow Standards[Regsize(Standards)],"AnalyteTable"
      row = tabhdrval(Standards[Regsize(Standards)],"AnalyteTable",\
                      "NumberOfRows")
      Settabtext Standards[Regsize(Standards)],"AnalyteTable",row,\
                "AnalyteName","pH"
      Settabval Standards[Regsize(Standards)],"AnalyteTable",row,\
```

```
                "Value",pH
    EndIf
  EndIf

EndMacro
```

The check for the last measurement activity can be done in the post-measurement macro hook with the ChemStation macro function currentmeasureactivity(). The return values are:

0 System is idle (function called outside hook)

1 Last measurement was a Blank

2 Last measurement was a Sample

3 Last measurement was a Standard

4 Last measurement was a Auxiliary

Controlling the Gilson Dilutor 401

You can also use the RS232 interface to control a Gilson Dilutor 401. However, the setup of the RS232 link described in "Configuring an RS232 Device" on page 154 does not work because the timing of the communication protocol required by the dilutor is very critical. To overcome this problem you must purchase from the Gilson company a device driver called GSIOC.SYS. This device driver executes the low-level communication protocol and the ChemStation can use this driver to communicate with the dilutor.

Use Notepad to add the line:

```
DEVICE=C:\GSIOC.SYS
```

to your CONFIG.SYS file. This loads the device driver when you turn on line power to your ChemStation. The communication from the macro level of the ChemStation occurs by *print output* to the file GSOC and *input* from the file GSIC. The driver does the rest of the control. The first step to start the communication is to open the two files as shown in the macro.

```
!  Control of the Gilson Dilutor using the GSIOC device driver

Name com_init                          ! initializes the fileslots

  Open "GSOC" for output as #3           ! 3 or another free fileslot
  Open "GSIC" for input as #4            ! 4 or another free fileslot
  Print #3,"D"                         ! init serial port
EndMacro
```

You must close the files before you close the ChemStation:

```
Name com_close                         ! closes the fileslots

  Close #3
  Close #4
EndMacro
```

The Gilson devices understand two different types of command. These are called *buffered* and *immediate* commands. The following two macros handle these commands at a higher level.

```
Name buff_cmd$               ! send a buffered command to Dilutor
  Parameter  cmd$,gsioc_unit
  Local SendCmd$, Receive$

  If len(cmd$) = 0 then
    Return "No command given"
```

```
  Else
    SendCmd$ = "B"+chr$(128+gsioc_unit)+cmd$
    Repeat
      Print #3, SendCmd$
      Input #4, Receive$
    Until Receive$[1] <> "#"
    Return  Receive$
  EndIf
EndMacro

!*************************************************************

Name imi_cmd$              ! send an immediate command to Dilutor
  Parameter  cmd$,gsioc_unit
  Local SendCmd$, Receive$

  If len(cmd$) = 0 then
    Return "No command given"
  Else
    SendCmd$ = "A"+chr$(128+gsioc_unit)+cmd$
    Print #3, SendCmd$
    Input #4, Receive$
    Return Receive$
  EndIf
EndMacro
```

You can now use the macro buff_cmd$ and imi_cmd$ to initiate the dilutor, set the flowrate, aspirate or dispose a certain volume. The next few macros give you an idea of how to control the dilutor. A complete list of all commands are given in the manual supplied with the Gilson Dilutor 401.

```
Name dilutor_init
  Parameter  syringevol
  Local SendCmd$

  Print imi_cmd$("Z",0)
  Print buff_cmd$("SP",0)
  SendCmd$="P"+val$(syringevol)
  Print buff_cmd$(SendCmd$,0)
EndMacro

Name dilutor_flow
  Parameter  flowrate
  Local SendCmd$

  SendCmd$="F"+val$(flowrate)
  Print buff_cmd$(SendCmd$,0)
EndMacro
```

```
Name aspirate
  Parameter volume
  Local SendCmd$

  Print buff_cmd$(”V1”,0)
  SendCmd$=”A”+val$(volume)
  Print buff_cmd$(SendCmd$,0)
  Print buff_cmd$(”V0”,0)
EndMacro

Name dispose
  Parameter volume
  Local SendCmd$

  SendCmd$=”D”+val$(volume)
  Print buff_cmd$(SendCmd$,0)
EndMacro
```

You can now use these macros to set up, for example, an automated titration system. You use the pre-measurement macro titrate to dispose the titrant in the cell and wait some time for mixing. The post-measurement macro pH_dilcorr corrects the measured spectrum for dilution and takes the pH value from a pH meter. The macro init_vol with the dialog myinitvol set the variables Startvol and Actualvol to their initial values.

To automate the complete sequence you can use the automation table with the sampling system Manual. You can start the macro ’init_vol’ and the macro for the aspiration of the titrant user macros. You must enter every measurement as a line in the automation table specifying the action Measure Sample and Source ID 0 (zero). It is important to set the Source ID to zero, because this suppresses the user prompt for the sample and takes the measurement immediately.

```
Name titrate
  Parameter volume, idle

  Actualvol=Actualvol+volume       ! Used for dilution correction
  dispose volume
  Print ”Wait time for mixing: ”,idle,” seconds”
  Sleep idle
EndMacro

!*************************************************************

Name ph_dilcorr       ! Measures a pH value and corrects dilution
  Local sb_factor
```

```
  sb_factor=Actualvol/Startvol
  scalarmultobj "Samples[Regsize(Samples)]",sb_factor
  store_ph                          ! see example for pHmeter

EndMacro

!****************************************************************

Name init_vol

  Define_my_dia1
  Startvolx$="2000"
  If showdialog("myinitvol")=1 then
    removedialog "myinitvol"
    Startvol= val(startvolx$)
  Else
    removedialog "myinitvol"
    startvol = 2000
  EndIf
  Actualvol = Startvol

EndMacro

!****************************************************************
Name Define_my_dia1

  BeginDialog "myinitvol",100,50,150,50,"Startvolume"
    Statictext 10,10,58,10,"Startvolume:"
    Editbox 60,9,50,12, Startvolx$
    OkButton 20,35,40,12,"OK"
    CancelButton 90,35,40,12,"Cancel"
  EndDialog

