DSWindows 2.2 Macros

User Guide

Issue 1

March 1997

NOTICE

Notice

CONTENTS

Alphabetic Index to Commands and Instructions vii

About this guide ix

What's new in DSWindows 2.2 ix
Who should use this guide xi
What you need to know xi
How to use this guide xii
Conventions xiii
Further information xiv

Introduction 1

What are Datastream macros? 2
Commands 2
Instructions 4
Labels 4
Constants and variables 4
Functions and expressions 5
Data files 5
Creating macros 8
Using the macro recorder 9
Editing macros 10
Running macros 10
Scheduling your macros using DSAGENDA 11
Summary of general macro rules 15
Tips on writing and editing macros 16

Macro commands and instructions 17

Conventions used in this section 18
Sending data to Datastream 19
Printing 25
Paging 26
Instructions (commands which control the flow of a macro) 27
Graphics 41
Graphic annotations 52
- Coordinates 53
- Using quotes 53
- Selecting/deselecting 55
- Amending items 59
- Creating new items 63
- Redrawing 69

Data Channel/Fundline 70
Capturing text 74
Arranging windows 76
- Activating windows 76
- Closing windows 76
- Minimizing windows 76
- Maximizing windows 77
- Restoring windows 77

Connecting to Datastream 78
Error recovery, logging errors, writing to file 82
Miscellaneous 85

Constants and variables 87

Introduction 87
Constants 87
Variables 92
Local and global variables 92
System variables 93

Functions and expressions 99

Functions 99
Manipulating strings 99
Converting strings 102
Expressions 105

How to make your macros more robust 107

Introduction 107
Techniques for making your macros more robust 108
&CONNECTSTATE 109
LOGERRORSTOFILE 109
Writing Trace information 111
&ATPROMPT 112
&AtOutput 113
&RECOVERYATTEMPTS 113
Adding a Waiting period 114
Ending recovery using the RECOVERSTOP and ENDALLMACROS commands 114
Potential problems with the SEND command 115
Starting recovery by calling a recovery macro 116
Failures in the recovery macro 117
Template recovery macro, RECOVER.MAC 117

Example macros 125

Example 1: DEMOGLST.MAC 125
Example 2: DEMO900.MAC 126
Example 3: DEMOGRPH.MAC 127
Example 4: DEMOLIST.MAC 127
Example 5: DEMOSAVE.MAC 128
Example 6: DEMOPSS.MAC 129
Example 7: DEMODATE.MAC 130
Example 8: DEMOSITA.MAC 131
Example 9: EX_PRNT.MAC 134
Example 10: EX_SAVE.MAC 135
Example 11: EX_LIST.MAC 136
Example 12: EX_SET.MAC 138
Example 13: EX_STRNG.MAC 139
Example 14: EX_SYSTM.MAC 140
Example 15: EX_MASTR.MAC 141
Example 16: EX_401X.MAC 142
Example 17: EX_GLIST.MAC 144
Example 18: EX_GSTYL.MAC 147
Example 19: EX_GANT.MAC 150
Example 20: EX_300C 153
Example 21: EX_GANT1.MAC 160
Example 22: EX_CSV.MAC 163
Example 23: EX_CLIP.MAC 165
Example 24: EX_EXCEL.MAC 167
Example 25: STARTUP.MAC 170
Example 26: LOGON.MAC 171
Example 27: EX_PSS.MAC 172
Example 28: EX_WRITE 174
Example 29: RECOVER.MAC 176
Example 30: EX_TIMES.MAC 176
Example 31: EX_DATEF.MAC 177
Example 32: EX_PROMT.MAC 178
Example 33: EX_CLOSE.MAC 179
Example 34: EX_900CO.MAC 179
Example 35: EX_CONN.MAC 180

Appendix A **181**

Converting control files into macros 181
Introduction 181
The conversion program 181

INDEX **189**

Alphabetic Index to Commands and Instructions

ACTIVATEBACKPAGES 76
ACTIVATEGRAPHICS 76
ACTIVATESAVEFILES 76
ACTIVATETERMINAL 76
ADDTOSELECTITEMS 57
ALLOWDUPLICATETIMESERIES 73
AUTOPAGE 26
AUTOPRINT 25
AUTOSAVE 43

BEEP 86

CALL 27
CAPTURE 74
CHANGEITEMS 61
CLOSEBACKPAGES 76
CLOSEDSWINDOWS 76
CLOSEGRAPHICS 76
CLOSESAVEFILES 76
CLOSETERMINAL 76
COMMENT 36
CONFIGUREDC 70
CONNECT 78
CONNECTNOQUEUE 79
CONNECTNOWAIT 79
CONSTTIMESERIES 72
COPYITEMS 60

DATA 32
DELETEITEMS 60
DESELECTGRAPH 51
DESELECTITEMS 58
DISCONNECT 81
DISPLAYLAYOUT 41
DISPLAYSINGLEGRAPH 41

END 31
ENDALLMACROS 83
ENDAUTOSAVE 44
ENDCAPTURE 75
ENDDATA 33
ENDDC 72
ENDPAGE 26
ENDPRINT 25
EXPORTGRAPHICS 44

GOTO 30
GRAPHDRAWOFF 69
GRAPHDRAWON 69
GRAPHPAGESETUP 47

IF 28
INPUT 33

LOADFILLSTYLES 49
LOADGRAPHFILE 46
LOADLAYOUT 48
LOADLAYOUTFILE 46
LOADLINESTYLES 48
LOADTEXTSTYLES 49
LOGERRORSTOFILE 83
LOGON 80

MAXIMIZEBACKPAGES 77
MAXIMIZEDSWINDOWS 77
MAXIMIZEGRAPHICS 77
MAXIMIZESAVEFILES 77
MAXIMIZETERMINAL 77
MESSAGE 86
MINIMIZEBACKPAGES 76
MINIMIZEDSWINDOWS 76
MINIMIZEGRAPHICS 76
MINIMIZESAVEFILES 76
MINIMIZETERMINAL 76
MOVEITEMS 59

NEWBOX 63
NEWLINE 68
NEWRECT 65
NEWTEXT 67

ONERROR 85
OPENDATA 32
OPENSAVEFILE 75
PRINTGRAPHFILE 45
PRINTGRAPHICS 44
PRINTLAYOUTFILE 45
PRINTSAVEFILE 25

RECOVERSTOP 83
RECOVERUSING 82
REFINESELECTITEMS 58
RESTOREBACKPAGES 77
RESTOREDSWINDOWS 77
RESTOREGRAPHICS 77
RESTORESAVEFILES 77
RESTORETERMINAL 77

SAVEGRAPHICS 42
SAVEWMF 43
SELECTGRAPH 51
SELECTITEMS 55
SENDANDCHECK 24
SET...TO 39
SETDATEEXPORTFORMAT 86
SETGLOBAL 40
SETGRAPHNAME 50
STARTDC 71
STARTPROGRAM 27

USERINPUT 37

WAIT 38
WRITETOFILE 84

About this guide

This user guide accompanies the DSWindows 2.2 User Guide and describes the DSWindows™ macro language.

It begins with an introduction to DSWindows macros, lists all the macro commands, variables, constants, functions and expressions. It includes a large number of sample macros, with explanations of what they do and how they work.

What's new in DSWindows 2.2

This section briefly outlines the main changes made to the DSWindows macros language and which collectively comprise the new features in version 2.2.

Macro recovery

- One of the prime objectives in 2.2 is to improve the robustness and recovery capability of DSWindows macros. A template recovery macro, `RECOVER.MAC`, has been added which includes a number of new macro commands, instructions and variables. For example:
  - a new command, **LOGERRORSTOFILE**, enables you to send error messages to a log file rather than the screen. This helps you to build self-reporting macros which do not require user interaction if an error occurs during execution of the macro.
  - a new instruction, **RECOVERUSING**, enables you to specify the name of the recovery macro which will be automatically triggered by a failed SEND or UPDATELOCALCODE command.
  - a new command, **RECOVERSTOP**, allows you to switch the recovery process off and revert to the normal macro behaviour - this is useful if you want to return control to the calling macro at the end of recovery.
  - a new system variable, **RecoveryAttempts**, allows you to control the number of times a recovery macro will attempt to resolve a problem (such as a failed SEND command), and also the way in which recovery is attempted.

- a new macro command, **ENDALLMACROS**, provides an alternative way of terminating the recovery process by simply killing all active macros.
- a new system variable, **&ATPROMPT,** enables you to test whether you are sitting at the Datastream prompt (Program Finder). This is especially useful in recovery macros where one of the main objectives is to recover the state of being at the prompt before returning control to the calling macro.

To support the introduction of these new elements, a new chapter has been added to this guide called **How to make your macros more robust.** This chapter provides a full explanation of all these new techniques and includes the complete `RECOVER.MAC`, together with some general explanation of how the macro works.

Date Macro commands

- a new macro command, SetDateExportFormat, allows you to set the DSWindows Export date format. The command takes a single input parameter as a date format string. This should match the date format strings listed in the short date styles options in the Windows control panel.

Data Channel Macro commands

- a new macro command, **ConstTImeSeries**, when executed after starting Data Channel translation, allows a constant value to be inserted in the next row/column.
- a new macro command, **AllowDuplicateTimeSeries**, enables you to control merging of requests for the same data but with different start/end dates.

Code Lookup

- Updates now repeat until all outstanding Code Lookup updates have been processed. The total duration of the update can be limited by setting a timeout parameter:
  - to control how long a PC is connected to the Datastream host for code updates, the **StartUpdate** command can now take a new parameter which specifies the number of minutes a code update can run for before being timed out.

Connect macros

- To support the release of DSGATE 3.0 and increase the range of connection options, the **CONNECT, CONNECTNOWAIT** and **CONNECTNOQUEUE** commands can now take either session or gateway name and queue name parameters.

Scheduling unattended running of macros

- DSWindows 2.2 includes a new default scheduler application (DSAGENDA) which enables users to create schedules for macros. The **Introduction** chapter now includes details of how to use DSAGENDA.

Macro variables

- A new macro constant, **&OS**, tells you what operating system DSWindows was built for - useful for writing a macro intended for use on multiple platforms.

Who should use this guide

This guide is for DSWindows users who want to use macros to automate the functions provided by DSWindows. It is assumed that users have the *DSWindows User Guide*.

What you need to know

In this guide, we assume that you know how to use your PC, and its keyboard. If not, please refer to the manufacturer's instructions. We assume also that you have used Microsoft Windows® and that you are familiar with Windows concepts and procedures. It is also assumed that you have used DSWindows and have some familiarity with what it does.

This guide does not tell you how to use Datastream™ programs, or how to use DSWindows. DSWindows provides an interface to Datastream's programs. The DSWindows macro language enables you to automate DSWindows functionality. If you need information on Datstream programs, please refer to the relevant user guide or codes manual. For information on how to use DSWindows interactively, please refer to the *DSWindows User Guide.*

If you are completely new to Datastream, please ask your Customer Services Executive or Client Liaison Executive for help with training and the provision of appropriate documentation.

How to use this guide

This guide provides a reference source for the DSWindows macro language.

It is split into seven sections:

- **About this guide (this section)**

  Tells you how to use the guide and how it is structured. It gives information on important keys and conventions used in the guide. It also tells you what you need to know and where to find further information.

- **Introduction**

  Explains what macros are, introduces the concepts used, describes how to create, edit and record macros, and how to create schedules for running them.

- **Macro instructions and commands**

  Lists all of the instructions and commands in the macro language, with detailed information on when to use them, the syntax, etc.

- **Constants and variables**

  Lists all of the constants and variables used in macros, with detailed information on when to use them, the syntax, etc.

- **Functions and expressions**

  Lists all of the functions and expressions used in macros, with detailed information on when to use them, the syntax, etc.

- **How to make your macros more robust**

  Details all the techniques available for writing self-reporting macros with inbuilt error handling and recovery capabilities. This section also includes documentation of the template recovery macro, RECOVER.MAC.

- **Sample macros**

  Gives a number of macros with explanations of what they do, and how they are written. You can use these macros as they are, or copy them and edit them to suit your own purposes.

- **Appendix A**

  This section is for users who have previously been using DSCOM™ or DSTERM™. It provides information on the program which converts DSCOM/DSTERM control files to macros.

NOTE *If the concept of macros is new to you, or if you have not written a DSWindows macro before, it is recommended that you read the Introduction before you start.*

Conventions

Keys	In Datastream user guides and online help, the names of keys are shown in small capital letters; for example, the function keys are F1 to F12 and keys identified on the keyboard with words are shown as, for example, ALT or ESC.
Keystrokes	When two or more keys need to be pressed simultaneously, the plus character is used to indicate simultaneous keystrokes; for example, holding down the CTRL key while pressing the 't' key is shown as CTRL+t.
Enter key	We use ENTER to refer to the key you press to transmit instructions to the computer. This may be marked on your keyboard as, for example, Return, Carriage Return, Enter or ↵.
Input	When describing other keyboard input, we show what you must type using a slightly different bold typeface; for example '...type **BMAH** and press ENTER'.
Screen displays	All screen messages and menu options referred to in the text are shown in bold type; for example '...select the **Print** command...'.

Further information

Other publications

For information on using DSWindows, please refer to the *DSWindows User Guide*.

For information on using the Datastream programs, please refer to the relevant user guide or manual:

- *Summary of programs*
- *Time Series Analysis User Guide*
- *Company Accounts Definitions Manual*
- *Company Accounts User Guide*
- *Data Channel User Guide*
- *Economics User Guide*
- *Graphics User Guide*
- *Economics Codes Manual (Vols. 1 - 4)*
- *Datastream Definitions Manual*
- *Indices, Interest and Exchange Rates Manual*

Online help

Online help is incorporated into all Datastream interface software and includes context sensitive help on menu commands and dialog boxes as well as a more general help system containing information on creating, editing and running macros. Online help now also includes a very large and comprehensive set of definitions covering terms used for all types of security, datatypes definitions (including I/B/E/S and MSCI), Datastream terminology and data sources and updating procedures. To access online help, click on the Help menu and select an appropriate option.

Telephone support

Datastream provides Helpline support for queries on programs, data, communications problems and so on. For an up-to-date list of telephone numbers you can use, refer to the back cover of this guide.

Training

Datastream provides a full range of hands-on training workshops, tailored to give you the knowledge, practice and confidence to make full use of the Datastream system. The workshops are constantly reviewed to meet changing market needs and to suit the differing requirements of each country in which the Datastream service is available. Contact your local Account Manager for complete and up to date information.

Introduction

This section introduces you to the Datastream macro language. If the concept of macros is new to you, or you have not created or edited macros before, it is recommended that you read this section before moving on to the detailed explanations of individual elements of the macro language which are given in the subsequent sections.

The Introduction includes:

- **What are macros ?**
  Introduces the main elements of the macro language (commands, instructions, labels, constants and variables, functions and expressions, data files).
- **Using macros**
  A short tutorial on creating, editing and running macros, using the macro recorder and scheduling your macros using DSAGENDA.
- **General macro rules**
  A brief list of the general rules which must be followed when creating or editing a macro.
- **Tips**
  A few suggested tips on writing good macros.
- **Examples**
  Examples of typical macros showing their structure and usage are given within each section.

What are Datastream macros?

Macros are short programs which enable you to automate Datastream functions. The Datastream macro language is an interpretive language which runs on your PC or workstation. It is loosely based on the Basic programming language, but has its own easy-to-use syntax. Users familiar with any simple programming or macro language you will find that the concepts used in Datastream macros are very similar.

You can create a macro in a suitable editor, such as the Windows Notepad, and then run it from the Terminal window in DSWindows. DSWindows also provides a macro recorder which enables you to create macros by recording the sequence of keystrokes required to achieve a particular task.

NOTE *The recorder only records the information sent to Datastream; commands such as PRINTGRAPHICS must be added manually (for an example, see page 142).*

The macro language consists of the following main programming elements:

- commands
- instructions
- labels
- constants and variables
- functions and expressions
- data files

Commands

You can use macro commands to perform a wide range of functions. For example:

- connecting to Datastream
- arranging windows
- sending data requests to Datastream
- paging
- capturing, saving and printing data
- downloading and exporting data

Examples

1. A simple macro command might run the Datastream 190A program. You can do this with the **SEND** command. For example, to retrieve company accounts data for ICI and BP, you could write a macro as follows, and run it from the Terminal window:

   ```
   SEND ("190A ICI")
   SEND ("190A BP")
   ```

2. Since you are likely to use the **SEND** command frequently, it has an abbreviated form in which the word '**SEND**' is replaced by the character '**>**' and no quotes are required around the input characters. The abbreviated form of the above macro is:

   ```
   >190A ICI
   >190A BP
   ```

3. To print the data you are receiving, you can include the **AUTOPRINT** and **ENDPRINT** commands in the macro, as follows:

   ```
   AUTOPRINT
   >190A ICI
   >190A BP
   ENDPRINT
   ```

4. You can include a command to save the program output to a save file, and print the contents of the save file. The command to start saving data is **CAPTURE**, and the command to print the contents of a save file is **PRINTSAVEFILE**. The name given to the save file here is "example1". The macro would then be:

   ```
   CAPTURE ( "example1" )
   >190A ICI
   >190A BP
   ENDCAPTURE
   PRINTSAVEFILE ( "example1" )
   ```

   This macro saves the text output from the two programs into the file called "example1", appends the default extension (.dst), and prints out the file.

Full descriptions of the macro commands are included in the section entitled 'Macro commands and instructions'.

Instructions

Instructions are a specific type of command which control the flow of the macro. For example, they are used to:

- assign variables
- make conditional evaluations
- input data from a data file
- include comments in a macro

Full descriptions of the macro instructions are included in the chapter entitled ‘Commands and instructions’.

Labels

A label identifies a place within a macro which is referred to from elsewhere in the macro. For example, you might use the label loop: to identify the start of a loop, and refer to it in a **GOTO** instruction; or you might use the label codes: and refer to it using an **INPUT** instruction.

NOTE *A label must be immediately followed by a colon (:).*

Constants and variables

Constants and variables are generally used in macros as mechanisms for testing for various conditions; for example, to test whether or not all the codes in a data file have been input, or a message has been sent.

Constants A constant is a string with a fixed value in the macro. For example, a constant might consist of the number 125, or the text, "This is an example".

Variables A variable is a placeholder for a value which can change while the macro is running. A system variable is a variable which has a specific meaning in a macro, such as identifying an ‘end of data’ state, or a day in the week. System variables can be changed by DSWindows, but not by a user’s macro.

For detailed information, please refer to the chapter ‘**Constants and Variables**’.

Functions and expressions

Functions Functions enable you to find required character strings by searching for them in relation either to their position within another character string, or their position on the screen. You can then manipulate these strings by, for example, extracting character(s) from within other strings and writing the extracted strings out to a file. Functions also allow you to convert strings into integers and vice-versa.

Expressions Expressions are formed by combining mathematical and logical operators with constants and variables.

For a detailed explanation of strings, integers, functions and expressions, please refer to the chapter 'Functions and Expressions'.

Data files

A list of Datastream codes or mnemonics which you want to use in a macro can be stored in one of two ways, either as part of the macro, or as a separate data file. In either case, the start of the data list is indicated by the instruction **DATA** and the end is indicated by the instruction **ENDDATA**.

NOTE *The advantage of storing such a list as a separate data file is that you can place references to it in any number of macros.*

Example 1 A list of company codes is included at the end of the macro, marked by the label codes: and bounded by the instructions **DATA** and **ENDDATA**. Each code in the data list becomes a variable which is accessed by an **INPUT** instruction when required.

The data file is opened using the **OPENDATA** instruction to reference the codes: label, and a save file ("example1") is opened using the **CAPTURE** instruction. A loop is used to test for the end of the data list and to repeat the data request until all the codes have been used. When the last code has been used, the **ENDDATA** instruction causes the system variable &ENDOFDATA to be set to TRUE. This causes the macro to exit the loop, close the save file (**ENDCAPTURE**) and stop the macro (**END**). The output is saved into the file "example1".

```
OPENDATA codes
CAPTURE ( "example1" )

Loop:
IF &ENDOFDATA = FALSE THEN
    INPUT code
    SEND ("190A " + code)
    GOTO Loop
ENDIF

ENDCAPTURE
END

codes:
DATA
    "ICI"
    "BP"
    "BOOT"
    "BMAH"
ENDDATA
```

Example 2

This example shows how to download data on the companies in the FTSE 100 using program 190A. The macro is the same as the one in the previous example, except that the list of codes is kept in a separate data file. (The example macro ex_strng shows how to generate such a list.)

The required data file (ftse.lst) is called using the **OPENDATA** instruction. As in Example 1, a save file is opened using the **CAPTURE** instruction. Each code becomes a variable which is accessed by the **INPUT** instruction when required, and a loop is used to reiterate the data request until all the codes have been used. The output is saved into a file called "ftsecos.dst".

```
OPENDATA "ftse.lst":
CAPTURE ("ftsecos.dst")

Loop:
IF &ENDOFDATA = FALSE THEN
    INPUT code
    SEND ("190A " + code)
```

```
    GOTO Loop
ENDIF

ENDCAPTURE
END
```

The data file (named ftse.lst) consists of the following (i.e. all the Datastream mnemonics for the companies in the FTSE 100):

```
DATA
    "ANL"
    "ALLD"
    "AW"
    etc...
    "WILM"
ENDDATA
```

NOTES *If you refer to a list of items contained in another file:*

1. *the filename in the* **OPENDATA** *command must be in quotes*
2. *the filename must also be followed by a colon (:), for example,* **OPENDATA** "/dswindow/myfiles/ftse.lst":
3. *if the file is in a directory other than your default working directory, you must include the full path name.*

Creating macros

The commands for creating, recording, running and editing macros are all available through the Macro menu on the Terminal window menu bar.

The last five macros run are listed in the macro menu. You can select them either by typing their number, or by clicking on the macro name.

Tool bar buttons are available for: *starting the macro recorder* 

running a macro 

You can create a macro in two ways

- open a file and type in the relevant text for the macro
- use the macro recorder to record the keystrokes you make when requesting data and using various functions

♦ To open a macro file

1. In the Terminal window, select **Macro>Edit macro...**
2. In the **Edit Macro File** dialog box, type the name of the new macro in the Selection field. By default, macros are held in the **\dswindow\files** directory.
3. Click on **OK**.

   Notepad is opened.

4. Type the text of the macro.

5. Save the macro. Make sure that you save the file with a .mac file extension.

Using the macro recorder

The Macro Recorder records the keystrokes you make when requesting data from Datastream. It records, for example, all the characters you type into fields in a program input screen, and strokes of the TAB key you make to move between the fields.

NOTE *The macro recorder recognises all keys but we recommend that you use* TAB*, rather than the mouse, to position the cursor at the start of a field.*

The keystrokes are recorded into a macro file (.mac), either a new one, or an existing one. If you record to an existing file, you can choose whether to append the keystrokes to the existing file, or to overwrite the file completely.

♦ To start recording

1. Select **Macro>Record Macro...** on the menu in the Terminal window (or click on the Macro Recorder icon)

2. In the **Record Macro** dialog box, either type a file name for the new macro in the **Filter** field, or select an existing file.

   If you select an existing file, a dialog box asks you to **Overwrite** or **Append**. Select as appropriate.

3. Click on **OK**.

From now until you stop recording, all keystrokes, apart from local functions such as mouse movements and menu or icon selections, are recorded into the file you specify. Remember to stop the macro recorder when you have finished building the macro.

♦ To stop recording

❑ Select **Macro>Stop Recording** or click on the Macro Recorder button again. Note that while the macro recorder is active the button is highlighted.

Editing macros

♦ **To edit a macro**

1. In the Terminal window, select **Macro>Edit macro...**
2. In the **Save Macro File** dialog box, type the name of the macro you want to edit in the Selection text box, or select it from the list of file names.
3. Click on **OK**.

   Notepad is opened with the text of the selected macro displayed.
4. Make the required changes to the macro, save the file and exit from the text editor.

Running macros

♦ **To run a macro**

1. In the Terminal window, select **Macro>Run Macro...** (or click on the Run Macro icon)
2. In the **Run Macro** dialog box, type the name of the macro you want to run in the Selection text box, or select it from the list of file names.
3. Click on **OK** to run the macro.

NOTE *When you click on the Macro menu option, the last five macros you have run are listed. You can select one of these macros by clicking on the macro name, or typing its number (1 - 5).*

Scheduling your macros using DSAGENDA

DSWindows 2.2 includes a new default update Scheduler utility called **DSAGENDA** which enables you to create a schedule for running macros.

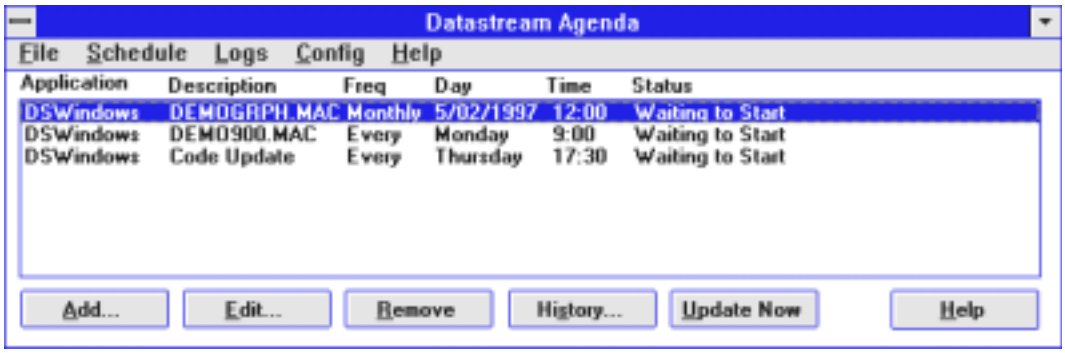
NOTES

1. *If you set a macro to run when you are not there (overnight for example), you must leave Agenda open (iconised if you prefer) and your PC switched on. You do not have to have Code Lookup or DSWindows running at the same time.*
2. *The first time you run Agenda a Welcome dialog will be displayed asking you to specify the location of the DSWindows executable and the Agenda control file (dsagenda.dsa). If no control file exists, one will be automatically created.*

♦ To start Agenda as a standalone application

❑ Double-click on the Agenda icon

The Agenda interface is shown in the following screenshot:



NOTE *This screenshot shows three scheduled items. Item 3 is a code update schedule, but notice that items 1 and 2 show the details of schedules for running macros.*

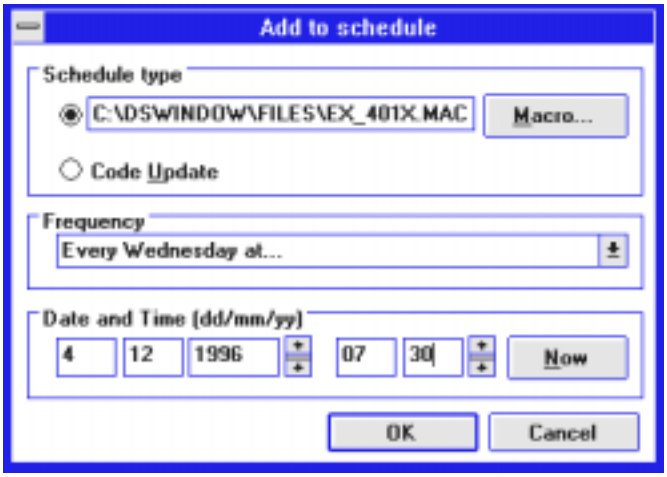
♦ To set a Schedule

Agenda enables you to set the day, the time and the frequency of the at which to run the macro. The day can be any day of the week; the time can be any time of the day or night - note that Agenda uses the 24-hour clock; the frequency can be once only, daily, every monday/tuesday/wednesday etc, monthly, that start of the month or quarterly.

