



SCRIPT

Mathematical Formula Formatter

User's Guide

S544-3306-00





SCRIPT

Mathematical Formula Formatter

User's Guide

S544-3306-00



This publication was produced using
IBM Publishing Systems BookMaster (5688-015)
under the **IBM Document Composition Facility** (5748-XX9), Release 3.2
with the **SCRIPT Mathematical Formula Formatter Feature**

The body of the publication is set in **Monotype Times New Roman** (5771-AAR).
Headings are set in **Helvetica** (5771-AAK).
Formula examples use **Characters for Math and Science** (5771-ACW).

The masters for printing were produced on the **4250/II Electro Compositor** using **electroNEG**.

Monotype and Monotype Times New Roman are registered trademarks of The Monotype Corporation, plc.
Helvetica is a registered trademark of Linotype A G or its subsidiaries.

First Edition (June 1989)

This edition applies to the **SCRIPT Mathematical Formula Formatter (SMFF)** feature of the **IBM Document Composition Facility (DCF)**, Release 3.2, Licensed Program 5748-XX9, which operates with VM/CMS and MVS/TSO systems. This edition applies to any subsequent release of this feature until otherwise indicated in new editions or technical newsletters.

Changes are made periodically to the information herein. Before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, 4300, and 9370 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's licensed program may be used. Any functionally equivalent program may be used instead.

Publications are stocked at the address given below; requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation
General Products Division
P. O. Box 1900
Department 588
Boulder, Colorado, U.S.A. 80301-9191.

IBM may use or distribute any of the information that you supply in any way that it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information that you supply.

© **Copyright International Business Machines Corporation 1984, 1987, 1989. All rights reserved.**
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Preface

This publication introduces the SCRIPT Mathematical Formula Formatter (SMFF) feature. SMFF is an optional feature of the IBM Document Composition Facility (DCF), Release 3.2, licensed program (Program Number 5748-XX9). To run with DCF Release 3.2, you must have the SCRIPT Mathematical Formula Formatter Feature at level 3.2, which will not run with earlier releases of DCF.

The Formula Formatter replaces the “reserved” SCRIPT/VS .EQ EQUATION“ control word with an “active” version of the same control word. The active .EQ control word provides the underlying interface to the *formula formatter language*, which provides keywords for describing mathematical and scientific formulas, and formats them for “all points addressable printers” supported by DCF Release 3.2. If a document that contains formulas is formatted for a line printer, the formulas are not formatted, but they are still checked for correct syntax.

A “Formula Formatter Starter Set” of GML tags is provided with the feature; it can be installed as an extension to the DCF starter set or it can be used by other DCF applications or source documents. The IBM Publishing Systems BookMaster product, (Program Number 5688-015), incorporates a SCRIPT Mathematical Formula Formatter tag set. These tag sets generate the .EQ and other appropriate DCF control words to control equation position and alignment, and to control whether the equation appears “inline” with text or in “display” form in a separate white space.

This book contains information about how to mark up formulas with the Formula Formatter GML tags and how to write the Formula Formatter language. It shows examples of all formula formatter functions, and describes the full function of the .EQ EQUATION“ control word.

Summary of changes

The description of the .EQ EQUATION“ control word has been moved from the *Program Description/ Operations Manual* of the old SCRIPT Mathematical Formula Formatter Program Offering to this book. This section has also been expanded to present all functions of the control word.

There are two program directories for SMFF; one is for a VM/CMS environment and another is for a MVS/TSO environment. The *Program Description/ Operations Manual* (PDOM) is being replaced by the program directories.

Major topics included in the directories are: documentation, machine readable material, program support, service classification, machine configurations, prerequisites, programming considerations, and installation procedures.

Publication Library Guide for the Document Composition Facility

The following table lists the Document Composition Facility publications by number as they relate to user tasks. **“DCF Publications” on page v lists the titles and the order numbers that correspond to the numbers listed in the table.**

Number	User Tasks	Typical Audience	Brief Description
(1) (2) (3) (20)	Planning and introducing DCF/DLF	Users, system planners	Provide a general overview of text processing, library facility, and available books.
(3) (4) (5) (12) (16) (20) (23)	Formatting documents (using the GML starter set)	Novice to experienced end users	Provide an introduction to the Generalized Markup Language (GML) starter set and describes the GML starter set tags and SCRIPT/VS messages.
(6)	Creating bar codes with DCF/GML	Experienced end users	Provides information about using GML to create bar codes.
(9) (10) (11) (12) (17) (18) (19) (22)	Formatting documents (using SCRIPT/VS control words)	Knowledgeable to experienced end users	Describe the function and use of all SCRIPT/VS control words, macros, diagnostic aids, and the formatting features and messages.
(14) (15)	Converting RFTDCA for SCRIPT/VS formatting	Novice to experienced RFTDCA users	Describe the function and use of the optional Office Document Feature, including diagnostic aids and messages.
(4) (5) (7) (9) (10) (11) (19) (22)	Modifying GML starter set ¹	Document administrator and text programmer ²	Contain material about GML starter set tags, SCRIPT/VS control words, and GML starter set modifications.
(4) (5) (7) (8) (9) (10) (11) (16) (19) (22)	Creating GML application processing functions	Document administrator and text programmer ²	Provide information about designing your own GML and about GML concepts, GML starter set tags, SCRIPT/VS control words, and usage guidelines.
(10) (11) (12) (13) (19) (21) (22)	Installing, modifying, and maintaining DCF	Systems programmer	Give information on error isolation, program tailoring, and use of SCRIPT/VS.
(23)	Creating mathematical formulas with SMFF	Experienced end users	Describes the function and use of the SCRIPT Mathematical Formula Formatter (SMFF), including .EQ control word and messages.

¹ Central Programming Service support and maintenance are provided *only* on the unmodified GML applications shipped with DCF. If you modify any of these GML applications shipped with DCF, it is recommended that you also maintain an *unmodified* copy for diagnostic purposes.

² The document administrator is responsible for defining markup conventions and procedures for an organization. The text programmer implements application processing functions (APFs) that provide the processing specified by the document administrator.

DCF Publications

Number	Titles and Order Numbers
(1)	<i>Document Composition Facility and Document Library Facility Executive Overview and Product Summary</i> , GX20-2332.
(2)	<i>Document Composition Facility and Document Library Facility General Information</i> , GH20-9158.
(3)	<i>Document Composition Facility: Introduction to Generalized Markup Language</i> , G544-3192.
(4)	<i>Document Composition Facility: Generalized Markup Language Starter Set User's Guide</i> , SH20-9186.
(5)	<i>Document Composition Facility: Generalized Markup Language Starter Set Reference</i> , SH20-9187.
(6)	<i>Document Composition Facility: Bar Code User's Guide</i> , S544-3115.
(7)	<i>Document Composition Facility: Generalized Markup Language Starter Set Implementation Guide</i> , SH35-0050.
(8)	<i>Document Composition Facility: Generalized Markup Language Starter Set Concepts and Design Guide</i> , SH20-9188.
(9)	<i>Document Composition Facility: SCRIPT/VS User's Guide</i> , S544-3191.
(10)	<i>Document Composition Facility: SCRIPT/VS Text Programmer's Guide</i> , SH35-0069.
(11)	<i>Document Composition Facility: SCRIPT/VS Language Reference</i> , SH35-0070.
(12)	<i>Document Composition Facility SCRIPT/VS Messages</i> , SH35-0048.
(13)	<i>Document Composition Facility: Diagnosis Guide and Reference</i> , LV32-0523.
(14)	<i>Document Composition Facility: Office Document Feature User's Guide</i> , G544-3129.
(15)	<i>Document Composition Facility: Office Document Feature Reference</i> , S544-3130.
(16)	<i>Using the Document Composition Facility</i> , SR21-0515 (Training Course 32291).
(17)	<i>Using DCF with the 4250 Printer</i> , SR20-8486 (Training Course 32908).
(18)	<i>Using DCF with Page Printers</i> , SR21-1211 (Training Course 32243).
(19)	<i>Document Composition Facility—Release 3 (SCRIPT/VS) for Document Administrators and Text Programmers</i> , SR20-7525 (Training Course).
(20)	<i>Document Composition Facility (SCRIPT/VS) Student Text</i> , SC20-1894 (Training Course).
(21)	DCF Program Directory.
(22)	<i>Document Composition Facility: TSO Enhancements Update Guide</i> , G544-3345.
(23)	<i>Document Composition Facility: SCRIPT Mathematical Formula Formatter User's Guide</i> , S544-3306.

Note: The following reference booklets and a three-ring binder are also available:

Document Composition Facility: GML Starter Set Quick Reference, SX26-3723.

Document Composition Facility: SCRIPT/VS Text Programmer's Quick Reference, SX26-3719.

Document Composition Facility Post-Processor Examples, S544-3484.

Document Composition Facility SCRIPT/VS, SH35-0086 (binder).

Contents

Introduction	1
Markup to Identify a Formula	3
The GML F Tag	3
The GML DF Tag	4
The GML DFDEF Tag	4
Attributes of :DF and :DFDEF	5
Formula Formatter Language Description	9
General Rules	9
Example Notation	9
Inline and Display Formulas	9
Keywords	10
Symbols	11
Blanks	12
Literal Text	12
Comments	12
Grouping	13
Operators	14
The &q0.Operator&eq0. Keyword	15
Ellipses	15
Subscripts and Superscripts	16
Piles	17
Alignment Variants	17
Vertical Spacing Variant	18
Fractions	19
Square Roots	20
Summation, Integral, and so On	20
Big Parentheses, Brackets, and so On	22
Embellished Characters	23
Overlays	23
Equation Numbers	24
Positioning Adjustments	24
Mathematical Functions	26
The &q0.Function&eq0. Keyword	26
Greek Letters	27
Special Characters	28
Creating Named Formula Symbols	29
Defining Symbols like \diamond with .NAMEF	30
Partial Formulas	31
Usage Notes for .NAMEF	32
The .EQ Control Word	33
GML Tags for Formulas	34
Utility Control Words	34
.EQ Control Functions	35
.EQ ?	35
.EQ SET	36
.EQ SYMBOL	39

.EQ TRACE	41
Appendix A. Formula Formatter Keywords	45
Special Break-Character Keywords	45
Regular Keywords	46
Appendix B. Required Formula Formatter Symbols	57
Appendix C. Other Symbols	59
Appendix D. Error Messages	61
Messages	62
Appendix E. Formula Formatter Examples	63
Example 1	64
Input for Example 1	64
Example 2	64
Input for Example 2	64
Example 3	65
Input for Example 3	65
Example 4	67
Input for Example 4	68
Appendix F. Programming Considerations	69
Tailoring the SMFF Feature for your Installation	69
Tailoring the SMFF Feature Profile	69
Index	73

Figures

1.	Attributes on :DF and :DFDEF	5
2.	Keywords for Operators	14
3.	Keywords for Mathematical Functions	26
4.	Keywords for Greek Letters	27
5.	Keywords for Special Characters	28
6.	The NAMEF Macro	29
7.	The .EQ Control Word	33
8.	The .EQ Control Functions	35
9.	The .EQ SET Control Function	36
10.	The .EQ SYMBOL Control Function	39
11.	The .EQ TRACE Control Function	41
12.	Special Break-Character Keywords	45
13.	Regular Keywords	46
14.	Required Formula Formatter Symbols	57
15.	Other Symbols Defined in DSMSPROF	59

Introduction

The Formula Formatter runs as a feature of Document Composition Facility (DCF) Release 3.2. It provides for “typesetting” of mathematical and scientific formulas. The Formula Formatter is designed to be easy to use for people who know neither mathematics nor typography.

Normal mathematical notation is the most efficient way to represent complex equations. But most SCRIPT/VS users use terminals that lack mathematical characters.

The structure of a formula is *described* in keywords, which can be spelled out fully for ease of reading, or abbreviated to minimize keystrokes. Mathematical operators and special characters also have symbolic names. Thus, a mathematical expression is described in words, roughly in the same way one would read it over the telephone.

In this sense, the Formula Formatter is perhaps more akin to SCRIPT/VS’s descriptive Generalized Markup Language (GML) than to the more procedural SCRIPT/VS formatting language.

The Formula Formatter does not check the mathematical validity of an expression, and indeed, will compose mathematically incorrect formulas (but they will be aesthetically pleasing).

SMFF will produce formulas for the 4250, 38PP, 3820, and PostScript physical devices. For other devices, if the SMFF code is installed, the formula input will be checked for errors but the formula will not be produced. Rather, the formula input will be printed “as is” in the output. If the formula has any syntax errors, the SMFF messages may help diagnose the problem.

The SMFF profile, DSMFPROF, defines fonts for the math symbols used when creating formulas. These symbols can be specified by keywords on the :DF (Display Formula) and :F (Formula) tags.

The fonts shown in the table below are used by SMFF. For the 4250 and 3820 fonts, an error message is issued by DSMFPROF if the fonts for the requested device are not present. For PostScript,³ the Symbol Font will be used if Lucida Math is not available.

Physical Device	Required Fonts
4250	Characters for Math & Science, 5771-ACW Typewriter and PI Fonts, 5771-AAW ⁴
3820/38PP	Mathematics and Science, 5771-ADT Pi and Specials, 5771-ABC ⁴
PostScript	Lucida Math [™]

When formatting for a 3800-3, you should use the 3820 font library if it is available at your installation. This is because the formula formatter sometimes produces undesirable results when using the 3800-3 fonts since there is not sufficient information in the fonts to place characters correctly. To use the 3820 fonts, specify the following options on the SCRIPT command:

```
DEVICE (38PPxxxx) FONTLIB (FONT3820) CHARS (X0T00395)
```

³ PostScript is a registered trade mark of Adobe Systems Incorporated.

⁴ These fonts are already required by the GML Starter Set.

[™] Lucida Math is the registered trademark of Bigelow and Holmes.

Markup to Identify a Formula

Input to the Formula Formatter can be identified with the GML tags, :DF and :F, which are provided with the Formula Formatter Starter Set. The :DFDEF. tag provides a way of declaring defaults that govern how formulas are formatted.

IBM Publishing Systems BookMaster, 5688-015, provides the same tags. So if you are using either the Formula Formatter Starter Set or BookMaster, you can mark up your formulas as described here. Your local installation may provide other tags that also invoke the Formula Formatter.

The formula GML tags identify where a formula starts and ends. The actual content of the formula is composed of the *formula language*. It makes no difference to the Formula Formatter whether it was invoked via the :F. tag or via the :DF. tag. The Formula Formatter language is the same in either case.

Normal SCRIPT/VS control words and GML tags are *not* valid input to the Formula Formatter, except for the ending control word or GML tag that signals the end of the formula. However, the formula is formatted according to the current environment. Such environmental values as the formatting mode, the current font, and the current line spacing, all affect the final appearance of the formula.

Once control passes to the Formula Formatter, it makes no difference how many lines of formula input you use to describe your formula. Spaces (blanks) and line ends are thrown away in the final output.

The GML F Tag

The GML :F tag identifies a formula. The formula is formatted as the next word to be placed in the current output line. Formulas that are marked up with the :F tag, and which are formatted in a line of ordinary text, are sometimes referred to as *inline* formulas.

The format of the :F tag is:

:F.*formula formatter input*:EF.

The tag and the end-tag do *not* have to be on the same line of your file. You can have as many lines of Formula Formatter input as you wish in between.

For example, this input:

The formula :f.x sup 2:ef. is formatted
as an :hpl.inline:ehpl. formula.

Is formatted as:

The formula x^2 is formatted as an *inline* formula.

The GML DF Tag

The GML `:DF` tag identifies a *display* formula. A display formula is one that is formatted on a separate line, apart from the surrounding body text. The syntax of the tag is as follows:

$$\begin{array}{l} :DF \left[\begin{array}{l} attribute-1= 'value-1' \\ \dots \\ attribute-n= 'value-n' \end{array} \right] .formula\ formatter\ input \\ :EDF. \end{array}$$

The attributes that may be specified on the tag are described in “Attributes of `:DF` and `:DFDEF`” on page 5.