EndMacro
```

12 Variables

Standard Variables

Standard variables are already defined global variables . They are used to pass values between macros. They can also be deleted by the Remove command. Existing standard variables should not be removed to assure system integrity. Generally within macros only local variables should be used. Table 6 and Table 7 show only the standard variables just after start of the ChemStation in advanced mode.

Table 6 Standard String Variables

Variable	Description
_ImportExportPath$	Path for import and export of spectra.
_SampleLogTable$	Name of current sample table.

Table 7 Standard Scalar Variables

Variable	Description
_AutoState	Set to 1 if automation is running
_CloseLevel	for internal use
I	common variable as counter in loops
_Exp10f	ln 10 = 2.30258509

System Variables

System variables are variables which can not be removed.

Table 8 System String Variables

Variable	Description
_AutoFile$	Name of current automation file.
_AutoPath$	Current path where automation tables are located.
_ConfigAutPath$	Path where automation files are located. Usually C:\HPCHEM\x\AUTOMATION\, where x is the instrument number. Used as default in macro command. Copied from configuration database. Cannot be changed by CP.
_ConfigMetPath$	Path where method files are located. Usually C:\HPCHEM\x\METHODS\, where x is the instrument number.
_DataFile$	Name of current data file.
_DataPath$	Path of current instrument data directory.
_DDEName$	DDE name of the application. This is HPUV-VISxxxxx, where xxxxx is a number that identifies the instance of the current software.
_DiagnosePath$	Path where diagnostics files are located. Usually C:\HPCHEM\x\DIAGNOSE\, where x is the instrument number.
_ERRCMD$	Name of command. The variable itself 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. Usually C:\HPCHEM\UVEXE\.
_InstName$	Instrument name. Copied from configuration database. Cannot be changed by CP.

Table 8 System String Variables (continued)

Variable	Description
_InstPath$	Path where files for current instrument are located. Usually C:\HPCHEM\x\, where x is the instrument number.
_MethFile$	Name of current method file.
_MethPath$	Path of current method.
_Operator$	Operator name.
_Product$	Product number of the software.
_Pg_FooterText$	Current printer footer text.
_Pg_HeaderText$	Current printer header text.
_ReportPath$	Path where report files are located. Usually C:\HPCHEM\x\REPORTS\, where x is the instrument number.
_SysMacPath$	Path where system macros are located. Usually C:\HPCHEM\SYSMACRO\.
_UserMacPath$	Path where user macros are located. Usually C:\HPCHEM\USERMAC\.
_Version$	Revision number of the software.

Table 9 System Scalar Variables

Variable	Description
_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.
_ERRLINE	Line number in a macro file. Variable is set if a macro aborts with an error
_ERROR	Error number
_Instance	Instance number of application. Cannot be changed by CP.

Table 9 System Scalar Variables (continued)

Variable	Description
_Instrument	May have value of 1-4 indicating the number of instruments configured.
_MENU_MEMORY	not used
_OffLine	Set to 0 if application is running offline. Set to 1 if application is online. Cannot be changed by C.
_Pg_BottomMargin	Current printer bottom margin settings (lines)
_Pg_FooterLines	Current printer footer length (lines)
_Pg_HeaderLines	Current printer header length (lines)
_Pg_LeftMargin	Current printer left margin (character)
_Pg_TopMargin	Current printer top margin settings (lines)
_SystemMode	Manager (1) or Operator (0) level

13 Windows

Table 10 Predefined Windows Parameters

Window Number	Type	Title	Register	Display Description Table
1-10	User	Untitled		
11	Graphic	Sample Spectra	Samples	
12	Table	Sample Spectra	Samples	DDTSamples
13	not used	Untitled		
14	Graphic	Processed Sample Spectra	ProcessedSamples_1	
15	Graphic	Processed Sample Spectra	ProcessedSamples_2	
16	Graphic	Processed Sample Spectra	ProcessedSamples_3	
17	Graphic	Processed Sample Spectra	ProcessedSamples_4	
18	Graphic	Math. Result	Arithm_Results	
19	Graphic	Auxiliary Spectra	Auxiliary	
20	Graphic	Last Blank Spectrum	Blank	
21	Table	Equation Results	Eval_Results_1	DDT_Equ_Smp_1
22	Table	Equation Results	Eval_Results_2	DDT_Equ_Smp_2
23	Table	Equation Results	Eval_Results_3	DDT_Equ_Smp_3
24	Table	Equation Results	Eval_Results_4	DDT_Equ_Smp_4

Table 10 Predefined Windows Parameters (continued)

Window Number	Type	Title	Register	Display Description Table
25	Table	Wavelength Results	WLResult_Smp_1	DDT_Afr_Smp_1
26	Table	Wavelength Results	WLResult_Smp_2	DDT_Afr_Smp_2
27	Table	Wavelength Results	WLResult_Smp_3	DDT_Afr_Smp_3
28	Table	Wavelength Results	WLResult_Smp_4	DDT_Afr_Smp_4
29	Table	Peaks	Task_Result	DDTPeaks
30	Table	Valleys	Task_Result	DDTValleys
31	Graphic	Peaks and Valleys of Selected Spectrum	Task_Result	
32	Graphic	(Dependent upon Task performed)	Task_Result	
33	Graphic	(Dependent upon Task performed)	Task_Result	
34	Table	Confirmation Analysis or Summary Result	Eval_Results	DDT_Confirm DDT_AllResults
35	Table	Compare Result		
36	Graphic	Mean	TestMethod_Result	
37	Table	Statistic	TestMethod_Result	DDT_TestMethod
38	Table	Tabular Data of Spectra	Any	DDT_SpecTab
39	Graphic	(Used for printer)		
40	Monitor	System Summary Status		
41	Monitor	Temperature Controller Status	Temco_Status	
42	Monitor	Spectrophotometer Status	Spectro_Status	
43	Dialog	Sample Table		
44	Graphic	Diagnose	UVDiag_Graph	
45	Table	Diagnose	UVDiag_Res	(Dependent on diagnostic task)
46	Table	Diagnose	UVDiag_Res	(Dependent on diagnostic task)

Table 10 Predefined Windows Parameters (continued)

Window Number	Type	Title	Register	Display Description Table
47	Graphic	Validation	Valid_Disp	
48	Graphic	Validation	(not used)	
49	Table	Validation	Validation_Result	(Dependent on validation test)
50	Table	Validation	Validation_Result	DDT_ValidRes DDT_ValidCFG
51	Graphic	Zero Cells Spectra	ZeroSpectra	
52	Graphic	Standard Spectra	Standards	
53	Table	Standard Spectra	Standards	DDTStds
54	Table	Analytes of Standard #	Standards	DDTAnalyteStd
55	Graphic	Processed Standard Spectra	ProcessedStandards_1	
56	Graphic	Processed Standard Spectra	ProcessedStandards_2	
57	Graphic	Processed Standard Spectra	ProcessedStandards_3	
58	Graphic	Processed Standard Spectra	ProcessedStandards_4	
59	Table	Equation Results	Eval_Results_Std_1	DDT_Equ_Std_1
60	Table	Equation Results	Eval_Results_Std_2	DDT_Equ_Std_2
61	Table	Equation Results	Eval_Results_Std_3	DDT_Equ_Std_3
62	Table	Equation Results	Eval_Results_Std_4	DDT_Equ_Std_4
63	Table	Wavelength Results	WLRESULTS_Std_1	DDT_Afr_Std_1
64	Table	Wavelength Results	WLRESULTS_Std_2	DDT_Afr_Std_2
65	Table	Wavelength Results	WLRESULTS_Std_3	DDT_Afr_Std_3
66	Table	Wavelength Results	WLRESULTS_Std_4	DDT_Afr_Std_4
67	Graphic	Calibration Curve	DataAnalysis_Param_1	
68	Graphic	Calibration Curve	DataAnalysis_Param_2	
69	Graphic	Calibration Curve	DataAnalysis_Param_3	
70	Graphic	Calibration Curve	DataAnalysis_Param_4	

Table 10 Predefined Windows Parameters (continued)

Window Number	Type	Title	Register	Display Description Table
71	Table	SCA Calibration Results	DataAnalysis_Param_1	
72	Table	SCA Calibration Results	DataAnalysis_Param_2	
73	Table	SCA Calibration Results	DataAnalysis_Param_3	
74	Table	SCA Calibration Results	DataAnalysis_Param_4	
75	Table	SCA Summary	DataAnalysis_Param_1	
76	Table	SCA Summary	DataAnalysis_Param_2	
77	Table	SCA Summary	DataAnalysis_Param_3	
78	Table	SCA Summary	DataAnalysis_Param_4	
79	Table	SCA Quantification Results	Eval_Results_Std_1	
80	Table	SCA Quantification Results	Eval_Results_Std_2	
81	Table	SCA Quantification Results	Eval_Results_Std_3	
82	Table	SCA Quantification Results	Eval_Results_Std_4	
83	Graphic	Residuals of Processed Standards	Eval_Results_Std_R	
84	Graphic	MCA Pure Component Spectra	Eval_Results_Std_D	
85	Table	MCA Calibration Results	Eval_Results_Std_1	DDTMCA_CalRes_1
86	Table	MCA Calibration Results	Eval_Results_Std_2	DDTMCA_CalRes_2
87	Table	MCA Calibration Results	Eval_Results_Std_3	DDTMCA_CalRes_3
88	Table	MCA Calibration Results	Eval_Results_Std_4	DDTMCA_CalRes_4
89	Table	Std.Dev. of Calibration	DataAnalysis_Param_1	DDT_MCAC_Sum
90	Table	Std.Dev. of Calibration	DataAnalysis_Param_2	DDT_MCAC_Sum
91	Table	Std.Dev. of Calibration	DataAnalysis_Param_3	DDT_MCAC_Sum
92	Table	Std.Dev. of Calibration	DataAnalysis_Param_4	DDT_MCAC_Sum
93	Graphic	Predicted Spectra of Sample	Eval_Results_D	

Table 10 Predefined Windows Parameters (continued)

Window Number	Type	Title	Register	Display Description Table
94	Table	MCA Quantification Results	Eval_Results_1	DDTMCA_QuaRes_1
95	Table	MCA Quantification Results	Eval_Results_2	DDTMCA_QuaRes_2
96	Table	MCA Quantification Results	Eval_Results_3	DDTMCA_QuaRes_3
97	Table	MCA Quantification Results	Eval_Results_4	DDTMCA_QuaRes_4
98	Graphic	Residual Spectrum of Sample	Eval_Results_R	

14 Registers

Overview of Registers used in the ChemStation Advanced Mode

The ChemStation uses a set of predefined registers as shown in Table 11 used by the ChemStation. Macros rely on the names of thses registers to function properly and you must take care when accessing these registers through macros.

Table 11 Name of registers used by the ChemStation

Type of Register	Register Name
System/Method Registers	Automation _Config DataAnalysis_Param_1 DataAnalysis_Param_2 DataAnalysis_Param_3 DataAnalysis_Param_4 FloatMenu GlobVars Meth_Descript ModeAdmin Report_Param Report_Status SampleLog SamplSys_Param Spectro_Param Spectro_Status StatMon Temco_Param Temco_Status

Table 11 Name of registers used by the ChemStation (continued)

Type of Register	Register Name
Data Registers	Arithm_Results
	Auxiliary
	Blank
	Eval_Results
	Eval_Results_1
	Eval_Results_2
	Eval_Results_3
	Eval_Results_4
	Eval_Results_Std_1
	Eval_Results_Std_2
	Eval_Results_Std_3
	Eval_Results_Std_4
	PP_Work
	ProcessedSamples_1
	ProcessedSamples_2
	ProcessedSamples_3
	ProcessedSamples_4
	ProcessedStandards_1
	ProcessedStandards_2
	ProcessedStandards_3
	ProcessedStandards_4
	Samples
	Samples_App
	Standards
	Standards_App
	WLResult_Smp_1
	WLResult_Smp_2
	WLResult_Smp_3
	WLResult_Smp_4
	WLResult_Std_1
	WLResult_Std_2
	WLResult_Std_3
	WLResult_Std_4

Table 11 Name of registers used by the ChemStation (continued)

Type of Register	Register Name
Temporary Registers	TaskFuncRes_Smp TaskFuncRes_Std Task_Result Task_Temp TestMethod_Result ZeroSpectra Eval_Results_R Eval_Results_Std_R Eval_Results_D Eval_Results_Std_D

Automation

The Automation register contains information on the current automation table. It contains one object only.

Table 12 Object header items of automation register object #1

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1=user specified
Title	string	r/w	Object title, not set.
AutomationTable	(table)	r/w	This is the automation table as entered from the user interface. The contents are listed below.
FromLine	integer ≥0	r/w	Line number in the automation table at which automation execution should start.
ToLine	integer ≥0	r/w	Line number in the automation table at which automation execution should stop.
AutoDesc	string	r/w	This is the description of the automation.

Table 13 Column Header Items of Automation Table

Header Item Name	Type/Range	Access	Meaning
SourceId	integer ≥0	r/w	Location of sample to be measured.
SampleName	string	r/w	Name of the sample
Action	string	r/w	Type of action to be performed, for example, Measure Samples, Load Method, and so on.
ExtParam	string	r/w	Any parameter required by the specified action, for example, the path and file name if Save is selected.

Arithm_Results

The Arithm_Results register contains the results of interactive mathematical processing of spectra. The number of objects in the register is equal to the number of spectra.

Auxiliary

This register contains auxiliary spectra.

Blank

The Blank register contains the last measured baseline spectrum. It normally contains one object only.

_Config

This register contains status information about windows, colors, axis styles, tables, and so on. It consists of 8 objects.

None of the information in this register belongs to the method, Load Method does not change this register. The _Config register is loaded from the standard CONFIG.REG file upon start up. . This CONFIG.REG file is in the UVEXE directory.

Table 14 shows the objects in the _Config register.

Table 14 Objects in _Config Register

#	Title	Description
1	User Interface	Contains information about the user interface such as colors, window descriptions, views and the realtionship between windows in a view, and so on.
2	Table Templates	Contains templates that are used to create a predefined table from scratch with the NewTab command. The user may add other table templates for their own use.