Example

This example shows how to set a schedule so that a macro is run every Wednesday at 7.30 a.m.

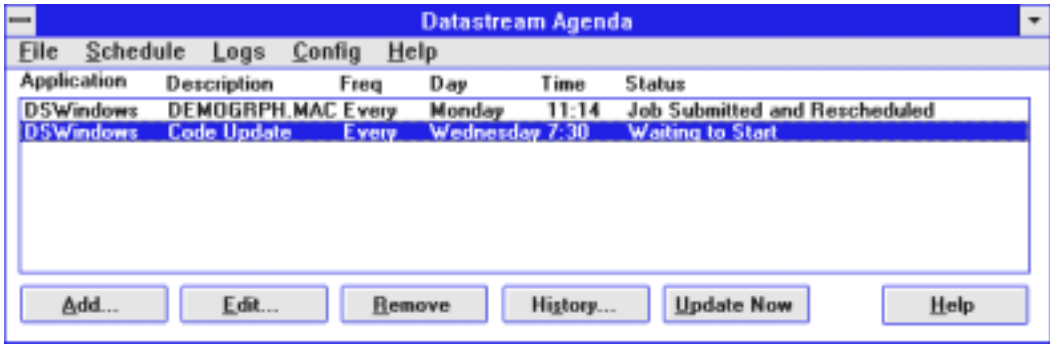
1. In the main window, click on the on the **Add..** button (or select Schedule>Add..) to display the Add to Schedule dialog



2. In Schedule type click on the **Macro...** button and select the name of the macro you wish to schedule
3. In the Frequency drop-down box, select **Every Wednesday at....**
4. In the Date and Time (dd/mm/yy) fields, leave the date as set by default - when you set the Frequency in the previous step, Agenda automatically sets the Date field to the date of the next Wednesday. In the Time field, position the cursor in the first box and either type the required hour or use the Up/Down arrow to select it. Repeat this process to set the minutes in the second box.

5. Click on **OK**.

The main window will now display showing the details of the schedule you have just set, with a status of 'Waiting to Start':



♦ To edit a scheduled task

- ❑ In the main window highlight the schedule entry you want to change.
- ❑ Click on **Edit...** (or select Schedule>Edit...) or double-click on the schedule entry.

  The Add to schedule dialog is opened with the details of the scheduled task displayed.
- ❑ Make the required changes and click on **OK**.

♦ To delete a scheduled task

- ❑ In the main window highlight the schedule entry you want to delete.
- ❑ Click on **Remove** (or select Schedule>Remove...)
- ❑ At the prompt, 'Are you sure you want to remove this scheduled item', click on **Yes** to confirm or **No** to cancel.

♦ To see a log of previous updates

You can check the status of completed jobs using the History log file.

- In the main window click on the **History** button (or select Schedule>History...)

  The Log of completed jobs window is displayed with a complete list of all items which Agenda has processed. The possible values in the Status field are as follows:

  "Aborted, status uncertain"
  "Code Update succeeded"
  "Code Update succeeded and rescheduled"
  "Failed"
  "Failed, Job Rescheduled"
  "Job Submitted"
  "Job Submitted and Rescheduled"
  "Missed"
  "Missed, Job Rescheduled"

The Log of completed jobs also provides an option to view the update log generated by Code Lookup providing the full details of an update. This log file is used for fault diagnosis and can be displayed in Notepad by clicking the **View Update Log** button.

General notes on Agenda operation and maintenance

- You can schedule more than one job to run at the same time, frequency and date - DSAgenda processes the jobs in the order they are listed and will complete each job before starting the next one.
- The status of each job is shown on both the main window and the Log of completed jobs.
- The details of all scheduled jobs remain displayed on the main Agenda window, even after they have been processed. The same details are also available using the Log of completed jobs. As a general principle, it is good practice to remove 'one-off' jobs from the Agenda list once they have been processed. Completed one-off jobs are displayed as 'Finished' in the Time field of their record. See the following point for further maintenance information.
- There is an upper limit of 100 scheduled jobs on Agenda at any one time. The limit on the Log of completed jobs is approx. 65,000.

Summary of general macro rules

The general rules you must adhere to when you write or edit a macro are:

- write one statement per line, or use \ as the last character before the return to indicate that the following line should be treated as part of the current line
- use spaces or tabs to separate key words and variable names
- enclose filenames in double quotes (for example, "logon.mac")
- use a semi-colon (;) to prefix a comment. All text to the right of a semi-colon up to the end of the line is treated as comment. Use comments to document your macros for future reference
- labels must be immediately followed by a colon (:)
- type Datastream input (such as codes or mnemonics) in upper case
- you can type instructions and commands in upper or lower case, or any mixture of the two
- you can use tabs to indent lines to make your macros more readable - the layout does not, however, effect the way in which the macro works

Tips on writing and editing macros

1. To simplify the process of understanding what a macro does and how it works, and to make debugging a macro easier, it is always good practice to:

   - document the macro using short statements which explain the overall objective of the macro and what each section of the macro does.
   - use indentation to group elements of a macro; for example, all statements inside an IF...THEN...ENDIF construction should be indented:

```
IF &ENDOFDATA=FALSE THEN
    INPUT CODE
    SEND ("900A "+CODE)
    SEND [CLEAR]
ENDIF
```

2. Always start a macro with the **>[CLEAR]** command to clear the screen before the macro is executed.

3. Get into the habit of testing the result of each command during the execution of the macro. This practise will help you to quickly identify errors, either in the macro itself or in the data.

Macro commands and instructions

NOTE *Please note that the Alphabetic index to commands and instructions (which was included at the start of this section in the previous version of this manual) can now be found as the last page of the Table of Contents on page vii.*

This section provides detailed information about Datastream macro commands and instructions. The commands and instructions are arranged in groups according to their functions and appear in the following order:

- Sending data to Datastream
- Printing
- Paging
- Instructions (commands which control the flow of the macro)
- Graphics
- Graphic annotations
- Data Channel
- Capturing text
- Arranging windows
- Connecting
- Error recovery, logging errors, writing to a file
- Code Lookup
- Miscellaneous

Conventions used in this section

Please note that, as an aid to reading and understanding the content of this section, the following typographic conventions have been adopted to illustrate the syntax of the Datastream macro language:

- command names and instructions are in upper case and bold font
- all parameters are in lower case italics
- mandatory parameters are in bold italics

Example **SAVEGRAPHICS***(**file:filename**,graph:graphname,append:flag)*

SAVEGRAPHICS is the command name

file:filename,*graph:graphname* and *append:flag* are parameters. ***filename*** is a required parameter. The others are optional.

Most parameters do not have names, but in a command in which the parameters do have names, such as the one above, the parameter names are usually optional. You need to use them only if one or more parameters are omitted, so that the parameters which *are* used can be identified. For example:

SAVEGRAPHICS(*file:"**test**",append:overwrite)*

The brackets around the parameters are always required.

Note also, that when you include a filename in a macro, it must always be surrounded by double quotes. (Double quotes are not included in the syntax, although they are included in the examples.)

Sending data to Datastream

SEND

Function

Sends the result of an expression to Datastream. This command can be used for a number of different functions, such as requesting a report or graph, connecting to Datastream via packet switching, or checking for a returned character or timeout period. Note that you can use the '>' character as an abbreviated form of the SEND command.

✪ **SEND** *can be used with two different parameter structures according to the intended function. These are explained and exemplified in Syntax 1 and Syntax 2 below.*

Syntax 1

SEND *(expression)*

expression — The data to be sent; for example, a program number and a code in expert mode. Please refer to the section 'Functions and Expressions' for details on the mathematical and logical operators you can use in building expressions. Two examples are given below.

Remarks

DSWindows automatically appends an ENTER to the string before it is sent, unless the parameter ends in **[ENTER]**, **[NOENTER]**, **[CLEAR]**, **[PAn]** or **[PFn]** keys.

In a **SEND** instruction, you can use special keys in quoted sequences, for example "**[CLEAR]**". These sequences represent single key-strokes.

Special keys are:

[BACKSPACE]	(remove last character)
[BACKTAB]	(return to last input field)
[BREAK]	(send a break signal)
[CLEAR]	(return to Datastream prompt)
[DELETE]	(delete current character)
[DOWN]	(cursor down)
[END]	(move cursor to last field)

[ENTER]	(send input to Datastream)
[ERASE_EOF]	(erase to end of field)
[ERASE_INPUT]	(erase all input fields)
[HOME]	(move cursor to first field)
[INSERT]	(insert characters before the current one)
[LEFT]	(cursor left)
[MOVETO col, row]	(moves the cursor to the column and row position specified. **col** can be between 1 and 80. **row** can be between 1 and 24. NB: It is preferable to use [TAB]s.) They are easier to use and they will still be effective if the positions of fields on the screen change.
[NEWLINE]	(cursor to next input line)
[NOENTER]	(no automatic <**Enter**>)
[PA1] - [PA3]	(the effect is program-specific)
[PF1] - [PF10]	(<**Alt_1**> to <**Alt_0**>)
[PF11] and [PF12]	(the effect is program-specific)
[QUOTE]	(sends a quote character without terminating the string)
[RESET]	(reset keyboard)
[RIGHT]	(cursor right)
[TAB]	(tab to next line)
[TTY]	(change to teletype mode)
[UP]	(cursor up)

Example 1 **SEND** ("401A BP")

Requests a 401A graph for BP using expert mode.

Example 2 **SEND** ("D:VW[TAB]DAXINDX[DOWN][DOWN][DOWN][TAB][TAB]2")

On a 401A input screen, requests a graph comparing Volkswagen with the DAX Index, with all values shown in index form starting at 100.

SEND continued - syntax 2

Syntax 2 **SEND** *(**text1**, WAITFOR: **text2,** TIMEOUT: n)*

Example 1 This is the first part of the example macro EX_PSS.MAC which is documented in full in the section 'Example Macros'. It illustrates how to use the **SEND** command to connect to Datastream via UK PSS. Note that the three **SEND** commands show variations on how to use the command parameters.

```
CONNECT                                   Open the port connecting DSWindows
                                          and Datastream.

start:

SEND( "ATZ", "OK", TIMEOUT:3)             Connect to the modem.

SEND( Text:"ATE1", "OK" )                 Turn echo off from the modem, and wait
                                          for the modem to respond with "OK".

SEND( "ATDxxxxxxxx", WaitFor:"CONNECT", 45 )
                                          After receiving "OK" from the modem,
                                          dial the phone no: ATDxxxxxxxx, and
                                          wait for the response: "CONNECT".

IF &sendComplete <> TEXTFOUND THEN
                                          If the PAD does not respond with
                                          "CONNECT", display the message:
                                          "DID NOT FIND 'CONNECT'", and
                                          start again.

    MESSAGE( "DID NOT FIND 'CONNECT' ", "EX_PSS.MAC")
    GOTO start
ELSE
    .....
```

Example 2

Illustrates how to use **SEND** with an expression requesting a report and a timeout control. The macro checks whether the request has been completed within the timeout period and, if not, disconnects from the Datastream mainframe, waits for 10 seconds, runs the LOGON macro and repeats the SEND request.

```
retry:
SEND ("900B ASDA,-10Y,,D", TIMEOUT:90)
IF &sendcomplete=TIMEOUT THEN
    DISCONNECT
    WAIT(10)
    LOGON
    GOTO retry
ENDIF
```

✪ *In this form of the SEND command, if the second parameter - WAITFOR:text2 - is not used in the command, you **must** include the word TIMEOUT: in any timeout declaration. However, if the second parameter is used, you can abbreviate the timeout declaration by omitting the word TIMEOUT. For example:*

```
SEND ("ATZ", "OK", 45)
```

✪ *If you use the SEND ("ATZ") command in a macro which uses RECOVERUSING to call a recovery macro, ensure that you issue the SEND ("ATZ") before recovery is enabled. When ATZ is issued to some modems a line drop occurs (DSR momentarily goes low) which triggers the recovery macro. Line drops are normally considered an error but in this case the user has effectively requested it. Please refer to RECOVERUSING for further information.*

Related commands **>, SENDANDCHECK**

>

Function — Use as an abbreviated form of the **SEND** command.

Syntax — ***> expression***

expression — The data to be sent; for example, a program number and a code in expert mode. Please refer to the section 'Functions and Expressions' for details on the mathematical and logical operators you can use in building expressions.

With the > command, the expression must not be surrounded by quotes: data is taken verbatim from the first to the last non-space character. This means that you cannot use string or variable operators, or quotes, colons or semi-colons within the data.

Remarks — If you are using strings and variables/constants, use the **SEND** command rather than this abbreviated version.

Example — **> 401A BP**

Requests a 401A graph, as in the example above.

Related commands — **SEND, SENDANDCHECK**

SENDANDCHECK

Function — Sends the text labelled TextToSend to Datastream, and awaits a response.

Syntax — **SENDANDCHECK *(TextToSend, TextToLookFor)***

TextToSend — A data request

TextToLookFor — The text string you want to find within the retrieved data.

Remarks — If the text labelled TextToLookFor is found in the results, the variable &TEXTFOUND is set to TRUE. Otherwise it is set to FALSE. If autopaging is ON then the TextToLookFor may appear on any of the pages of the report. Since the keyboard must be unlocked before TextToLookFor can be found, this command is only suitable for VT100 - not TTY (while dialling up).

Example 1

```
SENDANDCHECK("900B "+code+",1/1/79,,D","$$"+CHR$(34)+"H0" )
IF &TEXTFOUND <> TRUE THEN
        MESSAGE ("Data Channel header not found")
ENDIF
```

In this example, A 900B request is sent. If the text $$"HO is not returned, the error message is displayed.

NOTES

1. *$$"HO always appears at the start of any Data Channel output and can therefore be used to check for receipt of data*
2. *CHR$(34) is the ASCII value for a double quotation mark - use this function when checking for a quote character*

Example 2

```
SENDANDCHECK("XABC999PASSWD","LOGON REJECTED")
IF &TEXTFOUND = TRUE THEN
        SEND("XABC100PASSWD")
ENDIF
```

An attempt is made to logon using the ID XABC999 and the password PASSWD. If the text "LOGON REJECTED" is found, an alternative ID is used.

Related commands — **SEND, >**

Printing

AUTOPRINT

Function — Starts autoprinting (output text and graphics is printed as it is received).

Remarks — Graphs are printed as they are received. If you want to annotate a graph or graphs, first make the annotations, and then use the **PRINTGRAPHICS** command.

Example

```
AUTOPRINT
>190A D:VW
>401B D:WV
ENDPRINT
```

In this example, **AUTOPRINT** is switched on, and profit and loss accounts for Volkswagen and a graph are requested. They are printed as they are recieved. **ENDPRINT** switches off the autoprinting.

Related commands — **ENDPRINT, PRINTGRAPHICS, PRINTGRAPHFILE, PRINTSAVEFILE, PRINTLAYOUT**

ENDPRINT

Function — Ends autoprinting.

Example — See above.

Related command — **AUTOPRINT, PRINTGRAPHFILE, PRINTSAVEFILE, PRINTLAYOUT, PRINTGRAPHICS**

PRINTSAVEFILE

Function — Prints the specified text file. This must be a DSWindows (.DST) file.

Syntax — **PRINTSAVEFILE** ***(filename)***

filename — The name of the file to be printed. It must be a .DST file. If you do not specify a filename, a dialog box will prompt you for the filename.

Example **PRINTSAVEFILE** ("file10")

Prints a save file called ‘file10.dst’. If you do not specify a path, save files are assumed to be in your configured save file directory.

Related commands **PRINTGRAPHFILE, PRINTLAYOUTFILE, PRINTGRAPHICS, CAPTURE, ENDCAPTURE**

Paging

AUTOPAGE

Function Starts the autopaging function so that all output pages are displayed sequentially and moved to backpages. [ENTER] is sent automatically after each page is received. **AUTOPAGE** is on by default when a macro runs. Use **ENDPAGE** and **AUTOPAGE** together if you want to interrupt the autopaging.

Example See **ENDPAGE**

Related command **ENDPAGE**

ENDPAGE

Function Ends autopaging.

Example
```
ENDPAGE
> 99Z
>
> [CLEAR]
AUTOPAGE
```

In this example, **ENDPAGE** switches off autopaging, program 99Z is run and only the first two pages are displayed. **AUTOPAGE** switches on autopaging again.

Related command **AUTOPAGE**

Instructions (commands which control the flow of a macro)

CALL

Function — Executes a child macro. When this has completed execution, control returns to the parent macro, at the statement following the **CALL**.

Syntax — **CALL *file***:

file — The name of the macro.

Example —
CALL "EX_PRNT.MAC":
CALL "EX_CLIP.MAC":

In this example, two macros are run, first EX_PRNT.MAC followed by EX_CLIP.MAC. If you do not specify a path, macros files are assumed to be in your configured macro directory (by default dswindow\files).

Related command — **GOTO**

STARTPROGRAM

Function — Starts up another Windows application.

Syntax — **STARTPROGRAM** ***(commandline,*** *show)*

commandline — The program to be started, with or without the extension or directory path. If no extension is specified, .EXE is assumed. If no directory path is specified, Windows will search for the program in each of the following places (in the order listed):

1. the current directory
2. the Windows directory (containing win.com)
3. the Windows system directory
4. the directory listed in your PATH environment variable
5. the list of directories mapped in a network

show Defines the size of the window for the program.

show can be: *SHOW_NORMAL*
SHOW_MAX
SHOW_MIN

If no SHOW is specified, the default is SHOW_NORMAL. Sets the value of &RESULT on completion. If the value of &RESULT is greater than 32, then the command was successful. If the value is less than 32, then an error occurred. Error values are:

0 Out of memory
2 File not found
3 Path not found
5 Attempt to dynamically link to a task
6 Library requires separate data segments for each task
10 Incorrect Windows version
11 Invalid .EXE file
12 OS/2 application
13 DOS 4.0 application
14 Unknown .EXE type
15 Attempt in protected mode to launch an .EXE created for an earlier version of Windows
16 Attempt to load a second instance of an .EXE containing multiple, writeable data segments

Example **STARTPROGRAM** ("clock")

Starts the Clock application.

Related commands **CALL**

IF...THEN...ELSE...ENDIF

Function Makes a conditional evaluation; if the condition is true, then the first action is performed (the one following **THEN**); if the condition is not true, then the second action is performed (the one following **ELSE**). **ENDIF** signifies the end of the condition.

Syntax

```
IF (e) THEN
    s1
ELSE
    s2
ENDIF
```

e	An expression - if it is true, the first statement will be executed.
s1	The statement to be executed if the expression is true.
s2	The statement to be executed if the expression is not true.

Remarks

1. **IF...THEN...ENDIF** are compulsory. The **ELSE** clause is optional.
2. **IF...THEN** must be on one line.
3. **ENDIF** must be on a line by itself.
4. The logical operators **AND, OR** and **NOT** can also be used.
5. You can nest conditional evaluations, as follows:

```
IF (e) THEN
    IF (e1) THEN
        s1
    ELSE
        s2
    ENDIF
ELSE
    s3
ENDIF
```

Example 1

```
IF &DAYOFMONTH = 1 THEN
        ;Get last month's figures
        SEND ( "401A ICI,,,-1M" )
ENDIF
```

In this example, the **IF** condition tests whether it is the first day of the month. If it is, the **SEND** command runs a 401A program to chart the share price for ICI over the last month. **ENDIF** marks the end of the condition.

Example 2

```
IF &DAYOFMONTH = 1 AND &MONTH = 1 OR &MONTH = 4\
        OR &MONTH = 7 OR &MONTH = 10 THEN
        MESSAGE ("GET LAST QUARTER'S DATA", "QUARTER END")
ELSE
        MESSAGE ("GET LAST MONTH'S DATA", "MONTH")
ENDIF
```

In this example, the **IF** condition tests whether the quarter has just ended. If so, it prompts the user to get the last three months' data. If not, it prompts the user to get the last month's data.

GOTO

Function

Continues execution at a named file, or a labelled line in a file. If you specify a file name, but no label, then execution starts at the first executable line in the named file. If you specify a label, but no filename, then execution continues at that label within the current file.

Syntax

GOTO *filename:label*

filename	The name of the file which the macro will jump to.
label	The point in that file (or the current file) which the macro needs to go to.

You must specify either a filename or a label, or both. If no filename is given, the label must be in the current file.

Remarks

1. A label must be on a line by itself, and it must be followed by a colon (:).
2. A label can be positioned above or below the **GOTO** statement.
3. Avoid using the same label more than once, but, if there are multiple lines with the same label, then the last one previously used is executed. If none has been used before, the first one in the file is executed.

Example 1

```
GOTO "C:\DSWINDOW\DATA\COMP":
```

In this example, **GOTO** directs the execution of the macro to the named file.

Example 2

```
GOTO LOOP
...
...
LOOP:
```

In this example, **GOTO** directs the execution of the macro to the place (in the current file) labelled LOOP:

Related commands **CALL**

END

Function Terminates execution of the macro, and marks the end of the main body of the macro.

Example

```
OPENDATA "companies"
loop:
IF &ENDOFDATA = FALSE THEN
        INPUT code
        SEND( "101A " + code )
GOTO loop
ENDIF
END
```

1. In this example, companies is a data file, containing a number of company codes.
2. loop is a label. **IF** tests whether there is more data in the data file. If there is more data, **INPUT** inputs the next line of data (variable code).
3. **SEND** sends a 101A request to Datastream, for the company whose code has replaced code in the macro.
4. **GOTO** returns the execution of the macro to the label loop:. This will continue to happen until all the data has been input (that is, until &ENDOFDATA is not FALSE).
5. **ENDIF** marks the end of the condition.
6. **END** marks the end of the macro.

DATA

Function — Defines the start of a set of data items which can be read by an **INPUT** instruction. The data items must be enclosed in quotes.

Example

```
demolist:
DATA
        "BP",           "FTSE100"
        "F:PGT",        "FRCAC40"
        "J:RH@N",       "JAPDOWA"
ENDDATA
```

Related commands — **OPENDATA, ENDDATA, INPUT**

OPENDATA

Function — Opens a data file, or a data list in the current macro, or a list in another named macro, for input of data.

Syntax — **OPENDATA** *filename: label*

filename: — The name of the file to be opened.

label — The point in that file (or the current file) which the macro needs to go to.

You must specify either a filename or a label, or both. If no filename is given, the label must be in the current file.

Remarks

1. A label must be on a line by itself and must be followed by a colon (:).
2. If you omit the filename then you must also omit the colon. The macro assumes that the label is in the current macro, and that it defines the data statement to be used for the next **INPUT** statement.
3. A label can be positioned before or after the **GOTO** statement.
4. Only one such file can be open at any time.
5. An error occurs if the file or label requested does not exist.

6. **OPENDATA** is normally used in conjuction with an **IF...THEN...ELSE** construction to determine when the end of the data list has been reached. This test sets the system variable **&ENDOFDATA** to **TRUE** or **FALSE**.

✪ *You must specify the directory in which files are held, unless they are held in the directory you have specified as the default. To check or change your default directory, select **Options>Configure>Macros>Macro directory**.*

Example 1 **OPENDATA** demolist

In this example, the data file demolist, located in the current macro, is opened.

Example 2 **OPENDATA** "C:\WORKLIST":

In this example, the data file WORKLIST, held in the C: directory, is opened.

Example 3 **OPENDATA** "C:\DSWINDOW\MACROS\ATEST.MAC":list

In this example, the data file list, held in the file "ATEST.MAC" in the C:\DSWINDOW\MACROS\ directory, is opened.

Related commands **DATA, ENDDATA, INPUT**

ENDDATA

Function Defines the end of a set of data items.

Example See **DATA**.

Related command **DATA, OPENDATA, INPUT**

INPUT

Function Inputs a line of data into named variables.

Syntax **INPUT *v1,v2,...***

v1,v2 Successive data items

Remarks 1. One input statement reads one line of data, starting with the first, and reading each line successively until all the data has been read. If there are not enough data

items then an error occurs. If there is too much data then the remainder is ignored. Up to 20 items can be input on one line.

2. You can input more than one type of variable per line. They must be separated by commas, and each data item must be enclosed in quotes.

3. When the last item of data has been input the system variable &ENDOFDATA is set to TRUE and the next use of **INPUT** would generate an error.

Example 1

```
OPENDATA companies
loop:
IF &ENDOFDATA = FALSE THEN
        INPUT code
        SEND ( "401A " + code )
        GOTO loop
ENDIF

companies:
DATA
    "BP"
    "BMAH"
    "ENTO"
    "UMAR"
ENDDATA
```

In this example:

- **OPENDATA** opens the data list companies at the end of the macro.
- loop: is a label which marks the start of the loop.
- The **IF** instruction tests whether the end of the data file has been reached. If it has not, then the **INPUT** instruction goes to the data file for the next variable.
- The **SEND** command sends a 401A request with the latest variable - Note that there must be a space between 401A and the closing quote (so that the statement will read **401A BP**. Datastream will not accept **401ABP**.)
- After each **SEND** command, the **GOTO** instruction causes the macro to loop back to the label loop:, to test the **IF** condition again.
- After all the data items have been input, the &ENDOFDATA variable is set to TRUE and the **IF** condition no longer applies. **ENDIF** marks the end of the condition.

- **DATA** marks the start of the data.
- companies: is the label referred to by the OPENDATA command at the start of the macro.
- **ENDDATA** marks the end of the data.

Example 2

```
OPENDATA companies
loop:
IF &ENDOFDATA = FALSE THEN
        INPUT code, index
        SEND ( "401B " )
        SEND  (code + "[TAB]" + index + "-3Y" )
        GOTO loop
ENDIF

companies:
DATA
"BP",           "FTAOILS"
"SBRY",         "FTASTOR"
"D:BMW",        "MOTGPBD"
"J:SO@N",       "ELTNCJP"
ENDDATA
```

- **OPENDATA** opens the data list companies at the end of the macro.
- loop: is a label which marks the start of the loop.
- The **IF** instruction tests whether the end of the data file has been reached. If it has not, then the **INPUT** instruction goes to the data file for the next variable.
- The first **SEND** command sends a 401B request
- The second **SEND** command sends the latest input variables: a code and an index.
- After each **SEND** command, the **GOTO** instruction causes the macro to loop back to the label loop:, to test the **IF** condition again.
- After all the data items have been input, the &ENDOFDATA variable is set to TRUE and the **IF** condition no longer applies. **ENDIF** marks the end of the condition.
- **DATA** marks the start of the data.