ALWAYS END THE MARKUP WITH A PERIOD! Ordinarily, the *content* that follows GML markup does not look like GML markup, so if you forget the “markup content separator” `SCRIPT/VS` can be forgiving. But Formula Formatter input is **much more likely to be mistaken for a GML attribute** if you do not clearly end the markup. For example, in the formula input

`a = sum from i=0 to infinity of j sub i`

the starting “a = sum” could be mistaken for a syntactically valid GML attribute if you leave off the period, and this would make the remaining input syntactically *incorrect* for the Formula Formatter.

The GML DFDEF Tag

The GML `:DFDEF` tag defines default values that govern how formulas will be formatted. Any formula will be formatted according to all relevant defaults, unless the default is temporarily overridden for that formula. The `:DFDEF` tag does not have any content. It has only attributes, and these are identical to the attributes you can specify on the `:DF` tag. The syntax of the tag is as follows:

$$:DFDEF \left[\begin{array}{l} attribute-1= 'value-1' \\ \dots \\ attribute-n= 'value-n' \end{array} \right] .$$

The meanings of the values of all attributes are the same as those of the `:DF` tag, and they are described in “Attributes of `:DF` and `:DFDEF`” on page 5. Any `:DF` tag that does not specify a particular attribute will use the “global” default, unless that has been reset with the `:DFDEF` tag.

For example, if you specify
`:dfdef align=center.`

once, at the beginning of your document, then all :DF tags will default to center alignment, unless overridden with the ALIGN attribute. If an alignment is specified for a particular :DF tag, then that alignment will override the default for that formula only.

In BookMaster, the :DFDEF tag can be used only in the prolog of the document when it has no ID. In BookMaster, the DFDEF tag has an ID attribute, and the DF tag has a REFID attribute. These attributes serve to make the BookMaster DFDEF operate the same way other BookMaster definition tags work. For more information on how to use definition tags, see the *BookMaster User's Guide*.

Attributes of :DF and :DFDEF

The attributes that can be specified on the :DF and :DFDEF tags are as follows:

$\left[\begin{array}{c} \text{ITALIC} \\ \text{NOITALIC} \end{array} \right]$		$\left[\text{FRAME} = \left\{ \begin{array}{c} \text{NONE} \\ \text{RULE} \\ \text{BOX} \end{array} \right\} \right]$	
$\left[\text{ALIGN} = \left\{ \begin{array}{c} \text{LEFT} \\ \text{CENTER} \\ \text{RIGHT} \\ \text{INSIDE} \\ \text{OUTSIDE} \\ [-]h \end{array} \right\} \right]$			
$\left[\text{METRICS} = \left[\begin{array}{cc} \text{blank} & c \\ \text{fclr} & factor \\ \text{fbar} & factor \\ \text{fpos} & factor \\ \text{gap} & h \\ \text{impos} & factor \\ \text{imsize} & factor \\ \text{kerning} & factor \\ \text{maxsize} & pointsize \end{array} \right] \right]$		$\left[\begin{array}{cc} \text{mbar} & factor \\ \text{minsize} & pointsize \\ \text{mpos} & factor \\ \text{msize} & factor \\ \text{sbar} & factor \\ \text{sclr} & factor \\ \text{space} & v \\ \text{vbar} & factor \\ \text{vspace} & v \end{array} \right] \right]$	

Figure 1. Attributes on :DF and :DFDEF

where

ITALIC and NOITALIC

are value attributes. NOITALIC specifies that no italic font will be used in this formula. The normal default is ITALIC, which causes letters (normally representing variables) to be set in italic.

FRAME=

specifies what kind of frame, if any, should be drawn around the formula. The meanings of the values of FRAME are the same as those of the FRAME attribute of the FIG tag in the DCF Starter Set.

ALIGN=

specifies how the display formula is to be aligned. If the alignment is specified as h in any valid space units, then the formula is indented the specified distance from the *left* margin. If $-h$ is specified, then the formula is indented the specified amount from the *right* margin.

The indention may be specified in picas and points, millimeters, inches, or any other valid space units.

METRICS=

specifies values that control how the Formula Formatter itself builds a formula.

The value of this attribute consists of a string enclosed in single quotes in the form:

keyword1 *value1* . . . **keywordn** *valuen* . . .

You may specify more than one **METRICS=** attribute if you need to specify more keyword-value pairs than will fit conveniently in one.

Valid keywords for the **METRICS** are as follows. The keywords may be abbreviated to the minimum truncation shown in **bold** type.

blank *c*

Specifies the character to be used as the formula blank character. Each occurrence of this character in a formula produces one “thin space” (about a quarter of an em-space) of white space in the formatted output.

“*c*” is any single character. The default character is “%.”

flr *factor*

Specifies the amount of “clearance” above and below a fraction bar as a *factor* of the current font’s point size. This is the amount of white space between the bottom of the numerator and the top edge of the fraction bar, and between the top of the denominator and the bottom edge of the fraction bar.

The initial factor is about .20 (2 points clearance for a 10-point font).

fbar *factor*

Specifies the thickness of fraction bars as a *factor* of the current font’s point size.

The initial factor is 0.07.

fpos *factor*

Specifies the vertical position of the fraction bar above the baseline, as a *factor* of the current font’s point size.

The initial factor is dependent on the device for which the document is being formatted, and is about 0.2.

gap *h*

Specifies the gap between an equation number and the formatted equation, so they don’t bump into each other.

“*h*” may be any horizontal space units. That is, it may be specified in picas and points, inches, millimeters, cicerós and didot points, or unqualified horizontal units. The initial value is 4, which is equivalent to the width of “0000” in the default font.

impos *factor*

Specifies the vertical position of superior “marks” in inline formulas, as a *factor* of the current font’s point size. “Mark” is a general term for superscripts, subscripts, plex limits, and so forth.

The initial factor is 0.3.

imsize *factor*

Specifies the size of smaller fonts used for “marks” in inline formulas, as a *factor* of the current font’s point size. “Mark” is a general term for superscripts, subscripts, plex limits, and so forth.

The initial factor is 0.6.

kerning *factor*

Specifies the kerning adjustment used in horizontal placement of superscripts and subscripts. Superscripts and subscripts placed on an italic base expression are adjusted horizontally (kerned), and the amount of adjustment is a fixed *factor* of the vertical offset of the superscript or subscript above or below a “zero-kern” baseline. (If the base expression is not italic, no kerning is needed.)

The *factor* is a measure of how slanty the italic letters are. The initial factor is 0.16.

maxsize *pointsize*

specifies the maximum point size the Formula Formatter may use. The initial value is 0, meaning “no maximum,” and the Formula Formatter may use the largest font it can find in the current font library.

mbar *factor*

Specifies the thickness of “mark bars” (used for the **BAR** and **BARUNDER** keywords), as a *factor* of the current font’s point size.

The initial factor is 0.05.

minsize *pointsize*

Specifies the minimum point size the Formula Formatter may use. The initial value is 0, meaning the Formula Formatter may use the smallest font it can find in the current font library.

mpos *factor*

Specifies the vertical position of superior “marks” in display formulas, as a *factor* of the current font’s point size. “Mark” is a general term for superscripts, subscripts, plex limits, and so forth.

The initial factor is 0.5.

msize *factor*

Specifies the size of smaller fonts used for “marks” in display formulas, as a *factor* of the current font’s point size. “Mark” is a general term for superscripts, subscripts, plex limits, and so forth.

The initial factor is 0.8.

sbar *factor*

Specifies the thickness of **SQRT** bar as a *factor* of the current font’s point size.

The initial factor is 0.06.

sclr *factor*

Specifies the amount of “clearance” below a square root bar produced by the **SQRT** keyword. as a *factor* of the current font’s point size.

The initial factor is about .15 (1.5 points clearance for a 10-point font).

space *v*

Specifies the amount of vertical space left above and below display formulas.

“*v*” may be any vertical space units. That is, it may be specified in ems, picas and points, inches, millimeters, cicerós and didot points, or unqualified vertical units.

The initial value is 0.5, which is equivalent to half of a line space in the current font with the current linespacing.

vbar *factor*

Specifies the thickness of the vertical bar part of *constructed* left and right brackets, as a *factor* of the current font’s point size.

The initial factor is 0.1.

Left and right brackets are constructed for the **LB** and **RB** keywords. The horizontal “tails” on the brackets are made half the thickness of the vertical part.

Note: The thickness of vertical bars constructed by the **VBAR** keyword is the same as the thickness of fraction bars.

vspace *v*

Specifies the amount of vertical space left between piled elements separated by the **VABOVE** keyword.

“*v*” may be any vertical space units. That is, it may be specified in ems, picas and points, inches, millimeters, cicerós and didot points, or unqualified vertical units.

The initial value is 0.25, which is equivalent to one quarter of a line space in the current font with the current linespacing.

Formula Formatter Language Description

General Rules

Example Notation

In this section, Formula Formatter input is identified in examples with this symbol:

>>>

This symbol represents the markup that identifies formula input to SCRIPT/VS. For the purposes of these examples, it makes no difference whether the Formula Formatter input is identified with the :F [FORMULA] tag, the :DF [DISPLAY FORMULA] tag, or some other tag that your installation has provided.

For example, the formula input

>>> x sup 2

might actually be specified in any of the following forms:

1. :f.x sup 2:ef.
2. :f.x
sup 2:ef.
3. :df.x sup 2:edf.
4. :df align=center.
x sup 2
:edf.

and so on.

All display examples in this chapter are centered, and have been marked up for formatting with

```
:df align=center.  
(input)  
:edf.
```

Inline and Display Formulas

The Formula Formatter formats display formulas and inline formulas differently. In general, the Formula Formatter tries to adjust the sizes of certain display characters to fit comfortably with the associated expression, and it tries to keep an inline formula as tight as possible. For example, superscripts and subscripts are smaller and closer to the middle of the base expression in an inline formula than in a display formula. The inline formula $\sqrt{x^i}$ is smaller than the centered display formula

$$\sqrt{x^i}$$

And the inline formula x_3^2 uses smaller superscripts and subscripts than its display equivalent

$$x_3^2$$

1. The `:DF` tag always produces a display formula.
2. The `:F` tag generally produces an inline formula, but it can produce a display formula if the resulting formula is formatted by itself on a centered or unformatted line.
3. A formula that has an equation number (the `EQNO` keyword) is always formatted as a display, even if it was marked up with the `:F` tag.
4. A *named* formula defined with the `NAMEF` macro, and used as a `SCRIPT/VS` symbol in regular text is formatted the same as formula marked up with the `:F` tag. It is generally an inline formula, but can be a display if it is used in unformatted or centered text.

The Formula Formatter input language is the same, regardless of which of the two tags were used to identify the formula or whether it is formatted as inline or display.

Keywords

The Formula Formatter uses keywords to describe Greek letters, mathematical operators, functions, and other elements of an equation. Appendix A, “Formula Formatter Keywords” on page 45 summarizes all Formula Formatter keywords.

All keywords except those for Greek letters are case-insensitive. That is, you can enter the `SUP` keyword in any of the following forms:

`sup` `suP` `sUp` `sUP` `Sup` `SuP` `SUP` `SUP`

The first letter of keywords for Greek letters is case-sensitive, and determines the case of the Greek letter printed. That is, the keyword `gamma` starts with a lowercase `g`, and produces a lowercase gamma (γ), while the keyword `Gamma`, with a capital `G`, produces a capital gamma (Γ). It makes no difference whether you type letters other than the first as capital or lowercase letters. See “Greek Letters” on page 27 for a list of all Greek letter keywords.

Regular keywords may be abbreviated by entering them with a leading period. Thus, you can type the `INTEGRAL` keyword in any of the following forms:

`integral` `.int` `.inte` `.integ`
`.integr` `.integra` `.integral`

Most keywords must be delimited with blanks in the input. There are some “special break-character” keywords that do not need blanks to delimit them. A complete list of these appears in the section “Special Break-Character Keywords” on page 45. Most of these keywords also have “spelled-out” equivalents in case your terminal lacks some of the special characters.

The input

```
>>> alpha + beta over gamma
```

produces

$$\alpha + \frac{\beta}{\gamma}$$

Notice that the numerator is just “ β ” because the blank before the input “beta” made it a separate “word.” But the input

```
>>> alpha+beta over gamma
```

produces

$$\frac{\alpha + \beta}{\gamma}$$

In this case, the special keyword “+” was recognized, and it also separated “alpha” from “beta,” allowing all three keywords to be recognized without blanks separating them. Because there were no blanks in the string “alpha+beta,” the whole thing was treated as one “word,” and this word was used as the numerator of the fraction.

Symbols

Many Formula Formatter keywords cause special characters such as Σ and f to be formatted in the output. These special characters come from several different fonts in the font library. Actually, the Formula Formatter knows nothing about where these characters reside in the font library. Keywords that produce special characters actually produce only a *symbol name* for the required character. The symbol name must be correctly defined in order to get the correct output.

The formula

```
>>> sum from i=0 of theta sub i
```

produces:

$$\sum_{i=0} \theta_i$$

But it only produces this output if the required symbols are correctly defined. If these symbols were undefined, the same input would produce:

$$\textit{sum}_{i=0}^{theta_i}$$

All of the required symbol definitions are provided in the Formula Formatter Starter Set profile and in BookMaster — so if you are using either of these, you need not worry about having the symbols you need.

You can define other characters, and then use the symbol name within a formula as if it were a keyword. You cannot abbreviate your own symbol names, however — the name must be spelled out fully.

In ordinary SCRIPT/VS usage, a symbol is substituted with a leading ampersand (&). Inside the Formula Formatter, do not use the ampersand. The Formula Formatter automatically substitutes ordinary strings, such as replacing “theta” with its value, θ . It also substitutes the names of *named formulas*. For information on how to define named formulas, see “Creating Named Formula Symbols” on page 29.

Blanks

Ordinary blanks in the input are thrown away, and do not appear in a formatted formula. You can cause a little extra white space to be added by using the character defined as the *formula blank*. Each blank character produces a “thin space” in the output. This character is by default the percent sign (%), but you can change it in the METRICS GML attribute. See “Attributes of :DF and :DFDEF” on page 5 for information on how to do this.

Literal Text

Alphabetic letters in a mathematical expression are set in italic, normally representing variables.

Certain operators and bracket-like characters have special meaning in parsing a formula.

Blanks in formula input have no significance to the output, and are removed.

You can cause any of these cases to be treated as literal text by entering a string as a *quoted string*. A string enclosed in single quote marks (') or double quote marks (") is taken as a single string of literal text. Blanks in a quoted string are preserved, and formatted as ordinary word spaces. No symbol names or keywords are recognized inside a quoted string.

A quoted string that is delimited with single quote marks (') is set in the normal body font that is in effect at the time the formula is processed.

A quoted string delimited with double quote marks (") is set in a **bold font**, if the body font has a bold variation, and otherwise in the normal body font.

Comments

The regular SCRIPT/VS comment delimiter “.” is also a Formula Formatter keyword, designating a comment. In normal SCRIPT/VS input, the “.” must appear at the beginning of a logical input line, but this is not true inside the Formula Formatter. The “.” keyword may appear at the beginning or a line or anywhere else on an input line. It causes the rest of that input line to be ignored.

Grouping

Normally, the Formula Formatter recognizes the end of a particular input string because it ends in a blank. If you want to cause several words to be treated a single string, or if you want to force the Formula Formatter to treat any complex expression as a single unit, you can bracket the expression in *grouping brackets*.

The “<” character is the normal *begin* grouping bracket, and the “>” character is the normal *end* grouping bracket. The characters “<” and “>” are sometimes called *metabrackets*.