3	Display Description Tables	Contains tables of information specifying the layout of tables when displayed for editing. They are used by the table editing commands.
4	System	Contains a translation table for system messages
5	Automation	Reserved for future use.
6	Acquisition	Contains information about sampling systems (for example, co-ordinates for vials in Gilson autosampler) and information for the validation procedure.
7	Data Analysis	Contains tables of parameters used by the tasks.
8	Report	Contains parameters used for the report.

Object #1: User Interface

Table 15 Object Header Items in _Config Register Object #1, User Interface

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1=user specified.
Title	string	r/w	Object title=User Interface
Version	string	r/w	Software revision
AxisStyle	(table)	r/w	Defines axes of graphical data displays.
Font	(table)	r/w	Defines fonts.
Color	(table)	r/w	Defines colors for graphical data displays.
Window	(table)	r/w	Defines appearance of windows.
View_Math	(table)	r/w	Defines MATHS view.
Def_Math	(table)	r/w	Default parameters
View_PP_Smp_1 (...4)	(tables)	r/w	Defines Processed Samples view.
Def_PP_Smp_1 (...4)	(tables)	r/w	Default parameters
View_PP_Std_1 (...4)	(tables)	r/w	Defines Processed Standards view
Def_PP_Std_1 (...4)	(tables)	r/w	Default parameters
View_Smp	(table)	r/w	Defines the Samples view.
Def_Smp	(table)	r/w	Default parameters
View_Std	(table)	r/w	Defines the Standards view.
Def_Std	(table)	r/w	Default parameters
View_Afr_Smp_1 (...4)	(tables)	r/w	Defines the Used Wavelengths Samples view
Def_Afr_Smp_1 (...4)	(tables)	r/w	Default parameters
View_Afr_Std_1 (...4)	(tables)	r/w	Defines the Used Wavelengths Standards view

Table 15 Object Header Items in _Config Register Object #1, User Interface (continued)

Header Item Name	Type/Range	Access	Meaning
Def_Afr_Std_1 (...4)	(tables)	r/w	Default parameters
View_Equ_Smp_1 (...4)	(tables)	r/w	Defines the Equation Results Samples view
Def_Equ_Smp_1 (...4)	(tables)	r/w	Default parameters
View_Equ_Std_1 (...4)	(tables)	r/w	Defines the Equation Results Standards view (reserved for future use)
Def_Equ_Std_1 (...4)	(tables)	r/w	Default parameters
View_Cal_SCA_1 (...4)	(tables)	r/w	Defines the Single Component Analysis Calibration view
Def_Cal_SCA_1 (...4)	(tables)	r/w	Default parameters
View_Eval_SCA_1 (...4)	(tables)	r/w	Defines the Single Component Analysis Evaluation Results view
Def_Eval_SCA_1 (...4)	(tables)	r/w	Default parameters
View_Cal_MCA_1 (...4)	(tables)	r/w	Defines the Multicomponent Analysis Calibration view
Def_Cal_MCA_1 (...4)	(tables)	r/w	Default parameters
View_Eval_MCA_1 (...4)	(tables)	r/w	Defines the Multicomponent Analysis Evaluation Results view
Def_Eval_MCA_1 (...4)	(tables)	r/w	Default parameters
View_TestMeth	(table)	r/w	Defines the Test Method Task Results view
Def_TestMeth	(table)	r/w	Default parameters
View_C_SCA_SCA	(table)	r/w	Defines Compare Single Component Analysis against Single Component Analysis view
Def_C_SCA_SCA	(table)	r/w	Default parameters
View_C_SCA_MCA	(table)	r/w	Defines Compare Single Component Analysis against Multicomponent Analysis view

Table 15 Object Header Items in _Config Register Object #1, User Interface (continued)

Header Item Name	Type/Range	Access	Meaning
Def_C_SCA_MCA	(table)	r/w	Default parameters
View_C_MCA_SCA	(table)	r/w	Defines Compare Multicomponent Analysis against Single Component Analysis view
Def_C_MCA_SCA	(table)	r/w	Default parameters
View_C_MCA_MCA	(table)	r/w	Defines Compare Multicomponent Analysis against Multicomponent Analysis view
Def_C_MCA_MCA	(table)	r/w	Default parameters
View_Validate	(table)	r/w	Defines Validation view
Def_Validate	(table)	r/w	Default parameters
View_Diagnose	(table)	r/w	Defines Diagnostics view
Def_Diagnose	(table)	r/w	Default parameters
View_Diag_Flow	(table)	r/w	Defines Flow Test view
Def_Diag_Flow	(table)	r/w	Default parameters
View_Diag_Adj	(table)	r/w	Defines Adjust Disparity and Lamp Bridge Adjustment view
Def_Diag_Adj	(table)	r/w	Default parameters
CompCalWindows	(table)	r/w	Defines Compare Calibrations view
GOMDATALINKS	(table)	r/w	Defines the usage of graphical elements
View_ST_Smp	(table)	r/w	Defines the Standard tasks Samples view
Def_ST_Smp	(table)	r/w	Default parameters
View_ST_Std	(table)	r/w	Defines the Standard tasks Standards view

Table 15 Object Header Items in _Config Register Object #1, User Interface (continued)

Header Item Name	Type/Range	Access	Meaning
Def_ST_Std	(table)	r/w	Default parameters
EventMacros	(table)	r/w	Defines macros called and order in which changes to registers are executed

AxisStyle Table

Register name	_Config
Object number	1
Table name	AxisStyle

The axis style table has no predefined header items. Table 16 shows the predefined columns in the axis style table.

Table 16 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

Table 16 Predefined Columns in Axis Style Table (continued)

Column Name	Type/Range	Meaning
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
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	X position of axis begin within window 0 is left edge 1 is right edge
AxisYBegin	numeric	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 all other graphic windows, unless otherwise noted.
  2: Y axis for all other graphic windows, unless otherwise noted.
  3: X axis for calibration curve.
  4: Y axis for calibration curve.

- 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 0-1 and are relative to the window. xref Examples of AxisXBegin and AxisYBegin Values shows you some examples.

Table 17 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 18.

Table 18 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 19 shows the predefined columns in the color table.

Table 19 Predefined Columns in Color Table

Column Name	Type/Range	Meaning
Window	rgb	background color for window area
Data	rgb	background color for data area

Table 19 Predefined Columns in Color Table (continued)

Column Name	Type/Range	Meaning
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

Notes About the Color Table

- Upon startup of the system, after loading the _Config register from the file, the row 1 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.

Font Table

Register Name	_Config
Object Number	1
Table Name	Font

The font table has no predefined header items. Table 20 shows the predefined columns in the font table.

Table 20 Predefined Columns in Font Table

Column Name	Type/Range	Meaning
FaceName	string	font name
FontStyle	integer	styl type
Size	integer	font size in points
Color	integer	reference to color table
Justify	integer	justification
Angle	integer	orientation

Window Table

Register Name _Config
Object Number 1
Table Name Window

Chapter 13, "Windows" gives the windows used by the ChemStation. Table 21 shows the predefined header items in the window table.

Table 21 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

Table 21 Predefined Header Items in Window Table (continued)

Item Name	Type/Range	Meaning
LineStyle2 LineStyle3 LineStyle4 LineStyle5	enumeration	similar to LineStyle1 for next objects in register
CurrentView	string 9-characters	view name of current active view
MaxWinNr	integer	maximum number of windows
WinStackStart	integer	for internal use only
OnWinActivate	integer	for internal use only
XAxisStyle_	string 15-characters	name of AxisStyle table
YAxisStyle_	string 15-characters	name of AxisStyle table
ZAxisStyle_	string 15-characters	name of AxisStyle table
Color_	string 15-characters	name of Color table
Font	string	font name

Table 22 shows the predefined columns in the window table.

Table 22 Predefined Columns in Window Table

Column Name	Type/Range	Meaning
DefWXLow	numeric 0-1	left hand edge of window
DefWXHigh	numeric 0-1	right hand edge of window
DefWYLow	numeric 0-1	bottom edge of window
DefWYHigh	numeric 0-1	top edge of window

Table 22 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 For an explanation see Microsoft Windows Software Development Kit
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 Pen 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 Pen parameter
Destination	string	destination device of window: SCREEN Window will be drawn to screen. DefWXLow through DefWYHigh 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
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 22 Predefined Columns in Window Table (continued)

Column Name	Type/Range	Meaning
Type	enumeration	Type of window: 0 unspecified 1 graphic (Draw command) 2 "living" window with hard-coded WinNum, i.e. all modeless dialog boxes (isoplot tnote rc1 ChemStation for LC Only (note, edit events,edit calib table) 3 signal monitor 1 (ShowSignal command) 4 signal monitor 2 (ShowSignal command) 5 spectrum monitor (ShowSpectra command)tnoteref rc1 6 EdTab command 7 EdObjTab command 8 EdDataTab 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)
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

Table 22 Predefined Columns in Window Table (continued)

Column Name	Type/Range	Meaning
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
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	for internal use.
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	integer	index in font table.
ActRegister	string	register used to create window.
DefCommand	string	default command string for creation of window. If empty, no default exists.
CpName	string	for internal use.

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
EdTab
EditAnn
FindNearestAnn
FindNearestObj
FreeWin
GetSelect
SetWinTitle
Zoom
ZoomOut

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 Agilent. The uppermost 8 bits (Topic32 = 0x80000000 to Topic25 = 0x01000000) may be used by the user to group annotations according to their purpose.
- The user must not insert or delete any row within the first 98 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. Agilent supplied macros will not use thesewindow numbers. The windows will not take part in the View concept, see View table and SwitchView command.
- 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, esp. 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, Ed...Tab.
- 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).
- 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.

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 23 shows the predefined header items in the view table.

Table 23 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.

Table 23 Predefined Header Items in View Table (continued)

Item Name	Type/Range	Meaning
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
RelatedGrWnd	integer	for internal use.
RelatedTabWnd	integer	for internal use.
ViewInvalid	integer	for internal use

Table 24 shows the predefined columns in the view table.

Table 24 Predefined Columns in View Table

Column Name	Type/Range	Meaning
WinNum	integer	row index into Window table
WXLow	numeric 0-1	left-hand edge of window
WXHigh	numeric 0-1	right-hand edge of window
WYLow	numeric 0-1	bottom edge of window
WYHigh	numeric 0-1	top edge of window
WinStyle	bitfield 32	window styles as defined in Window table
Visible	boolean	0 window visible 1 window not visible

Specific Commands for the View Table

SwitchView

Notes About the View Table

1 The View tables are used to control the display management. The major elements of a particular view are the graphic and table windows. Each of these windows must be represented by a row in the Window table.

2 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].

3 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.

4 The user should not change any view table because this could then be inconsistent with the display status of the visible windows.

5 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 unhide 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.

CompCalWindows Table

Table 25 Table Header Items in CompCalWindows Table.

Header Item Name	Type/Range	Access	Meaning
FirstSel	string	r/w	Name of the first selected data analysis calibration for comparison
SecondSel	string	r/w	Name of the first selected data analysis calibration for comparison
Win1 (...6)	integer	r/w	The window numbers used for each of the six windows in the view
ForcedSingleCal	boolean	r/w	for internal use

Table 25 Table Header Items in CompCalWindows Table. (continued)

Header Item Name	Type/Range	Access	Meaning
SpecialMCAPM_1	string macro	r/w	for internal use
SpecialMCAPM_2	string macro	r/w	for internal use

Table 26 contains the window numbers for windows used by the SCA and MCA calibration views of the four possible data analyses methods. These are the windows which are used to build the Compare Calibrations View.

Table 26 Column Header Items CompCalWindows Table

Header Item Name	Type/Range	Access	Meaning