- companies: is the label referred to by the OPENDATA command at the start of the macro.
- **ENDDATA** marks the end of the data.
- The input data may be held in a separate file (for example, a file named TEST.LST and held in the C:\DSWINDOW\DATA directory). In this case, the first line of the macro should read:

  OPENDATA "C:\DSWINDOW\DATA\TEST.LST":companies

NOTE *You must specify the directory in which files are held, unless they are held in the directory you have specified as the default. Select the **Macro Directory** sub-command of the **Configure** command on the Options menu to check or change your default directory.*

Related commands **DATA, ENDDATA, OPENDATA**

;

Function — Defines the start of a comment. The comment is assumed to end at the end of the line. You can include comments anywhere in a macro (the macro processor ignores any text following **;** up to the end of the line).

Example

```
AUTOPAGE
; request profit and loss accounts
>190A D:VW
ENDPAGE
```

Remarks — If you want to use the ; character as a semi-colon, enclose it in quotes. For example, **SEND** ("This is the title;")

USERINPUT

Function — Instructs the macro to wait for the user to provide input. A dialog box is opened, providing an edit line for the user to type the input.

Syntax — **USERINPUT *title, prompt, variable***

title	The title for the dialog box.
prompt	The prompt in the dialog box.
variable	A string - the name of the variable which will be replaced by the user's input.

Example

```
USERINPUT "Graphics request", "Please type a code", x
SEND ("401A " + x)
```

- **USERINPUT** opens a dialog box with the title "Graphics request", prompting the user to type in a code.
- The **SEND** command requests a 401A chart of the instrument whose code the user has typed in. (The code replaces variable *x* in the macro.)

Related commands — **MESSAGE, INPUT**

WAIT

Function — Instructs the macro to wait until the specified time and date, or for a specified number of seconds, before continuing.

Syntax — **WAIT** *(**time**, date)* / **WAIT** *(**forSeconds**)*

time	The time of day to wait until. It must be in the format: hh:mm:ss and be surrounded by quotes.
date	The date to wait until. It must be in the configured format and be surrounded by quotes. If the date is the next day, or a particular number of days in the future, you can specify it in the format "+1D", etc. If this is a Saturday, the macro will adjust and run on the following Monday.
forSeconds	The number of seconds to wait.

Example 1 — **WAIT** ("16:42:0", "06/12/93")

In this example, **WAIT** instructs the macro to wait until 16:42 on the 6th December 1993.

Example 2 — **WAIT** (10)

In this example, **WAIT** instructs the macro to wait for 10 seconds.

Example 3: — **WAIT** ("01:15:0", "+1D")

In this example, **WAIT** instructs the macro to wait until 01:15 on the following day.

SET...TO...

Function — Assigns a value to a variable; the value can be a constant, expression, or another variable.

Syntax — **SET *v1 TO v2***

v1 — The variable to which you want to assign a value.

v2 — The value to be assigned.

Remarks — Use mathematical operators to form expressions. Refer to "Functions and Expressions" for details.

Example

```
LOADLAYOUT ("four")
SET count TO 0
OPENDATA list
Loop:
IF &ENDOFDATA = FALSE THEN
        INPUT code
        SEND ("401A" +" code")
        SET count TO count + 1
        IF count = 4 THEN
                PRINTGRAPHICS
                SET count TO 0
        ENDIF
        GOTO Loop
ENDIF
```

- A layout of four slots is loaded.
- 'count' is set to a value of 0.
- Company codes are requested from a data file called 'list'.
- Each time a chart is drawn, the value of 'count' increments by one.
- When the value of 'count' reaches 4 (i.e. the layout is full), the layout is printed, and the value of 'count' is set to 0 again.

Related commands — **SETGLOBAL...TO**

SETGLOBAL...TO...

Function — Assigns a value to a global variable; the value can be a constant, expression, or another variable. See the chapter 'Constants and variables' for further information.

Syntax — **SETGLOBAL *v1* TO *v2***

v1	The variable to which you want to assign a value.
v2	The value to be assigned.

Example — **SETGLOBAL** x **TO** 10

In this example, x becomes 10. In this and any **CALL**ed macros, x will have the value 10.

Related commands — **SET...TO, CALL**

Graphics

NOTE *This section assumes that you know how to generate, display, configure and annotate graphs within DSWindows, and that you understand the concepts of slots and layouts. For details on these subjects, please refer to the DSWindows 2.1 User Guide.*

DISPLAYSINGLEGRAPH

Function — Sets the mode of the Graphics window to display a single graph.

Example

```
LOADLAYOUTFILE ("C:\DSWINDOW\FILES\5LAY","5 GRAPH LAYOUT")
DISPLAYSINGLEGRAPH
```

- **LOADLAYOUTFILE** loads the file '5LAY' and the layout '5 GRAPH LAYOUT'.
- **DISPLAYSINGLEGRAPH** displays the first graph in the layout as a single graph.

Related commands — **DISPLAYLAYOUT**

DISPLAYLAYOUT

Function — Sets the mode of the Graphics window to display a layout.

Example

```
LOADGRAPHFILE\
 ("C:\DSWINDOW\FILES\RETAIL",2,"MARKS & SPENCER")
DISPLAYLAYOUT
```

- **LOADGRAPHFILE** loads the single graph 'Marks & Spencer' from the file 'Retail'.
- **DISPLAYLAYOUT** displays it in slot 2 in a layout.

Related command — **DISPLAYSINGLEGRAPH**

SAVEGRAPHICS

Function — Saves a single graph..

SAVEGRAPHICS*(file:* ***filename****, graph: graphname,append:flag)*

filename	The name of the save file. (It may be relative or a full path name). If you not provide an extension, the default extension (.DSG) is provided. If no filename is specified, the command is ignored.
graphname	The name for the graph. This is optional. If a graph of this name already exists in the file then an alphabetic character is added to the name. If no name is specified the existing graph name is used, or the first piece of text in the graph.
flag	OVERWRITE or APPEND (Default = APPEND) OVERWRITE = a new file is created APPEND = the graph is appended to the specified file.

✪ *Use this command if you are saving annotations. Do not use AUTOSAVE if you want to save annotations.*

Example 1 — Saves a graph with the name "DEMO GRAPH" to the file "DEMO401.DSG", overwriting the previous contents of the file.

```
SAVEGRAPHICS ( "DEMO401.DSG", "DEMO GRAPH",OVERWRITE )
DISPLAYGRAPH
```

Example 2 — Saves a layout with the name "DEMO LAYOUT" to the file named "DEMO401.DSG", overwriting the previous contents of the file.

```
SAVEGRAPHICS ( "DEMO401.DSG","DEMO LAYOUT",OVERWRITE )
DISPLAYLAYOUT
```

Related commands — **SAVEWMF, AUTOSAVE, EXPORTGRAPHICS, PRINTGRAPHFILE, PRINTLAYOUTFILE, LOADGRAPHFILE, LOADLAYOUTFILE**

SAVEWMF

Function — Saves a single graph or layout in Windows Metafile format (.WMF).

Syntax — **SAVEWMF*(filename)***

filename — The name of the save file. It may be relative or a full path name. If you not provide an extension, the default extension (.WMF) is provided. If no filename is specified, the command is ignored.

Example — **SAVEWMF** ("DEMO401.WMF")

- The current graph is saved into a Windows Metafile named "DEMO401.WMF". By default, any file of the same name will be overwritten.

Related commands — **SAVEGRAPHICS, AUTOSAVE, EXPORTGRAPHICS**

AUTOSAVE

Function — Starts autosaving: all graphs received following this command will be saved to the specified file. Any annotations made using the Graphics annotation commands will not be saved using this command - use the **SAVEGRAPHICS** command to save these.

Syntax — **AUTOSAVE** *(file: **filename**,append:flag)*

filename — The name of the save file.

If no filename is specified, the command is ignored.

flag — OVERWRITE or APPEND (Default = APPEND.)

OVERWRITE = a new file is created
APPEND = the graph is appended to the specified file.

Example —
AUTOSAVE ("DEMO401.DSG")
ENDAUTOSAVE

- All graphs received from now will be saved into the file "DEMO401.DSG", until the **ENDAUTOSAVE** command is given.

Related commands **SAVEGRAPHICS, SAVEWMF, ENDAUTOSAVE**

ENDAUTOSAVE

Function Ends autosaving.

Example See above.

Related command **AUTOSAVE**

EXPORTGRAPHICS

Function Exports the currently displayed graphics (a single graph or a layout) to a file in a format specified by the filter. Encapsulated Postcript (EPS) and Computer Graphics Metafile (CGM) are currently supported.

Syntax **EXPORTGRAPHICS** ***(filtername,filename)***

filtername	The name of the export filter (for example, DSCGM.DLL or DSEPS.DLL). If it is not on the same path as DSWindows, you must specify the full path name.
filename	The file into which the graph or layout is to be saved. The extension is added by the export filter (for example, .CGM).

Remarks If DSWindows fails to load the filter (for example, because it cannot find it on the specified or default path), then the command is ignored.

Example **EXPORTGRAPHICS** ("dscgm.dll","BTJAN93.CGM")

- The currently displayed graph is exported as a .CGM file called 'BTJAN93.CGM'.

Related commands **SAVEGRAPHICS, AUTOSAVE, SAVEWMF**

PRINTGRAPHICS

Function Prints the currently displayed graphic (a single graph or a layout).

Example **> 401H BT**
PRINTGRAPHICS

- Requests a high/low/close chart for BT and **prints it.**

Related commands **PRINTGRAPHFILE, PRINTLAYOUTFILE, AUTOPRINT**

PRINTGRAPHFILE

Function Prints graphs from the file specified.

Syntax **PRINTGRAPHFILE** *(**filename**,GraphName1,GraphName2,.....)*

filename The name of the file in which the graphs are stored.

GraphName1, GraphName2 The names of the graphs to be printed.

You can specify any number of graph names. If no graph names are specified then all the graphs in the file are printed.

Example **PRINTGRAPHFILE** ("BP401.DSG","JAN","FEB","MAR")

- Three graphs named "JAN", "FEB" and "MAR", stored in the file "BP401.DSG", are printed.

Related commands **PRINTGRAPHICS, PRINTLAYOUTFILE, AUTOPRINT**

PRINTLAYOUTFILE

Function Prints layouts from the file specified.

Syntax: **PRINTLAYOUTFILE** *(**filename**,Layout1,Layout2,.....)*

filename The name of the file in which the layouts are stored.

Layout1,Layout2 The names of the layouts to be printed.

You can specify any number of layout names. If no layout names are specified then all the layouts in the file are printed.

Example **PRINTLAYOUTFILE** ("BP401.DSG","BPMAR")

- The layout named "BPMAR", stored in the file "BP401.DSG", is printed.

LOADGRAPHFILE

Function Loads graphs from the specified file into the main Graphics window.

Syntax **LOADGRAPHFILE** *(**filename**,Slotnumber,GraphName1,GraphName2,...)*

filename	The name of the file in which the graphs are stored.
Slotnumber	The number of the first slot into which the graphs are to be loaded. If no slot number is specified then the graphs are loaded into slot zero and if a second graph is specified, the first is moved up to slot -1, and so on.
GraphName1 *GraphName2*	The names of the graphs to be loaded.

You can specify any number of graph names. If no graph names are specified, all graphs in the file are loaded.

Example **LOADGRAPHFILE** ("BP401.DSG",5,"JAN","FEB","MAR")

Three graphs named "Jan", "Feb" and "Mar", stored in the file "BP401.DSG", are loaded into slots 5, 6 and 7.

Related command **LOADLAYOUTFILE**

LOADLAYOUTFILE

Function Loads a layout from the file specified.

Syntax **LOADLAYOUTFILE** *(**filename**,LayoutName)*

filename	The name of the file in which the layout is stored.
LayoutName	The name of the layout to be loaded.

If no layout is specified then the last layout in the file is loaded.

Example **LOADLAYOUTFILE** ("BP401.DSG","BPMAR")

- The layout named "BPMAR", stored in the file "BP401.DSG", is loaded.

Related command **LOADGRAPHFILE**

GRAPHPAGESETUP

Function Sets up the Graphics printer page.

Syntax **GRAPHPAGESETUP** *(left:l,right:r,top:t,bottom:b,maptoblack:m,orientation:o)*

left:l The left margin setting, in 1/100 inches.
l can be any number from 1 to 32,767.

right:r The right margin setting, in 1/100 inches.
r can be any number from 1 to 32,767.

top:t The top margin setting, in 1/100 inches.
t can be any number from 1 to 32,767.

bottom:b The bottom margin setting, in 1/100 inches.
b can be any number from 1 to 32,767.

maptoblack:m Specifies whether colours are to be mapped to black.

m can be:
- 0 do not map to black. Dithering is used.
- 1 for map to black.

orientation:o Defines the orientation of the printout.

o can be:
- "portrait"
- "landscape"

All parameters are optional; if you omit any, the previously specified settings will apply. Parameter names must be included.

Example
```
GRAPHPAGESETUP\
(left:50,right:50,top:70,bottom:100,maptoblack:1,orientation:"landscape")
```

- ❑ The graphics printer page is set up to print:

  left and right margins of 0.5"
  top margin of 0.7"
  bottom margin of 1"
  colours mapped to black
  landscape format.

Related commands **PRINTGRAPHICS, AUTOPRINT, PRINTGRAPHFILE**

LOADLAYOUT

Function Loads a layout (that is, an arrangement of slots saved with a layout name).

Syntax **LOADLAYOUT** *(layoutname)*

layoutname The name of the layout to be loaded.

If no name is specified, then the default layout is loaded.

Remarks This command does not change which graphs are displayed; it changes the position of the slots according to the layout named.

Example **LOADLAYOUT** ("3equal slots")

- ❑ Loads a layout stored under the name "3equal slots".

Related commands **LOADFILLSTYLES, LOADTEXTSTYLES, LOADLINESTYLES**

LOADLINESTYLES

Function Loads a line style.

Syntax **LOADLINESTYLES** *(stylename)*

stylename The name of the set of line styles to be loaded.

If no style name is specified, then the default set of line styles is loaded.

Example **LOADLINESTYLES** ("Daily-Report")

- ❑ A user-defined line style called "Daily-Report" is loaded.

Related commands **LOADFILLSTYLES, LOADTEXTSTYLES**

LOADFILLSTYLES

Function Loads a set of fill styles.

Syntax **LOADFILLSTYLES** *(stylename)*

stylename The name of the fill styles are loaded.

If no style name is specified, then the default fill style is loaded.

Example **LOADFILLSTYLES** ("Daily-Report")

- the set of user-defined fill styles called "Daily-Report" is loaded.

Related commands **LOADLINESTYLES, LOADTEXTSTYLES**

LOADTEXTSTYLES

Function Loads a set of text styles.

Syntax: **LOADTEXTSTYLES** *(stylename)*

stylename The name of the text style to be loaded.

If no style name is specified, then the default text styles are loaded.

Example: **LOADTEXTSTYLES** ("Daily-Report")

- a set of user-defined text styles called "Daily-Report" is loaded.

Related commands **LOADLINESTYLES, LOADFILLSTYLES**

SETGRAPHNAME

Function — Sets the name for the current graph (not the name of the file in which the graph is stored). This applies only to single graphs.

Syntax — **SETGRAPHNAME** *(name)*

name	The new name for the current graph. It must be enclosed in quotes.

Remarks — Once set, the graph name can be used to select / deselect a graphics slot using **SELECT / DESELECT**, specify which graph to load using **LOADGRAPHICS**, or which graph to print using **PRINTGRAPHICS**.

If the name of a graph is not set using **SETGRAPHNAME**, the graph title is used. If there is no graph title, the first piece of text in the graph (usually the legend) is used.

Example — Requests a graph using the code for British Telecom, sets the name of the graph (BT 11/93 - 11/96) and saves the graph in a file called 'BT'.

```
> 401A BT
SETGRAPHNAME ("BT 11/93 - 11/96")
SAVEGRAPHICS ("C:\DSWINDOW\FILES\BT","BT 11/93 - 11/96")
```

Related commands — **SELECTGRAPH, DESELECTGRAPH, LOADGRAPHICS, PRINTGRAPHICS**

SELECTGRAPH

Function — Selects a graph for display in single graph mode, or adds a slot to a layout.

Syntax — **SELECTGRAPH** *(GRAPH:**name**)* *Or* **SELECTGRAPH** *(GRAPH:**number**)*

name — The name of the graph to be loaded. It must be enclosed in quotes. For details on how graph names are set, refer to **SETGRAPHNAME**.

number — The number (0 to -7) of the graph to be displayed.

Example 1 — **SELECTGRAPH** (GRAPH: "CONSTRUCTION1992")

Example 2 — **SELECTGRAPH** (GRAPH:-5)

Related commands — **DESELECTGRAPH, SETGRAPHNAME**

DESELECTGRAPH

Function — Hides a selected graph in a layout.

DESELECTGRAPH *(GRAPH:**name**)* Or *(GRAPH:**number**)*

name — The name of the graph to be removed. It must be enclosed in quotes. For details on how graph names are set, refer to **SETGRAPHNAME**.

number — The number (0 to -7) of the graph to be removed.

Example 1 — **DESELECTGRAPH** (GRAPH: "CONSTRUCTION1992")

Example 2 — **DESELECTGRAPH** (GRAPH:-5)

Related commands — **SELECTGRAPH**, **SETGRAPHNAME**

Graphic annotations

The commands used to annotate graphs are organised in four functional groups:

- **Selecting/deselecting**: select the items to be modified, deleted, moved or copied. (An item on a graph can be a segment of text, a line, a box, a pie chart, etc.)
- **Amending**: change the attributes of items, delete, move or copy items
- **Creating new items**: you can create items which apply to a whole layout, for example, the title of the layout in a box.
- **Redrawing**: suspend redrawing, so that a number of amendments are redrawn in one operation.

1 Annotations apply to the currently selected graph; you can apply any annotation command to a single graph.

2 Only commands which create **new** *items can be applied to a layout. In layout mode, any items you create belong to the layout, not to an individual graph in the layout.*

♦ To load a single graph for annotation

- Use the **LOADGRAPHFILE** command.

♦ To load a layout for annotation

- Use the **LOADLAYOUTFILE** command.

Items

Each of the following elements on a graph is considered to be an "item":

- The title
- The sub-title
- The X axis
- The Y axis
- The grid
- Each segment of text on the X axis
- Each segment of text on the Y axis

- Each line in a line chart
- Each bar in a bar chart
- Each legend
- Each segment of a pie

The annotation commands enable you to identify these items by their coordinate position in the window, or by their type, contents or style, and to change their position, contents or styles.

The following introductory paragraphs describe general features which apply to the annotation commands.

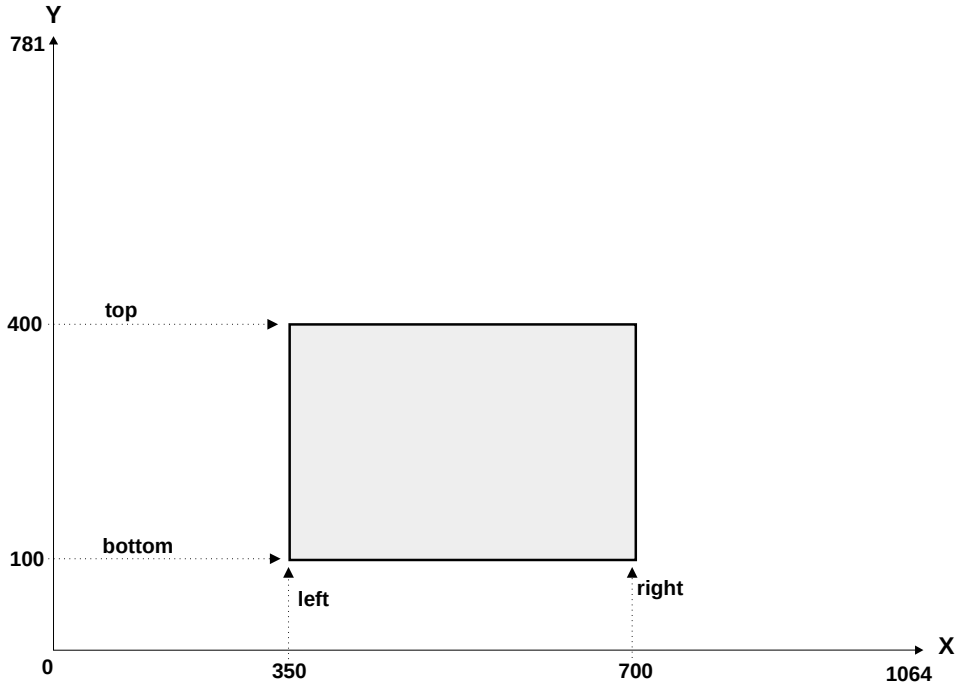
Coordinates

Every item on a graph is positioned by default in a certain location in the Graphics window. The locations are defined by a coordinate system, in which the window consists of 1,065 points (0 - 1064) on the horizontal (X) axis, and 782 points (0 - 781) on the vertical (Y) axis.

The position of any item is defined by the coordinate positions of the left, top, right and bottom sides of the item. When you move an item, its recorded position in the window is changed.

Using quotes

When you identify or change text items you must enclose the text in quotes. Similarly, when you identify an item by its style, the style name must be enclosed in quotes.



Graphic coordinate system

Selecting/deselecting

SELECTITEMS

Function

Selects and highlights one or more items in the current graph. A number of optional parameters enable you to select different kinds of item, either by location on the graph, by the type of item, by the style (fill, line or text), or by the whole or part of the text in text items. If no items match the specifications, then none are selected.

Syntax

SELECTITEMS *(AT:x,y, IN:l,t,r,b, TYPE:string, STYLE:string, TEXT:string, STARTS: string, CONTAINS: string)*

AT:x,y

Identifies the coordinate position of an **item**.

x and y are the coordinates.

x can be between 0 and 1064, the left and right margins respectively.

y can be between 0 and 781, the bottom and top margins respectively.

The item closest to this position is selected. Alternatively use IN to select one or more items within a rectangle.

IN:l, t, r, b

Identifies a **rectangle**.

l, t, b, and t are the left, right, bottom and top positions of a rectangle

l and r are coordinate positions between 0 and 1064.

b and t are coordinate positions between 0 and 781

TYPE:string

Identifies a type of item.

String can be any of the following:

polygon (e.g. the shading below a line or pie segments).
line
text
arc
rectangle
box
vertical text

STYLE:string — Identifies items by fill, line or text style, where string is the new style, enclosed in quotes.

The style name must be as shown in the annotation list boxes. You can use only one style per command.

TEXT:string — Identifies complete text items where string is the text, enclosed in quotes. Capital and lower case characters must match.

STARTS:string — Identifies items by a text string at the start of the item, where string is the text, enclosed in quotes. Capital and lower case characters must match.

CONTAINS:string — Identifies items by a text string within the item, where string is the text, enclosed in quotes. Capital and lower case characters must match.

Remarks

- Use one or more parameters to define the item(s) to be selected.
- In any one use of the command, most parameters are optional. Defaults are assumed for missing parameters as necessary.
- IN is an alternative to AT.
- Inappropriate parameters are ignored.
- Commands fail only if mandatory parameters are missing.
- All items matching the definition are selected. Equally, any items you want to select must match the parameter(s) you specify. Any previously selected item is deselected.
- If you specify no parameters, all items are selected.

Example 1 **SELECTITEMS** *(TYPE:"LINE",IN:100,200,100,200)*

- All lines within the rectangle defined by the coordinates 100, 200, 100, 200 are selected.

Example 1 **SELECTITEMS** *(CONTAINS:"FTSE")*

- Any text string which contains the characters "FTSE" is selected.

Related commands **ADDTOSELECTITEMS, REFINESELECTITEMS, DESELECTITEMS**

ADDTOSELECTITEMS

Function Adds specified item(s) to currently selected items, i.e. any previous selection is maintained.

Syntax **ADDTOSELECTITEMS** *(AT:x,y, IN:l,t,r,b, TYPE:string, STYLE:string, TEXT: string, STARTS: string, CONTAINS: string)*

(Parameters) As for **SELECTITEMS**

Remarks As for **SELECTITEMS**

Example 1 **ADDTOSELECTITEMS** *(TYPE: "POLYGON")*

- All pie charts are added to the current selection.

Example 2
```
SELECTITEMS (STYLE: "Title")
CHANGEITEMS (TEXT: "FTSE")
ADDTOSELECTITEMS (STYLE: "Sub-title")
NEWBOX
```

- The title is selected, the text of the title is changed, the sub-title is selected also, and a box is drawn round the two items.

Related commands **SELECTITEMS, REFINESELECTITEMS, DESELECTITEMS**

REFINESELECTITEMS

Function — Refine the currently selected items, i.e. only those items which match the parameters specified remain selected.

Syntax — **REFINESELECTITEMS** *(AT:x,y, IN:l,t,r,b, TYPE:string, STYLE:string, TEXT: string, STARTS: string, CONTAINS: string)*

Parameters — As for **SELECTITEMS**.

Remarks — As for **SELECTITEMS**.

Example —
SELECTITEMS *(TYPE:"LINE")*
REFINESELECTITEMS *(STYLE:"Line Style 1")*

- **SELECTITEMS** selects all lines.
- **REFINESELECTITEMS** selects from those all lines with "Line Style 1".

Related commands — **SELECTITEMS, ADDTOSELECTITEMS, DESELECTITEMS**

DESELECTITEMS

Function — Deselects any currently selected item which matches the parameters specified.

Syntax — **DESELECTITEMS** *(AT:x,y, IN:l,t,r,b, TYPE:string, STYLE:string, TEXT: string, STARTS: string, CONTAINS: string)*

Parameters — As for **SELECTITEMS**.

If no parameters are specified all items are deselected.

Remarks — As for **SELECTITEMS**.

Example — **DESELECTITEMS**

Deselects all items.

Related commands — **SELECTITEMS, ADDTOSELECTITEMS, REFINESELECTITEMS**

Amending items

MOVEITEMS

Function — Moves currently selected items.

Syntax — **MOVEITEMS** *(TO:x,y)*

TO:x,y — Defines a new coordinate position for the top left corner of the selected items. (The other coordinates follow accordingly.)

x and y are the coordinates

x can be between 0 and 1064, the left and right margins respectively

y can be between 0 and 781, the bottom and top margins respectively

OR:

MOVEITEMS *(BY:dx,dy)*

BY:dx,dy — Defines the number of points by which the selected items are to move.

dx and dy are the coordinates

dx can be between -1064 and 1064

dy can be between -781 and 781

Remarks

- If you specify no parameters, nothing is moved.
- You can use negative coordinates.
- You can move items off the visible area of the screen (and move them back again.)
- Items remain selected after moving.

Example 1 **MOVEITEMS** *(TO:500,500)*

- All the selected items are moved to the coordinate position 500:500.

Example 2 **MOVEITEMS** *(BY:100,100)*

- All the selected items are moved 100 coordinate points up and 100 coordinate points right towards the top right corner of the screen.