Left and right parentheses, brackets, braces, and metabrackets, all act as grouping brackets. That is, any “left” character — ([{ < — acts as a *begin* grouping bracket. Any “right” character —)] } > — acts as an *end* grouping bracket.

These left and right characters are really special keywords which are identical in function to the spelled-out keywords **LPAREN**, **LBRACKET**, **LBRACE**, **RPAREN**, **RBRACKET**, and **RBRACE**.

The regular grouping brackets < and > (metabrackets) are not printed in the output. Parentheses, braces, and brackets are printed. The begin and end are outside the bracketed expression. The notation:

`(x + 1)`

and the notation

`lparen x + 1 rparen`

are both the same as:

`< '(' x + 1 ')' >` or: `< (x + 1) >`

A begin and end do not have to “match.” That is, you can start a bracketed expression with a left parenthesis and end it with a right brace. If you want a left parenthesis to appear merely as a text character without implying the begin of a bracketed expression, you can do this in two ways:

1. Make it a quoted string, as in `' ('`.
2. Close the left parenthesis with a right nonprinting metabacket, as in `(>`.

The **LEFT** and **RIGHT** keywords bracket a left-right expression in their own right. Any left or right character that immediately follows the **LEFT** or **RIGHT** keyword is treated merely as the character that should be selected from a large enough font to enclose the entire left-right expression, and does not in this case have any grouping bracket flavor.

Thus,

`left (x right A`

is a balanced expression. There is no need to close the left parenthesis.

Operators

Most arithmetic operators are formatted with some white space on either side. Thus,

```
>>> x+y==p-q
```

is formatted as

$$x + y \equiv p - q$$

Plus and minus signs, entered with the `+` and `-` keywords (as opposed to the **PLUS** and **MINUS** keywords) need not have any blanks surrounding them. If one of these keywords has a blank to its *left* and not to its *right*, then it is recognized as a unary plus or minus sign, and does not receive the usual operator padding space.

The input

```
>>> y = a + -b
```

gives

$$y = a + -b$$

The Formula Formatter has the following operator keywords:

Keyword	gives:	Keyword	gives:	Keyword	gives:
<code>+</code>	$+$	<code>GT</code>	$>$	<code>MP</code>	\mp
<code>*</code>	\times	<code>IDENTICAL</code>	\equiv	<code>NE</code>	\neq
<code>-</code>	$-$	<code>INTERSECTION</code>	\cap	<code>PLUS</code>	$+$
<code>=</code>	$=$	<code>JOIN</code>	\cup	<code>PM</code>	\pm
<code>==</code>	\equiv	<code>LARROW</code>	\leftarrow	<code>PROPORTIONAL</code>	\propto
<code>BULLET</code>		<code>LE</code>	\leq	<code>RARROW</code>	\rightarrow
<code>CONGRUENT</code>	\cong	<code>LOR</code>	$ $	<code>SIMILAR</code>	\sim
<code>DIVIDE</code>	\div	<code>LT</code>	$<$	<code>SUCHTHAT</code>	\ni
<code>ELLIPSIS</code>	\dots	<code>MEET</code>	\cap	<code>TIMES</code>	\times
<code>EQUALS</code>	$=$	<code>MEMBEROF</code>	\in	<code>UNION</code>	\cup
<code>GE</code>	\geq	<code>MINUS</code>	$-$		

Figure 2. Keywords for Operators

The `&q0.Operator&eq0`. Keyword

The `OPERATOR` keyword causes any character (or grouped expression, for that matter) to be formatted as an arithmetic operator.

```
>>> z theta 2 %%% z operator theta 2
```

produces

$$z\theta 2 \quad z \theta 2$$

Ellipses

The `ELLIPSIS` keyword produces “three dots” in your formula:

```
>>> (x sub 1 , ellipsis , x sub n)
```

is formatted as

$$(x_1, \dots, x_n)$$

Sometimes, the ellipsis should not be placed on the formula’s baseline, but it should be raised. The Formula Formatter does this automatically when the ellipsis follows a regular operator (but not when it follows the `OPERATOR` keyword):

```
>>> x sub 1 + ellipsis + x sub n
```

produces

$$x_1 + \cdots + x_n$$

Subscripts and Superscripts

Identify superscripts and subscripts with the **SUP** and **SUB** keywords:

```
>>> x sup 2 + y sub k
```

gives

$$x^2 + y_k$$

The keywords **SUP** and **SUB** may be entered in capital or lowercase letters. The **SUP** keyword has a synonym of “**”:

```
>>> x**2 + y sub k
```

also gives

$$x^2 + y_k$$

The **PRESUP** and **PRESUB** keywords are variants of **SUP** and **SUB**, and they place the superscript and subscript *before* the base expression:

```
>>> x presup 2 presub 3
```

is

$${}_3^2x$$

The **MIDSUP** and **MIDSUB** variants place the superscript and subscript directly over and under the base expression. A single expression can have any combination of **SUP**, **SUB**, **PRESUP**, **PRESUB**, **MIDSUP**, and **MIDSUB**. Thus,

```
>>> a 'HHH' sub 1 sup 2 midsup 3
      presup 4 presub 5 midsub 6 c
```

Produces:

$$a_5^4 {}_6^3 \text{HHH}_1^2 c$$

Notice that all of the superscripts and subscripts apply to the previous *base expression*, not the previous word. In the preceding example, all superscripts and subscripts applied to “HHH.” If you want a superscript on a superscript, you must use grouping brackets:

```
>>> x sup <2 sup <alpha + beta>>
```

produces

$$x^{2^{\alpha+\beta}}$$

Piles

To pile one thing on top of another, use the **ABOVE** keyword. You can pile as many elements on top of each other as you wish, using the **ABOVE** keyword to separate the pieces.

```
>>> 1 - a above 'BBBBBB' above <i omega t>
```

is

$$\begin{array}{c} a \\ 1 - \text{BBBBBB} \\ i\omega t \end{array}$$

The vertical spacing between the elements is the nominal line spacing in effect for ordinary text. The entire pile is vertically centered in the formula.

Alignment Variants

By default, each element is horizontally centered in the width of the pile, which is determined by the widest element. The **ABOVE** keyword has alignment variants to change this. The **LABOVE**, **CABOVE**, and **RABOVE** variants specify left alignment, centering (the same as **ABOVE**), and right alignment, respectively.

To left align the same pile, use

```
>>> 1 - a labove 'BBBBBB' labove <i omega t>
```

to get

$$\begin{array}{c} a \\ 1 - \text{BBBBBB} \\ i\omega t \end{array}$$

In addition, the **HABOVE** variant aligns two elements at an arbitrary point which you identify with the **HERE** keyword. “Here” alignment is useful in aligning elements on a particular character, such as an equal sign. For example,

```
>>> <x here = y sup 2> habove <here = gamma sub j>
```

produces

$$\begin{array}{c} x = y^2 \\ = \gamma_j \end{array}$$

Each **ABOVE** keyword or variant controls how the two elements on either side of it are aligned with respect to each other. A single pile can have one alignment between one pair of elements, and a different alignment between another pair. Thus,

```
>>> aaaaa rabove b labove ccccc
      rabove d labove eeeee rabove f labove ggggg
```

will produce

$$\begin{array}{c} aaaaa \\ b \\ ccccc \\ d \\ eeeee \\ f \\ ggggg \end{array}$$

Vertical Spacing Variant

The vertical spacing of piled elements is normally fixed, and it is the same as the current nominal line spacing for ordinary text. The “V” variant on any **ABOVE** keyword causes *variable* line spacing. For example,

```
>>> <integral of theta> above <sum from i=0 to infinity>
```

is formatted with fixed line spacing, without regard to the height of the elements, as

$$\int_{\theta} \sum_{i=0}^{\infty}$$

This looks poor, since the actual height of the two elements is not taken into account in positioning them. With the **VABOVE** keyword, the piled height of the elements determines the vertical spacing.

```
>>> <integral of theta> vabove <sum from i=0 to infinity>
```

is formatted with variable line spacing as

$$\int_{\theta} \sum_{i=0}^{\infty}$$

You may combine the “V” variant with any of the horizontal alignment variations. That is, for a left-aligned, variably-spaced pile, use **LVABOVE**. For a here-aligned, variably-spaced pile, use **HVABOVE**.

Thus, the **ABOVE** keyword with all of its variants could be written as **ABOVE**, **LABOVE**, **CABOVE**, **RABOVE**, **HABOVE**; or **VABOVE**, **LVABOVE**, **CVABOVE**, **RVABOVE**, **HVABOVE**.

Fractions

To draw a fraction, use the keyword **OVER**:

```
>>> a+b over 2c = 1
```

gives

$$\frac{a+b}{2c} = 1$$

The fraction bar is made the right length and positioned automatically. The numerator and denominator are positioned according to their actual sizes:

```
>>> y = x over x    x over b
      p over x    p over b
```

produces

$$y = \frac{x}{x} \frac{x}{b} \frac{p}{x} \frac{p}{b}$$

Here also, grouping brackets can be used:

```
>>> <a over b + beta>over<sin(x)>
```

is

$$\frac{\frac{a}{b} + \beta}{\sin(x)}$$

OVER is of a lower precedence than **ABOVE** or **SUP** and **SUB**, so

```
>>> -b sup 2 over omega sub k
```

needs no brackets to be unambiguously

$$\frac{-b^2}{\omega_k}$$

The numerator and denominator are normally centered on the fraction bar. The **LOVER** and **ROVER** variants of the **OVER** keyword specify that the fraction should be left aligned or right aligned. For symmetry, there is also a **COVER** variant for centered alignment, but it acts the same as the default for **OVER**.

```
>>> y = <a + b sup 2> rover 2c
```

is formatted as:

$$y = \frac{a + b^2}{2c}$$

Fractions group to the left. That is,

```
>>> 0 + a over b over c + 0
```

is equivalent to

```
>>> 0 + <a over b> over c + 0
```

and produces

$$0 + \frac{\frac{a}{b}}{c} + 0$$

Square Roots

To draw a square root, use the **SQRT** keyword:

```
>>> sqrt a sub i + sqrt<ax**2 + bx + c>
```

is

$$\sqrt{a_i} + \sqrt{ax^2 + bx + c}$$

Square roots group to the right. That is,

```
>>> sqrt sqrt sqrt sqrt sqrt 'Hi Mom'
```

is equivalent to

```
>>> sqrt <sqrt <sqrt <sqrt <sqrt 'Hi Mom'>>>>
```

and produces

$$\sqrt{\sqrt{\sqrt{\sqrt{\sqrt{\text{Hi Mom}}}}}}$$

Summation, Integral, and so On

Summations, integrals, and similar constructions are identified in the general form of

```
>>> sum from i=0 to infinity of x sub i
```

which produces

$$\sum_{i=0}^{\infty} x_i$$

The Formula Formatter refers to all of these constructions as *plexes*. The keywords **SUM**, **PRODUCT**, and **INTEGRAL** all produce similar constructions.


```
>>> sum from i=0 to n of J sub i %%%
      product from j=0 to n of P sub j %%%
      integral from 0 to n of f(x) dx
```

produces

$$\sum_{i=0}^n J_i \prod_{j=0}^n P_j \int_0^n f(x) dx$$

In addition to these keywords, the general **PLEX** keyword identifies constructions that are formatted like a summation, and the **INTPLEX** keyword identifies a slightly different form used for integrals. With these, you can use any character as the “base” character:

```
>>> plex <X(n)> from <n rarrow infinity> %%%
      intplex 'H' from 0 to 1
```

is

$$\lim_{n \rightarrow \infty} X(n) H_0^1$$

Notice the use of grouping brackets in the “plex” and in the “from” above to cause an expression or a string that contains blanks to be treated as a single “word.”

Within an expression described by the **PLEX** and **INTPLEX** keywords, the subsidiary keywords **FROM**, **TO**, **OF**, and **WITH** are allowed. The **FROM** and **TO** parts (limits) are both optional, there can be multiple limits, and they can appear in any order. You must enclose the limits in grouping brackets if they contain any blanks. It is never wrong to do so.

The expression which undergoes summation, integration, etc., can optionally be indicated by **OF**, and must appear last. The size of the plex character is based on the size of the **OF** expression, so if you do not use the **OF** keyword, the Formula Formatter has nothing upon which to calculate the size of the plex character. In this case, it assumes that the plex character should be sized to fit the base font.

If you want multiple sums that all apply to a single expression, you can use the **WITH** keyword between them. The “of expression” that belongs to the last sum is then used to determine the size of all of them. If a plex expression has **WITH**, it cannot also have **OF**.

```
>>> sum from i=0 from j=0 to n of x sub <i j> %%%
      sum from i=0 to n with
      sum from j=0 to n of <a sup 2 over x sub <i j>>
```

produces

$$\sum_{i=0}^n \sum_{j=0}^n x_{ij} \sum_{i=0}^n \sum_{j=0}^n \frac{a^2}{x_{ij}}$$

Big Parentheses, Brackets, and so On

The **LEFT** and **RIGHT** keywords identify characters that are made big enough to enclose the expression inside. The sized characters are often characters like parentheses and brackets that have in themselves the property of grouping brackets.

The **LEFT** and **RIGHT** keywords bracket a left-right expression in their own right. Any left or right character that immediately follows the **LEFT** or **RIGHT** keyword is treated merely as the character to be sized to fit the left-right expression, and does not in this case have any grouping bracket flavor.

```
>>> left lbrace a over b right rbrace =  
      left lparen c over d right rparen +  
      left lbracket e sub i right rbracket
```

is

$$\left\{ \frac{a}{b} \right\} = \left(\frac{c}{d} \right) + [e_i]$$

And

```
>>> A = left lbracket aaa rabove b rabove ccc  
      % = above = above = %  
      27 labove 1888 labove alpha right rbracket
```

produces

$$A = \left[\begin{array}{l} aaa = 27 \\ b = 1888 \\ ccc = \alpha \end{array} \right]$$

In addition to these characters, which are selected from available fonts, the Formula Formatter can *construct* vertical bars and brackets of any required height. The **VBAR**, **LB**, and **RB** keywords are used for this.

```
>>> left lb  
      p over b + 1  
      right rb %%%  
      left vbar  
      'You' above 'can' above 'pile'  
      above 'this' above 'as' above 'high'  
      above 'as' above 'you' above 'wish!'  
      right vbar
```

is

$$\left[\frac{p}{b} + 1 \right]$$

You
can
pile
this
as
high
as
you
wish!

Embellished Characters

The keywords **DOT**, **DOTDOT**, **DOTUNDER**, **DOTDOTUNDER**, **HAT**, **PRIME**, **TILDE**, and **VECTOR** all produce embellishments on characters.

In addition to these, the **BAR** and **BARUNDER** keywords draw a horizontal rule over or under the base expression.

```
>>> x dot + X dot + y hat + y dotdot + z dotunder
      + z dotdotunder + x+y bar + alpha-beta barunder
      + z tilde + z bar dotdot + x prime - x vector
```

gives

$$\dot{x} + \dot{X} + \hat{y} + \ddot{y} + \tilde{z} + \bar{z} + \overline{x+y} + \underline{\alpha} - \underline{\beta} + \tilde{z} + \ddot{z} + x' - \vec{x}$$

The **OVERMARK** and **UNDERMARK** keywords are general keywords that place *any* character over or under the base expression. The **PRIMEMARK** keyword places *any* character next to the base expression.

```
>>> M overmark , + H undermark j + 86 primemark degree
```

is

$$\dot{M} + \underset{j}{H} + 86^\circ$$

Overlays

The **OVERLAY** keyword causes one base expression to be centered and overlaid on another. The **LOVERLAY** and **ROVERLAY** variations cause the left or right sides, respectively, of the two expressions to be aligned.

```
>>> bullet loverlay lor %%%
      bullet overlay lor %%%
      bullet roverlay lor
```

produces

| | |

Equation Numbers

The **EQNO** keyword causes the following expression to be formatted as a right-aligned equation number. The **REQNO** keyword is equivalent to **EQNO**. The **LEQNO** keyword formats the following expression as a left-aligned equation number.