SCATab	integer	r/w	The single component analysis results table
SCACalSum	integer	r/w	The single component analysis summary results table
SCACalCurve	integer	r/w	The single component analysis calibration curve graphic
MCATab	integer	r/w	The multicomponent analysis results table
MCACalSum	integer	r/w	The multicomponent analysis summary results table

Table 27 Column Header Items EventMacros Table

Header Item Name	Type/Range	Access	Meaning
Register	string	r	The name of the register that will be changed
Macro	string macro	r	The name of the macro called when a register is changed

Table 27 Column Header Items EventMacros Table (continued)

Header Item Name	Type/Range	Access	Meaning
Priority	integer 1 to 255	r	If more than one register is changed, this defines the sequence. The value 255 has the highest priority.
Enable	boolean	r	Flag controlling whther the macro is executed or not. TRUE = yes. FALSE = no.

Object #2: Table Templates

Table 28 Object Header Items in _Config register object #2, Table Templates.

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title = Table Templates
Window	(table)	r/w	Defines the appearance of windows
AxisStyle	(table)	r/w	Defines axes of graphical data displays
Color	(table)	r/w	Defines colors for graphical data displays
View_	(table)	r/w	Defines the appearance of views
DDT	(table)	r/w	Defines the appearance of generic tables
ODT	(table)	r/w	for internal use
Font	(table)	r/w	for internal use
DDTMCA_CalRes	(table)	r/w	Defines the appearance of the MCA calibration results table
DDTMCA_QuaRes	(table)	r/w	Defines the appearance of the MCA quantification results table

Table 28 Object Header Items in _Config register object #2, Table Templates. (continued)

Header Item Name	Type/Range	Access	Meaning
DDT_AllResults	(table)	r/w	Defines the appearance of the multiple analyses summary results table
DDT_Confirm	(table)	r/w	Defines the appearance of the confirmation analysis summary results table
AnalyteTable	(table)	r/w	Defines the appearance of the analytes table

These tables provide templates for fast generation of tables. Detailed descriptions of the tables can be found under the descriptions of tables in Object#1 and #3.

Object #3: Display Description Tables

Table 29 Object Header Items in _Config Register Object #3, Display Description Tables

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title = Display Description Tables
DDT_SpecTab	(table)	r/w	Defines the appearance of the Tabular List of a spectrum with standard deviation table
DDT_SpecTab_S	(table)	r/w	Defines the appearance of the Tabular List of a spectrum without standard deviation table
DDTSamples	(table)	r/w	Defines the appearance of the Samples table

Table 29 Object Header Items in _Config Register Object #3, Display Description Tables (continued)

Header Item Name	Type/Range	Access	Meaning
DDTStds	(table)	r/w	Defines the appearance of the Standards table
AnalyteTable	(table)	r/w	Defines the appearance of the Analytes table
DDTAnalyteSmp	(table)	r/w	Defines the specific appearance of the Analytes table for samples
DDTAnalyteStd	(table)	r/w	Defines the specific appearance of the Analytes table for standards
DDTAnalyteRes	(table)	r/w	Defines the appearance of the Analytes table for quantification results
DDTAnalyteResCo	(table)	r/w	Defines the appearance of the Analytes table for quantification results corrected for different path lengths
DDT_Afr_Smp_1 (...4)	(tables)	r/w	Defines the appearance of the Used Wavelengths Results - Samples tables (for methods 1 to 4)
DDT_Afr_Std_1 (...4)	(tables)	r/w	Defines the appearance of the Used Wavelengths Results - Standards table (for methods 1 to 4)
DDT_Equ_Smp_1 (...4)	(tables)	r/w	Defines the appearance of the Equations Results - Samples tables (for methods 1 to 4)
DDT_Equ_Std_1 (...4)	(tables)	r/w	Defines the appearance of the Equations Results - Standards tables (for methods 1 to 4). Reserved for future use

Table 29 Object Header Items in _Config Register Object #3, Display Description Tables (continued)

Header Item Name	Type/Range	Access	Meaning
DDTEquation	(table)	r/w	Defines the appearance of the Equations when printed
DDTSCA_CalRes	(table)	r/w	Defines the appearance of the Single-component Analysis Results table
DDTSCA_CalResP	(table)	r/w	Defines the appearance of the Single-component Analysis Results table when path length correction is used
DDT_SCAC_Sum	(table)	r/w	Defines the appearance of the Single-component Analysis Summary Results table in advanced mode
DDT_ST_SCA_SUM	(table)	r/w	Defines the appearance of the Single-component Analysis Summary Results table in standard mode
DDT_Compare	(table)	r/w	Defines the appearance of the Compare Results table
DDTSCA_CalRes_S	(table)	r/w	Defines a summary version of the Single Component Analysis Results table
DDTSCA_QuaRes	(table)	r/w	Defines the appearance of the Single Component Analysis Results table
DDTSCA_QuaResP	(table)	r/w	Defines the appearance of the Single Component Analysis Results table when path length correction is used
DDTMCA_CalRes	(table)	r/w	Defines the appearance of the Multicomponent Calibration Results table

Table 29 Object Header Items in _Config Register Object #3, Display Description Tables (continued)

Header Item Name	Type/Range	Access	Meaning
DDTMCAC_SUM	(table)	r/w	Defines the appearance of the Multicomponent Analysis Summary Results table
DDTMCA_QuaRes	(table)	r/w	Defines the appearance of the Multicomponent Analysis Results table
DDTAnalytes	(table)	r/w	Defines the appearance of the Analytes table
DDTPeaks	(table)	r/w	Defines the appearance of the Peakfind Task - Peaks Results table in advanced mode
DDTValleys	(table)	r/w	Defines the appearance of the Peakfind Task - Valleys Results table in advanced mode
DDT_ST_Peaks	(table)	r/w	Defines the appearance of the Peakfind Task - Peaks Results table in standard mode
DDT_ST_Valleys	(table)	r/w	Defines the appearance of the Peakfind Task - Valleys Results table in standard mode
DDTConcMatrix	(table)	r/w	Defines the appearance of the Compose Task - Concentration Matrix table
DDTAnalyteTable	(table)	r/w	Defines the appearance of the Analytes table
DDTPreProc_WL	(table)	r/w	Defines the appearance of the Spectral Processing table in the Data Analysis dialog box
DDTAnalytical	(table)	r/w	Defines the appearance of the Analytical Functions table in the Used Wavelengths section of the Data Analysis dialog box

Table 29 Object Header Items in _Config Register Object #3, Display Description Tables (continued)

Header Item Name	Type/Range	Access	Meaning
DDTEvaluation	(table)	r/w	Defines the appearance of the Evaluation table in the Data Analysis dialog box
DDTDaStatSingle	(table)	r/w	Defines the appearance of the table used for status monitor display with single analysis.
DDTDaStatConf1 (...3)	(table)	r/w	Defines the appearance of the table used for status monitor display with confirmation analysis
DDTDaStatDA2 (...4)	(table)	r/w	Defines the appearance of the table used for status monitor display with multiple analyses
DDTAuto	(table)	r/w	Defines the appearance of the Automation table
PDTAUTO	(table)	r/w	Defines the appearance of the printed Automation table
DDTSampleLogTab	(table)	r/w	Defines the appearance of the Samples table in Automation
DDTAnalyteTab2	(table)	r/w	Defines the analyte table of the Sample table
DDTWavelTab	(table)	r/w	Defines the appearance of the table used for the Wavelength Range table in the Used Wavelengths section of Data Analysis
DDTWavelList	(table)	r/w	Defines the appearance of the table used for the Wavelength List table in the Used Wavelengths section of Data Analysis

Table 29 Object Header Items in _Config Register Object #3, Display Description Tables (continued)

Header Item Name	Type/Range	Access	Meaning
DDTMCA_CalibPar	(table)	r/w	Defines the appearance of the MCA Calibration Parameters table
DDT_ConfirmDlg	(table)	r/w	Defines the appearance of the Confirmation Analysis Parameters table
PDT_ConfirmDlg	(table)	r/w	Defines the appearance of the printed Confirmation Analysis Parameters table
DDT_Confirm	(table)	r/w	Defines the appearance of the Confirmation Analysis Results table
DDT_AllResults	(table)	r/w	Defines the appearance of the Confirmation Analysis Results table when multiple data analyses are used
DDT_TestMethod	(table)	r/w	Defines the appearance of the Test Method Results table
DDT_Statistics	(table)	r/w	Defines the appearance of the Evaluation Results Statistics table
DDT_UVDIAG_TAB	(table)	r/w	Defines the appearance of the Lamp Diagnostics screen
DDT_UVDIAG_TAB2	(table)	r/w	Defines the appearance of the Spectrophotometer Internal Diagnostics screen
DDT_UVFLOW	(table)	r/w	Defines the appearance of the Flow Test screen
DDT_UVDIAG_WL	(table)	r/w	Defines the appearance of the screens which list the wavelengths at which intensity is too low/high

Table 29 Object Header Items in _Config Register Object #3, Display Description Tables (continued)

Header Item Name	Type/Range	Access	Meaning
DDT_VALIDT1 (...6)	(table)	r/w	Defines the appearance of the Validation Test Results table
DDT_VALIDCFG	(table)	r/w	Defines the appearance of the Validation Configuration table
DDT_VALIDRES	(table)	r/w	Defines the appearance of the Validation Results Files table
DDT_ST_CalRes	(table)	r/w	Defines the appearance of the Calibration Results table in standard mode
DDT_ST_QuantRes	(table)	r/w	Defines the appearance of the quantitative Results table in standard mode
DDT_ST_Stand	(table)	r/w	Defines the appearance of the Standrds table in standard mode
DDT_ST_Samples	(table)	r/w	Defines the appearance of the Sample table in standard mode
FBIDT	(table)	r/w	Generic
DDT_ddt	(table)	r/w	Generic
PDT_Default	(table)	r/w	Generic
PDT_Properties	(table)	r/w	Defines the appearance of the printed user entered analytes table
PDT_AnalyteTab	(table)	r/w	Defines the appearance of the evaluated analytes results table
PDT_AnalTabConf	(table)	r/w	Defines the appearance of the evaluated analytes results table with confirmation analysis
PDT_Path	(table)	r/w	Defines the appearance of the path length table in the calibration report

Table 29 Object Header Items in _Config Register Object #3, Display Description Tables (continued)

Header Item Name	Type/Range	Access	Meaning
PDT_SCASummary	(table)	r/w	Defines the appearance of the printed Single Component Analysis Summary Results table
PDT_CalibTable	(table)	r/w	Defines the appearance of the calibration table for an analyte in the calibration report
PDT_CalibResSCA	(table)	r/w	Defines the appearance of the Single Component Analysis calibration results summary in the calibration report
PDT_SCAData	(table)	r/w	Defines the appearance of the calibration data table for Single Component Analysis in the calibration report
PDT_MCAData	(table)	r/w	Defines the appearance of the calibration data table for Multicomponent Analysis in the calibration report
PDT_CalibResMCA	(table)	r/w	Defines the appearance of the calibration coefficients table for Multicomponent Analysis in the calibration report
PDT_SampleData	(table)	r/w	Defines the appearance of the sample data table in the results report
PDT_Def_ObjTab	(table)	r/w	generic
PDT_Def_data	(table)	r/w	generic
PDT_ValidT1 (...6)	(table)	r/w	Defines the appearance of the printed Validation Test Results table
CurrentSCACDTT	string	r/w	DDTSCA_CalResP
CurrentSCAQDTT	string	r/w	DDTSCA_QuaResP

Table 29 Object Header Items in _Config Register Object #3, Display Description Tables (continued)

Header Item Name	Type/Range	Access	Meaning
CurrentMCAADTT	string	r/w	
AFR_StdDevOn	boolean	r/w	

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.

Display Description Table

Register name	_Config
Object number	3
Table name	any table (specified as DispTabName in commands)

Table 30 shows the predefined header items in the display description table.

Table 30 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 characters 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.

Table 30 Predefined Header Items in Display Description Table (continued)

Item Name	Type/Range	Meaning
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 31 shows the predefined columns in the display description table.

Table 31 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 user makes any changes in the field, s/he has to be warned that all changes will be ignored. 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. 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.

Table 31 Predefined Columns in Display Description Table (continued)

Column Name	Type/Range	Meaning
		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. 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 characters to be reserved for the column
Format	string	Format string for the column, according to the standard C function printf.
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.