Related commands **COPYITEMS**

COPYITEMS

Function Copies the currently selected items to the position specified.

Syntax **COPYITEMS** *(TO:x,y)* Or **COPYITEMS** *(BY:dx,dy)*

Parameters As for **MOVEITEMS**.

Remarks

- As for **MOVEITEMS**.
- If you specify no parameters, the new items are positioned on top of the original ones.
- The new items remain selected.

Example

SELECTITEMS *(AT:100,100,TYPE:"LINE")*
COPYITEMS *(TO:500,500)*

- **SELECTITEMS** selects the line closest to the coordinate position 100,100.
- **COPYITEMS** copies the line to the position 500,500.

Related commands **MOVEITEMS**

DELETEITEMS

Function Deletes the currently selected items.

Example 1

SELECTITEMS *(STYLE: "Sub-title")*
DELETEITEMS

- **SELECTITEMS** selects the sub-title.
- **DELETEITEMS** deletes it.

Example 2 **SELECTITEMS**
DELETEITEMS

- **SELECTITEMS** selects all items.
- **DELETEITEMS** deletes all items (that is, the whole graph).

Related commands **SELECTITEMS**

CHANGEITEMS

Function Changes the attributes of the currently selected items.

Syntax **CHANGEITEMS** *(TEXT:text,STYLE:style,FILL:fill,SHADOW:shadowstyle, POSITION:textposition,SHADOWBORDER:linestyle,BORDER:linestyle, BORDERGAP:gap,SHADOWOFFSET:offset,WIDTH:textitemlength)*

TEXT:text	Defines a new text string, where text is the new string, enclosed in quotes.
STYLE:style	Defines a new fill, line or text style, where style is the new style, enclosed in quotes. The style name must be as shown in the annotation list boxes. You can use only one style per command.
FILL:fill	Defines a new fill style where fill is the new style, enclosed in quotes. The style name must be as shown in the annotation list boxes. You can use only one style per command.
SHADOW:shadowstyle	Defines a new fill style for shadows around boxes and rectangles, where shadowstyle is the new style, enclosed in quotes. The style name must be as shown in the annotation list boxes. You can use only one style per command.
POSITION:textposition	Defines the alignment of text.

textposition can be:	r, l or c for right, left or centre, enclosed in quotes.
SHADOWBORDER: linestyle	Defines the linestyle for the edge of boxes and rectangles, where linestyle is the new style, enclosed in quotes. The style name must be as shown in the annotation list boxes. You can use only one style per command.
BORDER:linestyle	Defines the linestyle for the edge of boxes and rectangles, where linestyle is the new style, enclosed in quotes. The style name must be as shown in the annotation list boxes. You can use only one style per comma
BORDERGAP:gap	Defines the gap between the contents and the borders of boxes and rectangles.
SHADOWOFFSET: offset	Defines the distance between boxes and rectangles and their shadows.
WIDTH:width	Defines the length of a text item.

Remarks

- The styles you specify must be as listed in the annotation list boxes.
- The selected items adopt any of the attributes which are applicable to them, even if the command includes other attributes which do not apply.
- A FILL: parameter overrides a fill style specified by the STYLE: parameter Use FILL as well as STYLE when a new box or a new rectangle has been created. The FILL applies to them.
- If you want to change the positions of items, use the **MOVEITEMS** command.
- Items remain selected, whether or not they have been changed.

Example

SELECTITEMS *(TYPE:"RECTANGLE")*
CHANGEITEMS *(SHADOW:"Fill Style 4")*

- **SELECTITEMS** selects all rectangles.
- **CHANGEITEMS** changes the fill style to Fill Style 4.

Related commands **MOVEITEMS, COPYITEMS**

Creating new items

NEWBOX

Function

Creates new boxes to enclose the currently selected items. The boxes are sized automatically to surround the items.

Syntax

NEWBOX*(FILL:fillstyle,BORDER:linestyle,SHADOW:fillstyle, SHADOWBORDER:linestyle,SHADOWOFFSET:offset,BORDERGAP:gap)*

FILL:fillstyle

Defines a fill style, where FILL is the fill style, enclosed in quotes. The style name must be as shown in the Fill Style Configuration list box. You can use only one style per command. To specify no fill for the box, use "No Fill".

BORDER:linestyle

Defines the line style for the edge, where linestyle is the style, enclosed in quotes. The style name must be as shown in the Line Style Configuration list box. You can use only one style per command. To specify no border for the box, use "No Style".

SHADOW: shadowstyle

Defines a fill style for shadows, where shadowstyle is the style, enclosed in quotes. The style name must be as shown in the Fill Style Configuration list box.You can use only one style per command. To specify no fill for the box, use "No Fill".

SHADOWBORDER: linestyle

Defines the linestyle for the shadow border, where linestyle is the style, enclosed in quotes. The style name must be as shown in the Line Style Configuration list box. You can use only one style per command. To specify no border for the box, use "No Style".

SHADOWOFFSET: offset

Defines the distance between the shadow and the rectangle.

BORDERGAP:gap Defines the gap between the contents and the borders. The size of the gap affects the size of the box.

Remarks

- The styles you specify must be as listed in the annotation list boxes.
- All parameters are optional.
- Default values are taken from the current system default, i.e. the last settings used for a box or rectangle.
- Default settings are updated every time you specify new parameters.

Example 1

SELECTITEMS *(STYLE:"Legend",TYPE:"TEXT")*
NEWBOX *(FILL:"Fill Style 3",BORDERGAP:30)*

- **SELECTITEMS** selects text items with the style "Legend".
- **NEWBOX** boxes it in a box with Fill Style 3 and a border gap of 30 points.

Example 2

SELECTITEMS *(IN:0,0,600,70)*
NEWBOX*(FILL:"No Fill",BORDER:"Axis",SHADOW:"No Fill",*
SHADOWBORDER:"Axis",SHADOWOFFSET:5)

- **SELECTITEMS** command is used to select all the item(s) in the area which has a lefthand co-ordinate (l) = 0, top (t) = 70, righthand (r) = 600 and bottom (b) = 0 - the legend in this instance
- **NEWBOX** boxes in the items with an unfilled (transparent) bordered box and an offset unfilled shadow box.

Related commands **SELECTITEMS**

NEWRECT

Function

Creates a new fixed-size rectangle.

Syntax

NEWRECT
*(**FROM:**x1,y1,**TO:**x2,y2,FILL:fillstyle,BORDER:linestyle,SHADOW:fillstyle, SHADOWBORDER:linestyle,SHADOWOFFSET:offset)*

FROM:x1,y1

Defines the first-drawn coordinate point of the rectangle (for example, the top left-hand corner).

x1 and y1 are the coordinates.

x1 can be between 0 and 1064.

y1 can be between 0 and 781.

TO:x2,y2

Defines the last-drawn coordinates of the rectangle (for example, the bottom right-hand corner).

x2 and y2 are the coordinates.

x2 can be between 0 and 1064.

y2 can be between 0 and 781.

FILL:fill

Defines a fill style, where fill is the fill style, enclosed in quotes. The style name must be as shown in the Fill Style Configuration list box. You can use only one style per command. To specify no fill for the box, use "No Fill".

BORDER:linestyle

Defines the linestyle for the edge, where linestyle is the style, enclosed in quotes. The style name must be as shown in the Line Style Configuration list box. You can use only one style per command. To specify no border for the box, use "No Style".

SHADOW:
shadowstyle
Defines a fill style for shadows, where shadowstyle is the fill style, enclosed in quotes. The style name must be as shown in the Fill Style Configuration list box. You can use only one style per command. To specify no fill for the box, use "No Fill".

SHADOWBORDER:
linestyle
Defines the linestyle for the shadow border, where linestyle is the style, enclosed in quotes. The style name must be as shown in the Line Style Configuration list box. You can use only one style per command. To specify no border for the box, use "No Style".

SHADOWOFFSET:
offset
Defines the distance between the shadow and the rectangle.

Remarks

- The styles you specify must be as listed in the annotation list boxes.
- All parameters are optional, except FROM and TO.
- Default values are taken from the current system default, i.e. the last settings used for a box or rectangle.
- Default settings are updated every time you specify new parameters.

Example

NEWRECT *(**FROM:**10,10, **TO:**100,100)*

- **NEWRECT** creates a new rectangle using the current default settings at the position specified.

Related commands **NEWBOX**

NEWTEXT

Function — Creates a new text string.

Syntax — **NEWTEXT** *(TEXT:newtext,AT:x,y,STYLE:textstyle,POSITION:textposition)*

TEXT:newtext	Defines the new text string where newtext is the new string, enclosed in quotes.
AT:x,y	Defines the coordinate position at which the start of the new text item is positioned. x can be between 0 and 1064 and defines the left margin position of the text on the screen. y can be between 0 and 781 and defines the height coordinate at which the text is positioned on the screen.
STYLE:textstyle	Defines the text style for the new text item, where textstyle is the name of the text style. It must be the name of a style in the annotation list box.
POSITION:textposition	Defines the alignment of text textposition can be: r, l or c for right, left or centre, enclosed in quotes.

Example — **NEWTEXT** *(TEXT:"Annual prices",POSITION:"r",AT:100,300)*

- The text "Annual prices" is added, right-aligned, at the coordinate point 100, 300.

Related commands — **NEWBOX, NEWRECT, NEWLINE**

NEWLINE

Function — Creates a new straight line defined by two points.

Syntax — **NEWLINE** ***(FROM:x1,y1,TO:x2,y2,****STYLE:linestyle)*

FROM:x1,y1	Defines the start point coordinates of the line: x1 can be between 0 and 1064 y1 can be between 0 and 781
TO:x2,y2	Defines the end point coordinates of the line: x2 can be between 0 and 1064 y2 can be between 0 and 781
STYLE:linestyle	Defines the linestyle for the new line, where linestyle is the style, enclosed in quotes. The style name must be as shown in the annotation list boxes. You can use only one style per command "Line style 1" is the default, if none is specified.

Example — **NEWLINE** *(FROM:10,10,TO:100,100,STYLE:"Arrow 1")*

- In this example, a line is drawn, starting at position 10,10 and ending at position 100,100, with the line style "Arrow 1".

Related commands — **NEWBOX, NEWRECT, NEWTEXT**

Redrawing

GRAPHDRAWOFF

Function — Suspends all re-drawing of the current graph. This is useful before a large number of annotation commands, because it speeds up the macro.

Remarks
- All redraws, including mouse-driven highlights, are suspended.
- The annotation commands must be followed by **GRAPHDRAWON**. The drawing is suspended until until this command has been processed.

Example

```
GRAPHDRAWOFF
SELECTITEMS
MOVEITEMS (BY:100,100)
CHANGEITEMS (STYLE:"Fill Style 4")
REFINESELECTITEMS (TYPE:"TEXT")
CHANGEITEMS (TEXT:"Something New")
GRAPHDRAWON
```

- **GRAPHDRAWOFF** suspends re-drawing.
- **SELECTITEMS** selects all items on the graph.
- **MOVEITEMS** moves all items by 100 coordinate points to the right and 100 up.
- **CHANGEITEMS** sets all fill styles to Fill style 4.
- **REFINESELECTITEMS** selects all text items.
- **CHANGEITEMS** changes all text items to the string "Something New".
- **GRAPHDRAWON** redraws the results of these changes.

Related command — **GRAPHDRAWON**

GRAPHDRAWON

Function — Starts re-drawing of the current graph (after a **GRAPHDRAWOFF**).

Example — See **GRAPHDRAWOFF**

Related command — **GRAPHDRAWOFF**

Data Channel/Fundline

Use the commands in this section to automate 900 (Data Channel) and 907 (Fundline) requests.

NOTE *The Data Channel display options, such as transposing column/row headings, are manually configured in the Configure Data Channel Translation dialog box (Options>Configure>Data Channel/Fundline Translator....).*

CONFIGUREDC

Function

Sets the various configurable options for the Data Channel programs.

Syntax

CONFIGUREDC *(Merge900A: n,* 1 = merge output into one table, 0 = dont
Transpose900A:n, 1 = transpose ON, 0 = transpose OFF
Titles900A: n, 1 = put date as title, 0 = don't
ColHeadings900A: n, 1 = put col headings in output, 0 = don't
RowHeadings900A: n, 1 = put row headings in output, 0 = don't
Merge900B: n, 1 = merge output into one table, 0 = dont
Transpose900B: n, 1 = transpose ON, 0 = transpose OFF
Titles900B: n, 1 = put start date, end date, frequency, 0 = don't
ColHeadings900B: n, 1 = put col headings in output, 0 = don't
RowHeadings900B: n, 1 = put row headings in output, 0 = don't
Codes900C: n, 1 = code/mnemonic for each co., 0 = name only
QuoteText: n, 1 = quotes around output text items, 0 = don't
QuoteNumbers: n, 1 = quotes around output numbers, 0 = don't
Prompt: n, 1 = display DC config dialog every time DC translation turned on, 0 = don't
Separator: "text" character(s) used to separate items
EndOfLine: "text" character(s) to be output at end of each line
DecimalSep: "text" character used to separate units from tenths
NotAvailable: "text" string to output if no value is available
QuoteChar: "text") character to output as quote character

NOTE: *Options for n: 1 or 0 as listed above*
Options for text: Separator: usually one of [SPACE] [TAB] , ;
EndOfLine usually one of [CR][LF] [LF] [CR]
DecimalSep usually one of , .

NotAvailable usually #n/a
QuoteChar usually one of: " '

1. *A string must be inside double quotes. Characters which are not quoted are keywords or identifiers for variables.*
2. *To embed a non-printable character or double quote in a string, CHR$ function should be used. For example, myStringWithQuote = "any" + CHR$(34), where 34 is the ASCII code for a double quote.*
3. *[SPACE], [ENTER] etc can be embedded inside the string for SEND-related commands only. No other commands, such as ConfigureDC, understand the embedded characters.*
4. *If the decimal separator is the same as the separator then semi-colons (;) will be used as the separator between items*
5. *Squared brackets are only used around [TAB], [SPACE], [CR] and [LF] - and only for the Separator and EndOfLine parameters.*

Example

CONFIGUREDC (*Merge900B: 1, 0, 0, 1, 0)*

- This example changes the setting for Merge900B and the four items which follow it. Note that if items are listed in the order specified above, the parameter labels can be omitted. This macro therefore sets 5 900B options:

Merge900B	output is merged into one rectangular table
Transpose900B	row of data for each series
Titles900B	don't output start date, end date and frequency
ColHeadings900B	include column headings in output
RowHeadings900B	don't include row headings in output

Remarks

- Any settings that you do not wish to change can be omitted.
- Refer to the *DSWindows 2.1 User Guide* for an explanation of the various settings - they are the parameters set via the Configure Data Channel Translator dialog box.

STARTDC

Function

Starts saving Data Channel/Fundline data.

Syntax

STARTDC(***destination***, *filename*,*flag*)

destination The destination of the data. Options are:

		CSVFILE	A .CSV file with comma separated values
		CLIPBOARD	No filename is required
		LISTFILE	A data or list file, to be used later with an INPUT statement. You can specify this format only when running program 900A.
	filename		A text string which defines the name of the file where the data is to be stored. You must give a filename if the destination is a CSVFILE or LISTFILE.
	flag	OVERWRITE	(Default) The specified file, if it already exists, is automatically overwritten
		APPEND	Add new data to an existing file

Example

```
STARTDC (LISTFILE, "FTSE.LST")
SEND ("900A FTSE,MNEM")
ENDDC
```

- In this example, the macro opens a list file called "FTSE.LST". The mnemonics of the companies in the FTSE are downloaded into the file. **ENDDC** closes the Data Channel.

Related commands **ENDDC**

ENDDC

Function Ends saving Data Channel/Fundline data.

Example See **STARTDC**

Related commands **STARTDC**

ConstTimeSeries

Function When executed after starting Data Channel translation, this command allows a constant value to be inserted in the next row/column.

Syntax **ConstTimeSeries** *("value", "start date", "end date", "frequency")*

Value		Any valid string, including spaces and separators. If not specified, defaults to a blank column/row.

Start date/End date/ frequency

If not specified, default to the the values used in the last successful 900B request following STARTDC. This is the preferred method of using ConstTimeSeries as it ensures a correct match with the previous number of downloaded Data Channel items.
If there is no previous 900B request the defaults are: "-1D", "" ,"D",

Example **ConstTimeSeries** *("MyLabel", "-1Y", "-6M", "W")*

See **EX_TIMES.MAC** for an example macro.

AllowDuplicateTimeSeries

Function The AllowDuplicateTimeSeries macro command is used to control the merging of 900B requests using the same datatype (e.g. ICI(P)).

AllowDuplicateTimeSeries takes one parameter, either True or False. When set to True, these time series will NOT be merged into a single column. By default, 900B requests using the same datatype are merged (AllowDuplicateTimeSeries set to False) in order to maintain backwards compatibility with earlier versions of DSWindows, EXCEPT where the start and end dates are also the same - it is assumed that two exactly identical requests are intentional and they are therefore both displayed.

Use of this macro command is illustrated in the example, EX_TIMES.MAC.

Example **AllowDuplicateTimeSeries (True)**

Capturing text

CAPTURE

Function

Saves text output to disk, either in a format for viewing within DSWindows, or in ASCII text format for importing into other packages.

Syntax

CAPTURE*(filename,filetype,page,flag)*

filename

A text string which defines the name of the file where the data is to be stored. If you are saving data to a drive/directory other than the default save file directory, then you must include the full path.

filetype

Numeric, 0 or 1

0 = a file which you want to review within DSWindows. The default extension is .DST

1 = an ASCII text file which you want to use in another package. The default extension is .TXT

If none is specified, 0 is the default.

page

Numeric, 0,1 or 2.

0 = Saves only output pages for a plain ASCII text file. Saves both input and output pages for a DSWindows save file

1 = Saves only output pages

2 = Saves both input and output pages

flag

QUERY, OVERWRITE or APPEND

QUERY = you are prompted, if the specified *filename* exists, whether to overwrite or append

OVERWRITE	the specified file, if it already exists, is automatically overwritten
APPEND	the saved pages are automatically appended to the file, if it already exists.

Example 1

```
CAPTURE("A:\testfile",1,1,overwrite)
> 301A BT,ICI
ENDCAPTURE
```

- A plain text file called a:\testfile.txt is created, and the output pages from the request for recent values and ranges for BT and ICI are saved into it. Any file of the same name will be overwritten. **ENDCAPTURE** closes the file.

Example 2 **CAPTURE**

- With no parameters, the macro stops, and you are prompted to supply a filename (an ASCII text file with a .TXT extension). The macro then continues.

Remarks If you want to capture Data Channel output please see **STARTDC** and **ENDDC**. If you want to capture Graphics output please see **AUTOSAVE/ENDAUTOSAVE** and **SAVEGRAPHICS**.

Related commands **ENDCAPTURE, PRINTSAVEFILE, ACTIVATESAVEFILES, OPENSAVEFILE**

ENDCAPTURE

Function Stops saving data.

Example See CAPTURE.

Related commands **CAPTURE**

OPENSAVEFILE

Function Opens the specified save file for viewing in the Save File window.

Syntax **OPENSAVEFILE *(filename)***

filename	The name of the file to be opened.

Example **ACTIVATESAVEFILES**
OPENSAVEFILE *("file10.dst")*

- The Save File window is opened and a file named "file10.dst" is loaded.

Related commands **CAPTURE**

Arranging windows

Activating windows

The following commands activate the windows. If the window has not been opened, then it is opened.

ACTIVATETERMINAL
ACTIVATEGRAPHICS
ACTIVATEBACKPAGES
ACTIVATESAVEFILES

Closing windows

The following commands close the windows and leave DSWindows.

CLOSEDSWINDOWS
CLOSETERMINAL
CLOSEGRAPHICS
CLOSEBACKPAGES
CLOSESAVEFILES

Minimizing windows

The following commands minimize the windows.

MINIMIZEDSWINDOWS
MINIMIZETERMINAL
MINIMIZEGRAPHICS
MINIMIZEBACKPAGES
MINIMIZESAVEFILES

Maximizing windows

The following commands maximize the windows.

MAXIMIZEDSWINDOWS
MAXIMIZETERMINAL
MAXIMIZEGRAPHICS
MAXIMIZEBACKPAGES
MAXIMIZESAVEFILES

Restoring windows

The following commands restore the windows to their normal size.

RESTOREDSWINDOWS
RESTORETERMINAL
RESTOREGRAPHICS
RESTOREBACKPAGES
RESTORESAVEFILES

TILE

Function — Tiles all child windows.

Related command — **CASCADE**

CASCADE

Function — Cascades all child windows.

Related command — **TILE**

ARRANGEICONS

Function — Arranges all child window icons at the foot of the main window.

Connecting to Datastream

CONNECT

Function — Connects to Datastream.

Syntax — **CONNECT** *("Sessionx")*

CONNECT *("Gatewayname")*

CONNECT *("Gateway,Queue")*

Sessionx	Use *Sessionx*, where *x* is a number in the range 1 to 8, to specify a previously configured session. It must be enclosed in quotes.
Gateway	The name of the gateway. It must be enclosed in quotes.
Gateway,Queue	These parameters are specifically to support connection via DSGATE 3.0, where *Gateway* is the name of the gateway and *Queue* is the name of the queue which the connecting workstation will use. The parameters must be enclosed in quotes.

Example — **CONNECT** *("DATASTREAM, ABC1111")*

- The PC connects to Datastream via the queue named ABC1111 on the gateway named DATASTREAM, where the gateway PC is running DSGATE3.0.

NOTE *If connecting to a gateway via a modem, use the* ***CONNECTNOWAIT*** *command*

Related commands — **DISCONNECT, CONNECTNOWAIT, CONNECTNOQUEUE**

CONNECTNOWAIT

Function — Special form of the **CONNECT** command for use when connecting to a modem.

Syntax — **CONNECTNOWAIT** *("Sessionx")*

CONNECTNOWAIT *("Gatewayname")*

CONNECTNOWAIT *("Gateway,Queue")*

Sessionx	Use *Sessionx*, where *x* is a number in the range 1 to 8, to specify a previously configured session. It must be enclosed in quotes.
Gateway	The name of the gateway. It must be enclosed in quotes.
Gateway,Queue	These parameters are specifically to support connection via DSGATE 3.0, where *Gateway* is the name of the gateway and *Queue* is the name of the queue which the connecting workstation will use. The parameters must be enclosed in quotes.

Example — **CONNECTNOWAIT** *("DS-GATE-1")*

Related commands — **CONNECT, DISCONNECT, CONNECTNOQUEUE**

CONNECTNOQUEUE

Function — This is a special form of the **CONNECT** command. It connects to Datastream, via the specified gateway, but will not wait in a queue. It allows the program to seek another gateway if the first one is busy. If no gateway is specified, then the one specified in the Configure Communication dialog is the default.

Syntax — **CONNECTNOQUEUE** *("Sessionx")*

CONNECTNOQUEUE *("Gatewayname")*

CONNECTNOQUEUE *("Gateway,Queue")*

Sessionx	Use *Sessionx*, where *x* is a number in the range 1 to 8, to specify a previously configured session. It must be enclosed in quotes.
Gateway	The name of the gateway. It must be enclosed in quotes.
Gateway,Queue	These parameters are specifically to support connection via DSGATE 3.0, where *Gateway* is the name of the gateway and *Queue* is the name of the queue which the connecting workstation will use. The parameters must be enclosed in quotes.

Example

```
CONNECTNOQUEUE ("DS-GATE-1")
IF &connectState = QUEUEING THEN
        DISCONNECT
        CONNECT ("DS-GATE-2")
ENDIF
```

- **CONNECTNOQUEUE** tries to connect via DS-GATE-1, but it will not wait if there is a queue.
- **IF** checks if there is a queue.
- If there is a queue, the macro disconnects.
- **CONNECT** tries to connect via another gateway (DS-GATE-2), queueing if necessary.

Related commands **CONNECT, DISCONNECT, CONNECTNOWAIT**

LOGON

Function Runs the default logon macro.

Example

```
ActivateBackpages
ActivateTerminal
MaximizeTerminal
LOGON
```

- The first two lines open the Backpages and Terminal windows.
- The third line maximizes the Terminal window.

- **LOGON** calls the default logon macro. This is LOGON.MAC, unless you have specified another one (by selecting **Options>Configure>Macros>Select logon macro**).

DISCONNECT

Function	Disconnects from Datastream.
Remarks	You can still use DSWindows, but you cannot request data.
Example	See **CONNECTNOQUEUE**.
Related commands	**CONNECT, CONNECTNOWAIT, CONNECTNOQUEUE, CLOSEDSWINDOWS**

Error recovery, logging errors, writing to file

RECOVERUSING

Function

Used with LOGERRORSTOFILE, this command enables you to specify how a macro will recover from an error state such as a communications problem. The command specifies the name of a recovery macro which will be triggered by an error condition (a failed SEND command). The recovery macro will typically attempt to perform a reconnect, return the user to the Datastream prompt and the calling macro will restart at the beginning of the line in which the original error occurred.

Syntax

RECOVERUSING *("filename")*

filename Name and path of the recover macro. If no path is specified, it defaults to the \dswindow\files\ directory.

RECOVERUSING *("recover.mac")*

Example

See the example macro, **EX_900CO.MAC**

Related commands

RECOVERSTOP, LOGERRORSTOFILE, ENDALLMACROS

NOTES

1. *RECOVERUSING only works when LOGERRORSTOFILE is active.*
2. *An example recover macro,* `RECOVER.MAC`*, is included on the installation disks and can be found in the \dswindow\files directory. This is a fully working macro and has been created as a template which you can edit to suit your own requirements.*
3. *When the recovery macro successfully restarts the calling macro, processing of the macro will recommence at the start of the line in which the original error occurred: users should be aware that, depending on how their original macro was structured, this can have an impact upon the running of the macro.*
4. *For detailed information and advice on writing robust macros and on creating a recovery macro, please refer to the Chapter, 'How to make your macros more robust'.*
5. *If no file name extension is specified it defaults to *.mac.*

RECOVERSTOP

Function — Use the RECOVERSTOP command to switch recovery off.

Syntax — **RECOVERSTOP**

Related commands — ENDALLMACROS, RECOVERUSING

ENDALLMACROS

Function — Kills all running macros. Use this command, for example, to stop both a recover macro and the calling macro.

Syntax — **ENDALLMACROS**

Related commands — RECOVERSTOP

LOGERRORSTOFILE

Function — Sends error messages to a log file rather than the screen. This ensures that normal dialogs are not displayed and macro execution is not interrupted when an error occurs. Used with RecoverUsing, this command prevents the situation in which a macro sits waiting for a user to click on OK before the processing of the macro can continue.

Syntax — **LOGERRORSTOFILE** *(filename,flag)*

filename	Name and path of the log file. If no path is specified it defaults to the configured save file directory with a .LOG file extension.	
flag	APPEND	Add new data to an existing file
	OVERWRITE	The specified file, if it already exists, is automatically overwritten.