An equation number forces the formula to be formatted as a display formula.

```
>>> eqno lbracket 1.2 rbracket  
      x identical sqrt x sup 2
```

produces

$$x \equiv \sqrt{x^2} \quad [1.2]$$

And

```
>>> legno lbracket 25 rbracket  
      x identical sqrt x sup 2
```

is formatted as

$$[25] \quad x \equiv \sqrt{x^2}$$

Positioning Adjustments

You can make small adjustments in the placement of things with the **ADJUST** keyword. The adjustment applies to the “word” that follows the **ADJUST** keyword and its values, but you can use grouping brackets to treat virtually any expression as a single entity.

With the **ADJUST** keyword, you can move something **UP**, **DOWN**, **LEFT**, or **RIGHT**. The general syntax is:

```
adjust ( up n left n ...)
```

All specifications to **ADJUST** must be enclosed in parentheses, and all must appear on the same input line. You must specify at least one direction of movement, but you need not specify any directions that do not apply. Directions may be specified in any order, and the direction keywords **UP**, **DOWN**, **LEFT**, and **RIGHT** may be truncated to **U**, **D**, **L**, and **R**.

That is, **L**, **LE**, **LEF**, and **LEFT** all mean to move the expression to the left.

For each direction, you must specify how much it is to be moved (“n”). The values of “n” are “adjustment units” that are equivalent to one tenth of the point size of the font used to format the adjusted entity. The following are valid adjustments:

```
adjust (up 2 right 3)  
adjust (down 1.5 left 2.7)
```

The actual adjustment is proportional to the size of the expression that is moved.

The input

```
>>> y = midsup adjust(u 2 r 10.5) 3 sqrt x sup 2
```

produces

$$y = \sqrt[3]{x^2}$$

Mathematical Functions

Mathematical function names such as “sin” and “log” are usually typeset in a roman typeface rather than italics. They also generally have some white space surrounding them. For example:

```
>>> sin x cos x + sin sup 2 theta
      + tan (2 theta) cot sup 2 (4 theta)
```

Produces

$$\sin x \cos x + \sin^2 \theta + \tan(2\theta) \cot^2(4\theta)$$

The Formula Formatter recognizes the following mathematical functions as keywords, and formats them this way:

ARG	EXP	MIN
COS	GLB	MOD
COSH	LIM	SEC
COT	LN	SIN
CSC	LOG	SINH
CTN	LUB	TAN
DET	MAX	TANH
DIM		

Figure 3. Keywords for Mathematical Functions

Note: The Formula Formatter formerly recognized, but has now dropped, the following function keywords: antilog, arc, colog, hom, and ker. If you need to use any of these words as mathematical functions, you must either use the **FUNCTION** keyword (see “The &q0.Function&eq0. Keyword”) or define a *named formula* (see “Creating Named Formula Symbols” on page 29).

The &q0.Function&eq0. Keyword

If you want to invent a new mathematical function and have it formatted like the functions listed above, use the **FUNCTION** keyword. For example,

```
>>> function foo x function bar x + sin sup 2 theta
```

Produces

$$\text{foo } x \text{ bar } x + \sin^2 \theta$$

Notice that “bar” is taken simply as a function name here, even though that word is also a regular Formula Formatter keyword. The **FUNCTION** keyword has the following rules:

1. The next word in the input is taken as the function name.
2. A function name must be given, and it must be on the same line of formula input as the **FUNCTION** keyword.
3. The function name must not be more than 12 letters long.

Greek Letters

Formula Formatter keywords are provided for the Greek letters. All other keywords are case-insensitive, but in the names of Greek letters, the first letter of the keyword is case-sensitive.

If you enter the name of a Greek letter with an initial capital letter, you will get the capital Greek letter. That is, the forms `alpha`, `aLpHa`, `alPHA`, and so forth, all refer to a lowercase alpha (α). The forms `Alpha`, `ALPHA`, `AlPhA`, and so on, all refer to a capital Greek alpha (A).

Keyword	symbol	Keyword	symbol
alpha, Alpha	α , A	nu, Nu	ν , N
beta, Beta	β , B	xi, Xi	ξ , Ξ
gamma, Gamma	γ , Γ	omicron, Omicron	o , O
delta, Delta	δ , Δ	pi, Pi	π , Π
epsilon, Epsilon	ϵ , E	rho, Rho	ρ , P
zeta, Zeta	ζ , Z	sigma, Sigma	σ , Σ
eta, Eta	η , H	tau, Tau	τ , T
theta, Theta	θ , Θ	upsilon, Upsilon	υ , Υ
iota, Iota	ι , I	phi, Phi	ϕ , Φ
kappa, Kappa	κ , K	chi, Chi	χ , X
lambda, Lambda	λ , Λ	psi, Psi	ψ , Ψ
mu, Mu	μ , M	omega, Omega	ω , Ω

Figure 4. Keywords for Greek Letters

Note: Some of the Greek capital letters look just like capital letters in the Roman alphabet. The Greek letters used for mathematics have “upright” capitals and “inclined” lowercase letters. A capital “A” in a mathematical formula would ordinarily be set in italic, while a capital Alpha would be “upright.” Thus, you can tell the difference between a capital “A” (A) and a capital Alpha (A) in a formula.

Special Characters

The Formula Formatter provides keywords that really do nothing except produce special characters. Each special character is defined as a SCRIPT/VS symbol with the same name as the keyword. All of the necessary symbols to support these keywords are defined as part of the Formula Formatter Starter Set profile, DSMFPROF.

Any symbol that does not have a keyword can still be used in a formula, but there are two advantages to having a keyword:

1. The keyword is case-insensitive. For example, the keyword **ALEPH** will produce the “aleph” character (\aleph), regardless of whether it is written in a formula as `aleph`, `Aleph`, `ALEPH`, or any other variation.
2. The keyword can be abbreviated. For example, the keyword **HAMILTONIAN** can be written as `.ham`, `.hami`, `.hamil`, `.hamilt`, `.hamilto`, `.hamilton`, `.hamiltoni`, `.hamiltonia`, `.hamiltonian`, or `hamiltonian`.

A symbol that does not have a keyword must be typed exactly as the symbol name has been defined. (Remember, inside a formula, do not prefix the symbol with `&`, as you do outside a formula.)

The Formula Formatter recognizes the special-character keywords:

Keyword	gives:	Keyword	gives:	Keyword	gives:
ALEPH	\aleph	DEL	∇	INFINITY	∞
ALL	\forall	EXISTS	\exists	LNOT	\neg
ANGSTROM	\AA	HAMILTONIAN	\mathcal{H}	PARTIAL	∂
APP	\approx	HYPHEN	-	SCRIPTL	ℓ
DEGREE	$^\circ$				

Figure 5. Keywords for Special Characters

Creating Named Formula Symbols

The **NAMEF** (Named Formula) macro allows you to create a special kind of **SCRIPT/VS** symbol that contains a formula or a part of a formula. When the symbol is used, the Formula Formatter interprets the contents of the symbol. You can create symbols that contain formulas and use them in any **GML** element — or inside another formula.

The **NAMEF** macro has three operands, **SYMBOL**, **F**, and **FPART**. These operands are specified just like labelled attributes on **GML** tags — that is, they take the form

LABEL = *value*

and the *value* must be enclosed in single quotes if it contains blanks or special characters. A **NAMEF** macro with all of its operands *must appear in a single input line*. You cannot stack the operands on multiple lines. The form of the macro is:

.NAMEF **SYMBOL** = *name* [**F** = '*formula*'
FPART = '*partial-formula*']

Figure 6. The **NAMEF** Macro

where

SYMBOL= *name*

identifies the name of the symbol (just the name, without any preceding & or trailing period) you want to create. It can be from 2 to 10 characters long, with no blanks or special characters; the first character must be a letter. This operand is required.

F= '*formula*'

identifies a formula to be assigned to the symbol. This formula will automatically be treated as though it had grouping brackets around it. Therefore, it must be a complete expression in its own right — but you can use it inside another formula almost as though it were a keyword, and be assured that it will be treated as a single entity.

FPART= '*partial-formula*'

identifies a fragment of a formula. A partial formula is *not* treated as though it had grouping brackets around it. It may be a partial expression that would produce errors if used alone, but which is valid in the context in which you use it.

If you do not specify either **F**= or **FPART**=, then the symbol is assigned the null string, and it will be invisible if used. If you specify both, the **F** specification is used, and the **FPART** specification is thrown away.

Defining Symbols like \diamond with .NAMEF

The Formula Formatter Starter Set and BookMaster both define a symbol called “diam” which will be used in examples of the use of the **NAMEF** macro. Whenever you want a diamond *outside of a formula*, you can type the name of the symbol as &diam., and this is substituted and formatted as \diamond .

Within a formula, you should never use the & or the period. Just use the *name* of the symbol, almost as though it were a keyword. Using the **OVERLAY** keyword, you could define a formula with

```
>>> diam overlay adjust (up 2) '.'
```

and it would be formatted as

\diamond

This same formula can be assigned to a symbol called “diamdot” with:

```
.namef symbol=diamdot f='diam overlay .adj (u 2) ''.'
```

Notice that within the quoted string representing the value of **F=**

1. no & or period was used for the “diam” symbol, since this is “inside a formula.”
2. the single quotes around the period were doubled. Single quotes that appear within a **NAMEF** quoted string must be doubled.

Now that the symbol is defined, you can use it the same way you would use any other symbol. If you want to get a “diamdot” symbol anywhere outside of a formula, you type it as &diamdot., which will then produce \diamond . The headline on this section was typed as:

```
:h3.Defining symbols like &diamdot. with .NAMEF
```

If you want to use the “diamdot” symbol *inside* a formula, then you do not use the & and the period. Also, since the symbol has implied grouping brackets, you can use it as a single keyword inside a formula:

```
>>> diamdot over diamdot %%% diamdot sup 2
```

produces

$\frac{\diamond}{\diamond} \diamond^2$

Again, this same formula could be used as the definition for another symbol. Let's define one called “dbldiam”:

```
.namef symbol=dbldiam f='diamdot over diamdot'
```

Now, anywhere *outside* of a formula, you can get the “dbldiam” symbol by typing &dbldiam., which produces $\frac{\diamond}{\diamond}$ —anywhere it is used.

As you can see, you can build up complex formula symbols from simpler ones, and define virtual keywords for frequently-used formula sequences.

Partial Formulas

The **FPART=** form is used when the formula to be defined is not a complete expression, but relies on its context. For example, you could define a symbol named “squared” with

```
.namef symbol=squared fpart='sup 2'
```

The string “sup 2” is not a complete expression, and if you used the “squared” symbol alone, the Formula Formatter would complain.

You can use it inside a formula as:

```
>>> x squared
```

which would produce

$$x^2$$

You can also use other symbols in conjunction with the **NAMEF** macro. For example, you might have a symbol named “small3” to produce a small number 3. This symbol could be defined with the **SCRIPT/VS .DV** control word as follows:

```
.DV small3 font mathp /3  
.namef symbol=cuberoo fpart='.adj(u 4 r 5) small3 sqrt'
```

Or if you have BookMaster, you could use the BookMaster **NAMEIT** macro to define it. In either case, the small3 can be used inside a formula to define a symbol for “cuberoo” as follows:

```
.nameit symbol=small3 size=7 text='3'  
.namef symbol=cuberoo fpart='.adj(u 4 r 5) small3 sqrt'
```

Notice that the symbol defined with **NAMEIT** (or with **.DV**) does not take an & or a period when used in a formula!

Now the “squared” and “cuberoo” symbols we have defined for both the **.DV** control word or the **NAMEIT** macro can be used as virtual keywords in a formula. The formula input

```
>>> rho = cuberoo gamma squared
```

is formatted as

$$\rho = \sqrt[3]{\gamma^2}$$

Usage Notes for .NAMEF

1. Be sure that your symbol does not use itself! If you define a symbol that uses itself, the Formula Formatter will detect that infinite recursion would result from continuously trying to resolve the symbols, and it will terminate SCRIPT/VS with a severe error message.

For example, this definition would cause a loop error:

```
.namef symbol=foo f='1 over foo'
```

A symbol does not have to use itself directly to cause a loop error. If you define two symbols that refer to each other, that too creates a loop.

2. An exception to the loop rule is the **FUNCTION** keyword. The **FUNCTION** keyword causes the next word to be taken as is, and not interpreted as a keyword or as the name of a symbol. Thus, you can safely define new function keywords such as:

```
.namef symbol='ker' f='function ker'  
.namef symbol='inf' f='function inf'
```

3. You cannot use the **NAMEF** macro to redefine a Formula Formatter keyword. For example, there is a **SUP** keyword, and if you define a symbol called “sup” it still will not take precedence over the built-in keyword. An occurrence of the word “sup” in a formula will be interpreted as the **SUP** keyword, not as a reference to the “sup” symbol.

The .EQ Control Word

The .EQ EQUATION“ control word identifies to SCRIPT/VS where an equation begins and ends, and it performs Formula Formatter control functions. The first time any .EQ control word is processed, the Formula Formatter is initialized. As part of initialization, the Formula Formatter defines the symbol &\$EQ to contain the level number of the Formula Formatter.

The form of the .EQ EQUATION“ control word is:

A diagram showing the structure of the .EQ control word. It consists of the text ".EQ" followed by a vertical list of options enclosed in large square brackets. The options are: "1" (with a horizontal line under it), "n" (with a horizontal line under it), "ON", "OFF", "TAG", and "line" (in italics).
$$.EQ \left[\begin{array}{c} \underline{1} \\ \underline{n} \\ \text{ON} \\ \text{OFF} \\ \text{TAG} \\ \textit{line} \end{array} \right]$$

Figure 7. The .EQ Control Word

A single number (*n*) indicates how many input lines will be processed as a formula, and the default is 1.

.EQ ON starts an open-ended formula, where all lines are taken as Formula Formatter input until .EQ OFF ends it.

.EQ TAG is valid only when executed within the processing macro for a GML start tag. It starts an open-ended formula, just like .EQ ON, except that the formula is ended by the end-tag that corresponds to the start tag whose processing macro issued the .EQ TAG control word. No other tags are processed. Any other GML markup within the scope of the .EQ is treated merely as a string, and will be formatted into the formula. .EQ OFF will always end a formula, even if it was started with .EQ TAG.

The *line* in the single-line form of .EQ EQUATION“ can either be a line of formula input, that is, an equation, or it can be one of the .EQ control functions.

It makes no difference how many input lines are used to describe your formula. Spaces (blanks) and line ends are thrown away in the final output. Thus, the following notations are all equivalent:

.eq x=y+z

.eq 2
x = y
+ z

.eq on
x = y
+ z
.eq off

All of these produce: x=y+z

GML Tags for Formulas

GML tags are provided for describing mathematical formulas. All of the content of a formula GML tag is in the formula language, and is processed directly by the Formula Formatter. The processing functions (macros) to implement these tags can set up any suitable formatting environment using standard SCRIPT control words, and then do a .EQ TAG. The formula will be built according to the current alignment (left, centered, right, inside, or outside), and according to the current indentation, current font, and so forth.

The Formula Formatter treats a formula as an “inline” formula in a context, such as a flowing paragraph, where input lines flow and spill, known in DCF as .FO ON. A formula is formatted as a “display” formula if input lines are not flowed (.FO OFF or .NF). The processing macro for a display formula GML tag does a .NF control word before doing the .EQ TAG control word, to cause the formula to be formatted as a display.

The .EQ TAG control word works in a way similar to the .WF TAG control word, allowing all input until the matching end-tag to be processed by the Formula Formatter.