Specific Commands for the Display Description Table

EdTab
EdObjTab
EdDataTab

Notes About the Display Description Table

- 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

- 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

Object #4: System

Table 32 Object Header Items in _Config Register Object #4, System

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title = System
CurrSampleName	string	r/w	The name of the last measured sample during an automated run
MethodChanged		r/w	Reserved for future use
StringTable	(table)	r/w	Lists error messages used by the software

StringTable Table

Table 33 Column Header Items in STRINGTABLE Table

Header Item Name	Type/Range	Access	Meaning
ID	integer	r/w	Error message identity number
String	string	r/w	Error message text

Object #5: Automation

(reserved for future use)

Table 34 Object Header Items in _Config Register Object #5, Automation

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title = Automation

Object #6: Acquisition

Table 35 Object Header Items _Config Register Object #6, Acquisition

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title = Acquisition
MaxDepth	numeric	r/w	Maximum depth in tenths of mm for Gilson autosampler, probe default = 1200
UnitID	integer	r/w	Identity number of Gilson autosampler, default is 10
RackType	string	r/w	Name of current rack type, for example, Rack Code 22
ActTriggerCmd	macro string	r/w	macros to be executed on trigger event
OnBlankButton	macro string	r/w	macros to be executed when the blank button is pressed on the instrument
OnStdButton	string	r/w	macros to be executed when the standard button is pressed on the instrument

Table 35 Object Header Items _Config Register Object #6, Acquisition (continued)

Header Item Name	Type/Range	Access	Meaning
OnSampleButton	macro string	r/w	macros to be executed when the sample button is pressed on the instrument
OnStopButton	macro string	r/w	macros to be executed when the stop button is pressed on the instrument
OnPowerOn	macro string	r/w	macros to be executed when the instrument is powered on
SmplSysTab	(table)	r/w	List of sampling system types and macros used to control them
RackCodes	(table)	r/w	List of rack types for Gilson autosampler
Gilson221	(table)	r/w	Information on rack dimensions when used with Gilson 221
Gilson222	(table)	r/w	Information on rack dimensions when used with Gilson 222
ValidCfgTbl_01	(table)	r/w	Information on instrument specifications, calibration standard values and other information for validation of a standard HP 8452A
ValidCfgTbl_02	(table)	r/w	Information on instrument specifications, calibration standard values and other information for validation of an HP 8452A Option 002
ValidCfgTbl_03	(table)	r/w	Information on instrument specifications, calibration standard values and other information for validation of an HP 8452A Option 003
ActSmplSysIndex	integer	r/w	Indexof current sampling system
ActSmplSys	string	r/w	Name of current selected sampling system

SamplSysTab Table

Table 36 Column Header Items in SamplSysTab Table

Header Item Name	Type/Range	Access	Meaning
Name	string	r/w	Key name of sampling system
Group	string	r/w	mode
DispText	string	r/w	Name of sampling system in user interface
Macro	string	r/w	Object title = System
GUIInfo	string	r/w	for internal use only
WaitForTemco	boolean	r/w	Flag to indicate if wait for temperature ready from Peltier has been selected. 0 = not activated. 1 = activated.
Pathlength1 (...7)	numeric	r/w	Configured path length in cm of cuvette(s)
CellType1 (...7)	string	r/w	measurement table for multiple cells

Rows for Manual, Sipper, Autosampler, Multicell Transport and Gilson Autosampler sampling systems.

RackCodes Table

Table 37 Column Header Items in RackCodes Table

Header Item Name	Type/Range	Access	Meaning
Name	string	r/w	Name of rack type
Code	integer	r/w	Number of rack type

Rows for Rack Codes 20, 21, 22, 23, 24, 28, 29, 30, 31, 32, 33, 34, and 22E.

Gilson221 and Gilson222 Tables

Table 38 Column Header Items in Gilson221 and Gilson222 Tables

Header Item Name	Type/Range	Access	Meaning
ZMAX	numeric	r/w	Maximum depth in tenths of mm for sampling probe
X1	numeric	r/w	X-axis position in tenths of mm of first vial
Y1	numeric	r/w	Y-axis position in tenths of mm of first vial
XStep	numeric	r/w	X-axis step in tenths of mm to next rows (Gilson 221) or columns (Gilson 222) of vials
YStep	numeric	r/w	Y-axis step in tenths of mm to next columns (Gilson 221) or rows (Gilson 222) of vials
NX	integer	r/w	Number of rows in X-direction
NY	integer	r/w	Number of rows in Y-direction

Rows for Rack Codes 20, 21, 22, 23, 24, 28, 29, 30, 31, 32, 33, 34, and 22E.

ValidCfgTbl_01 (...3) Tables

Table 39 Column Header Items in ValidCfgTbl_01 (...3) Tables

Header Item Name	Type/Range	Access	Meaning
Name	string	r/w	Name of validation parameter
Value	numeric	r/w	Value of validation parameter

Object #7: Data Analysis

Table 40 Object Header Items in _Config Register Object #7, Data Analysis

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title = Data Analysis
PreProcSteps	(table)	r/w	The possible spectral processing functions
IntRefWavel	(table)	r/w	The wavelength(s) used for internal reference
ScatterWavel	(table)	r/w	The wavelengths used for scatter correction
EvalTasks	(table)	r/w	The evaluation process
EquationParam	(table)	r/w	Parameters for equation evaluation
CompareParam	(table)	r/w	The parameters for comarison task
CalibParam	(table)	r/w	The calibration parameters
AnalWavel	(table)	r/w	The analytical wavelengths
RefWavel	(table)	r/w	The reference wavelengths

Table 41 Column Header Items in PreProcSteps and EvalTasks Tables

Header Item Name	Type/Range	Access	Meaning
Name	string	r/w	Name of the spectral processing type
MacroName	string	r/w	Name of macro that executes the spectral processing
FileName	string	r/w	File name of macro that executes the spectral processing
HelpIndex	integer	r/w	Index to help file
Description	string	r/w	Text decribing the process

Table 41 Column Header Items in PreProcSteps and EvalTasks Tables (continued)

Header Item Name	Type/Range	Access	Meaning
CmdClass	integer	r/w	Type of process: 50 = unitary 51 = binary 102 = multiple spectra
SelCond	string	r/w	Indicates number of spectra required for the command class: ≥1 for unitary =2 for binary ≥2 for multiple
NbrParam	integer	r/w	Number of parameters used by the function (1 to 5)
Param1 (...5)	numeric	r/w	Values of the parameters

Table 42 Column Header Items in AnalWavel, IntRefWavel,and ScatterWavel Tables

Header Item Name	Type/Range	Access	Meaning
Start	numeric	r/w	Start of wavelength range
To	numeric	r/w	End of wavelength range
Step	numeric	r/w	Sampling step for values within the specified wavelength range
Factor	numeric	r/w	Multiplication factor

Table 43 Column Header Items in EquationParam Table

Header Item Name	Type/Range	Access	Meaning
AnalyteName	string	r/w	Analyte name, user entered
Equation	string	r/w	Equation, user entered
Unit	string	r/w	Result units, user entered

Table 44 Header items of the CompareParam Table

Header Item Name	Type/Range	Access	Meaning
CmpFrom	numeric	r/w	Lower limit of wavelength range to be used for comparison task
CmpTo	numeric	r/w	Upper limit of wavelength range to be used for comparison task
NormalizePoint	numeric	r/w	Normalization wavelength for Compare (Normalization) task
IntRefWavel	numeric	r/w	Internal reference wavelength for Compare (Normalization) task
Threshold	numeric	r/w	Threshold for Compare (Regression) task
WavelengthShift	numeric	r/w	Wavelength shift in nm for Compare (regression) task

Table 45 Header Items in CalibParam Table

Header Item Name	Type/Range	Access	Meaning
CalibMethod	string	r/w	The calibration method SCA or MCA
CalibCurveType	string	r/w	The calibration curve type
WeightingMethod	string	r/w	The weighting method LSQ or MLH
StandardsSource	string	r/w	Source of standards to be used for the calibration, usually the Standards register

Table 46 Header Items in CompareParam Table

Header Item Name	Type/Range	Access	Meaning
AnalyteName	string	r/w	The names of the Analytes

Object #8: Report

Table 47 Object Header Items in _Config Register Object #8, Report

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title = Report
Left_Margin	numeric	r/w	The width of the left margin in number of characters, default is 8
Top_Margin	numeric	r/w	The depth of the top margin in number of lines, default is 6
Bottom_Margin	numeric	r/w	The height of the bottom margin in number of lines, default is 3
Gutter_Margin	numeric	r/w	The width of the gutter margin in number of characters, default is 3
Left_Indent	numeric	r/w	The width of the indent of the first line in a paragraph in characters, default is 4
PrinterID	numeric	r/w	The printer file slot
FileSlotBasis	numeric	r/w	The start file slot
FileSlot	numeric	r/w	The current file slot
MF_Width	numeric	r/w	The metafile graphics width
MF_Height	numeric	r/w	The metafile graphics height
GraphToFile	numeric or boolean	r/w	Flag indicating whther graph should be sent to printer or file: 0 = printer. 1 = file.
Print_Window	numeric	r/w	The number of the graphics window used to generate the graphic in the report
Point_Size	numeric	r/w	The character size

Table 47 Object Header Items in _Config Register Object #8, Report (continued)

Header Item Name	Type/Range	Access	Meaning
Max_Annotation	numeric	r/w	The maximum number of spectra that will be annotated when overlaid spectra are printed, default is 4
OK	string	r/w	Text used in report, default is OK
Cancel	string	r/w	Text used in report, default is Cancel
Config_Report	string	r/w	Text used in report, default is Configure Report
Font	string	r/w	Font used in report parameter dialog box, default is Helv
Mtd_Param_Dlg	string	r/w	Text used in report, default is Method Report Parameter
Cal_Param_Dlg	string	r/w	Text used in report, default is Calibration Report Parameter
Res_Param_Dlg	string	r/w	Text used in report, default is Result Report Parameter
Operator	string	r/w	Text used in report, default is The operator name
Last_Update	string	r/w	Text used in report, default is Last update
Modified	string	r/w	Text used in report, default is (modified)
MF_Prefix	string	r/w	Picture file prefix, default is Pic_
MF_Suffix	string	r/w	Picture file suffix for standard metafile, default is .SMF
MF_Incl_Begin	string	r/w	Start Text used in report to indicate meta file name, default is {
MF_Indl_End	string	r/w	End Text used in report to indicate meta file name, default is }
Res_Report_Hdr	(list)	r/w	Text used at start of report
End_Report	(list)	r/w	Text used at end of report

Table 47 Object Header Items in _Config Register Object #8, Report (continued)

Header Item Name	Type/Range	Access	Meaning
Res_Report	string	r/w	Text used in report, default is Results Report
NumSmps	string	r/w	Text used in report, default is Number of Samples
MethodHeader	string	r/w	Text used in report, default is Statistical Information
Data_Modified	string	r/w	Text used in report, default is (raw data modified)
MtdLSQID	(list)	r/w	Items used in report, default is StdErrorRegr, IndependStds
MtdSQLB	(list)	r/w	Text used in report, default is Std.Dev.Residual, Independence of Stds
MtdMLHID	(list)	r/w	Items used in report, default is StdErrorRegr, RelFitError, IndependStds
MtdMLHLB	(list)	r/w	Text used in report, default is Std. Dev. Residual, Rel. Fit error, Independence of Stds
MtdInfoID	(list)	r/w	Items used in report, default is RelFitError, IndependStds
MtdInfoLB	(list)	r/w	Text used in report, default is Rel. Fit error, Independence
SmpData	string	r/w	Text used in report, default isUsed Wavelength Results
Smp_Result	string	r/w	Text used in report, default isData Analysis Result
SmpInfoID	(list)	r/w	Items used in report, default is SampleName, Operator, Date, TimeOfDay, SolventName, PathLength, PathLengthUnit, Comment

Table 47 Object Header Items in _Config Register Object #8, Report (continued)

Header Item Name	Type/Range	Access	Meaning
SmpInfoLbl	(list)	r/w	Text used in report, default is Sample Name, Operator, Date, Time, Solvent, Path Length, Path Length Unit, Comment
SpecSmpLB	string	r/w	Text used in report, default isSample Spectrum
SpecSmpProLB	string	r/w	Text used in report, default isProcessed Sample Spectrum
Smp_Residual	string	r/w	Text used in report, default isResidual Spectrum
SpecOverlayLB	string	r/w	Text used in report, default is overlaid
Res_Summary	string	r/w	Text used in report, default isResults Summary
Smp_Comp_Stat	string	r/w	Text used in report, default is Evaluation Results Statistics
Method	string	r/w	Text used in report, default is Method Report
Method_File	string	r/w	Text used in report, default isMethod file
Product	string	r/w	Text used in report, default is Product
Version	string	r/w	Text used in report, default is Version