NOTES

1. *To start logging errors to a file, use LOGERRORSTOFILE with the file name as a parameter*
   To switch the log process off and revert to normal display mode, use the command without the parameter.
2. *Unless you switch LOGERRORSTOFILE off, the log process stays active for the duration of the complete macro.*
3. *If LOGERRORSTOFILE is not switched off, any MESSAGE command, for example will also be sent to the log file - care must therefore be taken in how the command is used, particularly if the macro is being used in interactive mode (ie not unattended).*

WRITETOFILE

Function — Writes text to a specified file

Syntax — **WRITETOFILE** ***(text,filename,****flag)*

text	text string to be written	
filename	name of the file to write text to	
flag	APPEND	Add new data to an existing file
	OVERWRITE	The specified file, if it already exists, is automatically overwritten

Example — **WRITETOFILE** ("Hello" + CHR$(13) + CHR$(10), "c:\dswindow\files\test.txt")

- This example writes the text string "Hello", together with a line feed character, to the file test.txt in the specified path.

ONERROR

Function — Defines the error status for the macro, when an error is found in a SEND or CONNECT command.

Syntax: **ONERROR** *(status)*

status — The error status. It can be 0 or 1:

0 = If an error has been found, the macro will stop running. This is the default.
1 = If an error has been found, the macro will continue running. The user should be aware that errors may have occurred during processing which may affect the result.

Example — **ONERROR (0)**

Miscellaneous

MESSAGE

Function — Displays a message box with the specified message and title. Both message and title are strings.

Syntax — **MESSAGE** *(message, title)*

message — A text string - the message to be displayed.

title — A text string - the title for the message box.

Example — **MESSAGE** ("Requested text not found","Text search")

- The macro is stopped until the message is acknowledged (by clicking on **OK**).

Related commands — **USERINPUT**

BEEP

Function — Generates a beep at the PC's speaker.

Syntax — **BEEP** *(beeps)*

beeps — A number specifying the number of beeps to generate. If no number is specified, the default is one.

Example — **BEEP*(3)***

- Three beeps are generated.

Related commands — **MESSAGE, WAIT**

SetDateExportFormat

Function — This command is used to set the DSWindows Export Date format. The date format should be the same as the date format strings listed in the short date styles options in the Windows Control Panel.

Syntax/Example — **SetDateExportFormat ("DD/MM/YY")**

Please refer to the EX_DATEF.MAC example macro to see the **SetDateExportFormat** command in use.

Constants and variables

This section includes:

- an introduction to the concepts of constants and variables in the Datastream Macro language
- a list of the available system constants and variables, instructions on how to use them and some examples

Introduction

System constants and variables are normally used in macros to test for a number of conditions. Constants are also used in certain commands as parameters to define output destinations and window sizes.

Experienced macro users can define their own constants and variables.

Constants

Constants are text strings, numbers or dates whose values are fixed in the macro and do not change. System constants are constants given meaningful names for commonly used values in Datastream macros. Note that you can use system constants only with certain instructions, commands and variables.

The available system constants are listed below with notes on when to use them together with examples.

TRUE

Usage — Use with the system variables &ENDOFDATA and &TEXTFOUND.

Example

```
IF &ENDOFDATA = TRUE THEN
        GOTO FINISH
ENDIF
FINISH:
```

FALSE

Usage — Use with the system variables &ENDOFDATA and &TEXTFOUND.

Example

```
LOOP:
IF &ENDOFDATA = FALSE THEN
        INPUT NEXTVALUE
        SEND (NEXTVALUE)
        GOTO LOOP
ENDIF
```

CONNECTED

Usage — Use with the system variable &CONNECTSTATE.

Example

```
CONNECT ("DSGATE-1")
IF &CONNECTSTATE <>CONNECTED THEN
        CONNECT
ENDIF
```

NOT_CONNECTED

Usage — Use with the system variable &CONNECTSTATE.

Example

```
CONNECT ("DSGATE-1")
IF &CONNECTSTATE = NOT_CONNECTED THEN
        CONNECT
ENDIF
```

QUEUEING

Usage — Use with the system variable &CONNECTSTATE.

Example

```
CONNECTNOQUEUE ("DSGATE-1")
IF &CONNECTSTATE = QUEUEING THEN
        DISCONNECT
        CONNECT("DSGATE-2")
ENDIF
```

UNLOCK

Usage — Use with the system variable &SENDCOMPLETE.

Example

```
SEND ("A212301202",TIMEOUT:30)
IF &SENDCOMPLETE<>UNLOCK THEN
        MESSAGE ("DATASTREAM did not send logon screen","Connect Error")
ENDIF
```

TEXTFOUND

Usage — Use with the system variable &SENDCOMPLETE.

Example

```
SEND ("A212301202",WAITFOR:"ADD",TIMEOUT:10)
IF &SENDCOMPLETE <> TEXTFOUND THEN
        MESSAGE ("The PAD did not send the 'ADD' prompt","Connect Error (5)")
ENDIF
```

TIMEOUT

Usage — Use with the system variable &SENDCOMPLETE.

Example

```
SEND ("A212301202",TIMEOUT:10)
IF &SENDCOMPLETE = TIMEOUT THEN
        MESSAGE ("We Timed Out","TIMEOUT")
ENDIF
```

QUERY

Usage — Use as a variable in the **CAPTURE** command, to display a prompt requesting whether you want to overwrite an existing save file.

Example

```
CAPTURE ("atest.txt",1,0,QUERY)
> [CLEAR]
> 99Z
ENDCAPTURE
```

OVERWRITE

Usage — Use as a variable, (e.g.with the **CAPTURE** command), to overwrite an existing save file.

Example

```
CAPTURE ("atest.txt",1,0,OVERWRITE)
> [CLEAR]
> 99Z
ENDCAPTURE
```

APPEND

Usage — Use as a variable, (e.g with the **CAPTURE** command), to append the new data to an existing file.

Example

```
CAPTURE ("atest.txt",1,0,APPEND)
> [CLEAR]
> 99Z
ENDCAPTURE
```

CSVFILE

Usage — Use as a variable in the **STARTDC** command, to save Data Channel data to a .CSV file.

Example

```
STARTDC (CSVFILE,"DEMO.CSV")
```

CLIPBOARD

Usage — Use as a variable in the **STARTDC** command, to save Data Channel data to the Clipboard.

Example — **STARTDC** (CLIPBOARD)

LISTFILE

Usage — Use as a variable in the **STARTDC** command, to save Data Channel data to a list file for later use.

Example — **STARTDC** (LISTFILE, "DEMO.LST")

SHOW_MIN

Usage — Use as a variable in the **STARTPROGRAM** command, to display the program window as minimized.

Example — **STARTPROGRAM** ("NOTEPAD.EXE",SHOW_MIN)

SHOW_MAX

Usage — Use as a variable in the **STARTPROGRAM** command, to display the program window as maximized.

Example — **STARTPROGRAM** ("NOTEPAD.EXE",SHOW_MAX)

SHOW_NORMAL

Usage — Use as a variable in the **STARTPROGRAM** command, to display the program window as normal sized.

Example — **STARTPROGRAM** ("NOTEPAD.EXE",SHOW_NORMAL)

Variables

Variables are named fields and act as placeholders for values to be determined during the operation of the macro. You must assign the value of a variable using a **SET** or **INPUT** instruction. The value of a variable can change as the macro is executed. For example, in the statement:

```
SET a TO ( b + c )
```

a is a variable, its value determined by the result of the expression "b + c".

Variable names can start with a letter or an underscore (_) and continue with letters, digits or underscores. They can be in upper or lower case, and there is no limit to their length.

Local and global variables

NOTE *The following applies where variables are used in 'child macros' (i.e. macros which are called and activated from within another macro).*

By default, variables are local; in other words, they only apply in the macro in which they are set. However, you can set a variable as global so that it applies not only in the macro in which it is set, but also in any other macro which that macro references using the **CALL** command.

- To set a global variable, use the **SETGLOBAL...TO...** command. For example,

  ```
  SETGLOBAL x TO 1
  ```

  When the variable x is subsequently used in the macro in which it is set, or in any **CALL**ed macro, it refers to the global variable x. If a variable has already been used before a **SETGLOBAL** command has set it, then this will be reported as an error.

- To prevent a variable being assigned as local and then changed to global by another macro:

  ```
  SET x TO 1
  SETGLOBAL x TO 10
  ```

- To use a global variable in an **INPUT** command in a child macro, the **SETGLOBAL** command must precede the **INPUT** command; for example,

  SETGLOBAL name **TO** ""
  INPUT name

  In this example, the variable name is set first to an arbitrary value ("" in this case) as it will be overwritten by the **INPUT** statement.

System variables

System variables are variables which are reserved for specific purposes. Their values are read-only (that is, the user cannot assign them), but they may be referenced from within a macro and the values may change according to the state of DSWindows.

A list of the system variables is given below.

&ENDOFDATA

Usage

Use with the **INPUT** instruction. It has one of two values: TRUE and FALSE. The value is set to TRUE when an **INPUT** instruction reaches the end of a data list.

Example

```
OPENDATA arglist
LOOP:
IF &ENDOFDATA = FALSE THEN
        INPUT arg
        SEND ("101B "+arg)
        SEND [CLEAR]
        GOTO LOOP
ENDIF
```

&SCREEN

Usage

Represents the screen as a large text string. Use this variable to check if a sub-string is embedded in it.

Example

```
IF MID$(&SCREEN, 1151, 12) = "PLEASE LOGON" THEN
        SEND ("DS")
ENDIF
```

- In this example, 1151 = 80*(row-1) + column where row contains the search-text, and column is the column where the text starts. 12 is the length of the string: "PLEASE LOGON".

&CONNECTSTATE

Usage

Set by the **CONNECT** and **CONNECTNOQUEUE** commands. &CONNECTSTATE has three possible values: CONNECTED, NOT_CONNECTED and QUEUEING. QUEUEING applies only with **CONNECTNOQUEUE**.

Example

```
IF &CONNECTSTATE = NOT_CONNECTED THEN
        CONNECT
ENDIF
```

&SENDCOMPLETE

Usage

Set by the **SEND** command. &SENDCOMPLETE has three possible values: TIMEOUT, TEXTFOUND and UNLOCK.

Example

```
SEND ("A212301202",TIMEOUT:10)
IF &SENDCOMPLETE = TIMEOUT THEN
        MESSAGE ("We Timed Out","TIMEOUT")
ENDIF
```

&TEXTFOUND

Usage

Set by the **SENDANDCHECK** command.

Example:

```
SENDANDCHECK ("99Z", "FRANCE")
IF &TEXTFOUND THEN
        MESSAGE ("FRANCE was mentioned", "TEXTFOUND")
ENDIF
```

- In this example the text "France" is sought in the output from the news program 99Z. If it is found, a message is displayed.

&DAYOFWEEK

Usage — &DAYOFWEEK can be used in any expression and has seven possible values (0 - 6), representing the values of the days of the week (Sunday to Saturday).

Example

```
IF &DAYOFWEEK = 2 THEN
        MESSAGE ("It is Tuesday","Day Of Week")
ENDIF
```

&DAYOFMONTH

Usage — &DAYOFMONTH can be used in any expression and has 31 possible values (1 - 31), representing the values of the days of the month.

Example

```
IF &DAYOFMONTH = 1 THEN
        MESSAGE ("It is the first of the month")
ENDIF
```

&MONTH

Usage — &MONTH can be used with any expression and has 12 possible values (1 - 12), representing the values of the months of the year (January to December).

Example

```
IF &MONTH = 3 THEN
        MESSAGE ("2nd quarter starts next month")
ENDIF
```

&YEAR

Usage — &YEAR can be used with any expression and is a 2-character number string, representing a year.

Example

```
SET today TO &DAYOFMONTH
SET this_month TO &MONTH
SET this_year TO &YEAR
SAVEGRAPHICS ("GR"+ STR$(today)+STR$(this_month) +
STR$(this_year), OVERWRITE)
```

- In this example a graph is saved to the file GRddmmyy.dsg where ddmmyy is today's date.

&RESULT

Usage

Set by the **STARTPROGRAM** command. It has a number of possible values; please refer to the section on **STARTPROGRAM** for the values and their meanings.

Example

```
STARTPROGRAM ("TEST.EXE",SHOW_NORMAL)
IF &RESULT = 2 OR &RESULT = 3 THEN
        MESSAGE ("Program Not Found","TEST.EXE")
ENDIF
```

&RECOVERYATTEMPTS

Usage

Used within a recovery macro, this variable generates an incremental count of the subsequent number of times a SEND or UPDATELOCALCODE command failed and generated a recovery attempt. (Failures in the recovery process itself are not included in the count.) This enables you to control the number of times a recovery attempt is made and also to vary the way in which the macro tries to recover, for example, by trying to connect to a different gateway on the third recovery attempt.

Example

```
Set MaxRecoveryTries To 3;

If (&RecoveryAttempts > MaxRecoveryTries) Then
  Set Msg To "Max (" + Str$(MaxRecoveryTries) + ") recovery attempts
    exceeded."
  Message (Msg, TraceCaption)
EndIf
```

&OS

Usage

Use to tell you what operating system DSWindows was built for. It can be one of: WINDOWS, HPUX, SOLARIS, SOLARIS2. &OS might be useful for writing a library macro intended for use on multiple plaftorms.

Example

```
If &OS = WINDOWS
        Set EOL To chr$(13) = chr$(10)
; carriage return and line feed
Else
        Set EOL To chr$(10)
EndIf
```

&ATPROMPT

Usage

Use this variable to test whether or not you are currently at the Datastream prompt (Program Finder). It is set to TRUE if you are and FALSE if not. Typical usage would be to test whether a recovery process has been successful in returning you to the prompt (see example below).

Example

```
IF (&ATPROMPT = FALSE) THEN
   WRITETOFILE ("recover.log", "Recovery failed")
ELSE
   WRITETOFILE ("recover.log", "Recovery succeeded")
```

Functions and expressions

This section explains the purpose of functions and expressions in Datastream macros and the rules governing their use. Lists of the available functions and the mathematical and logical operators for use in expressions are given.

Functions

You can use functions to:

- manipulate strings
- convert strings into integers and vice-versa.

Manipulating strings

Use one of the following functions to extract a string from another string:

- left$(str, i)
- right$(str, i)
- mid$(str, i,,j)
- len(str)
- Instr (lookIn, lookFor)

These functions enable you to define the portion of the string which you want to extract. The following section explains the use and syntax of each of the functions. Examples showing how to use them are given at the end of the ‘Manipulating strings’ section.

left$

Usage — Extracts a number of characters from a string, starting from the left of the string.

Syntax — **left$ (str,i)**

str	The name of the original string
i	The number of characters in the new string

right$

Usage — Extracts a number of characters from a string, starting from the right of the string.

Syntax — **right$** ***(str,i)***

str	The name of the original string
i	The number of characters in the new string

mid$

Usage — Extracts a number of characters from a string, starting from the position you specify.

Syntax — **mid$** ***(str,i,j)***

str	The name of the original string
i	The character position, from the left, with which the new string is to begin
j	The number of characters in the new string.

len

Usage — Returns the length of a string.

Syntax — **len** ***(str)***

str	The name of the string

Instr

Usage

Searches for a string within another string. The result of the expression is either the position of the lookFor string within the lookIn string, or 0 if lookIn does not contain lookFor.

Syntax

Instr (lookIn, lookFor)

lookIn a string to search

lookFor a string to look for in the string lookIn

Examples

Example 1

This set of commands illustrates how to use the left$, right$ and mid$ functions to extract characters from a string.

```
SET test TO "Now run List File demo"
SET tmp1 TO left$( test,8 )
SET tmp2 TO right$( test,7 )
SET tmp3 TO mid$( test,5,3 )
```

- The value of the variable test is set to the string "Now run List File demo"
- The value of the variable tmp1 is set to the string "Now run " - the first 8 characters in the variable test.
- The value of the variable tmp2 is set to the string "le demo" - the last 7 characters in the variable test.
- The value of the variable tmp3 is set to the string "run" - three characters, starting with the fifth, in the variable test.

Example 2 This macro illustrates how to use the len(str) function, and how to convert integers into strings.

```
SET test TO "Now run List File demo"
SET tmp4 TO len( test )
MESSAGE ("The length of the file 'test' = +str$(tmp4))
```

- The value of the variable test is set to the string "Now run List File demo"
- The value of the variable tmp4 is set to the integer 22 using the len(str) function because the variable test contains a 22 characters string. Note that the len(str) function has returned an integer value to the tmp4 variable.
- To display the value of the variable tmp4 (ie the integer 22), the **MESSAGE** command is used to generate output. Because the **MESSAGE** command parameters must be strings and the value of tmp4 is currently an integer value, the **str$** function is used to convert the value of tmp4 to a string value. Note that the str$ function is explained in detail in the following section, 'Converting strings'.

Example 3 Set position To **InStr** (&screen, Hong Kong)

Converting strings

The conversion functions enable you to convert strings to integers and vice-versa, and to convert characters into ASCII numbers and vice-versa.

The coversion functions are:

- **str$(i)**
- **chr$(i)**
- **val(str)**
- **asc(str)**

NOTE *The result of functions with a* $ (str$, chr$, mid$, left$, right$*) is always a string, and the result of functions without a* $ (val, asc, len) *is always an integer.*

str$

Usage

Converts an integer value to a string. For example, if a variable has an integer value of 99, to display the value of the variable (using the **MESSAGE** command) you must first convert it to a string using the **str$** function. See the example at the end of this section. To perform the reverse of this function, use **val(i)** (see below).

Syntax

str$ ***(i)***

i An integer, such as 99, or a variable with an integer value.

chr$

Usage

Converts an integer value to the ASCII character represented by the integer. For example, **SET** X TO **CHR$**(65) sets the value of the variable X to the letter "A" where the ASCII code for an "A" = 65. To perform the reverse of this function, use **asc$** ***(i)*** (see below).

Syntax

chr$ ***(i)***

i An integer, such as 65, or a variable with an integer value

NOTE *Useful examples of the* chr$ *function are:*
chr$(10) *line feed character*
chr$(13) *carriage return character*
chr$(34) *the double quote (") character*

val

Usage

Converts a character string to its integer value. For example, to convert the string "99" to an integer value, you could use the command **SET** X TO **VAL**("99")**.** To perform the reverse of this function, use **str$(i)** (see above).

Syntax

val ***(i)***

i A string, such as "99", or a variable with a string value.

NOTES

1. val(i) *only recognises numeric characters in a string - it ignores everything else*
2. val(i) *only recognises integers (whole numbers). Numbers containing decimal points are not understood and anything after a decimal point is ignored.*

3. The Datastream Macro language recognises integers in the range -32767 to 32767.

asc

Usage

Converts the first character in a string to its ASCII value. For example, the command **SET** X **TO asc** ("ABC") would set the value of the variable X to the ASCII value of the character "A" (e.g. the integer 65).

Syntax

asc ***(i)***

i The characters in the string

Example 1

```
SET tmp1 TO 65
SET tmp2 TO str$( tmp1 )
SET tmp2 TO chr$( tmp1 )
```

- The value of the variable tmp1 is set to the integer 65.
- The value of the variable tmp2 is set to the string "65".
- The value of tmp2 is set to the string "A" where A is the character whose ASCII value is 65 (the value of the variable tmp1)

Example 2

```
SET tmp1 TO "66"
SET tmp2 TO val( tmp1 )
SET tmp2 TO asc( tmp1 )
```

- The value of the variable tmp1 is set to the string "66".
- The value of the variable tmp2 is set to the integer 66.
- The value of the variable tmp2 is set to the ASCII value (an integer) of the first 6 in the string "66", which is 54.

Example 3

```
SET X TO "ABC"
SET Y TO asc(X)
MESSAGE ("The value of Y = " +str$(Y))
```

- The value of the variable X is set to the string "ABC".
- The value of the variable Y is set to the ASCII value (an integer) of the first character in the variable X (the letter "A").
- The **MESSAGE** command displays the value of the variable Y after converting it to a string value using the **str$** function ("The value of Y = 65").

Expressions

You can use mathematical and logical operators with constants and variables to form expressions in macros. The following are valid mathematical and logical operators:

SYMBOL	MEANING	SYMBOL	MEANING
+	Add	**>**	Greater than
–	Subtract	**>=**	Greater than or equal to
*****	Multiply	<>	Not equal to
/	Divide	AND	Logical "and"
%	Remainder	OR	Logical "or"
<	Less than	NOT	Logical "not"
<=	Less than or equal to	**(**	Open parenthesis
=	Equal to	**)**	Close parenthesis

NOTES

1. *In the* **SET...TO...** *command you can use only* **+**, **-**, ***,/** *and* **%** *in the* **SET...TO..** *command.*
2. *When dealing with strings, the comparison operators you can use are =, <>, < and >.*
3. *Use + to link two strings together (for example, a + b where a = "Good" and b = " Morning" would result in "Good Morning"). Note that when you concatenate two strings you must put in any spaces that are required; the operator + does not do this automatically.*
4. *You can combine comparison operators and logical operators to form complex expressions, for example:*

   IF v1 < v2 **AND** v2 < (v3 + v4)

5. *Because the macro language does not recognise fractions or decimal points, when you use the division symbol as an operator, you can show the remainder of the calculation using the % symbol.*

How to make your macros more robust

Introduction

The DSWindows macro language has been extended in version 2.2 to enable you to create macros with inbuilt error-handling and recovery capabilities. You can now build macros, for example, that detect errors (such as communications problems caused by modem or gateway faults), pass control to a recovery macro which attempts to solve or route around the problem and which then hands control back to the calling macro to continue its processing. A typical scenario in which this technique will be most useful is a macro which is run unattended, such as an overnight download, and which fails for a relatively trivial and predictable reason. And, because there is nobody available to correct the fault and restart the macro, it simply fails and leaves you to discover it in the morning.

This chapter is intended to help you to avoid this type of situation and to introduce a number of more general techniques to make your macros more robust.

- **Techniques for making your macros generally more robust**

  'Robust', in the context of this chapter, is used to mean macros which are, for example, self-reporting, require no human intervention, can detect a range of error states and recover from them, and which don't attempt to make a Datastream connection where one already exists.

- **A complete recovery macro (`RECOVER.MAC`)**

  The objective of this recovery macro is to take over when things are going wrong, to try all means to get back to the Datastream prompt (the Program Finder) and to return control to the original macro at the point where the problems arose, allowing it to seamlessly continue. The recovery macro, `(RECOVER.MAC)` is included on the installation disk and can be found in your **\dswindow\files** directory; it is intended to provide a template which users can customise according to their own specific requirements.

It is assumed that the reader has a good basic knowledge of DSWindows and the DSWindows macro language.

Techniques for making your macros more robust

This section provides general tips which represent good practice in building robust macros and ensuring that faults and errors can be effectively traced. All of the techniques discussed can be found in RECOVER.MAC included in the second part of this chapter and many of the examples given are also taken from this macro. The section is divided into the following elements:

- &CONNECTSTATE
- LOGERRORSTOFILE
- Writing trace information
- &ATPROMPT
- &ATOUTPUT
- &RECOVERYATTEMPTS
- Adding a waiting period
- Ending recovery using RECOVERSTOP and ENDALLMACROS
- Calling a recovery macro (RECOVERUSING)
- Potential problems with the SEND command
- Failures in the recovery macro

NOTE *All commands and variables discussed here are also described in the relevant section of the manual.*

&CONNECTSTATE

The &CONNECTSTATE variable is available at all times and is described in the "Constants and Variables" section of this manual.

To protect your macro against the possibility that it experiences connection problems, you can use the &CONNECTSTATE variable to try an alternative communication mechanism or to give up the attempt to connect:

```
CONNECT()
IF (&CONNECTSTATE = NOT_CONNECTED) THEN
   CONNECT ("MyOtherGateway")
ENDIF
IF (&CONNECTSTATE = NOT_CONNECTED) THEN
  END
ENDIF
SEND ("DS")
SEND ("ABCD123password")
```

LOGERRORSTOFILE

One of the problems with running macros unattended (e.g. overnight) is that if any kind of message is displayed during its execution it requires user intervention (e.g. someone to click on the OK button) to clear it before the macro can continue. DSWindows 2.2 has a new macro command that allows you to divert everything that would have appeared as a message on screen to a file instead.

This new command is called LOGERRORSTOFILE. For example, the follow macro:

```
DISCONNECT()
SEND ("99Z")
```

would result in the error message, ("`Send Macro Failed: Please connect to Datastream first`"), being displayed on the screen in a message box with a Macro Error title caption).

However, if we divert the messages using LOGERRORSTOFILE,

```
LOGERRORSTOFILE ("recover.log")
DISCONNECT()
SEND ("99Z")
```

nothing at all appears on screen. Instead, the message appears in the log file :

```
 *** Message on Mon 10 Mar 1997 at 16:52:33 ***
-----------------------------------------------------
Macro Error
-----------------------------------------------------
Send Macro Failed: Please connect to Datastream first
-----------------------------------------------------
```

Message on Tue 24 Sep 1996 at 16:41:40
Macro Error
Send Macro Failed: Please connect to Datastream first

The following macro will not work in unattended mode if the first CONNECT() fails unless messages have been diverted to a file instead of the screen.

```
CONNECT()
;if above connect fails, then a message will appear now
IF (&CONNECTSTATE = NOT_CONNECTED) THEN
   CONNECT ("MyOtherGateway")
ENDIF
```

Under normal operation, this would generate a message that the user would have to manually clear before trying the second gateway. However, if error messages were being diverted to a file, the macro would silently carry on after the first failure and try the second gateway. We recommend then that, if a macro is intended to run in unattended mode, you should use the LOGERRORSTOFILE command.

```
LOGERRORSTOFILE ("recover.log")
IF (&CONNECTSTATE = NOT_CONNECTED) THEN
   CONNECT()
ENDIF
;no message appears now, even if above connect fails
IF (&CONNECTSTATE = NOT_CONNECTED) THEN
   CONNECT ("MyOtherGateway")
ENDIF
```

NOTES *1. To switch LOGERRORSTOFILE off, use* **LOGERRORSTOFILE ("")**

2. *If LOGERRORSTOFILE is switched on, then all messages, including those created using the MESSAGE command, are sent to the specified log file. Users running macros in interactive mode should be aware that no messages will be displayed on the screen until the command is switched off.*

3. *The LOGERRORSTOFILE command can take an OVERWRITE or APPEND parameter*

Writing to a log file also enables your macro to handle errors in a more robust way. For example, you can divert output to any file that you choose and even change which file during macro execution :

```
LOGERRROSTOFILE ("pete.txt")
;Errors will go to file pete.txt
LOGERRROSTOFILE ("dave.txt")
;Errors will now go to file dave.txt
LOGERRROSTOFILE ("")
;Errors will appear back on screen again
LOGERRROSTOFILE ("pete.txt")
;Errors will now go to file pete.txt
```

Using this method you could, for example, write to different log files for different macros, or for different sections of the same macro. This can be useful for writing trace information which can subsequently help you to examine the progress of the macro (see next section).