Utility Control Words

The .EQ EQUATION“ control word primarily processes the formula language. There are also three utility functions and a comment delimiter that are recognized by the .EQ control word, but are not part of the formula language. Ordinarily, line endings in the input file are not important to the Formula Formatter, except that keywords in the language may not span input lines. The exceptions are:

1. The special keyword .* is a comment delimiter. It causes itself and the rest of the input line on which it occurs to be thrown away.
2. The utility functions SYMBOL and SET cause the remainder of the input line on which they occur to be taken and processed as conventional SCRIPT/VS control words. They are not part of the Formula Formatter language.

It is recommended that the utility functions be coded in the “one-line” form of the .EQ control words. That is, this form is not recommended:

```
.eq on
symbol fred /x sup 2
.eq off
```

Instead, it is more efficient to code the one-line form as:

```
.eq symbol fred /x sup 2
```

.EQ Control Functions

The *line* in the single-line form of .EQ EQUATION“ can either be a line of formula input, that is, an equation, or it can be one of the .EQ *control functions*:

$$\text{.EQ} \left[\begin{array}{l} ? \\ \text{SET } \textit{set-parameters} \\ \text{SYMBOL } \textit{symbol-parameters} \\ \text{TRACE } \textit{trace-parameters} \end{array} \right]$$

Figure 8. The .EQ Control Functions

These functions are not part of the Formula input language, and they are never parsed as part of an actual formula. Instead, the control functions, while under the .EQ control word, operate as though they were entirely separate control words in their own right.

Actually, the keywords SET, SYMBOL, and TRACE are recognized even if they occur in the middle of a long formula. The keyword and *the rest of that input line* will be separated from the rest of the formula input, and processed immediately as control words. They will not become part of the formula in any case. You should always write the EQ control functions as separate one-line control words, however, for greater clarity.

The .EQ ? control word *must* be written in the one-line form; the “?” has no special meaning inside a long formula.

.EQ ?

The control word .EQ ? causes the Formula Formatter to display an identification message that indicates its level and date of creation, for example:

SCRIPT Mathematical Formula Formatter Feature, DCF 3.2 - 08/18/89

.EQ SET

The Formula Formatter chooses the sizes of fonts for superscripts, the thickness of rules for fraction bars, the minimum allowed font size, and several other things, from built-in factors and values. The .EQ SET function is used to define these governing values. With this function, you can control many details of the style of your typeset formulas. The form of this control function is as follows:

$$.EQ \ SET \left\{ \begin{array}{l} \text{SAVE} \\ \text{RESTORE} \\ \text{setkey}_1 \ \text{setval}_1 \ \dots \ \text{setkey}_n \ \text{setval}_n \end{array} \right\}$$

Figure 9. The .EQ SET Control Function

where

SAVE

Saves all of the current values in the .EQ SET “style environment,” but does not change any.

RESTORE

Restores the .EQ SET “style environment” that was saved by the most recent .EQ SET SAVE, and removes it from the list (stack) of saved style environments. If there are no saved values available, then a message is issued, and the initial values are restored.

setkey setval

Sets actual values in the .EQ SET “style environment” with keyword/value pairs. Each keyword must have a value, and a single .EQ SET control word can have as many keyword/value pairs as will fit conveniently on one line.

A GML interface to the .EQ SET keywords is provided in the **METRICS** attribute of the **:DF** and **:DFDEF** tags. That attribute is actually implemented with the .EQ SET control word, and the keywords on the **METRICS** attribute are the same as these. The keywords may be abbreviated to the minimum truncation shown in **bold** type.

blank *c*

Specifies the character to be used as the formula blank character. Each occurrence of this character in a formula produces one “thin space” (about a quarter of an em-space) of white space in the formatted output.

“*c*” is any single character. The default character is “%.”

flr *factor*

Specifies the amount of “clearance” above and below a fraction bar as a *factor* of the current font’s point size. This is the amount of white space between the bottom of the numerator and the top edge of the fraction bar, and between the top of the denominator and the bottom edge of the fraction bar.

The initial factor is about .20 (2 points clearance for a 10-point font).

fbar *factor*

Specifies the thickness of fraction bars as a *factor* of the current font’s point size.

The initial factor is 0.07.

fpos *factor*

Specifies the vertical position of the fraction bar above the baseline, as a *factor* of the current font's point size.

The initial factor is dependent on the device for which the document is being formatted, and is about 0.2.

gap *h*

Specifies the gap between an equation number and the formatted equation, so they don't bump into each other.

"*h*" may be any horizontal space units. That is, it may be specified in picas and points, inches, millimeters, ciceros and didot points, or unqualified horizontal units. The initial value is 4, which is equivalent to the width of "0000" in the default font.

impos *factor*

Specifies the vertical position of superior "marks" in inline formulas, as a *factor* of the current font's point size. "Mark" is a general term for superscripts, subscripts, plex limits, and so forth.

The initial factor is 0.3.

imsize *factor*

Specifies the size of smaller fonts used for "marks" in inline formulas, as a *factor* of the current font's point size. "Mark" is a general term for superscripts, subscripts, plex limits, and so forth.

The initial factor is 0.6.

kerning *factor*

Specifies the kerning adjustment used in horizontal placement of superscripts and subscripts. Superscripts and subscripts placed on an italic base expression are adjusted horizontally (kerned), and the amount of adjustment is a fixed *factor* of the vertical offset of the superscript or subscript above or below a "zero-kern" baseline. (If the base expression is not italic, no kerning is needed.)

The *factor* is a measure of how slanty the italic letters are. The initial factor is 0.16.

maxsize *pointsize*

specifies the maximum point size the Formula Formatter may use. The initial value is 0, meaning "no maximum," and the Formula Formatter may use the largest font it can find in the current font library.

mbar *factor*

Specifies the thickness of "mark bars" (used for the **BAR** and **BARUNDER** keywords), as a *factor* of the current font's point size.

The initial factor is 0.05.

minsize *pointsize*

Specifies the minimum point size the Formula Formatter may use. The initial value is 0, meaning the Formula Formatter may use the smallest font it can find in the current font library.

mpos *factor*

Specifies the vertical position of superior "marks" in display formulas, as a *factor* of the current font's point size. "Mark" is a general term for superscripts, subscripts, plex limits, and so forth.

The initial factor is 0.5.

msize *factor*

Specifies the size of smaller fonts used for “marks” in display formulas, as a *factor* of the current font’s point size. “Mark” is a general term for superscripts, subscripts, plex limits, and so forth.

The initial factor is 0.8.

sbar *factor*

Specifies the thickness of **SQRT** bar as a *factor* of the current font’s point size.

The initial factor is 0.06.

sclr *factor*

Specifies the amount of “clearance” below a square root bar produced by the **SQRT** keyword. as a *factor* of the current font’s point size.

The initial factor is about .15 (1.5 points clearance for a 10-point font).

space *v*

Specifies the amount of vertical space left above and below display formulas.

“v” may be any vertical space units. That is, it may be specified in ems, picas and points, inches, millimeters, ciceros and didot points, or unqualified vertical units.

The initial value is 0.5, which is equivalent to half of a line space in the current font with the current linespacing.

vbar *factor*

Specifies the thickness of the vertical bar part of *constructed* left and right brackets, as a *factor* of the current font’s point size.

The initial factor is 0.1.

Left and right brackets are constructed for the **LB** and **RB** keywords. The horizontal “tails” on the brackets are made half the thickness of the vertical part.

Note: The thickness of vertical bars constructed by the **VBAR** keyword is the same as the thickness of fraction bars.

vspace *v*

Specifies the amount of vertical space left between piled elements separated by the **VABOVE** keyword.

“v” may be any vertical space units. That is, it may be specified in ems, picas and points, inches, millimeters, ciceros and didot points, or unqualified vertical units.

The initial value is 0.25, which is equivalent to one quarter of a line space in the current font with the current linespacing.

.EQ SYMBOL

The .EQ SYMBOL function causes the Formula Formatter to define a regular SCRIPT/VS symbol whose value is a formula or a part of a formula. The NAMEF macro in the Formula Formatter Starter Set and in BookMaster provides a GML interface to .EQ SYMBOL.

The .EQ SYMBOL function also defines the pieces of special adjustable-size characters such as braces and brackets, so the Formula Formatter can build these characters to any height required. This form of .EQ SET requires a suitable font in the font library. The syntax of the function is:

$$\text{.EQ SYMBOL } symname \left\{ \begin{array}{l} \left[\begin{array}{c} \text{GROUP} \\ \text{FRAGMENT} \end{array} \right] / formula-input \\ \text{FONT } fontname \left[\begin{array}{c} \text{SINGLES } c_1 \dots c_4 \\ \text{PARTS } c_{top} c_{bot} [c_{ext} [c_{mid}]] \end{array} \right] \\ \text{DELETE} \end{array} \right\}$$

Figure 10. The .EQ SYMBOL Control Function

where

symname

is the name of the EQ symbol to be defined or deleted. A valid EQ symbol name:

- Must be at least two characters long
- Must be no longer than ten characters
- Must not start with a number
- Must contain only characters that are valid in any SCRIPT/VS symbol name.

A symbol defined with .EQ SYMBOL in the GROUP or FRAGMENT form resides in the regular SCRIPT/VS symbol table. It replaces a symbol of the same name that resides in the regular symbol table, regardless of how it was defined. .EQ SYMBOL, .SE SET SYMBOL, and .DV DEFINE VARIABLE all set the same symbol names in the SCRIPT/VS symbol table. The three differ only in how the value of the symbol will be interpreted. There can be only one symbol of a particular name in the SCRIPT/VS symbol table.

A symbol defined in the FONT form can coexist with a symbol of the same name that was defined by other means. A symbol in the FONT form does not reside in the regular SCRIPT/VS symbol table. Unlike other symbols, a “FONT” symbol has a very limited context where it is valid. It cannot be used outside a formula, and inside one, it is used only for adjustable-size characters, such as a brace or a bracket in a “Left-Right” expression.

GROUP

is the default, and it indicates that the value of this symbol is a complete expression that can stand alone. For example, “x sup 2” could be defined as a formula group, and the symbol that contains this expression could be used alone anywhere. If used inside a formula, this symbol acts as though it had grouping brackets around it, so it behaves like a single letter.

FRAGMENT

specifies that the value of this symbol might not be a complete expression that can be correctly parsed in its own right. For example, “sup 2” is a fragment, and will

cause a syntax error if not used in a suitable context. A formula fragment must be used inside a formula in a context where the fragment becomes part of a complete expression.

/ is a delimiter that signals the start of a formula string. It can be any character, even a character that occurs within the *formula-input*. The inner contents of the string are not examined, but the *last* character is checked to see if it is the same delimiter, and if it is, it is deleted. A closing delimiter is needed only if the string ends in the same character as was used as the delimiter, or if it has trailing blanks. It is never wrong to use a closing delimiter.

formula-input

is an expression or a part of an expression in the Formula Formatter language.

FONT

specifies a font that contains characters that form parts of adjustable-size characters such as braces and brackets. The *fontname* is the name of the font as previously defined with `.DF DEFINE FONT`.

The FONT form of `.EQ SYMBOL` takes the following additional keywords and values:

SINGLES

the font may contain up to four single characters of varying heights. These must be specified in order of increasing height, the shortest first, the tallest last.

PARTS

the font may contain from two to four characters that form pieces of a constructed form of the character. These are the top piece, the bottom piece, the extender piece, and the middle piece. The character codepoints of the parts must be given in that order. All piece characters must have at least a top and a bottom. If there is an extender that can be used to “telescope” the character to any required height, it must be specified next. If the characters is a left or right brace, it also has a middle piece, which is defined last.

c font character used for the adjustable-size formula components. It can be a single character or a 2-character hexadecimal code such as C1.

DELETE

causes the symbol named in *symname* to be deleted if it “belongs” to `.EQ SYMBOL`. Any symbol that was defined with `.EQ SYMBOL` is deleted, but if a symbol of the same name exists, but it was defined with `.SE SET SYMBOL` or `.DV DEFINE VARIABLE` it is left unchanged.

See the discussion of the NAMEF macro in the the section “Creating Named Formula Symbols” on page 29 for examples of the use of `.EQ` symbols in the GROUP and FRAGMENT forms.

.EQ TRACE

The Formula Formatter provides a diagnostic trace facility that displays information about the formula parse tree, the fonts the Formula Formatter selects, the state of various .eq symbols, and so forth. All trace information is delivered as DCF messages, and will go to the same destination as all other diagnostic messages. That is, the trace messages will be displayed at your terminal, or delivered to the same file as all other messages if the MESSAGE(DELAY) option of the SCRIPT command is in effect.

The .EQ TRACE function turns on and off various trace requests, and provides some immediate information as well. The syntax of the function is:

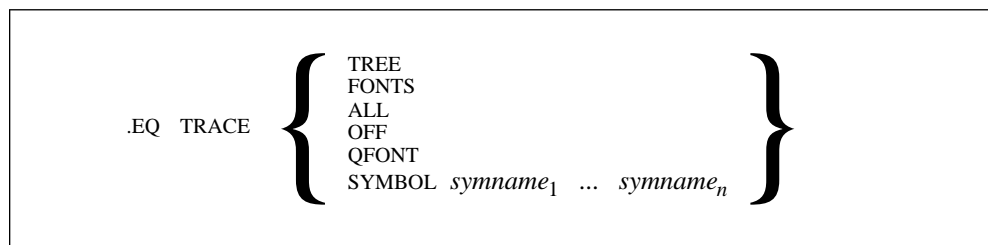


Figure 11. The .EQ TRACE Control Function

where

TREE

tells the Formula Formatter to display the parse trees of all formulas after they have been completely parsed, and before they are formatted. Tree tracing works even if the current output device is not supported for actual formula formatting.

Ordinarily, if the current output device is not suitable for actual formula formatting, the formula *input* is printed “as is” as the output. If you request Tree tracing, then the Formula Formatter does not print the unformatted input. For devices that do allow full formula formatting, a trace of the tree does not affect the full formula output.

A Tree trace represents the binary parse tree in outline form. Extra white space is shown as the percent character (%), regardless of the actual EQ blank character currently defined. The tree display shows all of the different node types in the formula parse tree. For example, if Tree tracing is in effect, the trace for the formula:

$x \supset 2 \text{ over } y \supset 2$

looks like this:

Parse Tree:

- A. Over node: Numerator, Bar, Denominator
 - 1. Text Expression node: Base, SUP
 - a. Body Text node: <x>
 - b. Mark Text node: <2>
 - 2. Rule node: (Fraction Bar)
 - 3. Text Expression node: Base, SUP
 - a. Body Text node: <y>
 - b. Mark Text node: <2>

This shows that the main expression is a fraction, which is held in the tree in an “Over node.” The Over node, A., has three child nodes, for the Numerator, the fraction bar, and the denominator. These three nodes are displayed as numbers 1., 2., and 3. in the display.

The Numerator, 1., is a “Text Expression node” which has two child nodes, the base expression and the superscript, which are shown as a. and b. The base expression is the letter x, and it is held in a “Body Text node,” meaning that it will be set in the normal body pointsize. The superscript is in a “Mark Text node,” indicating that it will be set in the smaller mark point size. The actual text associated with a Text node is shown inside pointy brackets.

A tree trace can show any Formula Formatter parse tree, to whatever nesting depth is necessary.

FONTS

tells the Formula Formatter to display the actual fonts it selects when it is allowed to choose a font on its own. Based on symbol definitions, and the constraints defined with .EQ SET, the Formula Formatter chooses the proper size for “mark” fonts (that is, the font for superscripts, subscripts, overmarks, limits on summations and integrals, and so forth), and also for characters whose size must be adjusted to fit an expression (that is, the braces and brackets in Left-Right expressions, summations, square roots, and so forth).