NumAux	string	r/w	Text used in report, default is Auxiliary Spectra
Report_Param	string	r/w	Text used in report, default is Report
Calibration	string	r/w	Text used in report, default is Calibration Report
NumStds	string	r/w	Text used in report, default is Number of Standards

Table 47 Object Header Items in _Config Register Object #8, Report (continued)

Header Item Name	Type/Range	Access	Meaning
Calib_Table	string	r/w	Text used in report, default is Calibration Table of
Calib_Data	string	r/w	Text used in report, default is Used Wavelength Results
Calib_Coeff	string	r/w	Text used in report, default is Coefficients
SpecStd	string	r/w	Text used in report, default is Standard Spectra
SpecStdProc	string	r/w	Text used in report, default is Processed Standard Spectra
Std_Residual	string	r/w	Text used in report, default is Residual Spectra
Pure_Spectra	string	r/w	Text used in report, default is Pure Analyte Spectra
CalibCurve	string	r/w	Text used in report, default is Calibration Curve
T_Tab	(table)	r/w	Translation table

Table 48 Column Header Items in T_Tab Table

Header Item Name	Type/Range	Access	Meaning
KeyWord	string	r/w	key word
Translation	string	r/w	translation of key word

DataAnalysis_Param_1 (...4)

These registers contain the data analysis parameters for data analysis methods 1 to 4. There is one object in each register for a data analysis which does not use an SCA or MCA method or is not calibrated. A calibrated SCA data analysis contains an additional 7 objects. A calibrated MCA data analysis contains an additional 5 objects and one for each component.

Objects in DataAnalysis_Param_X Registers

Table 49 Objects in the DataAnalysis_Param Register

#	Title	Description
1	Data Analysis Param 1(...4)	Contains the parameters for the data analysis as shown above
2	(not set)	The calibration results
3	(not set)	For internal use
4	(not set)	For internal use
5	(not set)	SCA: Information for the data points in the calibration graph
6	(not set)	SCA: Information for the calibration curve line in the calibration graph. MCA: Pure component spectrum of component 1.
7	(not set)	SCA: Information for the lower confidence interval line in the calibration graph. MCA: Pure component spectrum of component 2.
8	(not set)	SCA: Information for the upper confidence interval line in the calibration graph. MCA: Pure component spectrum of component 3.

Object #1 in DataAnalysis_Param_X

Table 50 Object Header Items in _DataAnalysis_Param_1 (...4) Register Object #1

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title = Data Analysis Param 1
Nbr	integer	r/w	The number of the data analysis
Ident	string	r/w	The identity of tha data analysis, user entered
Calibrated	boolean	r/w	Flag to indicate if the method is calibrated or not: 0 = no, 1 = yes.
AuxLinks	boolean	r/w	Flag to indicate if Auxiliary spectra are used: 0 = no, 1 = yes
ProcessCap		r/w	Capability of method
PreProc_WL	(table)	r/w	Spectral processing functions
Analytical	(table)	r/w	Type of analytical function
AnalWavel	(table)	r/w	Analytical wavelengths
AFWSINGLE	(table)	r/w	Use wavelengths, single wavelength
AFWLIST	(table)	r/w	Use wavelengths, wavelength list
AFWRANGE	(table)	r/w	Use wavelengths, wavelength range
AFWFUNC	(table)	r/w	Use wavelengths, analytical function
RefWavel	(table)	r/w	Reference wavelengths and function
Evaluation	(table)	r/w	Type of evaluation, Equation, SCA or MCA
EquationParam	(table)	r/w	Equation evaluation parameters, only present if equation evaluation is selected
AnalyteNames	(table)	r/w	Analyte names , only present if an evaluation step is selected

Table 50 Object Header Items in _DataAnalysis_Param_1 (...4) Register Object #1 (continued)

Header Item Name	Type/Range	Access	Meaning
StandardsNames	(table)	r/w	Standards names, only present if an evaluation step is selected
IntRefWavel	(table)	r/w	Wavelengths used for Internal Reference function, only present if Internal Reference is selected as a spectral processing step
ScatterWavel	(table)	r/w	Wavelengths used for Scatter Correction function, only present if Scatter Correction is selected as a spectral processing step
Calibparam	(table)	r/w	Analyte names and concentration unit

Table 51 Column Header Items in PreProc_WL and Evaluation Tables

Header Item Name	Type/Range	Access	Meaning
Name	string	r/w	Name of spectral processing function
MacroName	string macro	r/w	Macro name of function
Param1 (...5)	numeric	r/w	Value of the parameters used by the function (0 to 5)

Rows = number of spectral processing steps.

Table 52 Column Header Items in Analytical Table

Header Item Name	Type/Range	Access	Meaning
Function	integer	r/w	0 = None 1 = Single 2 = List 3 = Range 4 = Analytical function

Table 53 Column Header Items inRefWavel, IntRefWavel, ScatterWavel, AnalWavel, AFWSINGLE, AFWLIST, AFWFUNC and AFWRANGE Tables

Header Item Name	Type/Range	Access	Meaning
Start	numeric	r/w	Start of wavelength range
To	numeric	r/w	End of wavelength range
Step	numeric	r/w	Sampling step for values within the specified wavelength range
Factor	numeric	r/w	Multiplication factor

Rows = number of wavelengths or wavelength ranges.

Table 54 Header Items in RefWavel Table

Header Item Name	Type/Range	Access	Meaning
FuncOperation	string	r/w	Operation type: Noreference Subtract Multiply or Divide
PrevFuncOp	string	r/w	For internal use

Table 55 Column Header Items in EquationParam Table

Header Item Name	Type/Range	Access	Meaning
AnalyteName	string	r/w	Analyte name, user entered
Equation	string	r/w	Equation, user entered
Unit	string	r/w	Result units user, entered

Rows = 4 (for up to 4 user entered equations).

Table 49 on page 233 shows the objects in a calibrated SCA DataAnalysis_Param register.

Object #2 in DataAnalysis_Param_X Registers

Object structure in case of SCA analysis:

Table 56 Object Header Items in DataAnalysis_Param_1 (...4) Register Object #2

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title (not set)
SCA_CalibRes	(table)	r	Results of calibration
SCA_Summary_Tab	(table)	r	Summary table of SCA results for display
DataCols	data block	r	Calibration coefficients

Table 57 Header Items in SCA_CalibRes Table

Header Item Name	Type/Range	Access	Meaning
AnalyteName	string	r	Analyte name, user entered
Unit	string	r	Result unit, user entered
PathlengthUnit	string	r	Path length unit, default is cm
NumbStandards	numeric	r	Number of standards
CalibCurveType	string	r	The calibration curve type used
WeightingMethod	string	r	The weighting method used
RelFitErr	numeric	r	The relative fit error value
StdErrorRegr	numeric	r	The standard error of regression value
CorrelCoeff	numeric	r	The correlation coefficient value
Uncertainty	data block	r	The uncertainty value

Table 58 Column Headers in SCA_CalibRes Table

Header Item Name	Type/Range	Access	Meaning
StandardName	string	r	Standard name, user entered
AnalFuncRes	numeric	r	Analytical function result
AnalFuncStdDev	numeric	r	Standard deviation of analytical function result
AnalyteValue	numeric	r	Analyte value, normally concentration
AnalyteStdDev	numeric	r	Standard deviation of analyte value
PathLength	numeric	r	Path length
FittedValue	numeric	r	calculated result
Residual	numeric	r	Residual
PerCentError	numeric	r	Percent error
ConfIntervall	numeric	r	95% Confidence interval
Leverage	numeric	r	Leverage
StudentResidual	numeric	r	Studenized residual
CooksDistance	numeric	r	Cooks distance

Object structure in case of MCA analysis:

Table 59 Object Header Items in DataAnalysis_Param_1 (...4) Register Object #2

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 2 = matrix
Title	string	r/w	not set
MCA_CalibRes	(table)	r	Standard deviation of Calibration
PathLengthUnit	string	r	Path length units default = cm
DataCols	data block	r	Calibration matrix

Table 60 Column Header Items in MCA_CalibRes Table

Header Item Name	Type/Range	Access	Meaning
AnalyteName	string	r	Analyte name, user entered
Unit	string	r	Analyte unit, user entered
StandErr	numeric	r	Standard error of calibration

Eval_Results

This register contains the summary of results from 1 to 4 analysis methods. It contains one object.

Table 61 Object Header Items in Eval_Results Register Object #1

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 2 = matrix
Title	string	r/w	Object title not set
Allresults	(table)	r/w	Results summary table
Statistics	(table)	r/w	Evaluation results statistics

Table 62 Column Headers in AllResults Table

Header Item Name	Type/Range	Access	Meaning
SampleName	string	r	Sample name, user entered.
AnalyteName	string	r	Name of the analyte in the sample
MethodType	string	r	Evaluation type and method number. For example, SCA 1 would be the confirmation method 1 using single component quantification.
AnalFuncRes	numeric	r	The analytical function result used for the evaluation step
Value	numeric	r	The value of the evaluation result
StdDev	numeric	r	The standard deviation of the result
Unit	numeric	r	The units of the result
AllreConfirmState	string	r	The confirmation state. If confirmation method is out of tolerance it contains >x%, where x is the user entered tolerance.

Rows = (number of samples) × (number of analytes) × (number of methods)

Table 63 Column Headers in Statistics Table

Header Item Name	Type/Range	Access	Meaning
AnalyteName	string	r	Name of the analyte in the sample
MethodType	string	r	Evaluation type and method number. For example, SCA 1 would be the confirmation method 1 using single component quantification.
Average	numeric	r	The average result for all samples
StdDev	numeric	r	The standard deviation of the average
RelError	numeric	r	The relative standard deviation of the average in %
Minimum	numeric	r	The minimum result of all samples
Maximum	numeric	r	The maximum result of all samples
Unit	numeric	r	The units of the result

Rows = (number of analytes) × (number of methods)

Eval_Results_1 (...4)

These registers contain the evaluation results for data analysis methods 1 to 4. The number of objects is equal to the number of sample spectra.

Table 64 Object Header Items in Eval_Results_1 (...4) Register

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 5 = spectrum
Title	string	r/w	Object title (not set)
SCA_QuantRes	(table)	r	Quantitative results (SCA only)
AnalyteTable	(table)	r	The results for each analyte in the sample
StdErrRegr	numeric	r	The relative fit error of regression for the calibration (MCA only)
RelFitError	numeric	r	The relative fit error for the calibration (MCA only)
IndependStds	numeric	r	The independence of the standards (MCA only)
DataType	enumeration	r	Data type of spectrum: 0(UNK): unknown 1(ABS): absorbance 2(INT): intensity 3(TRANS): transmittance 4(FLUOR): fluorescence, phosphorescence, or chemiluminescence
DilutionFactor	numeric	r	The independence of the standards
SampleName	string	r	The sample name user entered

Table 64 Object Header Items in Eval_Results_1 (...4) Register (continued)

Header Item Name	Type/Range	Access	Meaning
Pathlength	numeric	r	The cell path length (MCA only)
PathlengthUnit	string	r	The units of the cell path length (MCA only)
DataCols	(data block)	r	The residual spectrum (MCA only)

Table 65 Column Header Items in Analyte Table

Header Item Name	Type/Range	Access	Meaning
AnalyteName	string	r	Analyte name, user entered
Value	numeric	r	Calculated result
StdDev	numeric	r	Standard deviation of the calculated result
Unit	string	r	Units of the result
PathValue	numeric	r	Result corrected for path length
ValueCorrPathl	numeric	r	Result corrected for path length and dilution
PathStdDev	numeric	r	Standard deviation of the result corrected for path length
PathCorrStdDev	numeric	r	Standard deviation of the result corrected for path length and dilution

Rows = number of analytes.

Table 66 Header Items in SCA_QuantRes Table

Header Item Name	Type/Range	Access	Meaning
Unit	string	r	Result units
PathlengthUnit	string	r	Cell path length units

Table 67 Column Header Items in SCA_QuantRes Table

Header Item Name	Type/Range	Access	Meaning
SampleName	string	r	Sample name, user entered
AnalyteName	string	r	Analyte name, user entered
AnalFuncRes	numeric	r	Used wavelength result
AnalFuncStdDev	numeric	r	Standard deviation of the used wavelength result
Value	numeric	r	Result
PredInterval	numeric	r	Prediction interval of the result
PathValue	numeric	r	Result corrected for path length
ValueCorrPathI	numeric	r	Result corrected for path length and dilution
PathStdDev	numeric	r	Standard deviation of the result corrected for path length
PathCorrStdDev	numeric	r	Standard deviation of the result corrected for path length and dilution
PathPredict	numeric	r	Prediction interval of the result corrected for path length
PathCorrPredInt	numeric	r	Prediction interval of the result corrected for path length and dilution

Eval_Results_Std_1 (...4)

