

FlexPDE Reference

Version 5
5/25/05

© 2005 PDE Solutions Inc.

Complying with all copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, or otherwise) without the express written permission of PDE Solutions Inc.

PDE Solutions may have patents, patent applications, trademarks, and copyrights or other intellectual property rights covering subject matter in this document. Except as provided in any written license agreement from PDE Solutions Inc., the furnishing of this document does not give you any license to these patents, trademarks, copyrights or other intellectual property.

PDE Solutions, and FlexPDE are either registered trademarks or trademarks of PDE Solutions Inc. in the United States and/or other countries.

Table of Contents

1. Foreword	1	4.3. Coordinates	47
2. Introduction	2	4.4. Variables	49
2.1. Preparing a Descriptor File	3	4.4.1. The THRESHOLD Clause	49
2.2. File Names and Extensions	4	4.4.2. Moving Meshes	50
2.3. Problem Descriptor Structure	4	4.4.3. The SIMPLEX Modifier	51
2.4. Problem Descriptor Format	5	4.5. Global Variables	51
2.5. Case Sensitivity	6	4.6. Definitions	52
2.6. "Include" Files	6	4.6.1. ARRAY Definitions	53
2.7. A Simple Example	7	4.6.2. Parameterized Definitions	54
3. The Elements of a Descriptor	10	4.6.3. STAGED Definitions	55
3.1. Comments	10	4.6.4. POINT Definitions	56
3.2. Reserved Words and Symbols	11	4.6.5. Data Import Definitions	57
3.3. Separators	12	4.6.6. The PASSIVE Modifier	63
3.4. Literal Strings	13	4.6.7. Mesh Control Parameters	64
3.5. Numeric Constants	14	4.7. Initial Values	65
3.6. Built-in Functions	14	4.8. Equations	66
3.6.1. Analytic Functions	15	4.8.1. Association between Equations, Variables and Boundary Conditions	67
3.6.2. Non-Analytic Functions	15	4.8.2. Modal Analysis and Associated Equations	67
3.6.3. Unit Functions	17	4.8.3. Moving Meshes	68
3.6.4. String Functions	17	4.9. Constraints	69
3.6.5. The FIT Function	18	4.10. Extrusion	70
3.6.6. The LUMP Function	19	4.11. Boundaries	71
3.6.7. The RAMP Function	19	4.11.1. Points	72
3.6.8. The SAVE Function	20	4.11.2. Boundary Paths	72
3.6.9. The SUM Function	21	4.11.3. Regions	74
3.6.10. The SWAGE Function	22	4.11.4. Excludes	79
3.6.11. The VAL and EVAL functions	23	4.11.5. Features	80
3.7. Operators	23	4.11.6. Ordering Regions	80
3.7.1. Arithmetic Operators	23	4.11.7. Numbering Regions	81
3.7.2. Relational Operators	24	4.11.8. Fillets and Bevels	81
3.7.3. String Operators	24	4.11.9. Boundary Conditions	82
3.7.4. Vector Operators	25	4.11.10. Fixed Points	90
3.7.5. Differential Operators	26	4.12. Front	90
3.7.6. Integral Operators	28	4.13. Resolve	91
3.8. Predefined Elements	32	4.14. Time	92
3.9. Expressions	33	4.15. Monitors and Plots	93
3.10. Repeated Text	34	4.15.1. Graphics Display and Data Export Specifications	94
4. The Sections of a Descriptor	37	4.15.2. Graphic Display Modifiers	97
4.1. Title	37	4.15.3. Controlling the Plot Domain	104
4.2. Select	37	4.15.4. Reports	107
4.2.1. Mesh Generation Controls	38	4.15.5. Window Tiling	107
4.2.2. Solution Controls	39	4.15.6. Monitors in Steady State Problems	108
4.2.3. Global Graphics Controls	44		

4.15.7. Monitors and Plots in Time Dependent Problems	108
4.15.8. Hardcopy.....	109
4.15.9. Graphics Export	109
4.15.10. Examples	110
4.16. Histories.....	110
4.17. End	111
5. Batch Processing	112
6. Running FlexPDE from the Command Line.....	113

1. Foreword

This document presents a detailed description of the components of FlexPDE problem descriptors. No attempt is made here to give tutorial explanations of the use of these components. See the companion volumes "Getting Started" for user interface information and "User Guide" for tutorial guidance in the use of FlexPDE.

2. Introduction

FlexPDE is a script-driven system. It reads a description of the equations, domain, auxiliary definitions and graphical output requests from a text file referred to as a "problem descriptor" or "script".

The problem descriptor file can be created either with the editor facility in FlexPDE, or with any other ASCII text editor. A word processor can be used only if there is an optional "pure text" output, in which formatting codes have been stripped from the file.

Problem descriptors use an easy to learn natural language originally developed by Robert G. Nelson. The language is also described in Dr. Gunnar Backstrom's book, "Fields of Physics by Finite Element Analysis - An Introduction". The language has been used in the PDS2 system at Lawrence Livermore National Lab and in the PDEase2 system.

As FlexPDE has evolved, a number of extensions have been added to extend its processing capabilities. The language as currently implemented in FlexPDE is described in this document.

While similar in some ways to a computer programming language, the natural language used in problem descriptors is much simpler. Most intermediate level college students, engineers, and scientists who have had at least an introductory course in partial differential equations can sufficiently master the language in less than an hour to prepare simple problem descriptor files and begin solving problems of their own devising.

The FlexPDE problem descriptor language can be viewed as a shorthand language for creating Finite Element models. The statements of the descriptor provide the information necessary for FlexPDE to assemble a numerical process to solve the problem.

It is important to understand that the language of FlexPDE problem descriptors is a *relational* language, not a *procedural* one. The user describes how the various components of the system relate to one another. He does *not* describe a sequence of steps to be followed in forming the solution, as would be done in a language such as C. Based on the relations between problem elements, FlexPDE decides on the sequence of steps needed in finding the solution.

FlexPDE makes various assumptions about the elements of the problem descriptor.

For example, if a variable is named in the VARIABLES section, it is assumed

- that this variable is a scalar field which takes on values over the domain of the problem,
- that it will be approximated by a finite element interpolation between the nodes of a computation mesh,
- that the values of the variable are continuous over the domain, and
- that a partial differential equation will be defined describing the behavior of the variable.

If a definition appears in the DEFINITIONS section, it is assumed that the named quantity

- is ancillary to the PDE system,
- that it may be discontinuous over the domain,
- that it does not (necessarily) obey any PDE.

In the chapters that follow, we describe in detail the rules for constructing problem descriptors.

2.1. Preparing a Descriptor File

Problem descriptor files for use with FlexPDE are most easily prepared and edited using FlexPDE's built-in editor.

To begin a new descriptor file, simply click "File | New Script" from the FlexPDE main menu bar.

To edit an existing descriptor, click "File | Open Script" instead.

A convenient way to create a new descriptor is to start with a copy of an existing descriptor for a similar problem and to modify it to suit the new problem conditions.

FlexPDE's built-in editor is similar to the Windows Notepad editor and produces a pure ASCII text file without any imbedded formatting characters. Descriptor files can also be prepared using any ASCII text

editor or any editor capable of exporting a pure ASCII text file. Descriptor files prepared with word processors that embed formatting characters in the text will cause FlexPDE to report parsing errors.

The built-in editor in FlexPDE uses syntax highlighting to enhance the readability of the user's script. Recognized keywords are displayed in red, comments in green, and text strings in blue.

2.2. File Names and Extensions

A problem descriptor file can have any name which is consistent with the host operating system.

Even though permitted by some operating systems, names with imbedded blank characters should be avoided.

It is best to choose a name that is descriptive of the problem.

Problem descriptor files must have the extension '.pde'.

When saving a file using FlexPDE's built-in editor, FlexPDE will automatically add the extension '.pde'.

When using a separate or off-line editor, be sure to give the file a '.pde' extension instead of the default extension.

2.3. Problem Descriptor Structure

Problem descriptors organize a problem by breaking it into sections of related items.

Each section is headed by a proper name followed by one or more statements which define the problem.

The permitted section names are:

TITLE	- defines the problem title
SELECT	- sets various options and controls
COORDINATES	- defines the coordinate system
VARIABLES	- names the problem variables
DEFINITIONS	- defines ancillary quantities and parameters
INITIAL VALUES	- sets initial values of variables
EQUATIONS	- defines the partial differential equation system
CONSTRAINTS	- defines optional integral constraints
EXTRUSION	- extends the domain to three dimensions
BOUNDARIES	- describes the 2D or projected 3D domain
RESOLVE	- optionally supplements mesh control
FRONT	- optionally supplements mesh control for advancing fronts
TIME	- defines the time domain
MONITORS	- selects interim graphic display
PLOTS	- selects final graphic display
HISTORIES	- selects time-summary displays
END	- identifies the end of the descriptor

The number of sections used in a particular problem descriptor can vary, subject only to the requirement that all files must contain a Boundaries section and an End section.

While some flexibility exists in the placement of these sections, it is suggested that the user adhere to the ordering described above.

DEFINITIONS and **SELECT** can appear more than once.

Because descriptors are dynamically processed from top to bottom, they cannot contain forward references. Definitions may refer to variables and other defined names, provided these variables and names have been defined in a preceding section or previously in the same section.

2.4. Problem Descriptor Format

While not strictly required, we suggest use of the following indentation pattern for all problem descriptors:

```

section 1
  statement
section 2
  statement 1
  statement 2
  *
  *
section 3
  statement 1
  statement 2
  *
  *

```

This format is easy for both the person preparing the file and for others to read and understand.