Writing Trace information

One very important aspect of making your macros more robust is having the ability to trace the exact progress of your macro by writing relevant information to a log file at each stage of the macro's processing. Making your macro self-reporting in this way will be of great benefit when things start to go wrong; it will help you to isolate where the problems arose and how much of the macro was successful.

The examples given throughout this chapter will all include suggested usage of this technique.

There are various ways to record useful information. If LOGERRORSTOFILE is switched on, you can take advantage of it to use the MESSAGE command to send, for example, status-type information to the log file:

```
LOGERRROSTOFILE (“pete.txt”)
MESSAGE “Starting macro”
;some work
MESSAGE “Finished part one”
;some more work
MESSAGE “Finished part two”
MESSAGE “Ending”
```

Alternatively, you can use the WRITETOFILE command. For example:

```
IF (&CONNECTSTATE = NOT_CONNECTED) THEN
      WRITETOFILE ("Default gateway is down","c:\dswindow\recover.log",APPEND)
      CONNECT ("MyOtherGateway")
END
```

&ATPROMPT

One of the objectives of the recover macro is to get you back to the Datastream prompt. To enable this, DSWindows 2.2 has a new system variable called &ATPROMPT. This is set to TRUE if we are currently sitting at the Datastream prompt (Program Finder), and FALSE if we are at any other screen. This can be used to good effect in the recovery process :

```
IF (&ATPROMPT = FALSE) THEN
   SEND (“[CLEAR]”)
ENDIF
```

You can also use this variable to see whether or not the recovery process was successful. Remember, the job of the recovery macro it to take us back to the Datastream prompt. So the last few lines of such a macro may read :

```
IF (&ATPROMPT = FALSE) THEN
   WRITETOFILE (“recover.log”, “Recovery failed”)
ELSE
   WRITETOFILE (“recover.log”, “Recovery succeeded”)
ENDIF
```

&AtOutput

The system variable &ATOUTPUT provides another method for adding robustness by testing whether a process has completed. For example, correct processing of the command **SEND 101A ICI** results in requested information being displayed in an output window. However, if the syntax of the command had been incorrect (for example, an invalid code had been used), the result would be that the 101A input screen is displayed with the cursor positioned in the field with the invalid entry.

NOTE *This variable is not used in RECOVER.MAC.*

&RECOVERYATTEMPTS

A new system variable in DSWindows 2.2, called &RECOVERYATTEMPTS, enables you to control the number of times the recovery macro attempts to recover from a failed command, on a 'per command' basis. &RECOVERYATTEMPTS generates an incremental count of the number of times a SEND or UPDATELOCALCODE command has failed and generated a recovery attempt. Note that failures in the recovery process itself are not included in the count.

```
Set MaxRecoveryTries To 3 ; Maximum number of times to try to recover.

If (&RecoveryAttempts > MaxRecoveryTries) Then
  If (ShowTrace) Then
     Set Msg To "Max (" + Str$(MaxRecoveryTries) + ") recovery attempts exceeded."
    Message (Msg, TraceCaption)
  EndIf
EndIf
```

You can also use the &RECOVERYATTEMPTS variable to vary the way in which the macro tries to recover, for example by trying to connect to a different gateway on the second recovery attempt.

```
Set SecondaryConnect To "DATASTREAM-GATE-2"

If (&RecoveryAttempts = 2) Then
  If (ShowTrace) Then
     Set Msg To "Connecting using : " + SecondaryConnect
     Message (Msg, TraceCaption)
  EndIf
  Connect (SecondaryConnect)
EndIf
```

Adding a Waiting period

It is always advisable to incorporate waiting periods at the start of an initial recovery, for example, or before starting subsequent recovery attempts. This is necessary to allow time for external events to change before a connection attempt or after a disconnection. For example, the following macro sets up a 10 second delay before making the first recovery attempt:

```
Set WaitInitialRecovery To 10 ; Secs to wait before recovering first time.
If (&RecoveryAttempts <= 1) Then
   If (ShowTrace) Then
      Set Msg To "Waiting for " + Str$(WaitInitialRecovery) + " seconds."
      Message (Msg, TraceCaption)
      EndIf
Wait (WaitInitialRecovery)
EndIf
```

Notice that in the RECOVER.MAC four waiting periods are defined:

```
Set WaitSendTimeOut To 30 ; Secs to wait before timing out sends.
Set WaitAfterDisconnect To 30 ; Secs to wait after a disconnection.
Set WaitInitialRecovery To 60 ; Secs to wait before recovering first time.
Set WaitSubsequentRecoverys To 120 ; Secs to wait before recovering subsequent
times.
```

Ending recovery using the RECOVERSTOP and ENDALLMACROS commands

The RECOVERSTOP and ENDALLMACROS commands are used to stop a macro in different ways. RECOVERSTOP simply switches off recovery and instructs the macro to make no further attempt to recover from any failed SEND command, without actually stopping the macro itself. ENDALLMACROS is used simply to kill all active macros.

RECOVER.MAC illustrates the use of these commands in the section which checks that the number of recovery attempts has not been exceeded. It uses TERMINATEATEND to enable you to specify how you want the process to react to this situation.

```
*************************************************************************************************
Set MaxRecoveryTries To 3 ; Maximum number of times to try to recover.
Set TerminateAtEnd To TRUE ; Whether to end all macros if recovery ultimatily fails.
*************************************************************************************************

If (&RecoveryAttempts > MaxRecoveryTries) Then
   If (ShowTrace) Then
      Set Msg To "Max (" + Str$(MaxRecoveryTries) + ") recovery attempts
exceeded."
      Message (Msg, TraceCaption)
   EndIf

   If (TerminateAtEnd) Then
      If (ShowTrace) Then
         Message ("Terminating recovery macro.", TraceCaption)
      EndIf
      EndAllMacros ()

   Else
      If (ShowTrace) Then
         Message ("Giving up recovery.", TraceCaption)
      EndIf

      RecoverStop
      Goto FINISH
   EndIf

EndIf
```

Note that, by default, `RECOVER.MAC` sets TERMINATEATEND to **TRUE** so that, in the event that the number of recovery attempts is exceeded, ENDALLMACROS is executed. If you prefer your recovery macro to use RECOVERSTOP, you must set TERMINATEATEND to **FALSE**.

Potential problems with the SEND command

1. Remember that the final job of `RECOVER.MAC` is to return control to the calling macro at the start of the line at which it failed. It is very important that users understand the impact of this on the way in which the calling macro is structured.

   Imagine a macro with SEND commands structured as follows:

   ```
   ....
   SEND 401A
   SEND BT
   ...
   ```

If the macro fails at the first line, perhaps due to a communications problem, RECOVER.MAC will carry out its job and restart the macro with the SEND 401A command being issued at the Datastream prompt. All of which is fine. However, if the macro fails at the second line, RECOVER.MAC will again do its job and restart the macro with the SEND BT command being issued at the Datastream prompt. So it is very important that SEND commands are structured in an atomic format which allow the recovery to start with a complete and correct command:

```
SEND("401A BT")
```

2. If you use the SEND ("ATZ") command in a macro which uses RECOVERUSING to call a recovery macro, ensure that you issue the SEND ("ATZ") *before* recovery is enabled. When ATZ is issued to some modems a line drop occurs (DSR momentarily goes low) which triggers the recovery macro. Line drops are normally considered an error but in this case the user has effectively requested it. Please refer to *RECOVERUSING* for further information.

3. To avoid the possibility that a SEND command never completes and the recovery process is therefore never started, we recommend that a TIMEOUT period is added to the SEND command. This can then be used with the &SENDCOMPLETE variable to test whether the SEND has been executed: for example:

```
SEND ("900B MKS(P),-4M,,D",TIMEOUT:10)
IF &SENDCOMPLETE = TIMEOUT THEN
    IF (SHOWTRACE) THEN
    MESSAGE ("SEND TIMED-OUT.", TRACECAPTION)
    ENDIF
ENDIF
```

Starting recovery by calling a recovery macro

The RECOVERUSING command has been introduced to enable you to start recovery by specifying the name of a recovery macro which will be executed in the event of a failed SEND or UPDATELOCALCODE command. Typically it is anticipated that customers will use the RECOVER.MAC supplied on the installation disks as a template for creating their own recovery macros. We recommend that the RECOVERUSING command is paired with LOGERRORSTOFILE and placed at the top of your macro.

```
LOGERRORSTOFILE ("Pete")
RECOVERUSING ("recover.mac")
```

Failures in the recovery macro

All recovery macros should be written so that any failures within the recovery process itself do not start a new recovery. `RECOVER.MAC`, for example, is written so that errors are simply accepted, appropriate logging information is written to the log file and control is returned to the calling macro.

Template recovery macro, RECOVER.MAC

These notes are intended simply to describe the basic structure and functions of `RECOVER.MAC`. For detailed information on individual commands or variables, please refer to the relevant passage earlier in this chapter, or to the appropriate definition elsewhere in this manual. Note that the Section numbers shown in the following text are purely for the purposes of this documentation - they are not part of the macro itself.

`RECOVER.MAC` is structured as eight sections, each with a specific function.

SECTION 1
Includes explanatory remarks and defines a set of variable values used within the macro. Please note that the variables are all configurable and have been set to typical values for general use. The macro is structured to enable you to change these values according to your own requirements without needing to amend the main body of the macro; for example, you can change any of the WAIT or CONNECT options, switch Tracing off, or change the behaviour of the macro when the maximum number of recovery attempts has been reached.

```
;*******************************************************************************
;* SECTION 1
;* Activate this recovery macro by placing the following two
;* lines at the top of the macro you want to make robust :
;*
;* LogErrorsToFile ("errors.log")
;* RecoverUsing "recover.mac":
;*
;* Feel free to alter any of the variables at the top of this
;* file to suit your specific set-up.
;*
```

```
;* When specifying an alternative connect method you have a
;* number of choices :
;* (1) Use "" to stick to the default comms configuration. (This is
;* used for the first recovery attempt regardless).
;* (2) Use another gateway name. i.e. "DATASTREAM-GATE-2".
;* (3) Use "SESSIONx" (where x is a number) to specify a different
;* session that you have previously configured.
;* (4) Use "DEVICE=x" to specify a different device. See the file
;* DSADP.INI in your windows directory to see the format for x. For example :
;* "DEVICE=XTEC,S,COM1,9600,E,7,1,X,4000,2666,1333,C,D"
;*;*****************************************************************************
Set SecondaryConnect To  "" ; Alternative connect methods second time. (See
comment above).
Set SubsequentConnects To  "" ; Alternative connect methods subsquent times.
(See comment above).
Set MaxRecoveryTries  To  3 ; Maximum number of times to try to recover.
Set TerminateAtEnd To TRUE ; Whether to end all macros if recovery ultimatily fails.
Set WaitSendTimeOut  To  30 ; Secs to wait before timing out sends.
Set WaitAfterDisconnect  To 30 ; Secs to wait after a disconnection.
Set WaitInitialRecovery To  60 ; Secs to wait before recovering first time.
Set WaitSubsequentRecoverys To  120 ; Secs to wait before recovering subsequent
times.
Set ShowTrace To TRUE ; Whether to write trace information to your logging file.
```

SECTION 2
Sets up the recovery logging process. This can be 'switched off' by setting the SHOWTRACE variable in SECTION 1 to FALSE.

```
;*******************************************************************************
;*SECTION 2: Log recovery process to file.
;*******************************************************************************
Set TraceCaption To "Recovery attempt No. " + Str$(&RecoveryAttempts)
If (ShowTrace) Then
   Message ("Starting recovery macro.", TraceCaption)
EndIf
```

SECTION 3
Checks the maximum number of times a recovery attempt has been made, on a 'per command' basis, and determines how the macro behaves if this exceeds the maximum allowed. By default, with TerminateAtEnd set to TRUE, `RECOVER.MAC` writes a 'Terminating recovery macro' message to the log file and then kills both recovery and calling macros. The alternative method (logging a 'Giving up recovery' message and the end state, then stopping recovery and returning control to the calling macro) can be set simply by changing the TerminateAtEnd variable to FALSE.

```
;*******************************************************************************
;* SECTION 3: Check we've not tried too many times.
;*******************************************************************************
If (&RecoveryAttempts > MaxRecoveryTries) Then
   If (ShowTrace) Then
      Set Msg To "Max (" + Str$(MaxRecoveryTries) + ") recovery attempts
      exceeded."
      Message (Msg, TraceCaption)
   EndIf
   If (TerminateAtEnd) Then
      If (ShowTrace) Then
         Message ("Terminating recovery macro.", TraceCaption)
      EndIf
      EndAllMacros ()
```

```
   Else
      If (ShowTrace) Then
         Message ("Giving up recovery.", TraceCaption)
      EndIf
      RecoverStop
      Goto FINISH
   EndIf
EndIf
```

SECTION 4

Checks the number of the current recovery attempt, logs appropriate messages and sets the relevant waiting period.

```
'*****************************************************************************
'* SECTION 4: We will wait at the start of recovery to allow time for events to change.
'*****************************************************************************
If (&RecoveryAttempts <= 1) Then
   If (ShowTrace) Then
      Set Msg To "Waiting for " + Str$(WaitInitialRecovery) + " seconds."
      Message (Msg, TraceCaption)
   EndIf
   Wait (WaitInitialRecovery)
Else
   If (ShowTrace) Then
      Set Msg To "Waiting for " + Str$(WaitSubsequentRecoverys) + " seconds."
      Message (Msg, TraceCaption)
   EndIf
   Wait (WaitSubsequentRecoverys)
EndIf
```

SECTION 5

Checks two things: are you connected to the Datastream host and are you at the Datastream prompt (Program Finder). It then attemps to get you to the prompt if you are not already there and, if that fails, logs suitable status information and disconnects you. If both conditions are true, this status is logged and the recovery ends.

```
;*******************************************************************************
;* SECTION 5: Try to get to the Datastream prompt if already connected.
;*******************************************************************************
If (&ConnectState = CONNECTED) Then
   If (&AtPrompt) Then
      If (ShowTrace) Then
         Message ("Datastream prompt found.", TraceCaption)
      EndIf
      Goto FINISH

   Else
      If (ShowTrace) Then
         Message ("Already connected. Sending [CLEAR].", TraceCaption)
      EndIf
      Send (Text:"[CLEAR]", Timeout:WaitSendTimeOut)

      If (&AtPrompt) Then
         If (ShowTrace) Then
            Message ("Datastream prompt found.", TraceCaption)
         EndIf
         Goto FINISH
      EndIf

   EndIf


   If (ShowTrace) Then
      Message ("Connected, but not responding. Disconnecting.", TraceCaption)
   EndIf
   Disconnect ()

   If (ShowTrace) Then
      Set Msg To "Waiting for " + Str$(WaitAfterDisconnect) + " seconds."
      Message (Msg, TraceCaption)
   EndIf
   Wait (WaitAfterDisconnect)

EndIf
```

SECTION 6

Attempts a reconnection and logon. The method it uses to connect is determined by the current recovery attempt count. Appropriate information is logged.

```
;*****************************************************************************
;* SECTION 6: Try reconnecting.
;*****************************************************************************
If (&RecoveryAttempts = 2) Then
   If (SecondaryConnect <> "") Then
      If (ShowTrace) Then
         Set Msg To "Connecting using : " + SecondaryConnect
         Message (Msg, TraceCaption)
      EndIf
      Connect (SecondaryConnect)
   EndIf
EndIf
If (&RecoveryAttempts > 2) Then
   If (SubsequentConnects <> "") Then
      If (ShowTrace) Then
         Set Msg To "Connecting using : " + SubsequentConnects
         Message (Msg, TraceCaption)
      EndIf
      Connect (SubsequentConnects)
   EndIf
EndIf
If (ShowTrace) Then
   Message ("Calling logon macro.", TraceCaption)
EndIf
Logon ()
```

SECTION 7
Logs the end state of the recovery process, based on whether you are connected and at the prompt.

```
;*******************************************************************************
;* SECTION 7: Finish. Log end state to file.
;*******************************************************************************
FINISH:
If (ShowTrace) Then
   Set Msg To "Recovery over."
   If (&ConnectState = CONNECTED) Then
      Set Msg To Msg + " Connected."
   Else
      Set Msg To Msg + " Disconnected."
   EndIf
   If (&AtPrompt) Then
      Set Msg To Msg + " At the prompt."
   Else
      Set Msg To Msg + " Not at the prompt."
   EndIf
   Message (Msg, TraceCaption)
EndIf
```

SECTION 8:
Ends the recovery macro.

```
;*******************************************************************************
;* SECTION 8: The End.
;*******************************************************************************
End
```

Example macros

This section documents a set of example macros which are are provided with DSWindows 2.2. The macros are intended to illustrate:

- the range of activities you can automate using Datastream macros
- how typical macros are constructed

The macros themselves reside in the \dswindow\files directory and you can edit them for your own purposes, or copy them and edit the copies as you like. The names of the macros indicate their function; for example, ex_prnt.mac illustrates a macro which produces printed output, ex_save.mac illustrates a macro which creates a save file.

Example 1: DEMOGLST.MAC

This macro runs program 401A three times, to download three graphs, each one comparing a stock with its market index, from 1st January 1991. The indices are rebased to the starting value of the stocks they are compared with. The list of stocks and indices is at the end of the macro, at the point labelled DEMOLIST.

```
OPENDATA DEMOLIST

Loop:

IF &ENDOFDATA = FALSE THEN
      INPUT STOCK, INDEX
      SEND( "401A "+ STOCK +","+ INDEX +",,1/1/91,,,3" )
      > [CLEAR]
GOTO Loop

ENDIF
```

```
DEMOLIST:

DATA
      "BP",            "FTSE100"
      "F:PGT", "FRCAC40"
      "J:RH@N",     "JAPDOWA"
ENDDATA
```

Example 2: DEMO900.MAC

This macro runs the Data Channel programs 900A and 900B, saving the output in a .CSV file named "DEMO900". (You can use a .CSV (Comma Separated Value) file to export data to spreadsheets, such as Excel.)

The macro begins by opening this save file, then runs 900A to download information (the number of shares in issue, NOSH and the beta coefficient, BETA) for two stocks, British Telecom (BT) and Marks & Spencer (MKS). Then, using program 900B, it downloads recent daily values for a number of stocks.

```
STARTDC (CSVFILE,"DEMO900.CSV")

> 900A

> BT,MKS

> NOSH,BETA

SEND ( "900B BT,-3M,,D" )

SEND ( "900B BT(MV),-3M,,D" )

SEND ( "900B MKS(P),-4M,,D" )

SEND ( "900B CTRP(P),-2M,,D" )

SEND ( "900B BP(P),-3M,,D" )

ENDDC
```

Example 3: DEMOGRPH.MAC

This macro runs program 401A three times to download a series of graphs showing values for the last two years for the FTSE 100 index, the Dow Jones Industrial index (DJINDUS) and the share price of BP compared to the FTSE 100. Then it runs the flexible format program 401X twice.

```
SEND ( "401A FTSE100" )
SEND ( "DJINDUS" )
SEND("BP[ERASE_EOF][TAB]FTSE100[DOWN][DOWN][DOWN][TAB][TAB]2"
SEND ( "[CLEAR]" )
SEND ( "401X 001G" )
SEND ( "401X 002G" )
```

The following macro (similar to the one above) uses the abbreviated form of the **SEND** command:

```
> 401A JAPDOWA
> ICI[ERASE_EOF][TAB]FTSE100[DOWN][DOWN][DOWN][TAB][TAB]2
> [CLEAR]
> 401X 003G
> 401X 005G
```

Example 4: DEMOLIST.MAC

This macro creates a local list file, called DEMO.LST, consisting of codes for equities in the German brewing sector. Then, for each of these equities, it runs program 900B to download data into a .CSV file, called DEMO.CSV.

```
MESSAGE( "Running 900A to create DEMO.LST","DEMOLIST.MAC" )
STARTDC( LISTFILE,"DEMO.LST" )
> 900A BREWSD
ENDDC
MESSAGE( "Now run List File DEMO.LST",      "DEMOLIST.MAC" )
STARTDC ( CSVFILE,"DEMO.CSV" )
```

```
OPENDATA "DEMO.LST":LIST

Loop:

IF &ENDOFDATA = FALSE THEN
      INPUT code
      SEND (  "900B " + code + ",1/1/91,30/1/91,D" )
      GOTO Loop
ENDIF

ENDDC

END
```

Example 5: DEMOSAVE.MAC

This macro contains 5 sections, which are concerned with capturing, saving and printing data.

The first section downloads data from 99FX1 and saves it in DSWindows format. When you run this, you will be prompted for a file name.

```
CAPTURE
> 99FX1
ENDCAPTURE
```

This section downloads data from 99FX2 and saves it as plain text for use in other Windows applications. When you run this, you will be prompted for a file name.

```
CAPTURE ( 1 )
> 99FX2
ENDCAPTURE
```

This section downloads data from 99FX3 and saves it in DSWindows format. The data will be saved into the file specified ("DEMO.DST")

```
CAPTURE ( "DEMO" )
> 99FX3
ENDCAPTURE
```

This section downloads data from 99FX4 and saves it as plain text for export. The data will be saved into a file specified and printed out as it is received.

```
CAPTURE ( "DEMO",1 )
AUTOPRINT
> 99FX4
ENDPRINT
ENDCAPTURE
```

This section prints out the data in the specified file.

```
PRINTSAVEFILE ( "DEMO" )
```

Example 6: DEMOPSS.MAC

Demo for connecting via UK PSS.

```
CONNECTNOWAIT()
START:
SEND( "ATZ", "OK", TIMEOUT:3)
SEND( TEXT:"ATE1", "OK" )

; DIAL YOUR LOCAL PAD
SEND( "ATDXXXXXXXX", WAITFOR:"CONNECT", 45 )
IF &SENDCOMPLETE <> TEXTFOUND THEN
MESSAGE( "DID NOT FIND 'CONNECT' ", "DEMOPSS.MAC")
GOTO START
ELSE
      SEND( "[ENTER][ENTER]SP[ENTER]", "LO" )
      Send( "[ENTER]", "NUI" )
```

```
; enter your NUI
      Send( "NUIXXXX", "ADD" )
      Send( "A212301202", "LOGON" )
; enter your user id and password
      Send( "UUUUUUUUPPPPPP" )
EndIf
```

Example 7: DEMODATE.MAC

Demonstrates the use of AddToDate, CompareDates, EndOfPriorPeriod, &dayOfMonth, &month, &year

```
; ******  Request data for every three working days  ******
; You must specify the date in the format that you are configured for.
; See Options>Configure>Dates in the terminal window.

Set date    To "1/1/95"
Set endDate To "31/1/95"

LOOP:

   Send( "900A FTSE,P," + date )
   AddToDate( date, 3, "WEEKDAY" )
   Set date To &result
   CompareDates( date, endDate )
   If &result > 0 Then
   Goto LOOP

   EndIf

;End of LOOP

; ******  Request data for the end of the last quarter  ******

Set date To str$(&dayOfMonth) + "/" + str$(&month) +"/" + str$(&year)
EndOfPriorPeriod( date, "QUARTER" )
Set endOfLastQuarter To &result
Send( "900A FTSE,P," + endOfLastQuarter )
```

Example 8: DEMOSITA.MAC

Demo illustrating how to connect to Datastream via SITA dial-up

NOTE *Before using this macro please set up the following 5 pieces of information. Put the values between double quotes. For example:*

Set sitadialPhoneNumber to "9 123 45678"

```
Set sitadialPadPhoneNumber To "xxxxxxxxxx"
Set sitadialNUI              To "xxxxxxxx"
Set sitadialPassword         To "xxxxxx"
Set datastreamLogonId        To "xxxxxxx"
Set datastreamPassword       To "xxxxxx"
Set error To ""
If &OS = "WINDOWS" Then
   Set EOL To chr$(13) + chr$(10)
Else

   Set EOL To chr$(10)
EndIf

ConnectNoWait()
Send( "ATZ", "OK", 3 )
Send( "ATE1", "OK", 3 )

Send( "ATD" + sitadialPadPhoneNumber, "CONNECT", 45 )

If &sendComplete <> TEXTFOUND Then

   Set errorNumber To "CMS1"
   Set error To "Could not connect to SITADIAL PAD - phone number "
   Set error To error + sitadialPadPhoneNumber + EOL
   Set error To "If the line is busy please try again."
   Goto DONE

EndIf
```

(continued...)

```
Wait( 2 )
Send( "...[ENTER]", "SITA", 10 )
If &sendComplete <> TEXTFOUND Then

   Set errorNumber To "CMS2"
   Set error To "Sita pad did not respond. Please try again."
   Goto DONE

EndIf

Send( "NUI " + sitadialNUI, "XXXXXX", 10 )
If &sendComplete <> TEXTFOUND Then

   Set errorNumber To "CMS3"
   Set error To "Your Sita NUI was not recognised. "
   Set error To error + "Please check the sitadialNUI in this macro."
   Goto DONE

EndIf

Send( sitadialPassword, "active", 10 )
If &sendComplete <> TEXTFOUND Then

   Set errorNumber To "CMS4"
   Set error To "Your Sita password was not recognised. "
   Set error To error + "Please check the sitadialPassword in this macro."
   Goto DONE

EndIf

Send( "9000132", "Dummy", 45 )
If &sendComplete <> UNLOCK Then

   Set errorNumber To "CMS6"
   Set error To "Could not contact Datastream host. "
   Set error To error + "Please try again."
   Goto DONE

EndIf
```

(Continued...)

```
Send( datastreamLogonId + datastreamPassword )
DONE:
If error <> "" Then

   Set error To "Error " + errorNumber + EOL + EOL + error
   Set error To error + EOL + EOL
   Set error To error + "If the problem persists please check your "
   Set error To error + "modem configuration." + EOL
   Set error To error + "If this does not solve the problem please "
   Set error To error + "call your Datastream representative."
   Message( error, "Connection error whilst running demosita.mac" )
   Disconnect()

EndIf
```

Example 9: EX_PRNT.MAC

This macro illustrates how to run simple Datastream programs and print the output.

```
SEND("401A JAPDOWA")                    To send a comment to Datastream either
                                        use the SEND command,

SEND("[CLEAR]")

>401A FTSE100                          or use the abbreviated form of the
                                        SEND command, >.

>[CLEAR]

ACTIVATETERMINAL

AUTOPRINT                               Turn printing on.

>190A D:VW                             Print report on a company,

SEND ("401A")                           and a performance chart.

SEND("D:VW[TAB]DAXINDX[DOWN][DOWN][DOWN][TAB][TAB]2")

>[CLEAR]

ENDPRINT                                Turn off printing.

END                                     End the macro.
```

Example 10: EX_SAVE.MAC

This macro demonstrates how to save text and graphics into separate files, and then how to reload and print them.

```
CAPTURE("ex_demo.dst",0,1,OVERWRITE)
```

Open a save file named "ex_demo.dst" to view again in DSWindows, containing only output pages; overwrite anything already in the file.

```
>301A  ICI
```

Retrieve data on ICI.

```
>[CLEAR]
>190E  ICI
>[CLEAR]
ENDCAPTURE
```

Close the text file.

```
ACTIVATEGRAPHICS
```

Make the Graphics window active.

```
AUTOSAVE("ex_demo.dsg",OVERWRITE)
```

The AUTOSAVE *command saves all subsequent graphics (until the* ENDAUTOSAVE *command). NB: These charts are saved as they arrive from Datastream, therefore subsequent annotations will not be saved - to save these use the* SAVEGRAPHICS *command.*

```
>401B ICI
```

Retrieve data in graphics form on ICI.

```
>[CLEAR]
>401F ICI
>[CLEAR]
ENDAUTOSAVE
```

Close the Graphics file.

```
PRINTSAVEFILE("ex_demo")
```

Print the text file.