This registers contain the same information for the standard spectra as the Eval_Results_1(...4) registers for the samples. They are using the identical object structure as the corresponding samples registers.

FloatMenus

This register contains information related to the different tasks, menus and graphics

WARNING **This register is for internal use only!**
Do not change register contents.

GlobVars

This register contains global variables used by the software.

WARNING **This register is for internal use only!**
Do not change register contents.

Meth_Descript

This register contains the user eneterd description and other information for the current method.

WARNING **This register is for internal use only!**
Do not change register contents.

MODEADMIN

System information to handle the different task modes.

WARNING **This register is for internal use only!**
Do not change register contents.

PP_Work

Temporary register during processing of spectra. For the object structure of spectral data see Table 75 on page 255.

ProcessedSamples_1 (...4)

These registers contain the spectral data of samples after processing by methods 1 to 4. Each of the objects in these registers have a corresponding object in the samples register. For the object structure of spectral data see Table 75 on page 255.

ProcessedStandards_1 (...4)

These registers contain the spectral data of standards after processing by methods 1 to 4. Each of the objects in these registers have a corresponding object in the standards register. For the object structure of spectral data see Table 75 on page 255.

Report_Param

This register contains the parameters for report generation. It includes three objects for methods, results and calibration reports.

Object #1: Method Results Parameters

Table 68 Object Header Items in Report_Param Register Object #1, Method Results Parameters

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title = Method Report Parameter
G1100A	(list)	r/w	List of general scanning report options: Information Checklist Instruments/Acquisition Data Analysis Report
G1101A	(list)	r/w	List of quantification method report options: Information Checklist Instruments/Acquisition Data Analysis Report Include Calibration Report

Table 68 Object Header Items in Report_Param Register Object #1, Method Results Parameters (continued)

Header Item Name	Type/Range	Access	Meaning
Parameters	(list)	r/w	Flags to indicate if an option has been selected: 0 = no, 1 = yes default values: 1,1,1,1,1,0
Help	numeric	r/w	Help context index

Object #2: Results Report Parameter

Table 69 Object Header Items in Report_Param Register Object #2, Results Report Parameter

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title = Results
G1100A	(list)	r/w	List of general scanning results report options: Sample Information Sample Spectra Processed Sample Spectra Used Wavelength Results Evaluation Results Confirmation Results Include Method Report All Sample Spectra Overlaid Summary

Table 69 Object Header Items in Report_Param Register Object #2, Results Report Parameter (continued)

Header Item Name	Type/Range	Access	Meaning
G1101A	(list)	r/w	List of quantification results report options: Sample Information Sample Spectra Processed Sample Spectra Used Wavelength Results Evaluation Results Statistical Information Residual Spectra Confirmation Results Include Method Report All Sample Spectra Overlaid Summary Evaluation Results Statistics
Parameters	(list)	r/w	Flags to indicate if an option has been selected: 0 = no, 1 = yes default values: 0,0,0,0,0,0,0,0,0,1,1,1
Help	numeric	r/w	Help context index

Object #3: Calibration Report Parameter

Table 70 Object Header Items in Report_Param Register Object #3, Calibration Report Parameter

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title: Calibration Report Parameter
G1100A		r/w	No report options

Table 70 Object Header Items in Report_Param Register Object #3, Calibration Report Parameter (continued)

Header Item Name	Type/Range	Access	Meaning
G1101A	(list)	r/w	List of calibration report options: Standard Spectra Processed Standard Spectra Path Length Table Data Analysis Parameters Coefficients Used Wavelength Results Calibration Table(s) of Analytes Curve Residual Spectra Diagnostics
Parameters	(list)	r/w	Flags to indicate if an option has been selected: 0 = no, 1 = yes default values: 1,1,1,1,1,1,1,1,1,1
Help	numeric	r/w	Help context index

Report_Status

This register contains information on the current report and is used temporary. It contains three objects.

Object#1: Path Length Table

Table 71 Header Items in Path Length Table Object

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title = Path Length Table

Object#2: Calib Table

Table 72 Header Items in Calib Table Object

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title = Calib Table

Object#3: Graphics

Table 73 Header Items in Graphics Object

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified.
Title	string	r/w	Object title = Graphics
GraphicsTable	(table)	r/w	Destination for report to file

SampleLog

The SampleLog register contains the sample log table for automation. It contains one Object per sample (only if entered by the user).

Table 74 Object Header Items in SampleLog Register

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title (not set)
SampleName	string	r/w	Sample name, user entered
SolventName	string	r/w	Solvent name, user entered
DilutionFactor	numeric	r/w	dilution factor, user entered
Comment	string	r/w	Comment text, user entered
AnalyteTable	(table)	r/w	The values for each analyte in the sample (Table 65)

Samples

The register contains the sample spectra. The number of objects in the register is equal to the number of spectra.

Table 75 Object Header Items of a Spectrum Object

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 5 = spectrum
Title	string	r/w	Object title (not set)
SampleName	string	r/w	The sample name, user entered
DataType	enumeration	r	Data type of spectrum: 0(UNK): unknown 1(ABS): absorbance 2(INT): intensity 3(TRANS): transmittance 4(FLUOR): fluorescence, phosphorescence, or chemiluminescence
DerivOrder	integer ≥1	r	Derivative order, not present if order is 0
InstrId	string	r	Instrument type identification
InstrNbr	numeric	r	Individual instrument number (user adjustable)
IntegrTime	numeric	r	The spectrophotometer integration time in seconds used for the measurement
OptBandwidth	numeric	r	The optical bandwidth of the spectrophotometer
TimeOfDay	string	r	The time when the spectrum was measured

Table 75 Object Header Items of a Spectrum Object (continued)

Header Item Name	Type/Range	Access	Meaning
Date	string	r	The date when the spectrum was measured
TimeSince1970	string	r	computer time stamp
RelTime	numeric	r	(for future use)
CellTemp	numeric	r	Current temperature of peltier controlled cell holder (only if temperature controller is online)
ExtTemp	numeric	r	Current temperature of external temperature sensor (only if temperature controller is online)
AnalyteTable	(table)	r/w	The results for each analyte in the sample (see Table 65)
SolventName	string	r/w	The solvent name, user entered
Comment	string	r/w	Comment, user entered
DilutionFactor	numeric	r/w	Dilution factor, user entered
PathLength	numeric	r	The cell path length
PathLengthUnit	string	r	The units of the cell path length
TempUnit	string	r	The units of the temperature values (only if temperature controller is online)
Operator	string	r	The operator name
DataCols	data block		The spectral data wavelength, absorbance and standard deviation for each data point

Samples_App

Buffers sample spectra from the spectrophotometer temporarily. For the object structure of spectral data see Table 75 on page 255.

SamplSys_Param

This register contains the current parameters for the selected sampling system. It contains one object.

Object#1: SamplingSystem

Table 76 Object Header Items in SamplSys_Param Register

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object name = SamplingSystem
PumpDir	string	r/w	Direction of pump: CW is clockwise,CCW is counter-clockwise
PumpTime	numeric	r/w	Pumping time in seconds, default is 20
WashTime	numeric	r/w	Wash time in seconds, default is 0
WaitTime	numeric	r/w	Wait time in seconds, default is 3
SampleReturn	numeric ≥0, ≤100	r/w	Return time in seconds, default is 0
AirSegment	numeric	r/w	Air segment time in seconds, default is 0

Spectro_Param

This register contains the parameters for data acquisition from the spectrophotometer. It contains one object.

Table 77 Object Header Items in Spectro_Param Register Object #1

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object name is 8452A
WavTable	(table)	r/w	The wavelengths for data acquisition if a wavelength list has been selected
DataType	integer	r/w	Type of data, default is 1 (absorbance)
GainSelect	integer	r/w	Specifies whether gain is set automatically during the blank measurement or fixed gain should be used: 0 = automatic 1 = fixed
Gain	integer 0-15	r/w	Value of the gain when fixed gain has been selected
GainAdjust	integer 0/1	r/w	Switch to set automatic gain setting: off = 0, on = 1(default)
LowerWL	integer	r/w	Lower wavelength limit in nm, default is : 190 HP 8452A 190 HP 8452A option #002 190offline 470 HP 8452A option #003

Table 77 Object Header Items in Spectro_Param Register Object #1 (continued)

Header Item Name	Type/Range	Access	Meaning
UpperWL	integer	r/w	Upper wavelength limit in nm, default is: 820 HP 8452A 510 HP 8452A option #002 1100 offline 1100 HP 8452A option #003
IntegrTime	numeric	r/w	Integration time for measurement in seconds, default is 0.5
CycleTime	numeric	r/w	The cycle time between measurements in seconds if time based measurements are selected, default is 1
DelayTime	numeric	r/w	The delay time before measurement starts in seconds if time based measurements are selected, default is 0
RunTime	numeric	r/w	The run time over which measurements are made in seconds if time based measurements are selected, default is 1
ShutterOpen	boolean	r/w	Flag to control whether shutter should stay open between measurements if time based measurements are selected: 0 = no, 1 = yes (default)
VarianceOn	boolean	r/w	Flag to control whether variance data should be calculated and transferred from the spectrophotometer: 0 = no, 1 = yes (default)
LampOn	boolean	r/w	Flag to control the deuterium lamp on/off status: 0 = no, 1 = yes (default)
MKEmode	boolean	r/w	Flag to control the lamp on/off status: 0 = no, 1 = yes (default)

Table 77 Object Header Items in Spectro_Param Register Object #1 (continued)

Header Item Name	Type/Range	Access	Meaning
MKEStirringOn	boolean	r/w	Flag to control the lamp on/off status: 0 = no, 1 = yes (default)
HoldTime	numeric	r/w	Initial hold time where the cycle time is not increased by the cycle factor
CycleFactor	numeric	r/w	Factor to increase the interval time between measurements
MatrixEnabled	boolean	r/w	Flag to control the lamp on/off status: 0 = no, 1 = yes (default)
Interval	boolean	r/w	time between measurements
TungstenOn	boolean	r/w	Flag to control the tungsten lamp on/off status: 0 = off, 1 = on (default)
StraylightCorr	boolean	r/w	Flag to control the straylight correction status: 0 = off, 1 = on (default)

Table 78 Column Header Items in WavTable Table

Header Item Name	Type/Range	Access	Meaning
WL	numeric	r/w	Wavelength for data acquisition

Spectro_Status

This register contains infomation on the current status of the spectrophotometer. It contains one object.

Table 79 Object Header Items in Spectro_Status Register Object #1

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title (not set)
InstrId	string	r/w	The instrument identification string
InstrNbr	integer 0 to 63	r/w	The instrument identity number read from the switch block inside the spectrophotometer
SamplingInterv	numeric	r/w	The sampling interval in nm of the spectrophotometer: 2 HP 8452A 2 HP 8452A option #003 1 HP 8452A option#002
MinWL	numeric	r/w	The lower wavelength limit in nm of the spectrophoto-meter: 190 HP 8452A 190 HP 8452A option #002 470 HP 8452A option #003 190 Agilent 8453
MaxWL	numeric	r/w	The upper wavelength limit in nm of the spectrophoto-meter: 510 HP 8452A #002 820 HP 8452A 1100 HP 8452A #003 1100 Agilent 8453
PowerOnTime	numeric	r/w	Time in seconds since the spectrophotometer was switched on

Table 79 Object Header Items in Spectro_Status Register Object #1 (continued)

Header Item Name	Type/Range	Access	Meaning
LampState	boolean	r/w	Deuterium lamp status: 0 = off, 1 = on
TungstenState	boolean	r/w	Tungsten lamp status: 0 = off, 1 = on
MeasureState	boolean	r/w	status: 0 = off, 1 = on
Reference	boolean	r/w	Flag to indicate if a valid blank has been measured: 0 = no, 1 = yes
MinIntTime	numeric	r/w	Minimum integration time in seconds (0.1 default)