2.5. Case Sensitivity

With the exception of quoted character strings, which are reproduced exactly as they appear in a problem descriptor, words, characters and other text items used in problem descriptors are NOT case sensitive.

Upper case letters and lower case letters are equivalent.

The text items **variables**, **VARIABLES**, **Variables** and mixed case text like **VaRiAbles** are all equivalent.

2.6. "Include" Files

FlexPDE supports the C-language mechanism of including external files in the problem descriptor. The statement

```
#INCLUDE "filename"
```

will cause the named file to be included bodily in the descriptor in place of the **#INCLUDE "filename"** statement.

If the file does not reside in the same folder as the descriptor, the full path to the file must be given.

An include statement can be placed anywhere in the descriptor, but for readability it should be placed on its own line.

This facility can be used to insert common definition groups in several descriptors.

Note: Although FlexPDE is not case sensitive, the operating system which is being asked for the included file may be case sensitive. The quoted file name must conform to the usage of the operating system.

2.7. A Simple Example

As a preview example to give the flavor of a FlexPDE descriptor file, we will construct a model of heatflow on a square domain.

The heatflow equation is

$$\text{div}(K*\text{grad}(\text{Temp})) + \text{Source} = 0$$

This equation is satisfied by the function

$$\text{Temp} = \text{Const} - x^2 - y^2$$

as long as K is constant and Source = 4*K.

We define a square region of material of conductivity K = 1, with a uniform heat source of 4 heat units per unit area.

We further specify the boundary value

$$\text{Temp} = 1 - x^2 - y^2$$

Since we know the analytic solution, we can compare the accuracy of the FlexPDE solution.

The text of the descriptor is as follows:

```
{ *****
  SIMPLE.PDE
  This sample demonstrates the simplest application of FlexPDE
  to
  heatflow problems.
  ***** }
```

TITLE "Simple Heatflow"

VARIABLES

temp { Identify "Temp" as the system variable }

DEFINITIONS

k = 1 { declare and define the conductivity }
 source = 4 { declare and define the source }
 texact = 1-x^2-y^2 { exact solution for reference }

INITIAL VALUES

temp = 0 { unimportant in linear steady-state problems,
 but necessary for time-dependent or nonlinear
 systems }

{ define the heatflow equation :}

EQUATIONS

div(k*grad(temp)) + source = 0

{ define the problem domain: }

BOUNDARIES

REGION 1 { ... only one region }
 { specify Dirichlet boundary at exact solution: }
 VALUE(temp)=texact
 START(-1,-1) { specify the starting point }
 LINE TO (1,-1) { walk the boundary }
 TO (1,1)
 TO (-1,1)
 TO CLOSE { bring boundary back to starting point }

MONITORS

CONTOUR(temp) { show the Temperature during solution }

PLOTS { write these plots to disk at completion: }

CONTOUR(temp) { show the solution }
 SURFACE(temp) { show a surface plot as well }
 { display the solution error :}
 CONTOUR(temp-texact) AS "Error"
 { show a vector flow plot: }
 VECTOR(-dx(temp),-dy(temp)) AS "Heat Flow"

END { end of descriptor file }

3. The Elements of a Descriptor

The problem descriptors or 'scripts' which describe the characteristics of a problem to FlexPDE are made up of a number of basic elements, such as names and symbols, reserved words, numeric constants, etc. These elements are described in the sections that follow.

3.1. Comments

Problem descriptors can be annotated by adding comments.

Multi-line comments can be placed anywhere in the file. Multi-line comments are formed by enclosing the desired comments in either curly brackets { and } or the paired symbols /* and */. Comments can be nested, but comments that begin with a curly bracket must end with a curly bracket and comments that begin with /* must end with */.

Example:

```
{ this is a comment  
so is this.  
}
```

End-of-line comments are introduced by the exclamation mark !. End-of-line comments extend from the ! to the end of the line on which they occur. Placing the line comment symbol ! at the beginning of a line effectively removes the whole line from the active portion of the problem descriptor, in a manner similar to 'rem' at the beginning of a line in a DOS batch file or "//" in C++.

Example:

```
! this is a comment  
this is not
```

Comments can be used liberally during script development to temporarily remove lines from a problem descriptor. This aids in localizing errors or focusing on specific aspects of a problem.

3.2. Reserved Words and Symbols

FlexPDE assigns specific meanings and uses to a number of predefined 'reserved' words and symbols in descriptors.

Except when they are included as part of a comment or a literal string, these words may only be used for their assigned purpose.

ABS	ALIAS	AND
ANGLE	ARC	ARCCOS
ARCSIN	ARCTAN	AS
AT	ATAN2	
BESSJ	BESSY	BINTEGRAL
BOUNDARIES	BY	
CDF	CENTER	CLOSE
CONIC	CONSTRAINTS	CONTOUR
COORDINATES	COS	COSH
CROSS	CURL	
DEBUG	DEFINITIONS	DEGREES
DEL2	DELTAT	DIFF
DIR	DIRECTION	DIV
DNORMAL	DOT	DTANGENTIAL
ELEVATION	ELSE	END
ENDTIME	EQUATIONS	ERF
ERFC	EXCLUDE	EXP
EXPINT	EXPORT	EXTRUSION
FEATURE	FILE	FINISH
FIT	FIXED	FOR
FROM		
GAMMAF	GLOBALMAX	GLOBALMIN
GRAD	GRID	
HISTORIES	HISTORY	
IF	INITIAL	INTEGRAL
INTEGRATE	INTSTRING	
JACOBIAN	JUMP	
LAMBDA	LAYER	LAYERED
LIMITED	LINE	LIST
LN	LOAD	LOG10
MAGNITUDE	MAX	MESH_DENSITY
MESH_SPACING	MIN	MOD
MONITORS	MOVE	
NATURAL	NEUMANN	NORMAL

NOT		
OFF	ON	OR
PERIODIC		
PI	PLOTS	POINT
PRINT	PRINTONLY	
RADIANS	RADIUS	RAMP
REGION	REPEAT	REPORT
RESOLVE		
SCALAR	SELECT	SIGN
SIMPLEX	SIN	SINH
SPLINE	SPLINETABLE	SPLINETABLEDEF
SQRT	STAGE	STAGED
START	SUM	SUMMARY
SURFACE	SWAGE	
TABLE	TAN	TANGENTIAL
TANH	TECPLOT	THEN
TIME	TITLE	TO
THRESHOLD	TRANSFER	TRANSFERMESH
UNORMAL	UPULSE	URAMP
USTEP		
VAL	VALUE	VALUES
VARIABLES	VECTOR	VELOCITY
VERSUS	VIEWANGLE	VIEWPOINT
VOID	VTK	VTKLIN
XCOMP		
YCOMP		
ZCOMP	ZOOM	

3.3. Separators

White Space

Spaces, tabs, and new lines, frequently referred to as "white space", are treated as separators and may be used freely in problem descriptors to increase readability. Multiple white spaces are treated by FlexPDE as a single white space.

Commas

Commas are used to separate items in a list, and should be used only where explicitly required by the descriptor syntax.

Semicolons

Semicolons are reserved to signify the end of a label or statement when it is not otherwise clear where the label or statements ends. If, while parsing equations in a problem descriptor, FlexPDE encounters two mathematical quantities separated by a white space without an intervening mathematical operator it will interpret this to mean that one equation has ended and another equation is about to begin. If, on the other hand, FlexPDE encounters two mathematical quantities with an intervening mathematical operator it will interpret this to mean a continuation of the equation even if the terms are placed on separate lines. If a new equation beginning with a mathematical operator (such as the negation operator '-') follows another equation, the first equation must be terminated with a semicolon to keep FlexPDE from interpreting the two equations as one equation.

3.4. Literal Strings

Literal strings are used in problem descriptors to provide optional user defined labels, which will appear on softcopy and hardcopy outputs.

The label that results from a literal string is reproduced on the output exactly (including case) as entered in the corresponding literal string.

Literal strings are formed by enclosing the desired label in either single or double quote marks. Literal strings that begin with a double quote mark must end in a double quote mark, and literal strings that begin with a single quote mark must end in a single quote mark.

A literal string may consist of any combination of alphanumeric characters, separators, reserved words, and/or symbols including quote marks, provided only that strings that begin with a double quote mark may contain only single quote marks and strings that begin with a single quote mark may contain only double quote marks.

Example:

TITLE "This is a literal 'string' used as a problem title"

3.5. Numeric Constants

Integers

Integers must be of the form XXXXXX where X is any decimal digit from 0 to 9. Integer constants can contain up to 9 digits.

Decimal Numbers

Decimal numbers must be of the form XXXXX.XXX where X is any decimal digit from 0 to 9 and '.' is the decimal separator. Decimal numbers must not include commas ','. Using the European convention of a comma ',' as a decimal separator will result in an error. Commas are reserved as item separators. Decimal numbers may include zero to nine digits to the left of the decimal separator and up to a total of 308 digits total. FlexPDE considers only the first fifteen digits as significant.

Engineering Notation Numbers

Engineering notation numbers must be of the form XXXXXEsYYY where X is any digit from 0 to 9 or the decimal separator '.', Y is any digit from 0 to 9, E is the exponent separator, and s is an optional sign operator. Engineering notation numbers must not include commas ','. Using the European convention of a comma ',' as a decimal separator will result in an error. Commas are reserved as item separators. The number to the left of the exponent separator is treated as a decimal number and the number to the right of the exponent separator is treated as an integer and may not contain a decimal separator or more than 3 digits. The range of permitted engineering notation numbers is 1e-307 to 1e308.