PRINTGRAPHFILE("ex_demo.dsg")	*Print the graphics file.*
END	*End the macro.*

Example 11: EX_LIST.MAC

This macro illustrates how a loop can be set up to print a number of charts (Global Formats) whose format numbers are held in a list. This list is at the end of the macro in this example - it could also be held in a separate file.

OPENDATA demolist	*The* OPENDATA *command identifies the source for the data.*
Loop:	*A label (*Loop:*) is placed at the start of the loop for the macro to return to at the end of the loop - using the* GOTO *command.*
IF &ENDOFDATA = FALSE THEN	*An* IF *statement is then used to test whether the system variable &ENDOFDATA has been set to TRUE. This occurs when the end of the data list is found, and the loop is ended after the* ENDIF *statement.*
INPUT format	*The first operation within the loop is to input the next value from the data list.*
SEND("401X " + format)	*The macro then requests the chart using the* INPUT*ed format from the data list,*
> [CLEAR]	
PRINTGRAPHICS	*prints,*
GOTO Loop	*and returns to the start of the loop.*

```
ENDIF

END

demolist:
```

*A label (*demolist:*) identifies the start of the data list and the data is held between the* DATA *and* ENDDATA *commands.*

```
DATA
      "001G"
      "002G"
      "003G"
      "004G"
      "005G"
      "006G"
      "007G"
      "008G"
      "009G"
      "010G"
      "011G"
      "012G"
ENDDATA
```

Example 12: EX_SET.MAC

This macro is similar to the EX_SAVE macro and EX_LIST macros, and illustrates how to use the **SET** command to assign a value to a variable. In this example a value for the APPEND/OVERWRITE indicator is set to OVERWRITE initially and then changed to APPEND.

Note: *The 101A program appears in the graphics window because /Y is appended to the request:*

```
SET app_over TO OVERWRITE

ACTIVATEGRAPHICS                    Note: the graphics window must be
                                    active before any graphics-related macro
                                    commands can be run.

OPENDATA demolist

Loop:

IF &ENDOFDATA = FALSE THEN
     INPUT code
     SEND("101A "+code+"/Y")
     >[CLEAR]
     SAVEGRAPHICS("ex_demo.dsg",APPEND:app_over)

                                    To change the variable to APPEND
                                    after the first new graph is saved.

     SET app_over TO APPEND
     GOTO Loop
ENDIF
PRINTGRAPHFILE("ex_demo.dsg")

END

demolist:

DATA
     "J:KB@N"
     "U:BUD"
```

```
        "GMET"
        "ALLD"
        "BASS"
        "J:ASBR"
        "R:SABJ"
        "H:HB"
        "O:OBRA"
ENDDATA
```

Example 13: EX_STRNG.MAC

This macro demonstrates how string operators can be used to remove full stops from mnemonics, which can then be used as file names for charts saved to disk.

STARTDC(LISTFILE,"ftse.lst")	*First, Data Channel is started and a file called "ftse.lst" is opened.*
> 900A FTSE,MNEM	*A local list of the mnemonics in the FTSE is generated, and saved in the file (see EX_LIST.MAC for more details on using data lists).*
ENDDC	*Data Channel is closed.*
OPENDATA "ftse.lst":list	*NB It may be necessary to add the path in the* OPENDATA *command.*
Loop:	*Set up a loop.*
IF &ENDOFDATA = FALSE THEN	
INPUT code,mnem	*Take each mnemonic in turn and amend the mnemonics, if necessary, by replacing the full stop with an underline.*
SET file TO mnem	

```
        SET I TO 1
        loop1:
        IF MID$(file,I,1)="." THEN
            SET file TO LEFT$(file,I-1)+"_"+MID$(file,I+1,4)
        ENDIF

        SET I TO I+1
        IF I <= 4 THEN
            GOTO Loop1
        ENDIF

        SEND(  "401B "+ mnem + ",FTSE100,1/1/87" )
        SAVEWMF (file +".wmf")          Generate the charts and save these to
                                        .WMF files.

        SEND("[CLEAR]")
        GOTO Loop
ENDIF

END
```

Example 14: EX_SYSTM.MAC

This macro illustrates how you can use the system variables &MONTH and &YEAR with the SEND command to fill in Datastream input fields.

The purpose of the macro is to graph the basic relationship between the near term futures contract on the DAX index future and the index. This macro uses the IF and OR commands to find the near term contract in the three month cycle.

```
IF &MONTH=3 OR &MONTH = 6 OR &MONTH = 9 OR &MONTH = 12
  THEN
    SET near_mth TO &MONTH
ENDIF

IF &MONTH=2 OR &MONTH = 5 OR &MONTH = 8 OR &MONTH = 11
THEN
```

```
    SET near_mth TO &MONTH + 1
ENDIF

IF &MONTH=1 OR &MONTH = 4 OR &MONTH = 7 OR &MONTH = 10
THEN
    SET near_mth TO &MONTH + 2
ENDIF
SEND( "401A " )

IF near_mth <=9 THEN
SEND("DAXINDX[TAB]GDX0"+STR$(near_mth)+STR$(&YEAR)+"[NOENTER]"
)
ELSE
SEND("DAXINDX[TAB]GDX"+STR$(near_mth)+STR$(&YEAR)+"[NOENTER]"
)

ENDIF
SEND("[TAB][TAB]-3M[TAB][TAB][TAB][TAB]CASH BASIS RELATIONSHIP" )
SEND("[CLEAR]")
END
```

Example 15: EX_MASTR.MAC

This macro illustrates how you can set up a ‘master’ macro to run other (child) macros. This technique enables you to keep your macro short and easy to maintain.

The CALL command is used to run the child macros, with control returning to the master after each child has run.

NOTE *All save files are closed after each child macro is executed.*

```
CALL "EX_PRNT.MAC":
CALL "EX_SAVE.MAC":
CALL "EX_LIST.MAC":
CALL "EX_SET.MAC":
END
```

Example 16: EX_401X.MAC

This macro illustrates how the keystroke recorder can be used to record the keystrokes used to fill in a 401X input screen.

The graph plots the share prices for the French company BIC and the CAC 40 index. Data for 6 months is requested, at daily frequency, grids for both the X and Y axes, daily tick marks on the X-axis and the second series rebased to the starting value of the first. The equivalent input screen is shown opposite.

To start the keystroke recorder, click on the Recorder button or select this option from the macro menu option on the terminal menu bar.

The [NOENTER] commands are supplied by the recording process - you do not have to enter these.

```
Send( "401X C " )

Send("F:BIC[TAB][TAB][TAB][TAB]FRCAC40[TAB][TAB][TAB][NOENTER]")

Send("[TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB]D[NOENTER]")

Send("-6M[TAB][TAB][TAB]BOTHD  [TAB][TAB][TAB][TAB][TAB][NOENTER]")

Send("[TAB][TAB][TAB]S")

PRINTGRAPHICS                    The graph can then be printed by adding
                                 a PRINTGRAPHICS command.

END
```

NOTE *The keystroke recorder only records the information sent to Datastream. You have to add the* PRINTGRAPHICS *command manually.*

401X input screen

Example 17: EX_GLIST.MAC

This macro illustrates how you can generate a chart for each company in a list (German Breweries with Market Value over 500M DM in this example) and load them into a layout of four charts and then print the layout.

Note: *A layout with four charts named "four" must be set up before this macro is run. The layout is then loaded using the LOADLAYOUT command.*

```
ACTIVATEGRAPHICS

LOADLAYOUT ("four")

SET grph_count TO 0          Next a counter (grph_count) is set to keep track of the number of charts loaded into each layout.

OPENDATA list

Loop:                        A loop is set up to generate the four charts in the layout - see EX_LIST.MAC for a detailed explanation of loops and associated data lists.

If &ENDOFDATA = FALSE THEN

    INPUT company

    >[CLEAR]

    SELECTGRAPH (grph_count)     The counter is used to specify which slot in the layout to make visible, and is incremented in each iteration of the loop.

    SEND( "401B " + company + ",BREWSSW" )

    SET grph_count TO grph_count+1
```

(Continued...)

```
        IF grph_count = 4 THEN
            PRINTGRAPHICS
            SET grph_count TO 0
            DESELECTGRAPH(1)
            DESELECTGRAPH(2)
            DESELECTGRAPH(3)
        ENDIF
        GOTO Loop
    ENDIF
    IF grph_count <> 4 THEN
        PRINTGRAPHICS
    ENDIF
    END
    list:
```

After four charts have been placed in the layout, the layout is printed and the counter is reset. The second, third and fourth slots in the layout are de-selected in case there are insuffucient companies to fill these. (It is assumed there are at least 4 companies in the list.)

A final PRINTGRAPHICS *command may be required to print the final layout, if the total number of graphs is not a multiple of four.*

(Continued...)

```
DATA

    "D:BBA"
    "D:PSC"
    "D:PTH"
    "D:BIN"
    "D:LBR"
    "S:DWB"

ENDDATA
```

Example 18: EX_GSTYL.MAC

This macro illustrates how different text, line and fill styles can be loaded in a macro. Examples of using 'special keys' such [ERASE_EOF] to erase to end of field, [QUOTE] to send an inverted comma and [NOENTER] to indicate that the input for that screen is continued on the next line - are also incuded

```
ACTIVATEGRAPHICS
```

New line styles (called Daily_reports) and text styles (called Daily_reports) are loaded (N.B. as with layouts these must be set up before this macro is run). For example, the text styles could be set so that the titles are underlined and the sub title and legends are in a smaller font, while the line and fill styles are set to different colours

```
LOADTEXTSTYLES ("Daily_reports")

LOADLINESTYLES ("Daily_reports")

LOADFILLSTYLES ("Daily_reports")
```

A loop is used to generate the four charts in the layout see EX_LIST.MAC for detailed explanation of loops and associated data lists

```
OPENDATA list

Loop:

If &ENDOFDATA = FALSE THEN

    INPUT company, index, name

    >[CLEAR]
```

Three charts are created in a 401X by amending the Global format 005G to show the performane of large software companies in different markets

```
    SEND( "401X A 005G" )
```

The first chart displays the share price

```
SEND("[ERASE_EOF]"+company+"[TAB][TAB][TAB][QUOTE][NOENTER]")

SEND("[TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB] [NOENTER]")

SEND( "[TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB] [NOENTER]")

SEND( "[TAB][TAB][TAB][TAB][TAB][TAB][ERASE_EOF][NOENTER]")

SEND( "SHARE PRICE[TAB][TAB][TAB][TAB][TAB][ERASE_EOF]"+name )
```

The second chart displays the market value

```
SEND("[ERASE_EOF]"+company+"(MV)[TAB][TAB][TAB][QUOTE] [NOENTER]" )

SEND( "[TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][NOENTER]")

SEND( "[TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][NOENTER]")

SEND( "[TAB][TAB][TAB][TAB][TAB][TAB][ERASE_EOF]MARKET VALUE")
```

The third chart displays the index performance

```
SEND( "[ERASE_EOF]"+index+"[TAB][TAB][TAB][ERASE_EOF][NOENTER]" )

SEND( "[TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB]
[NOENTER]" )

SEND( "[TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB]
[NOENTER]"

SEND( "[TAB][TAB][TAB][TAB][TAB][TAB][ERASE_EOF]" )
```

The charts are saved to disk so that they can be displayed at a later date using the Graphics Browse facility.

```
            SAVEGRAPHICS ("softw.dsg",,APPEND)
            GOTO Loop
ENDIF
END

list:
DATA
            "@MSFT","S&PCOMP",    "MICROSOFT"
            "F:CGS",  "PARCACG",    "CAP GEMINI SOGETI"
            "J:ZV@N",       "JAPDOWA",    "CSK CORP."
            "MFCS",         "FTALLSH",     "MICRO FOCUS GP."
            "C:SHKT","TTOCOMP",    "SHL SYSTEMHOUSE"
ENDDATA
```

Example 19: EX_GANT.MAC

This macro illustrates how you can customise a Datastream chart using the graphics annotation macro commands introduced in DSWindows 2.0. These charts are then saved as .WMF files and in a DSWindows save file for display in the Browse facility.

The SAVEGRAPHICS command is used in this macro in preference to AUTOSAVE, because AUTOSAVE does not save annotations.

Note: *See* EX_SYSTM.MAC *for an example of how to use system variables.*

```
SEND( "[CLEAR]")

OPENDATA list

Loop:

IF &ENDOFDATA = FALSE THEN

INPUT mnemonic

Send( "401X C")                          The tabs required to complete this 401X
                                         screen can be recorded using the macro
                                         keystroke recorder.

SEND( mnemonic + "[TAB][TAB][TAB][TAB]FTSE100[TAB][TAB][NOENTER]")

SEND( "[TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][TAB][NOENTER]")

SEND( "[TAB]D-1Y[TAB][TAB][TAB]BOTH[TAB][TAB][TAB][NOENTER]")

SEND( "[TAB][TAB][TAB]1,75 [TAB][TAB][TAB]I[TAB][TAB][TAB][TAB]N")

SELECTITEMS (CONTAINS:"FTSE")            The SELECTITEMS command is used
                                         to select the item(s) containing the text
                                         "FTSE".
```

(Continued.....)

```
CHANGEITEMS (TEXT:"FTSE 100")
```

The CHANGEITEMS *command is used to change these item(s) to "FTSE 100".*

```
SELECTITEMS (IN:0,0,600,70)
```

The SELECTITEMS *command is used to select all the item(s) in the area which has a lefthand co-ordinate (l) = 0, top (t) = 70, righthand (r) = 600 and bottom (b) = 0 - the legend in this instance.*

```
MOVEITEMS (TO:780,450)
```

The MOVEITEMS *command is used to move the selected items to the right of the chart.*

```
NEWBOX(BORDER:"Axis",SHADOW:"Fill Style 2",\
    SHADOWBORDER:"Axis",SHADOWOFFSET:5)
```

The NEWBOX *command is used to add a box around these items.*

```
DESELECTITEMS
```

The DESELECTITEMS *commands de-select these items.*

```
SAVEWMF (mnemonic+".wmf")
```

Each graph is saved in a separate .WMF file using the mnemonic to form the name.

```
SAVEGRAPHICS( "brw"+STR$(&DayOfMonth)+"_"+STR$(&Month), APPEND
```

The graphs are also saved in a .DSG file. The name of this file is determined by the date. For example, if the date is the 17th of May, the file will be called BRW17_5.DSG.

```
    SEND( "[CLEAR]")

    GOTO Loop

ENDIF
```

(Continued.....)

```
END

list:                                   This is a list (abbreviated) of UK
                                        brewery companies.

DATA
      "ALLD"
      "BASS"
      "BODD"
      ...

      "WOLV"
      "YNGBA"
      "YNGBNV"
ENDDATA
```

Example 20: EX_300C

This macro illustrates how time series data can be uploaded to the Datastream mainframe using the 300C program.

NOTE *This macro assumes that the series starts on a MONDAY (for daily data) and that values are available for ALL subsequent WEEKDAYS*

```
SEND( "[CLEAR]")

USERINPUT "UPLOADING TO 300C","Enter series name - TSXXXXXX",seriesn

USERINPUT "UPLOADING TO 300C","Enter series title",title

USERINPUT "UPLOADING TO 300C","Enter management group",mgmt

USERINPUT "UPLOADING TO 300C","Enter frequency (D W or M)",freq

; Initialise the series counter

SET seriesc TO 0

SEND("300C")

; Enter series details and check if the mnemonic is already in use

SENDANDCHECK("1"+seriesn + mgmt,"ALREADY IN USE")

IF &TEXTFOUND=TRUE THEN

        SEND ("[CLEAR]")

        MESSAGE ("MNEMONIC ALREADY IN USE - ABORTING")

        GOTO abort

ENDIF
```

```
; Open the text file and request the start date, Week day, Frequency and data.

OPENDATA LIST

INPUT st_date

INPUT W_day

INPUT freq

INPUT series

; Set the number of fields per input page according to the frequency

IF freq = "M" THEN

; Fix for monthly frequencies
; This checks the 'month' value of the start date, and reduces the index
; number 'nof_fields' so that the macro knows to expect the correct number
; of fields in the 300C input screen. A similar method exists for weekly
; frequencies, but the value for the day of the week is taken from the
; data list (at end of file)
; Additionally, another variable is introduced. The index 'nof_fields' holds
; the number of fields on the first input screen, taking into account the
; starting month (as explained above). The index 'nos_fields' holds the
; number of fields in the subsequent, standard 300C input screen for that
; frequency.

        SET month TO MID$ (st_date,4,2)

        SET monthv TO VAL(month) ;Get value of starting month

        IF monthv > 1 THEN

        SET nof_fields TO (4 * 12) - monthv + 1 ;Reduce nof_fields by the number
        of months

        ELSE

            SET nof_fields TO 4 * 12

        ENDIF
```

```
            SET nos_fields TO 4 * 12
ELSE
IF freq ="W" THEN
            SET nof_fields TO 3 * 16
            SET nos_fields TO 3 * 16
ELSE
        SET nof_fields TO 5 * 16  + 2 - VAL(W_Day)
        SET nos_fields TO 5 * 16
ENDIF
ENDIF
; Enter start date and frequency
SEND("[TAB][TAB]"+freq+st_date)
; ENTER DECIMAL PLACES NEEDED IF DEFAULT IS NOT NEEDED
SEND(title+"[TAB][TAB][TAB]4")
; Enter series data
SET no_fields to nof_fields
Loop:
            SEND (series + "[TAB][NOENTER]")
            SET seriesc TO seriesc + 1
            IF no_fields = seriesc THEN
                  SEND("[ENTER]")
                  SET seriesc TO 0
```

```
                SET no_fields TO nos_fields
            ENDIF
            INPUT series
            IF &ENDOFDATA = FALSE THEN
            GOTO Loop
ENDIF
SEND (series + "[TAB][NOENTER]")
SEND("END")
Abort:
END

LIST:
DATA
"01/06/84"
"1"
"M"
"-5.6319"
"-7.2329"
"-8.6191"
"-9.4071"
"-10.537"
"-11.611"
"-12.796"
"-13.528"
"-14.076"
"-13.452"
"-12.589"
"-11.843"
```

"-11.093"
"-9.5674"
"-8.8451"
"-8.2106"
"-5.9607"
"-4.017"
"-3.096"
"-1.251"
"0.84922"
"1.78146"
"3.34402"
"5.49007"
"6.08029"
"7.20487"
"8.46825"
"9.26583"
"9.31617"
"10.7251"
"11.5098"
"12.325"
"14.2966"
"17.1919"
"18.8057"
"19.6419"
"19.3555"
"17.3954"
"14.062"
"11.4768"
"9.20832"
"7.06202"
"6.12312"
"5.74271"
"3.82016"
"1.4657"
"2.4531"
"1.86986"
"1.28188"
"2.05614"
"3.76479"
"2.67455"

"2.04128"
"1.53731"
"0.58442"
"-0.3153"
"-1.9787"
"-2.9644"
"-4.0972"
"-3.954"
"-3.0299"
"-0.9994"
"0.58839"
"2.2076"
"2.55317"
"2.84521"
"2.54214"
"2.25084"
"3.23574"
"4.6838"
"3.385"
"2.22372"
"0.97498"
"-2.9168"
"-6.0828"
"-8.0827"
"-9.2986"
"-10.72"
"-10.11"
"-11.171"
"-11.168"
"-11.795"
"-12.757"
"-13.543"
"-13.165"
"-11.428"
"-9.9368"
"-7.8014"
"-5.5606"
"-3.7873"
"-5.1211"
"-5.8271"

"-6.4294"
"-7.0471"
"-5.7303"
"-3.4643"
"-2.4217"
"-1.8796"
"-1.6259"
"-2.4321"
"-2.8379"
"-2.7359"
"-0.9785"
"0.29922"
"1.31007"
"1.03924"
"2.03509"
"2.03702"
"2.35086"
"3.23467"
"5.15004"
"6.48559"
"6.71982"
"7.15133"
"7.32432"
"7.80322"
"6.9294"
"7.99737"
"7.36332"
"5.73407"
"4.13655"
"2.92505"
"-0.3697"
"-1.6997"
"-1.7588"
"-1.7008"
"-1.7086"
"-0.4392"
"0.54434"
"0.79026"
"-0.3415"

ENDDATA

Example 21: EX_GANT1.MAC

This is a further example of how you can use the graphics annotation macro commands to customise the look of a chart (and in this case to add page numbers). The GRAPHPAGESETUP command is also used to change the printer configuration.

```
DISPLAYSINGLEGRAPH          NB The Graphics Window should be in
                            Single Graph mode.

SET count TO 0              Initialise a counter for the page numbers.

OPENDATA list

loop:

IF &ENDOFDATA=FALSE THEN

    INPUT index

    SEND ("401A "+ index)

    GRAPHDRAWOFF            The GRAPHDRAWOFF command
                            stops the screen being redrawn as the
                            chart is annotated.

    SELECTITEMS(CONTAINS:"225",STYLE:"Title")

    CHANGEITEMS(TEXT:"NIKKEI - PRICE INDEX")

    SELECTITEMS(TYPE:"TEXT",STYLE:"Title")

                            The SELECTITEMS and
                            CHANGEITEMS commands are used to
                            change title text.

    CHANGEITEMS(STYLE:"Style 1")
```

(Continued .....)

```
ADDTOSELECTITEMS(TYPE:"TEXT",STYLE:"Sub-Title")
```

The SELECTITEMS *and* CHANGEITEMS *commands are used to change the title text style (font, size, etc).*

```
NEWBOX(BORDER:"Line Style 1",SHADOW:"Fill Style 1", \
SHADOWBORDER:"Line Style 1",SHADOWOFFSET:5)
```

The NEWBOX *command puts a box around the title and the sub-title. Note the \ at the end of the first line, to indicate that the command continues on the next line.*

```
MOVEITEMS(BY:0,-30)

DESELECTITEMS

SELECTITEMS(TYPE:"TEXT",STYLE:"Sub-Title")

CHANGEITEMS(TEXT:"Daily over the last year")

DESELECTITEMS
```

To add the page numbers at the top of the chart:

```
SET count TO count + 1

NEWTEXT(TEXT:"PAGE " +STR$(COUNT), AT:500,755,STYLE:"Style 1")

DESELECTITEMS

GRAPHDRAWON
```

The GRAPHDRAWON *command re-enables the graph display.*

(Continued .....)

```
        GRAPHPAGESETUP(LEFT:200,TOP:200,RIGHT:200,BOTTOM:200,\
        ORIENTATION:"LANDSCAPE")
```

The GRAPHPAGESETUP *command sets the PAGE SETUP parameters. N.B. The numbers are the margin sizes in 100ths of an inch (i.e. 50=1/2 inch)*

```
        PRINTGRAPHICS

        >[CLEAR]

        GOTO loop

ENDIF

END

list:

DATA
        "DJINDUS"
        "JAPDOWA"
        "HNGKNGI"
        "FTSE100"
ENDDATA
```

Example 22: EX_CSV.MAC

This macro illustrates how to convert the codes from a Datastream (mainframe) list into a local list file, and then use it to download values for each of these series - saving the output as a .CSV file.

To convert the Datastream list in a local list on your PC use the STARTDC(LISTFILE, filename) and ENDDC commands, together with the 900A program and the list name (French banks).

```
STARTDC(LISTFILE,"DEMO.LST")

>900A BANKSF,MNEM

ENDDC

STARTDC(CSVFILE,"EX_CSV.CSV")
```

Use STARTDC(CSVFILE, filename)*, to start saving and converting the downloaded data. Use* STARTDC(CLIPBOARD) *if you prefer to transfer to the Clipboard.*

```
OPENDATA "DEMO.LST":LIST

Loop:
```

Set up a loop and open the data file as illustrated in EX_LIST.MAC*. NB it may be necessary to add the path in the* OPENDATA *command.*

```
IF &ENDOFDATA = FALSE THEN

    INPUT code,mnem

    >[CLEAR]
```

(Continued.....)

```
        SENDANDCHECK("900B "+mnem+",1/1/86,,D","$$"+CHR$(34)+"H0" )
```

A special form of the SEND *command (*SENDANDCHECK*) is used to check that the requested data is available from Datastream. (Note, as more than 15 years of daily data is required, two 900B requests are made). Also a* [CLEAR] *is sent to Datastream to ensure that it is ready for the request and the CHR$(34) denotes an inverted coma.*

```
        IF &TEXTFOUND=FALSE THEN
```

After each request the data returned from Datastream is checked to ensure that the request was valid.

```
        MESSAGE ("The data for this 900B request is not available")

        ENDIF

        >[CLEAR]

        SENDANDCHECK("900B "+mnem+",1/1/76,31/12/86,D","$$"+CHR$(34)+"H0")
```

A second request is made for the rest of the data.

```
        IF &TEXTFOUND=FALSE THEN
            MESSAGE ("There is no data for "+mnem+" prior to 1985")
        ENDIF

        GOTO Loop

    ENDIF

    ENDDC
```

An ENDDC *command is required to close the file and convert the data to CSV format. This file can then be loaded into Excel, 123/W, etc.*

```
    END
```

Example 23: EX_CLIP.MAC

This macro uses the USERINPUT command to request input for the 900B Data Channel program.

```
USERINPUT "DOWNLOAD TO CLIPBOARD",\
    "ENTER CODES (COMMA SEPARATED)",codes

USERINPUT "DOWNLOAD TO CLIPBOARD",\
    "ENTER START DATE (D/M/Y or -1M)",st_date

USERINPUT "DOWNLOAD TO CLIPBOARD",\
    "ENTER FREQUENCY (D,W,M,Q,Y)",freq

STARTDC(CLIPBOARD)

SET L TO 1

SET codes TO codes + ",!"          Add a terminator (!) to the string
                                   containing the list of codes. This is used
                                   later within the loop to test for the end
                                   of the code string.

loop:                              A loop using the SET and string
                                   manipulation commands is used to parse
                                   the 'codes' string to make multiple
                                   900B requests.

IF MID$(codes,L,1)="," THEN        When a comma is found,

    SET cd TO LEFT$(codes,L-1)     cd is set to be the code to the left of it.

    SET codes TO MID$(codes,L+1,60) The codes string is reset so that it
                                   consists of everything to the right of the
                                   comma.

    >[CLEAR]
```

(Continued .....)

```
        SENDANDCHECK("900B "+cd+","+st_date+",,"+freq,"$$"+CHR$(34)+"H0" )
```

A 900B request is made, using cd*, i.e. the current code.*

```
        IF &TEXTFOUND=FALSE THEN
            MESSAGE (cd + " is not a valid code")
```

If the Data Channel output is not found, display the message.

```
        ENDIF
        SET L TO 1
    ENDIF
    IF MID$(codes,L,1)="!" THEN
```

If the terminator is found at the end of the code string,

```
        GOTO endloop
```

the macro will continue processing at the label endloop*.*

```
    ENDIF
    SET L TO I+1
    GOTO loop
    endloop:
    ENDDC
    END
```

Example 24: EX_EXCEL.MAC

This macro illustrates how you can start another application, such as Excel, from a macro and open saved data from Excel. It also illustrates how you can number the .CSV files using a counter.

```
SET count TO 1                          Initiate a variable for the counter called
                                        count.

OPENDATA list

Loop:                                   Set up a loop and open the data file as
                                        explained in EX_LIST.MAC.

If &ENDOFDATA = FALSE THEN

    STARTDC(CSVFILE,"ex_xls"+STR$(count)+".csv")

                                        Use the value of the counter in the file
                                        name.

    INPUT mnemonic

    >[CLEAR]

    SEND ("900A")

    SEND (mnemonic)

    SEND ("NAME,P,PE,EPS,DY,DCV,DPSC,MV//")

    SET count TO count+1                Increment the value of the counter.

    ENDDC

    GOTO Loop
```

(Continued .....)

```
ENDIF

SET total_count TO count

SET count TO 2

SET csvfiles TO "ex_xls1.csv"
```

To append the file names (ex_xls...) to the program name (EXCEL) in the STARTPROGRAM *command, variables are set for the total number of files and the file names are concatenated.*
NB: If the Save file directory is not on the path, you should add the full path before the file name (ex_xls).

```
Loop1:

IF count< >total_count THEN

    SET csvfiles TO left$(csvfiles,100)+" ex_xls"+STR$(count)+".csv"
```

With each iteration of the loop (loop1) a further file name is added.

```
    SET count TO count+1

    GOTO Loop1

ENDIF
```

(Continued .....)

```
STARTPROGRAM ("c:\xl4\excel "+csvfiles)
```

The STARTPROGRAM *command is used to start EXCEL and open the CSV files.*
NB Excel will be loaded only if it is on the path or if you specify the path. Excel looks in the 'macros' directory for the .CSV files, since that is now the current directory. The .CSV files would, by default, be saved to the save file directory.

```
END

list:

DATA

    "BREWSA"
    "BREWSD"
    "BREWSH"
    "BREWSJ"
    "BREWSK"
    "BREWSS"
    "BREWSW"
    "BREWSZ"

ENDDATA
```

*The data for this list was created using the Extract facility in Code Lookup (entering *BREWERIES in the 'Description' field and ALL in the 'coverage' field for Industry Lists), then selecting all the extracted codes (by holding down the* <**Shift**> *key) and copying these to the Clipboard (NB: The Clipboard options in Code Lookup must be set to 'return codes' and 'quotes'.)*

Example 25: STARTUP.MAC

This macro runs when you load DSWindows.