MaxIntTime	numeric	r/w	Maximum integration time in seconds (25.5 default)
InstState	boolean	r/w	Status of the spectrophoto-meter: 1 = offline 2 = not ready 4 = power fail 8 = ready 16 = error
ErrorString	string	r/w	If an error exists, the error text string from the spectrophotometer

Standards

The register contains the standard spectra. The number of objects in the register is equal to the number of spectra. Its structure is identical to the samples register.

Standards_App

Buffers standard spectra from the spectrophotometer temporarily. For the object structure of spectral data see Table 75 on page 255.

StatMon

This register contains the information on the system status. It contains one object.

Table 80 Object Header Items in StatMon Register Object #1

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title (not set)
TabDAStat	(table)	r	The Data Analysis status
DDTDaStat	string	r	Defines the appearance of the Data Analysis part of the Status Monitor window

Table 81 Object Header Items in TabDAStat Table

Header Item Name	Type/Range	Access	Meaning
Subject	string	r	Items in method window of the status monitor: Spectral Processing Use Wavelengths Evaluation Analyzed Calibrated
Method1 (...4)	string	r	Up to four data analysis methods

TaskFuncRes_Smp

Temporary register used by the Optimize Wavelength task.

Table 82 Object Header Items in TaskFuncRes_Smp Register Object #1

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 2 = matrix
Title	string	r/w	Object title (not set)
InvalidData	string	r	Is TRUE in case of invalid data in some spectra
SampleNames	(table)	r	The sample names, user entered
DataCols	(data block)	r	The spectral data; wavelength, absorbance, and standard deviation for each data point

TaskFuncRes_Std

The TaskFuncRes_Std register contains the results of the Evaluate Standards task.

Table 83 Object Header Items in TaskFuncRes_Std Register Object #1

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 2 = matrix
Title	string	r/w	Object title (not set)
InvalidData	string	r	Is TRUE in case of invalid data in some spectra
SampleNames	(table)	r	The sample names, user entered
DataCols	(data block)	r	The spectral data; wavelength, absorbance and standard deviation for each data point

Task_Result

The Task_Result register contains the results of any task performed. The contents of the object vary depending upon the task which has been performed.

Find Peaks/Valleys

The Task_Result Find Peaks/Valleys register contains one object. It is a copy of the selected spectrum with two tables with the results of the peak and valley find operation added. For the object structure of spectral data see Table 75 on page 255.

Table 84 Additional Object Header Items in Task_Result Register Object #1

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 5 = spectrum
...	...	...	...
MinTable	(table)	r	The valleys found
MaxTable	(table)	r	The peaks found.
DataCols	data block	r	The spectral data; wavelength, absorbance and standard deviation for each data point

Table 85 Column header Items in MinTable and MaxTable Tables

Header Item Name	Type/Range	Access	Meaning
X	numeric	r	The wavelength in nm of the minimum or maximum
Y	numeric	r	The amplitude value of the minimum or maximum

Rows = number of peaks or valleys found.

Compare Normalization

The Compare Normalization creates two objects. Both objects are spectra. The first one is the residual data spectrum. It uses the primary object structure of a spectrum as described in Table 75 on page 255. In addition the two header items indicated in Table 86 are added. The second object is a pseudo spectrum indicating zero deviation.

Table 86 Additional Object Header Items in Task_Result Register Object #1

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 5 = spectrum
...	...	...	...
WindowTitle	string	r/w	The window title is Compare(Normalization) <Spectrum A> against <Spectrum B>
CmpnormTab	(table)	r/w	Compare results summary
DataCols	data block	r	The residual spectral data; wavelength, absorbance and standard deviation for each data point

Table 87 Column header Items in CmpnormTab Table

Header Item Name	Type/Range	Access	Meaning
Name	string	r	Description
Value	string	r	Actual value

Table 88 Object Header Items in Task_Result Register Object #2

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 5 = spectrum
Title	string	r/w	Object title (not set)
SampleName	string	r/w	The sample name Zero is set
DataCols	(data block)	r	zero values in the specified wavelength range

Compare Regression

The Task_Result register used with the Compare Regression operation contains four objects. Object#1are the regression data along with a Compare Regression Summary table. Object#2 is a pseudo spectrum containing the calculated regression curve. Object#3 contains the residual spectrum. Object#4 is the zero spectrum described in Table 88.

Table 89 Object Header Items in Task_Result Register Object #1

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 5 = spectrum
Title	string	r/w	Object title (not set)
WindowTitle	string	r/w	The window title is <Spectrum A> versus <Spectrum B>

Table 89 Object Header Items in Task_Result Register Object #1 (continued)

Header Item Name	Type/Range	Access	Meaning
SampleName	string	r/w	The sample name is Spectrum versus Spectrum
CmpRegrTab	(table)	r	The results of the Compare (Regression) task
DataCols	(data block)	r	Spectral data of <Spectrum A> against spectral data of <Spectrum B> at corresponding wavelength

For the structure of the CmpRegrTab see Table 87.

Table 90 Object Header Items in Task_Result Register Object #2

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 5 = spectrum
Title	string	r/w	Object title (not set)
SampleName	string	r/w	The sample name is Calculated regression
DataCols	(data block)	r	Data of the regression line in graphic

Table 91 Object Header Items in Task_Result Register Object #3

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 5 = spectrum
Title	string	r/w	Object title (not set)
SampleName	string	r/w	The sample name = Residual

Table 91 Object Header Items in Task_Result Register Object #3 (continued)

Header Item Name	Type/Range	Access	Meaning
WindowsTitle	string	r/w	The window title is Compare (Regression): <Spectrum A> – x.xxxxxx + y.yyyyyyy<Spectrum B>
DataCols	(data block)	r	The residual spectral data; wavelength, absorbance and standard deviation for each data point

Task_Temp

This regiser is used by the Compose task temporarily.

Temco_Param

This register holds the set points for the peltier temperature controller.

Object#1: 89090A

Table 92 Object Header Items in Temco_Param Register Object #1

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title = 89090A
SetTemp	numeric	r/w	Set temperature for cell holder
SetTempLow	numeric	r/w	The minimum allowed settable temperature
SetTempHigh	numeric	r/w	The maxium allowed settable temperature
StirrerSpeed	numeric	r/w	The stirrer speed in rpm default is 40
TempUnit	enumeration	r/w	The temperature unit being used: 0 for Celsius, 1 for Kelvin 2 for Fahrenheit

Table 92 Object Header Items in Temco_Param Register Object #1 (continued)

Header Item Name	Type/Range	Access	Meaning
Stirrer	boolean	r/w	Flag to switch stirrer on/off: 0 for off, 1 for on (default)
Peltier	boolean	r/w	Flag to switch Peltier on/off: 0 for off, 1 for on (default)

Temco_Status

This register contains information about the current status of the 89090A peltier temperature controller.

Object#1: 89090A

Table 93 Object Header Items in Temco_Status Register Object #2

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title = 89090A
InstrID	string	r	Instrument identification string
ExtTemp	numeric	r	Temperature of the external sensor
ExtSensor	boolean	r	Flag to indicate if external sensor is connected: 0 = no, 1 = yes
ExtTempText	string	r	Current sensor temperature and units
CellTempText	string	r	Current cell holder temperature and units
Celltemp	numeric	r	Cell holder temperature
StirrerState	boolean	r	Flag to indicate stirrer status: 0 = off, 1 = on
PeltierState	boolean	r	Flag to indicate Peltier status: 0 = off, 1 = on
RemoteMode	boolean	r	Flag to indicate if Peltier controller is in remote mode: 0 = no, 1 = yes

Table 93 Object Header Items in Temco_Status Register Object #2 (continued)

Header Item Name	Type/Range	Access	Meaning
InstState	enumeration	r	Status of the Peltier controller: 1 = off line 2 = not ready 4 = power fail 8 = ready 16 = error
ErrorString	string	r	If there is an error the text string of the error message from the Peltier controller

TestMethod_Result

The TestMethod_Result register contains the results of the Test Method task. This register contains one to four objects, all identical in structure, dependend upon whether single or multiple data analysis have been set.

Table 94 Object Header Items in TestMethod_Result Register Object

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 1 = user specified
Title	string	r/w	Object title (not set)
Regress	(table)	r	Results of linear regression
WindowTitle	string	r/w	The window title = Data Analysis - Mean of xxxxx
SampleName	string	r/w	The sample name, user entered
Statistics	(table)	r	Summary of all regression results
DataCols	(data block)	r	Results of samples

Table 95 Header Items in Regress Table

Header Item Name	Type/Range	Access	Meaning
Type	string	r	Type of regression
tPercentage	numeric	r	Percentage point of t distribution
CorrCoeff	numeric	r	Correlation coefficient
StdDev	numeric	r	Standard deviation of regression
K1	numeric	r	Parameter K1 of the regression curve
StdDevK1	numeric	r	Standard deviation of K1
RelStdDevK1	numeric	r	Relative standard deviation of K1

Table 96 Column Header Items in Regress Table

Header Item Name	Type/Range	Access	Meaning
Name	string	r	Name of statistical value: Type of regressio Percentage point of t distribution Correlation coefficient Standard deviation of regression Parameter K1 of the regression curve Standard deviation of K1 Confidence interval of K1 Relative standard deviation of K1
Value	numeric	r	The value of the named statistic

Rows = number of statistical values

Table 97 Table header Items in Statistics Table

Header Item Name	Type/Range	Access	Meaning
AnalyteName	string	r	The analyte name, user entered
MethodType	string	r	The Data Analysis Method SCA/MCA
Value	numeric	r	The mean value
StdDev	numeric	r	The standard deviation of the mean value
Unit	string	r	The unit of the result values

Rows = (number of data analysis) x (number of analytes)

WLResult_Smp_1(...4)

These registers contain the data specified with the Use Wavelength parameters for all samples. Dependent upon the active data analysis up to four registers are used. Two objects are created. The first object contains all sample data used in analysis and the second object the corresponding standard deviations. The objects are organized such that the object number of the sample in the samples register corresponds to the row index in the data matrix in the WLResult_Smp register. The columns are representing the data values specified with the Use Wavelength parameters. Function result values (used in SCA), are accesible in column -1.

Table 98 Object Header Items in WLResult_Smp_1 (...4) Register Object#1

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 2 = matrix
Title	string	r/w	Object title (not set)
InvalidData	string	r	Is TRUE in case of invalid data in some of the spectra
SampleNames	(table)	r	The sample name, user entered
DataCols	(data block)	r	Used Wavelength values
DataRows	(data block)	r	Samples

Table 99 Object Header Items in WLResult_Smp_1 (...4) Register Object#2

Header Item Name	Type/Range	Access	Meaning
ObjClass	integer	r	Type of object 2 = matrix
Title	string	r/w	Object title (not set)
InvalidData	string	r	Is TRUE in case of invalid data in some of the spectra

Table 99 Object Header Items in WLResult_Smp_1 (...4) Register Object#2 (continued)

Header Item Name	Type/Range	Access	Meaning
SampleNames	(table)	r	The sample name, user entered
DataCols	(data block)	r	Used Wavelength standard deviations
DataRows	(data block)	r	Samples

WLResult_Std_1 (...4)

These registers contain the results of the Use Wavelengths function for the standards for one to four analyses. Their structures are identical to the WLResult_Smp_1 (...4) registers.

Index

D

E

F

G

I

L

M

N

O

P

R

S

T

U

V

W

Z

www.agilent.com

In This Book

This book describes how you work with commands to customize your Agilent ChemStation for UV-visible spectroscopy, making its operation more flexible. It explains programming techniques and uses frequent examples to show how these techniques work in actual applications.

Printed in Germany
10/2003

G1116-90006

Agilent Technologies