3.6. Built-in Functions

Functions and Arguments

All function references must include at least one argument. Arguments can be either numerical constants or expressions that evaluate to numerical values. The following functions are supported in problem descriptors:

3.6.1. Analytic Functions

The following analytic functions are supported by FlexPDE:

<u>Function</u>	<u>Comments</u>
ABS(x)	Absolute value
ARCCOS(x)	returns radians
ARCSIN(x)	"
ARCTAN(x)	"
ATAN2(y,x)	Arctan(y/x)
BESSJ(order,x)	Bessel Function J
BESSY(order,x)	Bessel Function Y
COS(x)	x is angle in radians *
COSH(x)	Hyperbolic cosine
ERF(x)	Error Function
ERFC(x)	Complementary Error Function
EXP(x)	Exponential function
EXPINT(x)	Exponential Integral Ei(x) for real x>0 **
EXPINT(n,x)	Exponential Integral En(x) for n>=0, real x>0 **
GAMMAF(x)	Gamma function for real x>0
GAMMAF(a,x)	Incomplete gamma function for real a>0, x>0
LOG10(x)	Base-10 logarithm
LN(x)	Natural logarithm
SIN(x)	x is angle in radians *
SINH(x)	Hyperbolic sine
SQRT(x)	
TAN(x)	x is angle in radians *
TANH(x)	Hyperbolic tangent

* Use for example COS(x DEGREES) to convert arguments to radians.

** as defined in Abramowitz & Stegun, "Handbook of Mathematical Functions".

Examples:

see Samples | Misc | Funtest.pde

3.6.2. Non-Analytic Functions

The following non-analytic functions are supported in FlexPDE:

MAX(arg1,arg2)

The maximum function requires two arguments. **MAX** is evaluated on a point by point basis and is equal to the larger of the two arguments at each point.

MIN(arg1,arg2)

The minimum function requires two arguments. **MIN** is evaluated on a point by point basis and is equal to the lessor of the two arguments at each point.

MOD(arg1,arg2)

The modulo function requires two arguments. **MOD** is evaluated on a point by point basis and is equal to the remainder of (**arg1**/**arg2**) at each point.

GLOBALMAX(arg)

The global maximum function requires one argument. **GLOBALMAX** is equal to the largest value of the argument over the problem domain. **GLOBALMAX** is tabulated and re-evaluated when components of the argument change.

GLOBALMAX_X(arg)**GLOBALMAX_Y(arg)****GLOBALMAX_Z(arg)**

The specified coordinate of the global maximum is returned. Global searches are tabulated by argument expression, and repeated calls to **GLOBALMAX** and its related coordinates do not cause repeated evaluation.

GLOBALMIN(arg)

The global minimum function requires one argument. **GLOBALMIN** is equal to the smallest value of the argument over the problem domain. **GLOBALMIN** is tabulated and re-evaluated when components of the argument change.

GLOBALMIN_X(arg)**GLOBALMIN_Y(arg)****GLOBALMIN_Z(arg)**

The specified coordinate of the global minimum is returned. Global searches are tabulated by argument expression, and repeated calls to **GLOBALMIN** and its related coordinates do not cause repeated evaluation.

RANDOM(arg)

The random function requires one argument. The result is a pseudo-random number uniformly distributed in (0,arg). The only reasonable application of the **RANDOM** function is in initial values. Use in other contexts will probably result in convergence failure.

SIGN(arg)

The sign function requires one argument. **SIGN** is equal to 1 if the argument is positive and -1 if the argument is negative.

3.6.3. Unit Functions

The following unit-valued functions are supported in FlexPDE:

USTEP(arg)

The unit step function requires one argument. **USTEP** is 1 where the argument is positive and 0 where the argument is negative. For example, **USTEP(x-x0)** is a step function at $x=x_0$.

UPULSE(arg1,arg2)

The unit pulse function requires two arguments. **UPULSE** is 1 where **arg1** is positive and **arg2** is negative and 0 everywhere else. **UPULSE(t-t0, t-t1)** is a pulse from t_0 to t_1 if $t_1 > t_0$.

URAMP(arg1,arg2)

The unit ramp function requires two arguments. **URAMP** is like **UPULSE**, except it builds a ramp instead of a rectangle..

Examples:

Samples | Misc | Ufuntest.pde

3.6.4. String Functions

FlexPDE provides minimal support for dynamically constructing text strings.

\$integer (i.e. <dollar> integer)

This function returns a text string representing the integer value **integer**. This function may be used in conjunction with the

concatenation operator "+" to build boundary or region names. For example

```
REPEAT i=1 to 4 do
  START "LOOP"+$i (x,y)
  <path_info>...
ENDREPEAT
```

This is equivalent to

```
START "LOOP1" (x,y) <path_info> ...
START "LOOP2" (x,y) <path_info> ...
START "LOOP3" (x,y) <path_info> ...
START "LOOP4" (x,y) <path_info> ...
```

Example:

See "Samples | Misc | ArrayRepeat.pde"

3.6.5. The FIT Function

The following two forms may be used to compute a finite-element interpolation of an arbitrary argument:

result = FIT(expression)

computes a Finite Element fit of the given **expression** using the current computational mesh and basis. Nodal values are computed to return the correct integral over each mesh cell.

result = FIT(expression,weight)

as with **FIT(expression)**, but with a smoothing diffusion with coefficient equal to **weight** (try 0.1 or 1.0, and modify to suit).

weight may be an arbitrary expression, involving spatial coordinates, time, or variables of the computation. In this way it can be used to selectively smooth portions of the mesh. The value of **weight** has a well-defined meaning: it is the spatial wavelength over which variations are damped: spatial variations with wavelength much smaller than **weight** will be smoothed, while spatial variations with wavelength much greater than **weight** will be relatively unmodified.

Note: FIT() builds a continuous representation of the data across the entire domain, and cannot preserve discontinuities in the fitted data. In some cases, multiplying the data by an appropriate material parameter can result in a continuous function appropriate for fitting.

FIT() may be used to smooth noisy data, to block ill-behaved functions from differentiation in the derivative computation for Newton's method, or to avoid expensive re-computation of complex functions.

See also the SAVE function, in which nodal values are directly computed.

Example:

Samples | Misc | fitweight.pde

3.6.6. The LUMP Function

The **LUMP** function creates a field on the finite element mesh, and saves a single value of the argument expression in each cell of the finite element mesh. The value stored for each cell is the average value of the argument expression over the cell, and is treated as a constant over the cell.

The **LUMP** function may be used to block ill-behaved functions from differentiation in the derivative computation for Newton's method, or to avoid expensive re-computation of complex functions.

The normal use for **LUMP** is in the **DEFINITIONS** section, as in

name = LUMP (expression)

Example:

Samples | Misc | lump.pde

3.6.7. The RAMP Function

The **RAMP** function is a modification of the **URAMP** function, intended to make the usage more nearly like an **IF..THEN** statement.

It has been introduced to provide an alternative to discontinuous functions like **USTEP** and the discontinuous **IF..THEN** construct.

Discontinuous switching can cause serious difficulties, especially in time dependent problems, and is strongly discouraged.

FlexPDE is an adaptive system. Its procedures are based on the assumption that by making timesteps and/or cell sizes smaller, a scale can be found at which the behavior of the solution is representable by polynomials. Discontinuities do not satisfy this assumption. A discontinuity is a discontinuity, no matter how close you look. Instantaneous turn-on or turn-off introduces high-frequency spatial or temporal components into the solution, including those which are far beyond the physical limits of real systems to respond. This makes the computation slow and possibly physically meaningless.

The **RAMP** function generates a smooth transition from one value to another, with a specified transition width. It can be thought of as a "fuzzy IF", and has a usage very similar to an IF.. THEN, but without the harsh switching characteristics.

The form is:

value = RAMP(expression, left_value, right_value, width)

This expression is logically equivalent to

value = IF expression < 0 THEN left_value ELSE right_value

except that the transition will be linear over **width**.

See the SWAGE function for a similar function with both smooth value and derivative.

Example:

see "Samples | Misc | Swagetest.pde" for a picture of the SWAGE and RAMP transitions and their derivatives.

3.6.8. The SAVE Function

The **SAVE** function creates a field on the finite element mesh, and saves the values of the argument expression at the nodal points for subsequent interpolation. **SAVE** builds a continuous representation of the data

across the entire domain, and cannot preserve discontinuities in the saved data.

The **SAVE** function may be used to block ill-behaved functions from differentiation in the derivative computation for Newton's method, or to avoid expensive re-computation of complex functions.

The normal use for **SAVE** is in the **DEFINITIONS** section, as in

name = SAVE (expression)

Note: **SAVE()** builds a continuous representation of the data across the entire domain, and cannot preserve discontinuities in the fitted data. In some cases, multiplying the data by an appropriate material parameter can result in a continuous function appropriate for saving.

See the **FIT()** function for a similar function with integral conservation and variable smoothing capabilities.

3.6.9. The SUM Function

The **SUM** function produces the sum of repetitive terms. The form is:

value = SUM(name, initial, final, expression)

The **expression** argument is evaluated and summed for **name = 0,1,2,...final**.

For example, the statement:

source = SUM(i,1,10,exp(-i))

forms the sum of the exponentials $\exp(-1)+\exp(-2)+\dots+\exp(-10)$.