ActivateBackpages	*Open the Backpages window.*
ActivateGraphics	*Open the Graphics window.*
ActivateTerminal	*Open the Terminal window.*
MaximizeTerminal	*Maximize the Terminal window.*
LOGON	*Call the LOGON macro.*

Example 26: LOGON.MAC

This macro runs when it is called by the startup macro, or when you select **Logon** on the Connect menu in the Terminal window. It connects your terminal to Datastream and logs on.

CONNECT	*Connect to Datastream*
SEND("DS")	*To logon to Datastream, send* DS.
SEND("XDSMnnnabcxyz")	*Then send your Datastream ID (*XDSMnnnn*) and password* (abcxyz*).*

Example 27: EX_PSS.MAC

This macro demonstrates how to connect via UK PSS. These commands should be inserted in your LOGON.MAC after (and replacing) the CONNECT command in LOGON.MAC.

```
CONNECTNOWAIT                          Open the port connecting DSWindows
                                       and Datastream.

start:

SEND( "ATZ", "OK", TIMEOUT:3)          Connect to the modem.

SEND( Text:"ATE1", "OK" )              Turn echo off from the modem, and wait
                                       for the modem to respond with "OK".

SEND( "ATDxxxxxxxx", WaitFor:"CONNECT", 45 )

                                       After receiving "OK" from the modem,
                                       dial the phone no: ATxxxxxxxx, and wait
                                       for the response: "CONNECT".

IF &sendComplete<>TEXTFOUND THEN
                                       If the PAD does not respond with
                                       "CONNECT", display the message:
                                       "DID NOT FIND 'CONNECT'", and
                                       start again.

    MESSAGE( "DID NOT FIND 'CONNECT' ", "DEMOPSS.MAC")

    GOTO start

    ELSE                               If the PAD responds with "CONNECT",
                                       logon to the PAD with:
                                       <Enter><Enter> <S> <P> <Enter>
                                       and wait for LO to be returned.
```

(Continued.....)

```
SEND("[ENTER][ENTER]SP[ENTER]", "LO")
```

*If the PAD responds with "***CONNECT***", logon to the PAD with:*
<Enter><Enter> <S> <P> <Enter>
and wait for LO to be returned.

```
SEND( "[ENTER]", "NUI" )
```

Press **<Enter>** *and wait for the* **NUI***: (Network User Identity) prompt***.**

```
; enter your NUI
SEND( "NUIXXXX", "ADD" )
```

Send your NUI and wait for the **ADD:** *(Network User Address) prompt.*

```
SEND( "A212301202", "LOGON" )
```

Send Datastream's user address, and wait for the **LOGON:** *prompt.*

```
; enter your user id and password
SEND( "UUUUUUUUPPPPPP" )
```

Send your Datastream User ID and password.

```
ENDIF
```

When the Program Finder is displayed, you are connected to Datastream.

Example 28: EX_WRITE

This macro illustrates the WriteToFile macro command. Problems are logged to a file that is displayed after the macro has run. To run this macro you must be logged on and at the Program Finder screen.

WriteToFile writes just what you ask it to. It does not automatically write carriage returns and line feeds to the file. The EndOfLine variable below is set to a string that contains a carriage return (13) and a line feed (10).

```
Set endOfLine     To chr$(13) + chr$(10)      ; Carriage return + line feed

Set quote         To chr$(34)                 ; ASCII value for a quote

Set tab           To chr$(9)                  ; ASCII value for a tab


; If the 900B program encounters an error it outputs $$"ER" on the second
line

Set errorString To "$$" + quote + "ER" + quote  ; $$"ER"
Set dateString  To str$(&dayOfMonth) + "/" + str$(&month) + "/" + str$(&year)
Set logFile     To "c:\dswindow\files\write.log"

WriteToFile( "Macro run date:" + dateString + endOfLine, logFile,
OVERWRITE )
WriteToFile( endOfLine, logFile )       ; Write a blank line

StartDC( CSVFILE, "write.csv" )

; Run through all the mnemonics listed at the foot of the macro. For each
; of them run the 900B program to request 10 years of weekly data to be
; included in the CSV files. If there is an error then write this to the
; log file.
```

```
OpenData LIST

LOOP:

   If &endOfData = FALSE Then
      Input mnemonic
      ; EndPage stops the 900B after one page of output
      EndPage
      Send( "900B " + mnemonic + ",1/1/85,1/1/90,W" )
            If mid$( &screen, 81, 6 ) = errorString Then
         Set fullError To mnemonic + tab + mid$( &screen, 81, 80 )
         WriteToFile( fullError + endOfLine, logFile )
         Send( "[CLEAR]" )

      Else
;Request the remaining pages of the 900B output
         AutoPage
          Send( "[ENTER]" )
      EndIf
      Goto LOOP
   EndIf

; End of LOOP

EndDC()

; Display the errors in notepad

StartProgram( "notepad c:\dswindow\files\write.log" )

LIST:
Data
   "@BOOT"          ; No data available between 1/1/85 and 1/1/90
   "@FOOT"          ; Should work without error
   "INVALID"        ; Invalid mnemonic
   "U:LEG"          ; Should work without error
   "ARM"            ; No data available between 1/1/85 and 1/1/90
EndData
```

Example 29: RECOVER.MAC

NOTE *The full text of RECOVER.MAC is documented in the section entitled,* ***'How to make your macros more robust****', pp. 107-123.*

Example 30: EX_TIMES.MAC

Demonstrates the ConstTimeSeries macro command which allows you to place a constant value (such as a string) in the next row/column.

```
STARTDC (CSVFILE,"consttim.csv")

   ConstTimeSeries("British Telecom","-6D","","D")
   SEND ("900B BT,-5D,,D")
   ConstTimeSeries("Primark","-6D","","D")
   SEND ("900B U:PMK,-5D,,D")
   ConstTimeSeries("Acorn Computers","-6D","","D")
   SEND ("900B ACRN,-5D,,D")

ENDDC ()

; If the AllowDuplicateTimeSeries is set to FALSE then a 900B request which
; has one time period wholly enclosed by the other (as in the example below)
; will produce only one column of output (i.e. the identical results are
; merged). If the value is set to TRUE the identical results will not be
; merged.
; NOTE: all changes to this option will be written to the ini file

ConfigureDC (Merge900B:1, Transpose900B:0, Titles900B:1, ColHeadings900B:0,
RowHeadings900B:0)

AllowDuplicateTimeSeries(FALSE) ; Activates on next call to ENDDC

STARTDC (CSVFILE,"dupltim1.csv")

   SEND ("900B BT,-2W,,D")
   SEND ("900B BT,-1W,,D")

ENDDC ()
```

```
AllowDuplicateTimeSeries(TRUE) ; Activates on next call to ENDDC

STARTDC (CSVFILE,"dupltim2.csv")

   SEND ("900B BT,-2W,,D")
   SEND ("900B BT,-1W,,D")

ENDDC ()

AllowDuplicateTimeSeries(FALSE) ; Activates on next call to ENDDC

STARTDC (CSVFILE,"dupltim3.csv")

   SEND ("900B BT ,-2W,,D")
   SEND ("900B ICI,-2W,,D")
   SEND ("900B BT ,-2W,,D")

ENDDC ()

;Resetting the value to default

AllowDuplicateTimeSeries(FALSE) ; Activates on next call to ENDDC
```

Example 31: EX_DATEF.MAC

Demonstrates the DateExportFormat macro command. Choose a date format as listed in the short date styles options in your Windows control panel.

NOTE *All changes to the date format will be written to the ini file.*

```
SetDateExportFormat("MM/dd/yy") ;  Activates on next call to STARTDC

STARTDC (CSVFILE, "expdate1.csv")
   >900B BT,-1Y,,D
ENDDC ()

SetDateExportFormat("dd.MM.yy") ;        Activates on next call to STARTDC
```

```
STARTDC (CSVFILE, "expdate2.csv")
   >900B BT,-1Y,,D
ENDDC ()

SetDateExportFormat("WINDOWS") ;        Activates on next call to STARTDC

STARTDC (CSVFILE, "expdate3.csv")
   >900B BT,-1Y,,D
ENDDC ()
```

Example 32: EX_PROMT.MAC

Example use of the new (2.2) system variable, &AtPrompt.

```
ActivateTerminal

If (&AtPrompt) ThenMessage ("Datastream prot found", "Test &AtPrompt")

Else
      Message ("Datastream prompt not found", "Test &AtPrompt")
EndIf

Message ("Sending Clear", "Test &AtPrompt")

>[CLEAR]

If (&AtPrompt) Then
      Message ("Datastream prompt found", "Test &AtPrompt")
Else
      Message ("Datastream prompt not found - you may not be logged on",
      "Test &AtPrompt")

EndIf
```

Example 33: EX_CLOSE.MAC

Demonstrates how to close DSWindows. This can be used to control DSWindows via DSAgenda. For example, it can be called from the last Macro scheduled or, more simply, run as the last scheduled job to ensure that DSWindows is closed down and, more importantly, that the session it was using is freed.

```
CloseDSWindows()
```

Example 34: EX_900CO.MAC

This macro demonstrates the use of continuous 900A transmission and compressed 900B transmission by adding the text "//C" to the end of each request. It also uses the default recovery macro, recover.mac, to attempt recovery.

NOTE *Both SEND commands are complete Datastram requests and so are fully recoverable. See 'Potential problems with the SEND Command' in the '**How to make your macros more robust**' section of this User Guide.*

```
LOGERRORSTOFILE ("recover.log")

RECOVERUSING "recover.mac":

STARTDC (CSVFILE, "ex_900co.csv")

SEND("900A FT30;NAME, MNEM, BDATE, INDG,P;1/1/97//C ")

SEND("900B @MSFT(P#S),-1Y,,D//C ")

ENDDC ()
```

Example 35: EX_CONN.MAC

This macro demonstrates how the connect command can be used to connect via alternate comms mechanisms.

```
; The following connects using the default comms mechanism :

Connect()

; The following connects using a named gateway which is not
; necessarily your default gateway. This assumes that your
; default comms mechanism is DSGATE 2 or DSGATE 3 :

Connect("MyOtherGateway")

; Further, DSGate 3 users can specify a queue name as follows :

Connect("MyOtherGateway,MyQueue")

; The following connects using a session configuration that
; was previously setup using the DSWindows menu option
; Options->Configure->Communications. Note that this
; technique may not be portable to other computers, since
; thier sessions may be configured differently :

Connect("SESSION2")

; The following connects using a named device type. These
; strings, describing the device, can be found in your
; dsadp.ini file (in the Windows directory).

Connect("DEVICE=XTEC,S,COM1,9600,E,7,1,X,4000,2666,1333,C,D")
```

Appendix A

Converting control files into macros

Introduction

This appendix gives information on how to convert existing DSCOM/DSTERM Control files into macros which you can use in DSWindows.

A conversion program is supplied with the DSWindows software which converts control files to DSWindows macros. You can also use the conversion program to convert list files into data files, in a format which can be used in a DSWindows **OPENDATA** command.

Most control file commands are translated automatically, but there may be some commands (for example, those regarding printer configuration and graphics display) for which there is no equivalent DSWindows command. We recommend that, after converting a control file, you test the new DSWindows macro. If it does not work as you intended you can then edit it in the Notepad, or another suitable editor. Comments in the converted file indicate commands which could not be converted.

The conversion program

Introduction

The conversion program is a DOS program called CONVERT.EXE which is stored in the DSWINDOW directory. It enables you to:

- Convert a single control file
- Convert a single list file
- Convert more than one control file
- Convert more than one list file
- Convert all control files in a directory
- Convert all list files in a directory

It also provides options which enable you to:

- Control the comments which are added to the macro
- Override extensions other than .LST
- Automatically overwrite existing macro files
- Automatically append converted list files to macros of the same name
- Save different types of data:
  - Data channel
  - Graphics
  - .TXT (plain text)
  - .DST (text for use in DSWindows)
- Print out a help screen

When you run the program, the resulting macro files have a .MAC extension, and you can specify the directory you want to store them in (if it is different from the source file directory).

Syntax

The syntax of the command is as follows:

convert *[filespec[destpath][options]]*

filespec the directory path and filename of the file(s) you want to convert. If you omit the filespec, you are prompted to supply it.

destpath the destination directory path, in which the converted macro file will be stored. It must appear after **filespec** in a convert command.

If you do not specify a destination, the destination file is stored in the same directory as the source file, with a **.mac** extension.

You can specify the destination path name with or without a terminating "****", for example, both:

C:\DSWINDOW\FILES and C:\DSWINDOW\FILES\ are acceptable.

options may be in any order, and you can write them in a command between the **filespec** and the **destpath**.

NOTE *We recommend that you convert your control files before you convert any list files associated with them (and which have the same name), because you can append converted list files to the corresponding macros. If you do not append list files to macros of the same name, you risk overwriting a macro with data, if the automatic overwrite option is specified.*

♦ **To convert a single control file**

❑ At the prompt, type: **convert path\filename.ctl destpath**.

Example **convert \dscom\dsfiles\search75.ctl c:\DSWINDOW\FILES**

This command converts the control file "search75.ctl" into a macro which will be stored in the \DSWINDOW\FILES directory as "search75.mac".

If you omit the filename extension, .ctl is added automatically.

♦ **To convert a single list file**

❑ At the prompt, type: **convert path\filename.lst destpath**.

Example **convert \dscom\dsfiles\grtest.lst c:\DSWINDOW\FILES**

This command converts the list file "grtest.lst" into a data file in the \DSWINDOW\FILES directory. If a macro file already exists with this name, than you are asked whether you want to append the data to the macro.

You must type in the extension (.lst), or the file will be assumed to be a control file. Alternatively, use the -L option (see page 185).

♦ **To convert more than one control file**

❑ Use the wildcard character (*****) to specify a number of files to be converted at once.

Example **convert \dscom\dsfiles\search*.ctl**

This command converts all control files beginning with the characters "search". In this example, no destination path is specified; the files will be stored as .mac files in the dscom\dsfiles directory.

♦ **To convert more than one list file**

❑ Use the wildcard character (*****) to specify a number of list files to be converted at once.

Example **convert 75code*.lst**

This command converts all list files beginning with the characters "75code" in the current directory. If macro files already exist with the same names, then you are prompted to append the data to the corresponding macros.

♦ **To convert all control files**

❑ Use the wildcard character (*****) to convert all the control files in the directory.

Example **convert *.ctl**

This command converts all control files in the current directory, into macro files in the same directory.

♦ **To convert all list files**

❑ Use the wildcard character (*****) to convert all the list files in the directory.

Example **convert *.lst**

This command converts all list files in the current directory, into macro files in the same directory (unless you append them to converted control files).

♦ **To control the comments which are added to the macro**

When you convert a control or list file, comments may be added to the translated macro. Such comments are inserted to point out, for example, that the command

cannot be translated or is not supported by DSWindows. You can also have the original code inserted into the macro in the form of comments.

The option **-Cn** (where n is 0, 1 or 2) controls the comments which are added to the final macro:

0	gives, in addition to comments, the original code. These comments appear in the macro before the code into which it has been translated.
1	gives comments but does not insert the original code. This is the default.
2	suppresses all comments, apart from a header (which is always added).

Example To translate the file search75.ctl with comments and original code, type:

convert search75.ctl -C0

♦ **To override extensions other than .lst**

❑ Use the option **-L** to convert files which do not have a .LST extension. The file extension is overridden, and the file is translated into a macro or a data file. You can convert one file (or a number of files at once) into a data file.

Example **convert * -L**

This command converts all the files in the current directory into data files (as if they were list files).

♦ **To overwrite existing files automatically**

❑ Use this option to overwrite files automatically. Normally the conversion program prompts you if the target macro file already exists and you must select whether to overwrite the file or not. If you include the overwrite option (**-O**) in a convert command, files will be overwritten without prompting.

Example **convert *.ctl -O**

This command converts all the control files in the current directory into macros, automatically overwriting any existing macros with the same names, without prompting.

♦ To append a list file to a macro automatically

- Use this option to append a list file to a macro of the same name. Normally the conversion program prompts you if a macro file of the same name exists, and you must select whether or not to append the list file to that macro. If you include the append option (**-A**) in a convert command, the data is appended automatically to the macro.

Example **convert *.lst -A**

This command converts all the list files in the current directory into data lists, automatically appended to the macros with which they share a name.

♦ To save different types of data

- Use this option to specify the type of data which is to be saved, when you convert a control file containing a @S or @C command. Four flags enable you to differentiate different types of data:
  - **-D** Data channel
  - **-G** Graphics
  - **-T** .TXT (plain text)
  - **-S** .DST (text for use in DSWindows)

Example **convert *.ctl -D**

This command converts all the control files in the current directory into macro files, with all @S commands converted to STARTDC and all @C commands converted to ENDDC.

The following table shows the effect of using the different flags:

Flag	Data Type	@S converts to:	@C converts to:
-D	Data Channel	STARTDC	ENDDC
-G	Graphics	AUTOSAVE	ENDAUTOSAVE
-T	Plain text	CAPTURE	ENDCAPTURE
-S	Text in .DST format	CAPTURE	ENDCAPTURE

- If you do not specify any flags on the command line, the **-T** and **-G** options take effect.
- If you want just the plain text option, specify **-T**.
- If you want just the graphics option, specify **-G**.

♦ To display a help screen

- Use this option to display a help screen about CONVERT and the options available.

Example **convert -?** or **convert -H**

♦ To print a help screen

Press the Print Screen key, or type the command, **convert -H>PRN:**

INDEX

&ATOUTPUT 113
&ATPROMPT 112
&CONNECTSTATE 94, 109
&DAYOFMONTH 95
&DAYOFWEEK 95
&ENDOFDATA 93
&MONTH 95
&RECOVERYATTEMPTS 113
&RESULT 96
&SCREEN 93
&SENDCOMPLETE 94
&TEXTFOUND 94
&YEAR 95

A

ACTIVATEBACKPAGES 76
ACTIVATEGRAPHICS 76
ACTIVATESAVEFILES 76
ACTIVATETERMINAL 76
WAIT 114
ADDTOSELECTITEMS 57
ALLOWDUPLICATETIMESERIES 73
APPEND 90
Arranging windows
  macro commands 76
AUTOPAGE 26
AUTOPRINT 25
AUTOSAVE 43

B

Basic (prog. language) 2
BEEP 86

C

CALL 27
CAPTURE 74
CHANGEITEMS 61
CLIPBOARD 91
CLOSEBACKPAGES 76
CLOSEDSWINDOWS 76
CLOSEGRAPHICS 76
CLOSESAVEFILES 76
CLOSETERMINAL 76
Commands 2
COMMENT 36
CONFIGUREDC 70
CONNECT 78
CONNECTED 88
Connecting to Datastream 19, 21, 78, 171
  via UK PSS 21, 172
  via SITA 131
CONNECTNOQUEUE 79
CONNECTNOWAIT 79
Constants 2, 87
ConstTimeSeries 72
Control files
  conversion 181
Conversion from
  DSCOM/DSTERM 181
CONVERT.EXE 181
Converting strings 102
Coordinates system 53
COPYITEMS 60
Creating macros 8, 9
CSVFILE 90

D

DATA 32
Data Channel 71, 126, 127
DELETEITEMS 60
DEMO900.MAC 126
DEMOGLST.MAC 125
DEMOGRPH.MAC 127
DEMOLIST.MAC 127
DEMOPSS.MAC 129
DEMOSAVE.MAC 128
DEMOSITA.MAC 131
DESELECTGRAPH 51
DESELECTITEMS 58
DISCONNECT 82
DISPLAYLAYOUT 41
DISPLAYSINGLEGRAPH 41
DSAGENDA 14
DSCOM
  converting from 181
DSTERM
  converting from 181

E

END 31
ENDALLMACROS 83, 115
ENDAUTOSAVE 44
ENDCAPTURE 75
ENDDATA 33
ENDDC 72
ENDPAGE 26
ENDPRINT 25
EX_300C.MAC 153
EX_401X.MAC 142
EX_900CO.MAC 179

EX_CLIP.MAC 165
EX_CLOSE.MAC 179
EX_CONN.MAC 180
EX_CSV.MAC 163
EX_DATEF.MAC 177
EX_EXCEL.MAC 167
EX_GANT.MAC 150
EX_GANT1.MAC 160
EX_GLIST.MAC 144
EX_GSTYL.MAC 147
EX_LIST.MAC 136
EX_MASTR.MAC 141
EX_PRNT.MAC 134
EX_PROMT.MAC 178
EX_PSS.MAC 172
EX_RECOVER.MAC 176
EX_SAVE.MAC 135
EX_SET.MAC 138
EX_STRNG.MAC 139
EX_SYSTM.MAC 140
EX_TIMES.MAC 176
EX_WRITE.MAC 174
EXPORTGRAPHICS 44
Expressions 2, 105
Extracting strings 99

F

FALSE 88
Functions 2, 100
  asc 104
  chr$ 103
  convert strings and integers 99, 102
  left$ 100
  len 100
  manipulating strings 99
  mid$ 100
  str$ 103
  val 103

G

GOTO 30
GRAPHDRAWOFF 69
GRAPHDRAWON 69
Graphics annotations 52
Graphics macro commands 41
GRAPHPAGESETUP 47

H

Help
  CONVERT.EXE 187
  online xiii
  online help xiv
  telephone support xiv
  training xiv

I

IF 28
INPUT 33
Instr 101
Instructions 2, 4
integers 99

L

Labels 2, 4
line breaks (\) 15
List files
  conversion 181
LISTFILE 91
LOADFILLSTYLES 49
LOADGRAPHFILE 46
LOADLAYOUT 48
LOADLAYOUTFILE 46
LOADLINESTYLES 48
LOADTEXTSTYLES 49
LOGERRORSTOFILE 83, 109
Logical operators 105
LOGON 80
LOGON.MAC 171

M

Macro recorder 2, 9
Macros 2 - 7
Mathematical operators 105
MAXIMIZEBACKPAGES 77
MAXIMIZEDSWINDOWS 77
MAXIMIZEGRAPHICS 77
MAXIMIZESAVEFILES 77
MAXIMIZETERMINAL 77
MESSAGE 86
MINIMIZEBACKPAGES 76
MINIMIZEDSWINDOWS 76
MINIMIZEGRAPHICS 76
MINIMIZESAVEFILES 76
MINIMIZETERMINAL 76
MOVEITEMS 59

N

NEWBOX 63
NEWLINE 68
NEWRECT 65
NEWTEXT 67
NOT_CONNECTED 88

O

ONERROR 85
OPENDATA 32
OPENSAVEFILE 75
OVERWRITE 90

P

Paging 26
PRINTGRAPHFILE 45
PRINTGRAPHICS 44
Printing 25
PRINTLAYOUTFILE 45
PRINTSAVEFILE 25
SEND 116

Q

QUERY 90
QUEUING 89
Quotes 53

R

RECOVER.MAC 107-124
RECOVERSTOP 83, 115
RECOVERUSING 82, 117
RECOVERYATTEMPTS 96
REFINESELECTITEMS 58
RESTOREBACKPAGES 77
RESTOREDSWINDOWS 77
RESTOREGRAPHICS 77
RESTORESAVEFILES 77
RESTORETERMINAL 77
rules 15
  summary of general macro rules 15

S

SAVEGRAPHICS 42
SAVEWMF 43
Saving 74
Scheduling a macro 11-14
SELECTGRAPH 51
SELECTITEMS 55
self-reporting 111
SENDANDCHECK 24
Sending data 19
SET...TO 39
SETDATEEXPORTFORMAT 86
SETGLOBAL 40
SETGRAPHNAME 50
SHOW_MAX 91
SHOW_MIN 91
SHOW_NORMAL 91
Simple examples 3
Special keys
  macros 19
STARTDC 71
STARTPROGRAM 27
STARTUP.MAC 170
strings 99
System variables 93

T

TEXTFOUND 89
TIMEOUT 89
Tips 16
TRACECAPTION 111
TRUE 88

U

UNLOCK 89
USERINPUT 37

V

Variables 2, 92

W

WAIT 38
Windows
  maximising windows 77
WRITETOFILE 84
recovery 107