When Font tracing is in effect, a mark font is traced with a message in the following form:

```
Base 10.0-pt., Using C1T05560: 6-pt. SONORAN SERIF, Medium. Cdpq: T1G
```

This message indicates what the base (body) font is at the time, and it shows that the mark size factor is probably set at .6.

If the Formula Formatter selects a font for a tall character whose height must enclose an expression, the font trace shows a nominal calculated point size and the actual point size used, as follows:

```
Calc 21.6-pt., Using C1M055L0: 22-pt. MATH FORMAT, Medium. Cdpq: T1M0
```

ALL

starts both Tree tracing and Font tracing.

OFF

stops all tracing.

QFONT

(i.e., Query Font) is an immediate request. It causes the Formula Formatter Trace facility to display the name, size, and attributes of the current font, whenever the control word is executed, in the following form:

```
>Font C1T05500: 10-pt. SONORAN SERIF, Medium. Cdpq: T1GI0395
```

A .EQ TRACE QFONT is generally done outside of a formula.

SYMBOL

displays the values of symbols that were defined with .EQ SYMBOL. If the symbol was defined with .EQ SYMBOL, the Formula Trace facility displays it in a form that looks similar to a .EQ SYMBOL control word that would define it.

For example, a Symbol trace of a symbol called “fred” might look like this:

```
>EQ Symbol fred Group /x sup 2
```

(If you changed the leading “>” character to a period, this line could be executed to define the symbol.)

The Symbol trace indicates that a symbol does not exist with a trace display of the following form:

```
>EQ Symbol lbrace = Undefined
```

This can mean that the symbol does not exist at all, or it can mean that it exists in the SCRIPT/VS symbol table, but not in a form that could have been defined with .EQ SYMBOL. If the symbol exists, and was defined with .SE SET SYMBOL“ or .DV DEFINE VARIABLE“, then .EQ TRACE considers it “undefined.”

Note: The .IT INPUT TRACE“ control word can display the value of any symbol that resides in the regular SCRIPT/VS symbol table.

If the symbol name is not valid, the Trace will display a message like this:

```
>EQ Symbol 5dots = Invalid name
```

The name can be perfectly valid for a regular symbol, but invalid as a EQ symbol.

symname

are names of the EQ symbols you want traced. A valid EQ symbol name:

- Must be at least two characters long

- Must be no longer than ten characters

- Must not start with a number

- Must contain only characters that are valid in any SCRIPT/VS symbol name.

Appendix A. Formula Formatter Keywords

Special Break-Character Keywords

The following keywords can be recognized *without* any blank before or after them. Many of these keywords are exactly the same as an equivalent “spelled-out” keyword, which you can use if your terminal lacks some of the special characters. The input for the example includes a comment to tell you the equivalent spelled-out keyword.

Keyword	Example	Input for Example
.*	x^i	x sup i .* The rest of the line ignored
($(a + b)$	(a + b) .* Same as lparen
+	$x + y$	x+y .* Same as plus
*	$x \times y$	x*y .* Same as times
**	x^i	x**i .* Same as sup
)	$(a + b)$	(a + b) .* Same as rparen
-	$x - y$	x-y .* Same as minus
/	$2/\gamma$	2 / gamma .* See also divide
	$a + b$	a +b .* Quoted string not italicized
=	$x = x$	x=x .* Same as equals
==	$a \equiv a$	a==a .* Same as identical
"	$\mathbf{a} + b$	"a"+b .* Quoted string not italicized
{	$\{a + b\}$	{ a + b } .* Same as lbrace
}	$\{a + b\}$	{ a + b } .* Same as rbrace
	$[a + b]$	a + b “ .* Same as lbracket
“	$[a + b]$	a + b “ .* Same as rbracket

Figure 12. Special Break-Character Keywords

Regular Keywords

The following keywords must be preceded and followed by blanks, break character keywords, or the end of an input line. If you want to group several blank-delimited words and keywords together as a single entity, use left and right grouping brackets.

These keywords may be abbreviated. To do this, enter the keyword with a period. Thus, **INTEGRAL** may be entered as .int, .inte, .integ, .integr, .integra, .integral, or integral. The minimum valid truncation is shown in **bold** type. Some keywords have no minimum truncation, and must be fully spelled out.

Keyword	Example	Input for Example
above	$\begin{pmatrix} a \\ b \end{pmatrix}$	left (a above b right)
adjust	$a \quad a$	a adjust(up 3 right 5) a
angstrom	5810Å	5810 angstrom
aleph	\aleph_0	aleph sub 0
all	$(\forall x)$	(all x)
alpha	$\frac{1}{\alpha} \approx 137$	1 over alpha app 137
Alpha	A^2	Alpha sup 2
app	\approx	app
arg	$a \arg \theta$	a arg theta
bar	$\sum_i (x_i - \bar{x})^2$	sum from i of (x sub i - x bar) sup 2
barunder	$\underline{x} \quad \underline{y}$	x barunder bullet y barunder
beta	$\frac{\beta}{\xi}$	beta over xi
Beta	$B(x, y)$	Beta (x, y)
bullet	$a \quad b$	a bullet b

Figure 13 (Part 1 of 10). Regular Keywords

Keyword	Example	Input for Example
cabove	$\begin{matrix} a + 1 \\ b \end{matrix}$	a+1 cabove b .* Same as above
chi	χ^2	chi sup 2
Chi	X^2	Chi sup 2
congruent	$ABC \cong CBA$	ABC congruent CBA
cos	$z \cos \theta$	z cos theta
cosh	$a \cosh x$	a cosh x
cot	$z \cot \theta$	z cot theta
cover	$\frac{a + 1}{b}$	a+1 cover b .* Same as over
coverlay	\emptyset	0 coverlay / .* Same as overlay
csc	$p \csc \theta$	p csc theta
ctn	$p \operatorname{ctn} \theta$	p ctn theta
cvabove	$\begin{matrix} a + x_2 \\ y^3 \end{matrix}$	<a+x sub 2> cvabove <y sup 3>
degree	$86^\circ 14'$	86 primemark degree 14 prime
del	$\nabla \times \vec{A}$	del * A vector
delta	$\exists \delta$	exists delta
Delta	Δz	Delta z
det	$q \det M$	q det M
dim	$q \dim M$	q dim M
divide	$a \div b$	a divide b
dot	\dot{x}	x dot
dotdot	\ddot{x}	x dotdot

Figure 13 (Part 2 of 10). Regular Keywords

Keyword	Example	Input for Example
dotdotunder	\ddot{x}	x dotdotunder
dotunder	\dot{x}	x dotunder
ellipsis	$a \dots b$	a ellipsis b
epsilon	$f(x + \epsilon) - f(x)$	f(x + epsilon) - f(x)
Epsilon	E	Epsilon
eqno	$y = x^i \quad (1)$	y = x sup i eqno (1)
equals	$a = a$	a equals a
eta	$\ddot{\eta}$	eta dotdot
Eta	\ddot{H}	Eta dotdot
exists	\exists	exists
exp	$\rho \exp i\theta$	rho exp i theta
from	$\sum_{i=1} x_i$	sum from i=1 of x sub i
function	$x \text{ foo } \theta$	x function foo theta
gamma	$\sqrt{\gamma}$	sqrt gamma
Gamma	$\Gamma(x + 1)$	Gamma (x+1)
ge	$a \geq b$	a ge b
glb	$\text{glb}\{x_i\}$	glb lbrace x sub i rbrace
gt	$a > b$	a gt b
habove	$a = b$ $= xyz$	 habove <here = xyz>
hamiltonian	\mathcal{H}	hamiltonian .* Symbol is hamilton
hat	\hat{x}	x hat

Figure 13 (Part 3 of 10). Regular Keywords

Keyword	Example	Input for Example
here	$a = b$ $= xyz$	 habove <here = xyz>
hvabove	$\sum_x \sum_y$	<here sum y> hvabove <sum x here>
hyphen	$a - b - c$	a - b hyphen c
identical	$a \equiv a$	a identical a
infinity	$\sum_{i=0}^{\infty}$	sum from i=0 to infinity
integral	$\int_0^{\pi} \cos \theta d\theta$	integral from 0 to pi cos theta % d theta
intersection	$X \cap Y$	X intersection Y
intplex	\int_{-}^{+}	intplex rb from - to +
iota	$F_{\iota\kappa\lambda}$	F sub <iota kappa lambda>
Iota	$I\Lambda I\Lambda\Sigma$	Iota Lambda Iota Alpha Sigma
join	$X \cup Y$	X join Y
kappa	$F_{\iota\kappa\lambda}$	F sub <iota kappa lambda>
Kappa	K_j	Kappa sub j
labove	$a + 1$ b	a+1 labove b
lambda	$F_{\iota\kappa\lambda}$	F sub <iota kappa lambda>
Lambda	Λ_2	Lambda sub 2
larrow	$x \leftarrow y$	x larrow y
lb	$[a + b]$	lb a + b rb .* Constructed bracket

Figure 13 (Part 4 of 10). Regular Keywords

Keyword	Example	Input for Example
lbrace	$\{a + b\}$	lbrace a + b rbrace
lbracket	$[a + b]$	lbracket a + b rbracket .* Bracket char
le	$a \leq b$	a le b
left	$\begin{bmatrix} a \\ b \end{bmatrix}$	left lb a above b right rb
leqno	(2) $y = x^i$	y = x sup i leqno (2)
lim	$\lim_{t \rightarrow \infty} \int_0^t$	lim midsub <t .rar .inf> .int from 0 to t
log	$x \log n$	x log n
lor	$a \mid b$	a lor b
ln	$x \ln n$	x ln n
lnot	$\neg \rho$	lnot rho
lover	$\frac{a + 1}{b}$	a+1 lover b
loverlay		bullet loverlay lor
lparen	$(a + b)$	lparen a + b rparen
lt	$a < b$	a lt b
lub	$\text{lub}\{x_i\}$	lub lbrace x sub i rbrace
lvabove	$\begin{matrix} a + x_2 \\ y^3 \end{matrix}$	<a+x sub 2> lvabove <y sup 3>
lvbar	$ a + b $	lvbar a + b rvbar
max	$\max(a, b)$	max (a, b)
meet	$X \cap Y$	X meet Y
memberof	$a \in A$	a memberof A

Figure 13 (Part 5 of 10). Regular Keywords

Keyword	Example	Input for Example
midsub	x_i	x midsub i
midsup	x^i	x midsup i
min	$\min(a, b)$	min (a, b)
minus	$a - b$	a minus b
mod	$a \equiv b \pmod{n}$	a == b (mod n)
mp	$a \mp b$	a mp b
mu	$0.8\mu\text{m}$	0.8 mu m
Mu	M	Mu
ne	$a \neq b$	a ne b
nu	x_ν	x sub nu
Nu	N	Nu
of	$\int \cos \theta \, d\theta$	integral of cos theta % d theta
omega	$i\omega t$	i omega t
Omega	Ω	Omega
omicron	o	omicron
Omicron	O	Omicron
operator	$a \oslash 2$	a .op omega 2
over	$\frac{a+1}{b}$	a+1 over b
overlay	\equiv	identical overlay /
overmark	\dot{M}	M overmark ,
partial	$\frac{\partial F}{\partial x}$	<partial F> over <partial x>

Figure 13 (Part 6 of 10). Regular Keywords

Keyword	Example	Input for Example
phi	$\cos \phi$	cos phi
Phi	Φ	Phi
pi	πr^2	pi r sup 2
Pi	$\Pi(x)$	Pi(x)
psi	$\frac{\psi}{\omega}$	psi over omega
Psi	Ψ	Psi
plex	$\bigcup_{i=1}^n S_i$	plex union from i=1 to n S sub i
plus	$a + b$	a plus b
pm	$a \pm b$	a pm b
presub	i^x	x presub i
presup	i^x	x presup i
prime	x'	x prime
primemark	x''	x primemark "
product	$\prod_{j=1}^{\infty} P_j$	product from j=1 to infinity of P sub j
proportional	$a \propto b$	a proportional b
rabove	$\begin{bmatrix} -1 \\ 1 \end{bmatrix}$	left lb -1 rabove 1 right rb
rarrow	$n \rightarrow \infty$	n rarrow infinity
rb	$[a + b]$	lb a + b rb .* Constructed bracket
rbrace	$\{a + b\}$	lbrace a + b rbrace
rbracket	$[a + b]$	lbracket a + b rbracket .* Bracket char

Figure 13 (Part 7 of 10). Regular Keywords

Keyword	Example	Input for Example
reqno	$y = x^i \quad (3)$	y = x sup i reqno (3)
rho	$\rho(x)$	rho (x)
Rho	P	Rho
right	$x $	left < x right vbar
rover	$\frac{a+1}{b}$	a+1 rover b
roverlay		bullet roverlay lor
rparen	$(a+b)$	lparen a + b rparen
rvabove	$\begin{matrix} a+x^2 \\ y^3 \end{matrix}$	<a+x sup 2> rvabove <y sup 3>
rvbar	$ a+b $	lvbar a + b rvbar
scriptl	ℓ	scriptl
sec	$a \sec \theta$	a sec theta
sigma	e^{σ^2}	e sup <sigma sup 2>
Sigma	Σ	Sigma
similar	$a \sim b$	a similar b
sin	$\int \sin \theta \, d\theta$	integral of sin theta % d theta
sinh	$a \sinh x$	a sinh x
sqrt	$\sqrt{x^i}$	sqrt x sup i
sub	x_i	x sub i
suchthat	$x \ni x \geq 0$	x suchthat x ge 0
sum	$\sum x_i$	sum of x sub i

Figure 13 (Part 8 of 10). Regular Keywords

Keyword	Example	Input for Example
sup	x^i	x sup i
tan	$\int \tan \theta d\theta$	integral of tan theta % d theta
tanh	$a \tanh x$	a tanh x
tau	$t - \tau$	t - tau
Tau	T	Tau
theta	$\int \cos \theta d\theta$	integral of cos theta % d theta
Theta	Θ	Theta
tilde	\tilde{x}	x tilde
times	$a \times b$	a times b
to	$\sum_{i=1}^n x_i$	sum to n of x sub i
undermark	$\underset{o}{u}$	o undermark join
union	$X \cup Y$	X union Y
upsilon	υ	upsilon
Upsilon	Υ	Upsilon
vabove	$\begin{matrix} a + x_2 \\ y^3 \end{matrix}$	<a+x sub 2> vabove <y sup 3>
vbar	$a b$	a vbar b
vector	\overrightarrow{x}	x vector
with	$\sum_{i=1} \sum_{j=1} A_{ij}$	sum from i=1 with sum from j=1 of A sub <i j>
xi	$\xi\eta\zeta$	xi eta zeta

Figure 13 (Part 9 of 10). Regular Keywords

Keyword	Example	Input for Example
Xi	$\Sigma\Xi$	Sigma Xi
zeta	$\xi\eta\zeta$	xi eta zeta
Zeta	Z	Zeta

Figure 13 (Part 10 of 10). Regular Keywords

Appendix B. Required Formula Formatter Symbols

Many Formula Formatter keywords cause special characters to be formatted in the output. It does this by looking up symbol definitions in the SCRIPT symbol table.

Each of the symbols in this list can be called for by a Formula Formatter keyword. In some cases, the keyword does nothing other than provide a convenient way of obtaining a special character, but in these cases, the keyword is case-insensitive, and it can be abbreviated.