The **SUM** function may be used with data **ARRAYs**, as in

DEFINITIONS

A = ARRAY(1,2,3,4,5,6,7,8,9,10)

source = SUM(i,1,10,A[i])

Example:

3.6.10. The SWAGE Function

The **SWAGE** function has been introduced to provide an alternative to discontinuous functions like **USTEP** and the discontinuous **IF..THEN** construct. Discontinuous switching can cause serious difficulties, especially in time dependent problems, and is strongly discouraged.

FlexPDE is an adaptive system. Its procedures are based on the assumption that by making timesteps and/or cell sizes smaller, a scale can be found at which the behavior of the solution is representable by polynomials. Discontinuities do not satisfy this assumption. A discontinuity is a discontinuity, no matter how close you look. Instantaneous turn-on or turn-off introduces high frequency spatial or temporal components into the solution, including those which are far beyond the physical limits of real systems to respond. This makes the computation slow and possibly physically meaningless.

The **SWAGE** function generates a smooth transition from one value to another, with a specified transition width. It also has smooth derivatives. It can be thought of as a "fuzzy IF", and has a usage very similar to an **IF.. THEN**, but without the harsh switching characteristics.

The form is:

value = SWAGE(expression, left_value, right_value, width)

This expression is logically equivalent to

value = IF expression < 0 THEN left_value ELSE right_value

except that the transition will be smeared over **width**.

See the **RAMP** function for a similar function which is smooth in value, but not in derivative.

Example:

see "Samples | Misc | Swagetest.pde" for a picture of the **SWAGE** and **RAMP** transitions and their derivatives.

3.6.11. The VAL and EVAL functions

There are two ways to evaluate an arbitrary expression at selected coordinates.

value = VAL(expression, x, y)
value = VAL(expression, x, y, z)

The value of **expression** is computed at the specified coordinates. *The coordinates must be constants.*

This form allows FlexPDE to compute implicit couplings between computation nodes referencing and evaluating the value. The value is computed and stored at each phase of the solution process, allowing efficient reference in many computations.

value = EVAL(expression, x, y)
value = EVAL(expression, x, y, z)

The Value of **expression** is computed at the specified coordinates. *The coordinates may be dynamically variable.*

This form does NOT allow FlexPDE to compute implicit couplings between computation nodes referencing and evaluating the value. The value is recomputed at each reference, possibly leading to increased run time.

3.7. Operators

3.7.1. Arithmetic Operators

The following customary symbols can be use in arithmetic expressions:

Operator Action

-	Unary negate, Forms the negative of a single operand
+	Binary add, Forms the sum of two operands
-	Binary subtract, Forms the difference of two operands
*	Binary multiply, Forms the product of two operands
/	Binary divide, Divides the first operand by the second
^	Binary power, Raises the first operand to the second

power
 ** Binary power, Alternative to ^

3.7.2. Relational Operators

The following operators may be used in constructing conditional expressions:

Relational Operators

<u>Operator</u>	<u>Definition</u>
=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to

Relational Combinations

<u>Operator</u>	<u>Definition</u>
AND	Both conditions true
OR	Either condition true
NOT	(Unary) reverses condition

Assignment Operator

In addition to its use as an equal operator, problem descriptors use the '=' symbol to assign (associate) values functions and expressions with defined names.

3.7.3. String Operators

The following operators can be used in expressions that construct string constants:

<u>Operator</u>	<u>Action</u>
+	Binary add, Forms the catenation of two text-string operands

3.7.4. Vector Operators

The following operators perform various transformations on vector quantities.

Vectors in three-dimensional problems are assumed to have one component in each of the coordinate directions.

Vectors in two-dimensional problems are assumed to have two components both of which lie in the plane of the problem. In some cases a third component is inferred to be a scalar, such as the result of a cross-product or a curl.

CROSS (vector1, vector2)

Forms the cross product of two **vectors** and returns the resulting vector. In 2D, **CROSS** returns a scalar value equal to the component of the vector cross product normal to the problem plane.

DOT (vector1, vector2)

Forms the dot product of two **vectors** and returns a scalar value equal to the magnitude of the vector dot product.

MAGNITUDE (vector)

Returns a scalar equal to the magnitude of a **vector** argument.

MAGNITUDE (argx, argy [, argz])

Returns a scalar equal to the magnitude of a vector whose components are **argx** and **argy** (and possibly **argz** in 3D).

NORMAL (vector)

Returns a scalar equal to the component of a **vector** argument normal to a boundary.*

NORMAL (argx, argy [, argz])

Returns a scalar equal to the boundary-normal component of a vector whose components are **argx** and **argy** (and possibly **argz** in 3D).*

TANGENTIAL(arg)

Returns a scalar equal to the component of a **vector** argument tangential to a boundary.*

TANGENTIAL (argx, argy [, argz])

Returns a scalar equal to the boundary-tangential component of a vector whose components are **argx** and **argy** (and possibly **argz** in 3D).*

VECTOR (argx, argy [, argz])

Constructs a **vector** whose components are the scalar arguments.

XCOMP (vector)

Returns a scalar whose value is the first component of the vector argument (regardless of the names of the coordinates).

YCOMP (vector)

Returns a scalar whose value is the second component of the vector argument (regardless of the names of the coordinates).

ZCOMP (vector)

Returns a scalar whose value is the third component of the vector argument, if it exists (regardless of the names of the coordinates).

* Note: **NORMAL** and **TANGENTIAL** operators may only be used in boundary condition definitions or in boundary plots or integrals.

3.7.5. Differential Operators

Differential operator names are constructed from the coordinate names for the problem, either as defined by the user, or as default names.

First derivative operators are of the form "D<name>", where <name> is the name of the coordinate.

Second-derivative operators are of the form "D<name1><name2>".

In the default 2D Cartesian case, the defined operators are "DX", "DY", "DXX", "DXY", and "DYY".

All differential operators are expanded internally into the proper forms for the active coordinate system of the problem.

D<n> (arg)

First order partial derivative of arg with respect to coordinate <n>, eg. DX(arg).

D<n><m> (arg)

Second order partial derivative of arg with respect to coordinates <n> and <m>, eg. DXY(arg).

DIV (arg)

Divergence of vector **arg**

DIV (argx, argy {, argz })

Divergence of the vector whose components are **argx** and **argy** (and possibly **argz** in 3D).

GRAD (arg)

Gradient of scalar **arg**

CURL (arg)

Curl of vector **arg**. In 3D, returns the vector curl; in 2D, returns a scalar value equal to the magnitude of the curl, which necessarily must lie normal to the computation plane.

CURL (scalar_arg)

Curl of a **scalar_arg** (2D only). Assumes **arg** to be the magnitude of a vector normal to the computation plane, and returns a vector result in the computation plane.

CURL (argx, argy {, argz })

Curl of a vector whose components in the computation plane are **argx** and **argy** (and possibly **argz** in 3D).

DEL2 (scalar_arg)

Laplacian of **scalar_arg**. Equivalent to DIV(GRAD(arg)).

3.7.6. Integral Operators

Integrals may be formed over volumes, surfaces or lines. The specific interpretation of the integral operators depends on the coordinate system of the current problem. Integral operators can treat only scalar functions as arguments. You cannot integrate a vector field.

Examples

Samples | Steady_State | Heatflow | HeatBdry.pde
Samples | Misc | 3D_Integrals.pde
Samples | Misc | Constraints | Bdry_Constraint.pde
Samples | Misc | Constraints | 3D_Constraint.pde
Samples | Misc | Constraints | 3D_Surf_Constraint.pde
Samples | Misc | Tintegral.pde

3.7.6.1. Time Integrals

The operators **TINTEGRAL** and **TIME_INTEGRAL** are synonymous, and perform explicit time integration of arbitrary scalar values from the problem start time to the current time:

TINTEGRAL (integrand)
TIME_INTEGRAL (integrand)

Note: *This operator cannot be used to create implicit linkage between variables. Use a GLOBAL VARIABLE instead.*

3.7.6.2. Line Integrals

The operators **BINTEGRAL** and **LINE_INTEGRAL** are synonymous, and perform line integrations.

The integral is always taken with respect to distance along the line or curve.

At present, line integrals are only meaningful in 2D problems. General 3D line integrals are not yet implemented.

In 2D Cartesian geometry, **LINE_INTEGRAL** is the same as **SURF_INTEGRAL**.

In 2D cylindrical geometry, **SURF_INTEGRAL** will contain the $2\pi r$ weighting, while **LINE_INTEGRAL** will not.

BINTEGRAL (integrand, named_boundary)
LINE_INTEGRAL (integrand, named_boundary)

The boundary specification may be omitted, in which case the entire outer boundary is implied.

Line integrals may be further qualified by specifying the region in which the evaluation is to be made:

LINE_INTEGRAL (integrand, named_boundary, named_region)

named_region must be one of the regions bounded by the selected boundary.

3.7.6.3. 2D Volume Integrals

The synonymous prototype forms of volume integral functions in 2D are:

INTEGRAL (integrand, region)
VOL_INTEGRAL (integrand, region)

Here **region** can be specified by number or name, or it can be omitted, in which case the entire domain is implied.

In two-dimensional Cartesian problems, the volume element is formed by extending the two-dimensional cell a single unit in the Z-direction, so that the volume integral is the same as the area integral in the coordinate plane.

In two-dimensional cylindrical problems, the volume element is formed as $2\pi r dr dz$, so that the volume integral is NOT the same as the area integral in the coordinate plane. For the special case of 2D cylindrical geometry, the additional operator

AREA_INTEGRAL (integrand, region)

computes the area integral of the integrand over the indicated region (or the entire domain) without the $2\pi r$ weighting.

3.7.6.4. 3D Volume Integrals

The synonymous prototype forms of volume integral functions in 3D are:

INTEGRAL (integrand, region, layer)
VOL_INTEGRAL (integrand, region, layer)

Here **layer** can be specified by number or name, or it can be omitted, in which case the entire layer stack is implied.

region can also be specified by number or name, or it can be omitted, in which case the entire projection plane is implied.

If **region** is omitted, then **layer** must be specified *by name* or omitted. If both **region** and **layer** are omitted, the entire domain is implied.

For example,

INTEGRAL(integrand, region, layer) means the integral over the subregion contained in the selected region and layer.

INTEGRAL(integrand, named_layer) means the integral over all regions of the named layer.

INTEGRAL(integrand, region) means the integral over all layers of the selected region.

INTEGRAL(integrand) means the integral over the entire domain.

3.7.6.5. 2D Surface Integrals

The synonymous prototype forms of surface integral functions in 2D are:

SINTEGRAL (integrand, named_boundary)
SURF_INTEGRAL (integrand, named_boundary)

Here **named_boundary** may be specified *by name*, or it can be omitted, in which case the entire outer boundary of the domain is implied.

In two-dimensional Cartesian problems, the surface element is formed by extending the two-dimensional line element a single unit in the Z-direction, so that the surface element is $dl \cdot 1$. In this case, the surface integral is the same as the line integral.

In two-dimensional cylindrical problems, the surface element is formed as $2\pi r \cdot dl$, so the surface integral is NOT the same as the line integral.

The region in which the evaluation is made can be controlled by providing a third argument, as in

SURF_INTEGRAL (integrand, named_boundary, named_region)

named_region must be one of the regions bounded by the selected surface.

3.7.6.6. 3D Surface Integrals

In three-dimensional problems, there are several forms for the surface integral:

1. Integrals over extrusion surfaces are selected by surface name or number and qualifying region name or number:

SINTEGRAL (integrand, surface, region)
SURF_INTEGRAL (integrand, surface, region)

If **region** is omitted, the integral is taken over all regions of the specified surface.

If both **surface** and **region** are omitted, the integral is taken over the entire outer surface of the domain.

Integrals of this type may be further qualified by selecting the layer in which the evaluation is to be made:

SURF_INTEGRAL (integrand, surface, region, layer)

layer must be one of the layers bounded by the selected surface.

2. Integrals over "sidewall" surfaces are selected by boundary name and qualifying layer name:

SINTEGRAL (integrand, named_boundary, named_layer)
SURF_INTEGRAL (integrand, named_boundary, named_layer)

If **layer** is omitted, the integral is taken over all layers of the specified surface.

Integrals of this type may be further qualified by selecting the region in which the evaluation is to be made:

SURF_INTEGRAL(integrand, named_boundary, named_layer, named_region)

named_region must be one of the regions bounded by the selected surface.

3. Integrals over entire bounding surfaces of selected subregions are selected by region name and layer name, as with volume integrals:

SINTEGRAL (integrand, named_region, named_layer)
SURF_INTEGRAL (integrand, named_region, named_layer)

If **named_layer** is omitted, the integral is taken over all layers of the specified surface.

3.8. Predefined Elements

The problem descriptor language predefines the following element:

PI 3.14159265358979

For Cartesian coordinates in which '**R**' is not specified as a coordinate name or a defined name, the problem descriptor language predefines the following elements:

R $R = \sqrt{x^2 + y^2}$! radius vector length in 2D

$R = \sqrt{x^2 + y^2 + z^2}$! radius vector length in 3D

THETA $THETA = \arctan(y/x)$! azimuthal angle in 2D or 3D

Note: If "R" or "Theta" appear on the left side of a definition before any use in an expression, then the new definition will become the meaning of the name, and the predefined meaning will be hidden.

In staged problems where "**stages = integer**" is declared in the SELECT section,

STAGE an internally declared index which steps between 1 and **integer**.

In modal analysis (eigenvalue and eigenfunction) problems where "**modes = integer**" is declared in the SELECT section,

LAMBDA an internally declared name which represents the various eigenvalues.

3.9. Expressions

Value Expressions

Problem descriptors can contain expressions made of one or more operators, variables, defined values and pairs of parentheses that evaluate to numerical constants. In evaluating value expressions, FlexPDE follows the algebraic rules of precedence in which unary operators are evaluated first, followed by binary operators in the following order:

power
multiplication and division
addition and subtraction
relational operators (<, <=, =, >, >=, >)
relational combinations (AND, OR)

When included in expressions, subexpressions enclosed in pairs of parentheses are evaluated first, without regard to the precedence of any operators which precede or follow them. Parentheses may be nested to any level, with inner subexpressions being evaluated first and proceeding outward. Parentheses must always be used in pairs.

Conditional-Value Expressions

Problem descriptors can contain conditional expressions of the form

IF condition THEN subexpression ELSE subexpression .

This form selects one of the two alternative values as the value of the expression. It is used in expressions like "y = IF a THEN b ELSE c", analogous to the expression "y = a ? b : c" in the C programming language.

It is *not* the procedural alternative construct "IF a THEN y=b ELSE y=c" familiar in procedural programming languages.

The **THEN** or **ELSE** subexpressions may contain nested **IF...THEN...ELSE** expressions. Each **ELSE** will bind to the nearest previous **IF**.

3.10. Repeated Text

The **REPEAT..ENDREPEAT** construct allows the repetition of sections of input text.

The syntax looks like a **FOR** loop in procedural languages, but we emphasize that in FlexPDE this feature constitutes a *textual* repetition, not a procedural repetition.

The form of a repeat clause is

REPEAT name = initial TO final
REPEAT name = initial BY delta TO final

These statements specify that the following lines of descriptor text should be repeated a number of times. The given **name** is defined as if it had

appeared in the **DEFINITIONS** section, and is given the value specified by **initial**.

The repeated section of text is terminated by the statement

ENDREPEAT

At this point, the value of **name** is incremented by **delta** (or by one, if no **delta** is given). If the new value is not greater than **final**, the repeated text is scanned again with the new value in place of **name**. If **delta** is negative, the value of **name** is decremented and the termination test is modified accordingly.

The **REPEAT** statement can appear in the following locations:

- in **BATCH** file lists
- in **VARIABLE** lists
- in **EXTRUSION** lists
- anywhere the **REGION**, **START** or **LINE** keywords are legal.
- around any plot command or group of plot commands.
- around any **DEFINITION** or group of **DEFINITIONS**.
- around any **REPORT** command or group of **REPORT** commands.
- around **AT** points in a **HISTORY** list

Use of **ARRAYS** and the **\$integer** string function can extend the power of the **REPEAT** loop.

Examples:

```
REPEAT xc=1/4 by 1/4 to 7/4
  REPEAT yc=1/4 by 1/4 to 7/4
    START(xc+rad,yc) ARC(CENTER=xc,yc) ANGLE=360
  CLOSE
ENDREPEAT
ENDREPEAT
```

This double loop constructs a 7 x 7 array of circles, all part of the same **REGION**.

See the sample problems:

Samples\Misc\Repeat.pde
Samples\Misc\ArrayRepeat.pde

[Note: **REPEAT..ENDREPEAT** replaces the older **FOR..ENDFOR** facility used in earlier versions of FlexPDE. The older facility is still supported for backward compatibility, but should not be used in new problem descriptors.]

4. The Sections of a Descriptor

The **SECTIONS** of a descriptor were outlined in the introduction. In the following pages we present a detailed description of the function and content of each section.

4.1. Title

The optional **TITLE** section can contain one literal string.

When a **TITLE** is used, the literal string it contains is used as a title label for all **MONITORS** and **PLOTS**.

If **TITLE** is not specified, the plots will not have a title label.

Example:

```
TITLE "this is my first model"
```

4.2. Select

The **SELECT** section, which is optional, is used when it is necessary to override some of the default selectors internal to the program.

Selectors are used to control the flow of the process used to solve a problem.

The **SELECT** section may contain one or more selectors and their associated values. The default selectors have been chosen to optimize how FlexPDE handles the widest range of problems.

The **SELECT** section should be used only when the default behavior of FlexPDE is somehow inadequate.

Unlike the other elements used in program descriptors, the proper names used for the selectors are not part of the standard language, and are not meaningful in other descriptor sections.

The selectors implemented in FlexPDE are specific to a version of FlexPDE, and may not correspond to those available in previous versions of FlexPDE or in other applications using the FlexPDE descriptor language.

4.2.1. Mesh Generation Controls

The following controls can be used in the SELECT section to modify the behavior of the mesh generator.

Logical selectors can be turned on by **selector = ON**, or merely mentioning the **selector**

Logical selectors can be turned off by **selector = OFF**.

ASPECT default: 2.0
Maximum cell aspect ratio for mesh generation in 2D problems and 3D surface meshes. Cells may be stretched to this limit of edge-size ratio.

CURVEGRID default: On
If ON, cells will be bent to follow curved boundaries, and a 3D mesh will be refined to resolve surface curvature.
If OFF, neither of these modifications will be attempted, and the computation will proceed with straight-sided triangles or flat-sided tetrahedra. (It may be necessary to turn this option OFF when surfaces are defined by TABLES, because the curvature is infinite at table breaks.)

GRIDARC default: 30 degrees
Arcs will be gridded with no cell exceeding this angle. Other factors may cause the sizes to be smaller.

GRIDLIMIT default: 8
Maximum number of regrid before a warning is issued. Batch runs stop at this limit.