All of these required symbols are defined in the Formula Formatter Starter Set, and also in BookMaster:

Symbol Name	Symbol	Symbol Name	Symbol
aleph	\aleph	integral	\int
all	\forall	intersect	\cap
angstrom	\AA	join	\cup
app	\approx	larrow	\leftarrow
bullet		lbrace	$\{$
congruent	\equiv	lbracket	$[$
degree	$^{\circ}$	le	\leq
del	∇	lnot	\neg
divide	\div	lor	$ $
divslash	$/$	lparen	$($
dot	\cdot	lt	$<$
dotdot	$..$	meet	\cap
ellipsis	\dots	memberof	\in
equals	$=$	minusop	$-$
exists	\exists	mp	\mp
ge	\geq	ne	\neq
gt	$>$	partial	∂
hat	\wedge	plusop	$+$
hamilton	\mathcal{H}	pm	\pm
hyphen	$-$	prime	$'$
identical	\equiv	product	\prod
infinity	∞	proportion	\propto

Figure 14 (Part 1 of 2). Required Formula Formatter Symbols

Symbol Name	Symbol
rarrow	\rightarrow
rbrace	$\}$
rbracket	$\]$
rparen	$)$
scriptl	ℓ
similar	\sim
sqrt	$\sqrt{}$
suchthat	\ni
sum	\sum
tilde	\sim
times	\times
union	\cup
vector	\rightarrow
alpha	α
Alpha	A
beta	β
Beta	B
gamma	γ
Gamma	Γ
delta	δ
Delta	Δ
epsilon	ϵ
Epsilon	E
zeta	ζ
Zeta	Z
eta	η
Eta	H
theta	θ
Theta	Θ
iota	ι
Iota	I

Symbol Name	Symbol
kappa	κ
Kappa	K
lambda	λ
Lambda	Λ
mu	μ
Mu	M
nu	ν
Nu	N
xi	ξ
Xi	Ξ
omicron	\omicron
Omicron	O
pi	π
Pi	Π
rho	ρ
Rho	P
sigma	σ
Sigma	Σ
tau	τ
Tau	T
upsilon	υ
Upsilon	Υ
phi	ϕ
Phi	Φ
chi	χ
Chi	X
psi	ψ
Psi	Ψ
omega	ω
Omega	Ω

Figure 14 (Part 2 of 2). Required Formula Formatter Symbols

Appendix C. Other Symbols

The *required* symbol names listed in Appendix B, “Required Formula Formatter Symbols” on page 57 are symbols whose names are automatically produced by the Formula Formatter from keywords. Those symbols must be defined for the Formula Formatter to work its best.

In addition to the *required* symbols, any other symbol that is defined can be used in a formula. You can use any symbol name in a formula almost as though it were a keyword. The value of the symbol will appear in the formatted formula.

If you use a symbol that is not also a Formula Formatter keyword, you must spell out the name fully. No abbreviation is possible except for keywords. Also, the value of a symbol that is not also a keyword will receive no special formatting, such as operator padding.

In addition to the required Formula Formatter symbols, the Formula Formatter Starter Set, and also BookMaster, provide definitions for the following:

Symbol Name	Symbol	Symbol Name	Symbol
euler	\mathcal{E}	doubleR	\mathbb{R}
hbar	\hbar	doubleZ	\mathbb{Z}
planck	\hbar	ltlt	\ll
weierstr	\wp	gtgt	\gg
fno	f	arc	\curvearrowright
nexists	\nexists	ltequiv	\leq
nmemberof	\notin	gtequiv	\geq
increase	\nearrow	percent	$\%$
decrease	\searrow	nidentical	\neq
urbarb	\uparrow	approxid	\approx
ulbarb	\uparrow	and	\wedge
lbarb	\leftarrow	angle	\angle
rbarb	\rightarrow	vardelta	∂
factorial	$!$	vartheta	ϑ
doubleC	\mathbb{C}	varsigma	ς
doubleN	\mathbb{N}	varphi	φ
doubleP	\mathbb{P}	or	\vee
doubleQ	\mathbb{Q}	rprime	\prime

Figure 15 (Part 1 of 2). Other Symbols Defined in DSMSPROF

Symbol Name	Symbol	Symbol Name	Symbol
ratio	:	cequal	\equiv
therefore	\therefore	ctimes	\otimes
because	\because	nsubset	$\not\subset$
uarrow	\uparrow	sqbullet	
darrow	\downarrow	nsuperset	$\not\supset$
nearly	\approx	ltrank	$<$
nnearly	\neq	gtrank	$>$
superset	\supset	bsl	\backslash
subset	\subset	nisubset	$\not\subset$
perpend	\perp	nisuperset	$\not\supset$
circ		lerank	\leq
scriptI	\mathcal{I}	gerank	\geq
riemann	\mathcal{R}	ldarrow	\Leftarrow
concat	\parallel	rdarrow	\Rightarrow
gtlt	\gtrless	zero	
diam	\diamond	isubset	\subseteq
determines	\mapsto	isuperset	\supseteq
cdot	\odot	nltrank	\prec
cplus	\oplus	nlerank	\nless
cminus	\ominus	iff	\Leftrightarrow

Figure 15 (Part 2 of 2). Other Symbols Defined in DSMSPROF

Appendix D. Error Messages

The Formula Formatter produces error messages when syntax errors in the input are detected. These messages are written out by the standard SCRIPT/VS error message processor, and they respond to the MESSAGE option of the SCRIPT command. Depending upon the current MESSAGE option in effect, messages can

- Display the message identifying number, or not

- Be displayed immediately, or delayed until the end of the document

- Display a trace of all nested imbedded files at the time of the condition.

See *Document Composition Facility Messages*, SH35-0048 for more information about messages issued by SCRIPT/VS.

Messages

In the following messages, variable information is shown in *italics*. In the message identifier, “xxx” refers to the module of the Formula Formatter that issues the message.

From the point of view of SCRIPT/VS, these messages diagnose syntax errors in the .EQ [EQUATION] control word. That control word provides the underlying Formula Formatter interface to the GML tags used to markup formulas.

Msg ID	Message text
DSMxxx602E	Invalid numeric parameter ' <i>parameter</i> ' in .EQ <i>function</i>
DSMxxx605E	Required parameter in .EQ ' <i>function</i> ' is missing
DSMxxx606W	The .EQ SET save stack is empty, initial values restored
DSMxxx607S	Infinite recursion in .EQ SYMBOL formulas
DSMxxx608E	Invalid parameter ' <i>parameter</i> ' in .EQ <i>function</i>
DSMxxx610W	Unable to start font ' <i>fontname</i> ' for .EQ
DSMxxx611W	No height table found for font ' <i>fontname</i> '. Cannot define symbol
DSMxxx620W	The .EQ ' <i>key</i> ' keyword is out of context, ERROR ' <i>num</i> '
DSMxxx621W	The .EQ ' <i>key</i> ' keyword must be preceded by a string
DSMxxx622W	The .EQ ' <i>key2</i> ' keyword must be preceded by a string; ' <i>key1</i> ' is not valid as a precedent
DSMxxx623W	A .EQ begin grouping bracket is not closed with a matching end grouping bracket
DSMxxx624W	A .EQ end grouping bracket does not have a matching begin grouping bracket
DSMxxx627W	The .EQ ' <i>key1</i> ' keyword must be followed by a string; ' <i>key2</i> ' is out of context
DSMxxx628W	The .EQ ' <i>key1</i> ' keyword must be followed by an expression; ' <i>key2</i> ' is out of context
DSMxxx629W	A .EQ ' <i>key</i> ' keyword appears not to be part of a PLEX or INTPLEX expression
DSMxxx630W	A .EQ ' <i>key</i> ' keyword must be followed by a PLEX keyword
DSMxxx631E	Unable to recover from .EQ errors, formula is discarded
DSMxxx632E	Stack overflow on .EQ
DSMxxx633E	Invalid SMFF parse state -- formula parsing is terminated
DSMxxx635W	Null formula -- ignored
DSMxxx636W	A .EQ LEFT keyword was found without a matching RIGHT keyword
DSMxxx637W	A .EQ RIGHT keyword was found without a matching LEFT keyword

Appendix E. Formula Formatter Examples

The following pages show some examples of formulas and the input used to generate those formulas.

The approach used for marking up the formulas in these examples uses a lot of NAMEF macros to define repeating elements. Doing this reduces typographical errors and makes the markup easier to read, because the underlying structure of the formula is not obscured by the details. It's also easier to update a formula that has well-planned named sections.

The method suggested here is not the only way it could be done.

Example 1

$$s(x) = (1 + \sigma)u_0 - \frac{q}{l} \left[1 + \delta b + 2 \sum_{k=1}^{\infty} \frac{a_k + b \tanh \alpha_k \delta}{\alpha_k (\alpha_k^2 + b \alpha_k \tanh \alpha_k \delta + 1)} \cos \alpha_k x \right]$$

Input for Example 1

```
.namef symbol=ask f='alpha sub k'
.*
:df.
s(x) = (1 + sigma) u sub 0 - q over l %%
left lb 1+ delta b+2
sum from <k=1> to infinity of
  <a sub k + b tanh ask delta>
    over
  <ask (ask sup 2 + b ask
    tanh ask delta + 1 ) > %%
cos ask x right rb
:edf.
```

Example 2

$$a_0 + \frac{b_1}{a_1 + \frac{b_2}{a_2 + \frac{b_3}{a_3 + \frac{b_4}{a_4 + \dots}}}}$$

Input for Example 2

```
:df align=center.
a sub 0 plus
b sub 1 over <a sub 1 plus
  <b sub 2 over <a sub 2 plus
    <b sub 3 over <a sub 3 plus
      <b sub 4 over <a sub 4 plus
        ....
      >>
    >>
  >>
>
:edf.
```

Example 3

$$\int_0^1 du' \sqrt{R} = \left[\frac{1}{2} + \frac{u(\xi\xi' - 1)}{2(1 + \xi'^2)} \right] \sqrt{u^2(1 + \xi^2) + 2u(\xi\xi' - 1) + (1 + \xi'^2)} \\ - \frac{u(\xi\xi' - 1)}{2(1 + \xi'^2)} u \sqrt{1 + \xi^2} + \frac{u^2(\xi + \xi')}{2(1 + \xi'^2)} \int_0^1 \frac{du'}{\sqrt{R(u')}}$$

Thus;

$$I_4 = \frac{1}{2} \int_0^1 du \sqrt{u^2(1 + \xi^2) + 2u(\xi\xi' - 1) + (1 + \xi'^2)} \\ + \frac{(\xi\xi' - 1)}{2(1 + \xi'^2)} \int_0^1 du \quad u \sqrt{u^2(1 + \xi^2) + 2u(\xi\xi' - 1) + (1 + \xi'^2)} \\ - \frac{(\xi\xi' - 1)\sqrt{1 + \xi^2}}{2(1 + \xi'^2)} \int_0^1 du \quad u^2 + \frac{(\xi + \xi')^2}{2(1 + \xi'^2)} \int_0^1 du \quad u^2 \int_0^1 \frac{du'}{\sqrt{R(u')}}$$

Note that if $R(u) \equiv u^2(1 + \xi^2) + 2u(\xi\xi' - 1) + (1 + \xi'^2)$

then

$$\frac{dR}{du} = 2u(1 + \xi^2) + 2(\xi\xi' - 1)$$

$$u = \frac{\left[\frac{dR}{du} - 2(\xi\xi' - 1) \right]}{2(1 + \xi^2)}$$

Input for Example 3

```
. *-----
.namef symbol=XXP    f='lparen xi xi prime - 1 rparen'
.namef symbol=XPS    f='lparen 1 + xi prime ** 2 rparen'
.namef symbol=XPS2   f='2 XPS'
.namef symbol=XISQ   f='lparen 1 + xi**2 rparen'
.namef symbol=XISQP1 f='1 + xi**2'
.namef symbol=XPXP   f='lparen xi + xi prime rparen'
.namef symbol=SQTXXX f='sqrt < u**2 XISQ + 2 u XXP + XPS>'
. *-----
. *
```

(Input for Example 3 continued on next page)

Input for Example 3 (continued)

```

:df metrics='vspace p9 maxsize 36'.
<< integral from 0 to 1 of du prime sqrt R >
  here = left lb 1
              over
              2      + < u XXP >
                      over
                      XPS2      right rb %% SQTXXX
> hvabove <
  here - < u XXP >
          over
          XPS2
% u sqrt XISQP1 + < u**2 XPXP >
              over
              XPS2
%% integral from 0 to 1 of < du prime
                        over
                        sqrt < R (u prime ) > >

:edf.
:p.Thus;
:df metrics='vspace p9 maxsize 36'.
<I sub 4 here = 1
              over
              2      %% integral from 0 to 1 of du SQTXXX
> hvabove <
  here + XXP
          over
          XPS2
%% integral from 0 to 1 of du bullet u SQTXXX
> hvabove <
  here - < XXP sqrt XISQP1 >
          over
          XPS2
%% integral from 0 to 1 of du bullet u**2 + XPXP ** 2
                        over
                        XPS2
%% integral from 0 to 1 of du bullet u**2
%% integral from 0 to 1 of du prime
                        over
                        sqrt < R(u prime) >>

:edf.
:p.Note that if :f.R(u)
  identical u**2 XISQ + 2u XXP + XPS:ef.
:p.then
:df metrics='maxsize 36'.
  dR
  over
  du      = 2u XISQ + 2 XXP
:edf.
:f.u = left lb  dR
              over
              du      - 2 XXP  right rb
              over
              <2 XISQ>
:ef.

```

Example 4

We now continue with the same induction as before:

$$\begin{aligned}
\sum_{s=t}^T \bar{\pi}_s b_s &= \sum_{s=t}^T [\bar{\lambda}_{t-1} \Pi_s b_s + \bar{\mu}_{t-1} \Upsilon_s b_s] \\
&= \bar{\lambda}_{t-1} \sum_{s=t}^T \Pi_s b_s + \bar{\mu}_{t-1} \sum_{s=t}^T \Upsilon_s b_s \\
&= \sum_{i=1}^I \bar{\lambda}_{t-1}^i \sum_{s=t}^T \bar{\pi}_s^{(i)} b_s + \sum_{j=1}^J \bar{\mu}_{t-1}^j \sum_{s=t}^T \bar{\sigma}_s^{(j)} b_s \\
&= \sum_{i=1}^I \bar{\lambda}_{t-1}^i \left[\bar{\pi}_t^{(i)} b_t + \sum_{s=t+1}^T \bar{\pi}_s^{(i)} b_s \right] + \sum_{j=1}^J \bar{\mu}_{t-1}^j \left[\bar{\sigma}_t^{(j)} b_t + \sum_{s=t+1}^T \bar{\sigma}_s^{(j)} b_s \right] \\
&= \sum_{i=1}^I \bar{\lambda}_{t-1}^i \left[\bar{\pi}_t^{(i)} b_t + \sum_{s=t+1}^T \bar{\pi}_s^{(i)} b_s \right] + \sum_{j=1}^J \bar{\mu}_{t-1}^j \left[\bar{\sigma}_t^{(j)} b_t + \bar{\omega}_t^{(j)} \hat{p}_t^{(j)} \right] \quad (\text{by theorem 2.1}) \\
&= \sum_{i=1}^I \bar{\lambda}_{t-1}^i \left[\bar{\pi}_t^{(i)} b_t + \bar{\lambda}_t^{(i)} q_t^{(i)} + \bar{\mu}_t^{(i)} p_t^{(i)} \right] + \sum_{j=1}^J \bar{\mu}_{t-1}^j \left[\bar{\sigma}_t^{(j)} b_t + \bar{\omega}_t^{(j)} \hat{p}_t^{(j)} \right] \quad (\text{by induction}) \\
&= \sum_{i=1}^I \bar{\lambda}_{t-1}^i q_t^{(i)} + \sum_{j=1}^J \bar{\mu}_{t-1}^j p_t^{(j)} \quad (\text{by (2.3.2) and (2.3.3)}) \\
&= \bar{\lambda}_{t-1} q_{t-1} + \bar{\mu}_{t-1} p_{t-1}
\end{aligned}$$

Thus we have our backward dual solution. It is now easy to show that when the algorithm terminates, it has solved the problem appropriately.