INITGRIDLIMIT default: 5
Maximum number of regridding passes in the initial refinement to define initial values. INITGRIDLIMIT=0 suppresses initial refinement.

NGRID
Specifies the number of mesh rows in each dimension. Use this control to set the maximum cell size in open areas. This is a

convenient way to control the overall mesh density in a problem. Default values are shown below:

	1D	2D	3D
Professional	100	15	10
Student	50	10	5

NODELIMIT

Specifies the maximum node count. If mesh refinement tries to create more nodes than the limit, the cell-merge threshold will be raised to try to balance errors across a mesh of the specified size. This control cannot be used to reduce the size if the initial mesh construction, which is dictated by NGRID, user density controls, and domain boundary feature sizes. Default values are shown below:

	1D	2D	3D
Professional	2000000	2000000	2000000
Student	100	800	1600

REGRID default: On
By default, FlexPDE implements adaptive mesh refinement. This selector can be used to turn it off and proceed with a fixed mesh.

SMOOTHINIT default: On
Implements a mild initial-value smoothing for time dependent problems, to help ameliorate discontinuous initial conditions.

STAGEGRID default: Off
Forces regeneration of mesh with each stage of a staged problem. FlexPDE attempts to detect stage dependencies in the domain and regenerate the mesh, but this selector may be used to override the automatic detection.

[Note:
See the "Mesh Control Parameters" section in this manual and the "Controlling Mesh Density" section in the User Guide for more discussion of mesh control.]

4.2.2. Solution Controls

The following controls can be used in the SELECT section to modify the solution methods of FlexPDE.

Logical selectors can be turned on by **selector = ON**, or merely mentioning the **selector**.

Logical selectors can be turned off by **selector = OFF**.

AUTOSTAGE default: On

In STAGED problems, this selector causes all stages to be run consecutively without pause. Turning this selector OFF causes FlexPDE to pause at the end of each stage, so that results can be examined before proceeding.

CHANGELIM default: 0.5(steady state),
2.0(time dependent)

Specifies the maximum change in any nodal variable allowed on any Newton iteration step (measured relative to the variable norm). In severely nonlinear problems, it may be necessary to force a slow progress toward the solution in order to avoid pathological behavior of the nonlinear functions.

CUBIC default: Off

Use cubic Finite Element basis (same as ORDER=3). The default is quadratic (ORDER=2). Cubic basis creates a larger number of nodes, and sometimes makes the system more ill-conditioned.

ERRLIM default: 0.002

This is the primary accuracy control. Both the spatial error control XERRLIM the temporal error control TERRLIM are set to this value unless over-ridden by explicit declaration.

[Note: ERRLIM is an *estimate* of the relative error in the dependent variables. The solution is not guaranteed to lie within this error. It may be necessary to adjust ERRLIM or manually force greater mesh density to achieve the desired solution accuracy.]

FIRSTPARTS default: Off

By default, FlexPDE integrates all second-order terms by parts, creating the surface terms represented by the Natural boundary condition. This selector causes first-order terms to be integrated by parts as well. Use of this option may require adding terms to Natural boundary condition statements.

FIXDT default: Off

Disables the automatic timestep control. The timestep is fixed at the value given in the TIME section.

HYSTERESIS default: 0.5

Introduces a hysteresis in the decay of spatial error estimates in time-dependent problems. The effective error estimate includes this fraction of the previous effective estimate added into the current instantaneous estimate. This effect produces more stable regridding in most cases.

ICCG default: On

Use Incomplete Choleski Conjugate-Gradient in symmetric problems. This method usually converges much more quickly. If ICCG=OFF or the factorization fails, then the Orthomin method will be used.

ITERATE default: 1000 (steady-state)
default: 500(time-dependent)

Primary conjugate gradient iteration limit. This is the count at which convergence-coercion techniques begin to be applied. The actual hard maximum iteration count is 4*ITERATE.

LINUPDATE default: 5

In linear steady-state problems, FlexPDE repeats the linear system solution until the computed residuals are below tolerance, up to a maximum of LINUPDATE passes.

MODES default: 0

Selects the Eigenvalue solver and specifies the desired number of modes. The default is *not* to run an Eigenvalue problem.

NEWTON default: (5/changelim)+40

Overrides the default maximum Newton iteration limit.

NONLINEAR default: Automatic

Selects the nonlinear (Newton-Raphson) solver, even if the automatic detection process does not want it.

NONSYMMETRIC default: Automatic

Selects the nonsymmetric Lanczos conjugate gradient solver, even if the automatic detection process does not want it.

NOTIFY_DONE default: Off

Requests that FlexPDE emit a beep and a "DONE" message at completion of the run.

NRMATRIX default: 5

Sets the maximum number of Newton-Raphson iterations before recomputing the coupling matrix in steady-state solutions. The matrix is recomputed whenever the solution changes appreciably, or when the residual is large.

NRMINSTEP default: 0.009

Sets the minimum fraction of the computed stepsize which will be applied during Newton-Raphson backtracking. This number only comes into play in difficult nonlinear systems. Usually the computed step is unmodified.

NRSLOPE default: 0.1

Sets the minimum acceptable residual improvement in Newton-Raphson backtracking of steady-state solutions.

NRUPDATE default: 3

Sets the maximum number of Newton-Raphson steps in each timestep in nonlinear time dependent problems. The default (3) seems to give the best balance between cost and stability. Well-behaved nonlinear problems may run more quickly with 1. This selector is set automatically by the PREFER_SPEED and PREFER_STABILITY selectors.

NRUPFIT default: Off

"ON" requests that FITs and SAVEs be recalculated at each Newton iteration of nonlinear time-dependent problems. Prior to version 2.20e, these items were computed once in each timestep. The default condition uses only one Newton step per timestep, so this selector is useful only if NRUPDATE is also set.

ORDER default: 2

Selects the order of finite element interpolation (2 or 3). The selectors QUADRATIC and CUBIC are equivalent to ORDER=2 and ORDER=3, respectively.

OVERSHOOT default: 0.001

Sub-iteration convergence control. Conjugate-Gradient solutions will iterate to a tolerance of OVERSHOOT*ERRLIM. (Some solution methods may apply additional multipliers.)

PRECONDITION default: On

Use matrix preconditioning in conjugate-gradient solutions. The default preconditioner is the diagonal-block inverse matrix.

PREFER_SPEED default: Off

Sets control parameters for time dependent problems to the best balance for speedy completion of most problems. Use PREFER_STABILITY for more difficult nonlinear problems. PREFER_SPEED is equivalent to NRUPDATE=1, TNORM=2.

PREFER_STABILITY default: On

Sets control parameters for time dependent problems to a slower but more stable configuration for difficult nonlinear problems. PREFER_STABILITY is equivalent to NRUPDATE=3, TNORM=4. Well-behaved nonlinear problems may run more quickly using PREFER_SPEED.

QUADRATIC default: On

Selects use of quadratic Finite Element basis. Equivalent to ORDER=2.

REINITIALIZE default: Off

Causes each Stage of a STAGED problem to be reinitialized with the INITIAL VALUES specifications, instead of preserving the results of the previous stage.

STAGES default: 1

Parameter-studies may be run automatically by selecting a number of Stages. Unless the geometric domain parameters change with stage, the mesh and solution of one stage are used as a starting point for the next.

SUBSPACE default: MIN(2*modes,modes+8)

If MODES has been set to select an eigenvalue problem, this selector sets the dimension of the subspace used to calculate eigenvalues.

TERRLIM default: 0.002

This is the primary temporal accuracy control. In time dependent problems, the timestep will be cut if the estimated relative error in time integration exceeds this value. The timestep will be increased if the estimated temporal error is smaller than this value. TERRLIM is automatically set by the ERRLIM control.

[**Note:** TERRLIM is an *estimate* of the relative error in the dependent variables. The solution is not guaranteed to lie within this error. It may be necessary to adjust TERRLIM to achieve the desired solution accuracy.]

TNORM default: 4
Error averaging method for time-dependent problems. Timestep control is based on summed (2^{TNORM}) power of nodal errors. Allowable values are 1-4. Use larger TNORM in problems with localized activity in large mesh.

UPFACTOR default: 1
Multiplier on upwind diffusion terms. Larger values can sometimes stabilize a marginal hyperbolic system.

UPWIND default: On
"Upwind" convection terms in the primary equation variable. In the presence of convection terms, this adds a diffusion term along the flow direction to stabilize the computation.

VANDENBERG default: Off
Use Vandenberg Conjugate-Gradient iteration (useful if hyperbolic systems fail to converge). This method essentially solves $(A^t A)x = (A^t)b$ instead of $Ax=b$. This squares the condition number and slows convergence, but it makes all the eigenvalues positive when the standard CG methods fail.

XERRLIM default: 0.002
This is the primary spatial accuracy control. Any cell in which the estimated relative spatial error in the dependent variables exceeds this value will be split (unless NODELIMIT is exceeded). XERRLIM is set automatically by the ERRRLIM selector.
[**Note:** XERRLIM is an *estimate* of the relative error in the dependent variables. The solution is not guaranteed to lie within this error. It may be necessary to adjust XERRLIM or manually force greater mesh density to achieve the desired solution accuracy.]

4.2.3. Global Graphics Controls

The following controls can be used in the SELECT section to modify the behavior of the graphics subsystem.

Logical selectors can be turned on by **selector = ON**, or merely mentioning the **selector**.

Logical selectors can be turned off by **selector = OFF**.

In the usual case, these selectors can be over-ridden by specific controls in individual plot commands (see Graphic Display Modifiers).

ALIAS (coord) = "name" default: Coordinate name
Defines an alternate label for the plot axes.

AUTOHIST default: On
Causes history plots to be updated when any other plot is drawn.

BLACK default: Off
Draw all graphic output in black only.

CDFGRID default: 51
Specifies the default size of CDF output grid (ie, 51x51).

CONTOURGRID default: 51
Resolution specification for contour plots. Actual computation cell sizes will be used unless they exceed the size implied by this resolution.

CONTOURS default: 15
Target number of contour levels. Contours are selected to give "nice" numbers, and the number of contours may not be exactly as specified here.

ELEVATIONGRID default: 401
Elevation plot grid size used by From..To elevation plots. Elevations on boundaries ignore this number and use the actual mesh points.

FEATUREPLOT default: Off
If this selector is ON, FEATURE boundaries will be plotted in gray. This was the default behavior in versions prior to 3.10b.

FINDERBINS default: 20
FlexPDE uses a banded subdivision of the box containing the domain to speed the search for plot points in the computation mesh and lookup points in TRANSFER files. In problems with meshes which are dense in localized areas, the default of 20 bands may be insufficient to speed the lookup, and a larger number of bands may be required. Use FINDERBINS to select a new number. [3.10a]

FONT default: 2

Font=1 selects sans-serif font. Font=2 selects serif font.

GRAY default: Off
Draws all plots with a gray scale instead of the default color palette.

HARDMONITOR default: Off
Causes MONITORS to be written to the hardcopy (.pg5) file.

LOGLIMIT default: 15
The range of data in logarithmic plots is limited to LOGLIMIT decades below the maximum data value. This is a global control which may be overridden by the local LOG(number) qualifier on the plot command.

MERGE default: On
Allows merging of low-error mesh cells. Only cells which have previously been split can be merged.

NOMINMAX default: Off
Deletes "o" and "x" marks at min and max values on all contour plots.

NOTAGS default: Off
Suppresses level identifying tags on all contour and elevation plots.

NOTIPS default: Off
Plot arrows in vector plots without arrowheads. Useful for bi-directional stress plots.

PAINTED default: Off
Draw color-filled contour plots. Plots can be painted individually by selecting PAINT in the plot modifiers.

PAINTGRID default: On
Draw color-filled grid plots. Colors represent distinct materials, as defined by parameters.

PAINTMATERIALS default: On
Synonymous with PAINTGRID, included for symmetry with individual PLOT modifiers.

PAINTREGIONS default: Off
Sets PAINTGRID, but selects a different coloring scheme. Colors represent logical regions in 2D, or logical (region,layer) compartments in 3D, instead of distinct material parameters.

PLOTINTEGRATE default: On
Integrate all spatial plots. Default is volume and surface integrals, using $2\pi r$ weighting in cylindrical geometry. Histories are not automatically integrated, and must be explicitly integrated.

PRINTMERGE default: Off
Send all stages or plot times of each EXPORT statement to a single file. By default, EXPORTS create a separate file for each time or stage. Individual EXPORTS can be controlled by plot modifiers.

SURFACEGRID default: 51
Selects the minimum resolution for Surface plots.

TEXTSIZE default: 35
Controls size of text on plot output. Value is number of lines per page, so larger numbers mean smaller text.

THERMAL_COLORS default: On
Sets the order of colors used in labeling plots. ON puts red is at the top (hot). OFF puts red at the bottom (lowest spectral color).

VECTORGRID default: 41
Sets minimum resolution of Vector plots.

VIEWPOINT (x, y, angle) default: negative X&Y, 30
Defines default viewpoint for SURFACE plots. Angle is in degrees. (In 3D, this specifies a position in the cut plane)

4.3. Coordinates

The optional **COORDINATES** section defines the coordinate geometry of the problem. The basic form of the section is:

COORDINATES geometry

where **geometry** may be any of the following:

<u>Name</u>	<u>Meaning</u>
CARTESIAN1	1D Cartesian coordinate named 'X'
CYLINDER1	1D Cylindrical coordinate named 'R'
SPHERE1	1D Spherical coordinate named 'R'
CARTESIAN2	2D Cartesian coordinates named 'X' and 'Y'.
XCYLINDER	2D Cylindrical coordinates with axial coordinate 'Z' lying along the horizontal (X) plot axis, and radial coordinate 'R' lying along the vertical(Y) plot axis.
YCYLINDER	2D Cylindrical coordinates with radial coordinate 'R' lying along the horizontal (X) plot axis, and axial coordinate 'Z' lying along the vertical(Y) plot axis.
CARTESIAN3	3D Cartesian coordinates named 'X', 'Y' and 'Z'.

Renaming Coordinates

A second form of the **COORDINATES** section allows renaming of the coordinates:

COORDINATES geometry ('Xname' [, 'Yname' [, 'Zname']])

In this case, the '**Xname**' argument renames the coordinate lying along the horizontal plot axis, and '**Yname**' renames the coordinate lying along the vertical plot axis. '**Zname**' renames the extrusion coordinate. **Names** may be quoted strings or unquoted names.

Renaming coordinates causes a redefinition of the differential operators. DX becomes D<Xname>, etc.

The **DIV**, **GRAD**, and **CURL** operators are expanded correctly for the designated geometry. Use of these operators in the **EQUATIONS** section can considerably simplify problem specification.

IF no **COORDINATES** section is specified, a **CARTESIAN2** coordinate system is assumed.

4.4. Variables

The **VARIABLES** section is used to define and assign names to all the primary dependent variables used in a problem descriptor. All names appearing in the **VARIABLES** section will be represented by a finite element approximation over the problem mesh. Each variable is assumed to define a continuous scalar field over the problem domain. It is further assumed that each variable will be accompanied by a partial differential equation listed in the **EQUATIONS** section.

In assigning names to the dependent variables, the following rules apply:

- Variable names must begin with an alphabetic character. They may not begin with a number or symbol.
- Variable names may be a single character other than the single character **t**, which is reserved for the time variable.
- Variable names may be of any length and any combination of characters, numbers and/or symbols other than reserved words.
- Variable names may not contain any separators. Compound names can be formed with the '_' symbol (e.g. temperature_celsius).
- Variable names may not contain the '-' which is reserved for the minus sign.

Example:

```
VARIABLES
U,V
```

4.4.1. The THRESHOLD Clause

An optional **THRESHOLD** clause may be associated with a variable name.

The **THRESHOLD** value determines the *minimum* range of values of the variable for which FlexPDE must try to maintain the requested **ERRLIM** accuracy. In other words, **THRESHOLD** defines the level of variation at which the user begins to lose interest in the details of the solution.

Error estimates are scaled to the *greater* of the **THRESHOLD** value or the observed range of the variable, so the **THRESHOLD** value becomes meaningless once the observed variation of a variable in the problem domain exceeds the stated **THRESHOLD**. If you make the **THRESHOLD** too large, the accuracy of the solution will be degraded. If you make it too small, you will waste a lot of time computing precision you don't need. So if you provide a **THRESHOLD**, make it a modest fraction of the expected range (max minus min) of the variable.

The **THRESHOLD** clause has two alternative forms:

```
variable_name ( THRESHOLD = number )
variable_name ( number )
```

[**Note:** In most cases, the use of **THRESHOLD** is meaningful only in time-dependent or nonlinear steady-state problems with uniform initial values, or that ultimately reach a solution of uniform value.]

4.4.2. Moving Meshes

FlexPDE version 5 allows the specification of equations to move the computation mesh.

In order to do this, you must assign a Variable as a surrogate for each coordinate you wish to modify. This specification uses the form

```
variable_name = MOVE ( coordinate_name ).
```

This declaration assigns **variable_name** as a surrogate variable for the **coordinate_name**. You may subsequently assign **EQUATIONS** and boundary conditions to the surrogate variable in the normal way, and these equations and boundary conditions will be imposed on the values of the selected mesh coordinate at the computation nodes.

Example:

```
VARIABLES
  U,V
  Xm = MOVE(X)
```

4.4.3. The SIMPLEX Modifier

A variable may be forced to be modeled with a linear basis, regardless of the basis of the computation. This modifier takes the form

```
variable_name ( SIMPLEX )
```

In some cases, the imposition of a lower-order basis on one selected variable can improve the stability of a computation.

4.5. Global Variables

The **GLOBAL VARIABLES** section is used to define auxiliary or summary values which are intricately linked to the field variables.

Each **GLOBAL VARIABLE** takes on a single value over the entire domain, as opposed to the nodal finite element field representing a **VARIABLE**.

GLOBAL VARIABLES differ from simple **DEFINITIONS** in that **DEFINITIONS** are algebraically substituted in place of their references, while **GLOBAL VARIABLES** are assigned a row and column in the master coupling matrix and are solved simultaneously with the finite element equations.

The **GLOBAL VARIABLES** section must follow immediately after the **VARIABLES** section. Rules for declaring **GLOBAL VARIABLES** are the same as for **VARIABLES**, and a **GLOBAL VARIABLE** may have a **THRESHOLD**.

Each **GLOBAL VARIABLE** will be associated with an entry in the **EQUATIONS** section, with rules identical to those for **VARIABLES**.

GLOBAL VARIABLES do not have boundary conditions. They may be either steady-state or time-dependent, and may be defined in terms of integrals over the domain, or by point values of other functions.

Examples:

Samples | Misc | Heaterssi.pde

[Note: In previous versions of FlexPDE, Global Variables were referred to as SCALAR VARIABLES. This usage is still allowed for compatibility, but the newer terminology is preferred.]