Input for Example 4

```

:p.We now continue with the same
induction as before:
.*
.*-----
.namef symbol=tml      fpart='sub t-1'
.namef symbol=sbs      fpart='sub s b sub s'
.namef symbol=stsi     fpart='sub t sup (i)'
.namef symbol=stsj     fpart='sub t sup (j)'
.namef symbol=leftb    fpart='% left lb'
.namef symbol=riteb    fpart='right rb %'
.namef symbol=lambar   fpart='lambda bar'
.namef symbol=mubar    fpart='mu bar'
.namef symbol=SumT     fpart='sum from s=t to T'
.namef symbol=SumI     fpart='sum from i=1 to I lambar tml sup i'
.namef symbol=SumJ     fpart='sum from j=1 to J mubar tml sup j'
.namef symbol=SumlT    fpart='sum from <s=t+1> to T'
.namef symbol=pibs     fpart='pi bar sub s sup (i) b sub s'
.namef symbol=pibt     fpart='pi bar stsi b sub t'
.namef symbol=sigs     fpart='sigma bar sub s sup (j) b sub s'
.namef symbol=sigt     fpart='sigma bar stsj b sub t'
.*-----
.*
:df metrics='vspace .75'.
<
    SumT pi bar sbs here = %
    SumT .lbrack lambar tml PI sbs
        + mubar tml Upsilon sbs .rbrack
> hvabove < here = %
    lambar tml
    SumT PI sbs + mubar tml SumT Upsilon sbs
> hvabove < here = %
    SumI SumT pibs
+ SumJ SumT sigs
> hvabove < here = %
    SumI leftb pibt + SumlT pibs riteb
+ SumJ leftb sigt + SumlT sigs riteb
> hvabove < here = %
    SumI leftb pibt + SumlT pibs riteb
+ SumJ leftb sigt + omega bar stsj p hat stsj riteb
        %%%% '(by theorem 2.1)'
> hvabove < here = %
    SumI leftb pi bar stsi b sub t
        + lambar stsi q stsi
        + mubar stsi p stsi riteb
+ SumJ leftb sigt + omega bar stsj p hat stsj riteb
        %% '(by induction)'
> hvabove < here = %
    SumI q tml sup i + SumJ p tml sup j
        %%%%%%%%%% '(by (2.3.2) and (2.3.3))'
> hvabove < here = %
    lambar tml q tml + mubar tml p tml sqbullet
>
:edf.
:p.Thus we have our backward dual solution.
It is now easy to show that when
the algorithm terminates, it has solved the problem appropriately.

```

Appendix F. Programming Considerations

Tailoring the SMFF Feature for your Installation

The Formula Formatter feature chooses the sizes of fonts for superscripts and subscripts, the thickness of rules for fraction bars, the minimum allowed font size, and several other things, from built-in factors and names. Many of these things can be altered with .EQ SET parameters to suit your own requirements.

For example, the size and positioning of superscripts is different for inline formulas than for display formulas. If you would like to make *all* superscripts use the size rule for display formulas, and the positioning rule for inline formulas, you could set these values with the control word

```
.eq set imsize 0.8 mpos 0.3
```

This control word could be placed:

Just before a particular set of formulas in one file, to change the rules for those formulas only

At the beginning of a particular SCRIPT file, to change the rules for all formulas in that document

In the installation's GML profile, to change the rules for *all* formulas formatted at the installation.

Tailoring the SMFF Feature Profile

The GML profile DSMFPROF, which comes with the Formula Formatter feature, is provided merely as an example of how to define all of the macros and all of the symbols necessary. The real work of defining the macros is done in DSMFMAC. DSMFMAC defines the macros and establishes the associations between the formula GML tags and those macros.

DSMFPROF uses the .IM [IMBED] control word to imbed:

1. DSMPROF3, the GML Starter Set profile. If you use a different profile, you could easily make DSMFPROF imbed your GML profile instead of DSMPROF3).
2. DSMFMAC.

The advantage of this method is that you can run the Formula Formatter feature without altering your regular profile. The disadvantage is that your users now have to think about two profiles, DSMFPROF if there are formulas, and DSMPROF3 (or whatever GML profile you normally use) if not.

Instead of using DSMFPROF, you can change your own regular GML profile so it imbeds DSMFMAC. If you do this, then your users can always use one GML profile, which might be more convenient than having a separate DSMFPROF profile.

In CMS, all you have to do is add two lines to DSMFPROF3 SCRIPT (or your regular GML profile, if it's not DSMFPROF3):

```
.im DSMFMAC
```

In TSO, you must specifically define the data set names for the macro and symbol files before imbedding them. Do this with the following .DD [DEFINE DATA FILE-ID] control word. In TSO, you could add the following lines to your regular GML profile:

```
.dd DSMFMAC dsn 'SCRIPT.R32.MACLIB(DSMFMAC) '  
.im DSMFMAC
```

All of the symbols used by the Formula Formatter feature are defined in the “DSMFPROF” file. The DSMFPROF file has two sets of definitions so it can define the required symbols according to what fonts are available in your font library. If you do not have the true math fonts, DSMFPROF tries to define the basic required symbols using only typewriter fonts. The characters available in the true math fonts are much better suited to mathematical formula formatting, and if you have these fonts, DSMFPROF will be able to do better symbol definitions.

If you want to define additional symbols that are not already defined in DSMFPROF, you can add them with the .DV [DEFINE VARIABLE] control word, following the format of the symbol definitions in DSMFPROF.

Index

Special Characters

? parameter of .EQ EQUATION" 35
.(period)
 keyword abbreviation delimiter 10
 markup content separator 4
 use to separate gml markup from content 4
.* (comment) 12
" (double quote) delimiter 12
' (single quote) delimiter 12
** (SUP keyword) 16
&\$EQ symbol 33
% keyword 12
+ (PLUS keyword) 14
- (MINUS keyword) 14

A

abbreviating keywords 10
ABOVE keyword 17
ADJUST keyword 24
 left, right, up, down 24
adjustments 24
ALIGN attribute 6
aligning a display formula 6
alignment
 of equal signs 17
 of equation numbers 24
 of fractions 19
 of overlays 23
 of piles 17
arithmetic operators 14
attributes
 ALIGN= 6
 FRAME= 6
 ITALIC= 5
 METRICS= 6
 NOITALIC= 5

B

back inferior
 See ?
back superior
 See ?
BAR keyword 23
BARUNDER keyword 23
big parentheses, brackets, and braces 22
blanks 12
 as word delimiter 13
 required 12
 thrown away 12

bold font 12
braces 13, 22
 sized to fit left/right expression 22
brackets 13, 22
 constructed 22
 grouping 13
 sized to fit left/right expression 22
break character keywords 10
break-character keywords
 minus 14
 plus 14

C

CABOVE keyword 17
centered fractions 19
centered overlays 23
centered piles 17
Commands
 in a formula
 See keywords
 SCRIPT/VS
 See control words
comments 12
constructed brackets 22
control words
 .EQ EQUATION" 33
courses
 self-study v
 training v
COVER keyword 19
COVERLAY keyword 23

D

DCF publications v
 library guide iv
delimiters
 blanks 13
 comment 12
 keyword abbreviation 10
 quoted string 12
describing a formula 3
DFDEF tag 4
DF tag 3, 4, 9
display formulas 9
 controlling alignment of 6
 controlling framing of 6
 controlling italics in 5
 controlling the metrics of the formatter 6
 defaults for 4
 forced by numbering 24

- display formulas (*continued*)
 - how determined 34
 - italics in 5
 - markup for 4, 6
- DOT keyword 23
- .* (comment) delimiter 12
- DOTDOT keyword 23
- DOTDOTUNDER keyword 23
- DOTUNDER keyword 23
- down adjustment 24

E

- ELLIPSIS keyword 15
- embellished characters 23
- .EQ EQUATION™ control word 33
 - parameters of
 - ? 35
 - OFF 33
 - ON 33
 - SET 36
 - SYMBOL 39
 - TAG 33
 - TRACE 41
- EQNO keyword 24
- equation numbers 24
- example notation 9

F

- fine adjustments 24
- fonts
 - bold 12
 - italic 12
 - roman 12
- formula formatter
 - commands
 - See* keywords
 - display 9
 - inline 9
 - keywords
 - See* keywords
 - language 3
 - markup for 3
 - tags for 3
- fractions 19
- FRAME attribute 6
- framing a display formula 6
- FROM keyword 21
- F tag 3, 9
- FUNCTION keyword 26
- functions 26

G

- GML attributes
 - ALIGN= 6
 - FRAME= 6
- GML tags
 - DF 3, 4, 9
 - DFDEF 4
 - F 3, 9
- Greek letters 10, 27
- grouping
 - description of 13
 - in fractions 19
 - in left/right expressions 22
 - in square roots 20
 - of superscripts 16
- grouping brackets 13

H

- HABOVE keyword 17
- HAT keyword 23
- HERE keyword 17

I

- Identification message 35
- inferior
 - See* ?
- inline formulas 9
 - how determined 34
 - markup for 3
- integral 20
- INTEGRAL keyword 20
- INTPLEX keyword 21
- intplexes 20
- ITALIC attribute 5
- italic font 12
- italics 5

K

- keywords
 - ** 16
 - abbreviation of 10
 - ABOVE 17
 - ADJUST 24
 - left, right, up, down 24
 - BAR 23
 - BARUNDER 23
 - break character 10
 - CABOVE 17
 - COVER 19
 - COVERLAY 23
 - DOT 23
 - DOTDOT 23
 - DOTDOTUNDER 23

keywords (*continued*)

DOTUNDER 23
ELLIPSIS 15
EQNO 24
for Greek letters 10, 27
for special characters 28
FROM 21
FUNCTION 26
HABOVE 17
HAT 23
HERE 17
INTEGRAL 20
INTPLEX 21
LABOVE 17
LB 22
LEFT 13, 22
LEQNO 24
LOVER 19
LOVERLAY 23
MIDSUB 16
MIDSUP 16
MINUS 14
OF 21
OPERATOR 15
OVER 19
OVERLAY 23
OVERMARK 23
PLEX 21
PLUS 14
PRESUB 16
PRESUP 16
PRIME 23
PRIMEMARK 23
PRODUCT 20
RABOVE 17
RB 22
REQNO 24
RIGHT 13, 22
ROVER 19
ROVERLAY 23
SQRT 20
SUB 16
SUM 20
SUP 16
TILDE 23
TO 21
UNDERMARK 23
upper- and lowercase in 10
VABOVE 18
VBAR 22
VECTOR 23
WITH 21

L

LABOVE keyword 17
language for describing a formula 3
large parentheses, brackets, and braces 22
LB keyword 22
left adjustment 24
LEFT keyword 13, 22
left-aligned fractions 19
left-aligned overlays 23
left-aligned piles 17
left/right expressions 22
LEQNO keyword 24
Level identification message 35
Level indicator 33
library guide
 for DCF publications iv
limits 21
literal text 12
LOVER keyword 19
LOVERLAY keyword 23

M

markup
 for display formulas 4
 for formulas 3
 for inline formulas 3
markup content separator 4
mathematical functions 26
mathematical operators 14
metabackets 13
metrics 6
METRICS attribute 6
MIDSUB keyword 16
MIDSUP keyword 16
MINUS keyword 14

N

named formulas 29
NAMEF macro 29
NOITALIC attribute 5
notation for examples 9
numbering formulas 24

O

OF keyword 21
OFF parameter of .EQ EQUATION“ 33
ON parameter of .EQ EQUATION“ 33
OPERATOR keyword 15
operators 14
options of .EQ EQUATION“
 ? 35
 OFF 33
 ON 33

options of .EQ EQUATION“ (continued)

SET 36
SYMBOL 39
TAG 33
TRACE 41

ordering publications ii

OVER keyword 19

overlaid characters 23

OVERLAY keyword 23

OVERMARK keyword 23

P

parameters of .EQ EQUATION“ 33, 35, 36, 39, 41

? 35

OFF 33

ON 33

SET 36

SYMBOL 39

TAG 33

TRACE 41

parentheses 13, 22

sized to fit left/right expression 22

% keyword 12

period

keyword abbreviation delimiter 10

markup content separator 4

use to separate gml markup from content 4

piles 17

arbitrary horizontal alignment 17

default alignment of 17

horizontal alignment of 17

variable linespacing 18

PLEX keyword 21

plexes 20

PLUS keyword 14

positioning adjustments 24

PRESUB keyword 16

PRESUP keyword 16

PRIME keyword 23

PRIMEMARK keyword 23

product 20

PRODUCT keyword 20

publications

for DCF v

how to order ii

Q

quoted strings 12

R

RABOVE keyword 17

RB keyword 22

REQNO keyword 24

requests for publications ii

required blank 12

right adjustment 24

RIGHT keyword 13, 22

right-aligned fractions 19

right-aligned overlays 23

right-aligned piles 17

roman font 12

ROVER keyword 19

ROVERLAY keyword 23

S

self-study courses v

SET parameter of .EQ EQUATION“ 36

Special characters 28

SQRT keyword 20

square roots 20

SUB keyword 16

subscripts 16

SUM keyword 20

summation 20

SUP keyword 16

superior

See ?

superscripted superscript 16

superscripts 16

SYMBOL parameter of .EQ EQUATION“ 39

symbols

produced by keywords 11

used in formulas 39

T

TAG parameter of .EQ EQUATION“ 33

tags

DF 3, 4, 9

DFDEF 4

F 3, 9

text formulas

See ?

thin space 12

three dots 15

TILDE keyword 23

TO keyword 21

TRACE parameter of .EQ EQUATION“ 41

training courses v

U

unary plus and minus 14

UNDERMARK keyword 23

up adjustment 24

upper- and lowercase

in keywords 10

V

VABOVE keyword 18
variably-spaced piles 18
VBAR keyword 22
VECTOR keyword 23
vertical bars 22
 sized to fit left/right expression 22

W

WITH keyword 21

Readers' Comments — We'd Like to Hear from You

SCRIPT

Mathematical Formula Formatter

User's Guide

Publication No. S544-3306-00

Use this form to provide comments about this publication, its organization, or subject matter. Understand that IBM may use the information any way it believes appropriate, without incurring any obligation to you. Your comments will be sent to the author's department for the appropriate action. Comments may be written in your language.

Note: IBM publications are not stocked at the location to which this form is addressed. Direct requests for publications or for assistance in using your IBM system, to your IBM representative or local IBM branch office.

	Yes	No
Does the publication meet your needs?	_____	_____
Did you find the information:		
Accurate?	_____	_____
Easy to read and understand?	_____	_____
Easy to retrieve?	_____	_____
Organized for convenient use?	_____	_____
Legible?	_____	_____
Complete?	_____	_____
Well illustrated?	_____	_____
Written for your technical level?	_____	_____
Do you use this publication:		
As an introduction to the subject?	_____	_____
As a reference manual?	_____	_____
As an instructor in class?	_____	_____
As a student in class?	_____	_____
What is your occupation?		

Thank you for your input and cooperation.

Note: You may either send your comments by fax to 1-800-524-1519, or mail your comments. If mailed in the U.S.A., no postage stamp is necessary. For residents outside the U.S.A., your local IBM office or representative will forward your comments.

Comments:

Name

Address

Company or Organization

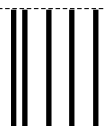
Phone No.



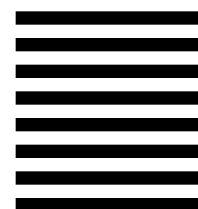
Fold and Tape

Please do not staple

Fold and Tape



NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

Information Development
The IBM Printing Systems Company
Department H7FE Building 004M
P O Box 1900
BOULDER CO 80301-9817



Fold and Tape

Please do not staple

Fold and Tape



Program Number
5748-XX9

File Number
S370-40

S544-3306-00