4.6. Definitions

The **DEFINITIONS** section is used to declare and assign names to special numerical constants, coefficients, and functions used in a problem descriptor.

In assigning names to the definitions, the following rules apply:

- Must begin with an alphabetic character. May *not* begin with a number or symbol.
- May be a single character other than the single character t , which is reserved for the time variable.
- May be of any length and any combination of characters, numbers, and symbols other than reserved words, coordinate names or variable names.
- May *not* contain any separators. Compound names can be formed with the '_' symbol (e.g. **temperature_celsius**).
- May *not* contain the '-' which is reserved for the minus sign.

Normally, when a definition is declared it is assigned a value by following it with the assignment operator '=' and either a value or an expression. Definitions are dynamic elements and when a value is assigned, it will be the initial value only and will be updated, if necessary, by the problem solution.

Example:

Viscosity = 3.02e-4*exp(-5*Temp)

Definitions are expanded inline in the partial differential equations of the **EQUATIONS** section. They are not represented by a finite element

approximation over the mesh, but are calculated as needed at various times and locations.

Redefining Regional Parameters

Names defined in the **DEFINITIONS** section may be given overriding definitions in some or all of the **REGIONS** of the **BOUNDARIES** section. In this case, the quantity may take on different region-specific values. Quantities which are completely specified in subsequent **REGIONS** may be stated in the **DEFINITIONS** section without a value.

See the User Guide section "Setting Material Properties by Region" for examples of redefined regional parameters.

4.6.1. ARRAY Definitions

Names may be defined as representing arrays or lists of values. The statement

name = ARRAY [value_1 , value_2 ... value_n]

will define **name** to be an n-element array of values **value_1 ... value_n**.

In subsequent text, these values may be referenced as

name [index]

The values given in the value list must evaluate to scalar numbers. They may not contain coordinate or variable dependencies.

[Note: In previous versions, the elements of an array were required to be constants computable at the time of reading the script. In version 5, the elements of an array need only be computable at the time they are used.]

Example:

definitions

xc=array(1/3, 2/3, 3/3, 4/3, 5/3) { a list of X-coordinates }

```

yc=array(1/3, 2/3, 3/3, 4/3, 5/3)    { a list of Y-coordinates }
...
boundaries
region 1
repeat i=1 to 5          { an indexed loop on X-position }
repeat j=1 to 5          { an indexed loop on Y-position }
start(xc[i]+rad,yc[j])  { an array of circular dots at the }
arc(center=xc[i],yc[j]) angle=360 { ... tabulated coordinates }
close
endrepeat
endrepeat

```

This text generates a 5 x 5 array of circles in the domain, all in region 1.

See also "Samples | Misc | arrayrepeat.pde".

4.6.2. Parameterized Definitions

Definitions can be made to depend on one to three explicit arguments, much as with a Function definition in a procedural language. The syntax of the parameterized definition is

```

name ( argname ) = expression
name ( argname1 , argname2 ) = expression
name ( argname1 , argname2 , argname3 ) = expression

```

The construct is only meaningful if **expression** contains references to the **argnames**. Names defined in this way can later be used by supplying actual values for the arguments. As with other definitions in FlexPDE, these actual parameters may be any valid expression with coordinate or variable dependences. The **argnames** used in the definition are local to the definition and are undefined outside the scope of the defining **expression**.

Note that it is never necessary to pass known definitions, such as coordinate names, variable names, or other parameters as arguments to a parameterized definition, because they are always globally known and are evaluated in the proper context. Use the parameterized definition facility when you want to pass values that are not globally known.

[**Note:** This construct is implemented by textual expansion of the definitions in place of the function reference. It is not a run-time call, as in a procedural language.]

Example:

DEFINITIONS

```
uu(arg) = arg*arg
```

...

EQUATIONS

```
div(a*grad(u)) + uu(u+1)*dx(u) + 4 = 0;
```

In this case, the equation will expand to

$$\text{div}(a*\text{grad}(u)) + (u+1)*(u+1)*dx(u) + 4 = 0.$$

See also "Samples|Misc|func.pde"

4.6.3. STAGED Definitions

FlexPDE can perform automated parameter studies through use of the "staging" facility. In this mode, FlexPDE will run the problem a number of times, with differing parameters in each run. Each stage begins with the solution and mesh of the previous stage as initial conditions.

The STAGES Selector

In the **SELECT** section, the statement

```
STAGES = number
```

specifies that the problem will be run **number** times. A parameter named **STAGE** is defined, which takes on the sequence count of the staged run. Other definitions may use this value to vary parameter values, as for example:

```
Voltage = 100*stage
```

STAGED Definitions

A parameter definition may also take the form:

param = STAGED (value_1, value_2, ... value_n)

In this case, the parameter **param** takes on **value_1** in stage 1, **value_2** in stage 2, etc.

If **STAGED** parameters are defined, the **STAGES** selector is optional. If the **STAGES** selector is not defined, the length of the **STAGED** list will be used as the number of stages. If the **STAGES** selector is defined, it overrides the length of the **STAGED** list.

See the example "Samples | Misc | Stages.pde".

STAGED Geometry

If the geometric domain definition contains references to staged quantities, then the solution and mesh will not be retained, but the mesh will be regenerated for the new geometry. History plots can still be displayed for staged geometries.

See the example "Samples | Misc | Stage_Geom.pde".

FlexPDE attempts to detect stage dependence in the geometrical domain definition and automatically regenerate the mesh. If for any reason these dependencies are undetected, the global selector **STAGEGRID** can be used to force grid staging.

4.6.4. POINT Definitions

A name may be associated with a coordinate point by the construct

point_name = POINT(a,b)

Here **a** and **b** must be computable constants at the time the definition is made. They may not depend on variables or coordinates. They may depend on stage number.

The name of the point can subsequently appear in any context in which the literal point (**a,b**) could appear.

4.6.5. Data Import Definitions

4.6.5.1. The TABLE Input function

FlexPDE supports a tabular data import function:

name = TABLE ('filename')

This statement imports a data table from the named file and associates the data with the defined **name**.

Normally, the statement appears in a parameter definition (in the **DEFINITIONS** section or as a regional parameter definition in a **REGION** clause), and the table data are associated with the given name.

TABLE can also be used in arithmetic expressions, in which case the data are unique to that specific evaluation.

TABLE data are interpolated with linear, bilinear or trilinear interpolation on the specified data grid.

FlexPDE can accept (import) and generate (export) non-analytic data through external one, two, or three dimensional table files. This feature is useful for modeling systems where experimental data is available and for interfacing with other software programs.

Table import files are ASCII text files, and can be generated with any ASCII text editor, by user programs designed to generate tables, or by FlexPDE itself, using the **EXPORT** plot modifier or the **TABLE** output statement (see MONITORS and PLOTS).

Modifying Table Coordinates

In the normal case, the name(s) of the table coordinate(s) are given in the table file itself. However, an alternative form of the **TABLE** input function can be used to rename the coordinates:

name = TABLE ('filename', coord1 [,coord2...])

In this case, the coordinates named in the file will be over-ridden by the stated coordinate names. These names must have been defined before their use in the **TABLE** statement.

When the parameter **name** is used in subsequent computations, the current values of the table coordinates will be used to interpolate the value. For instance, if the table coordinates are the spatial coordinates X and Y, then during computations or plotting, the named parameter will take on a spatial distribution corresponding to the table data spread over the problem domain.

See the **TABLEDEF** statement for an alternative form of table input.

See the **SPLINETABLE** statement for an alternative form of table interpolation.

Examples:

Samples | Misc | Table.pde

4.6.5.2. The TABLEDEF input statement

The **TABLEDEF** input statement allows the import of tabular data, similar to the **TABLE** input function.

The format is

```
TABLEDEF('filename',name1 [,name2,..])
```

Unlike the **TABLE** statement, the **TABLEDEF** statement can be used to directly define one or several parameters from a multi-valued table file. Whereas in the **TABLE** statement the additional arguments are coordinate reassignments, in the **TABLEDEF** statement the additional arguments are the names to be defined and associated with the table data. In this, the **TABLEDEF** statement is syntactically identical to the **TRANSFER** statement.

See TABLE File Format for a definition of the table file format.

4.6.5.3. The SPLINETABLE function

SPLINETABLE is a variation of the **TABLE** input function, and uses the same table format.

```
name = SPLINETABLE ( 'filename' )
```

SPLINETABLE causes the data to be interpolated using cubic splines. The interpolating functions are constructed so that the interpolated value and its first and second derivatives are all continuous across table grid positions. A condition of zero curvature is applied at the table edges.

Only tables of one or two dimensions are currently supported.

Examples:

See Samples | Misc | Spline1.pde and Samples | Misc | Splineable.pde

4.6.5.4. TABLE File format

Data files for use in **TABLE**, **TABLEDEF** or **SPLINETABLE** input must have the following form:

```
{ comments }
name_coord1 datacount1
  value1_coord1 value2_coord1 value3_coord1 ...
name_coord2 datacount2
  value1_coord2 value2_coord2 value3_coord1 ...
name_coord3 datacount3
  value1_coord3 value2_coord3 value3_coord3 ...
data { comments }
data111 data211 data311 ...
data121 data221 data321 ...
data131 data231 data331 ...
...      ...      ...
...      ...      ...
data112 data 212 data312 ...
data122 data 222 data322 ...
data132 data 232 data 332 ...
...      ...      ...
...      ...      ...
```

where

name_coordN is the coordinate name in the N direction. Typically, **name_coord1** is **x**, **name_coord2** is **y**.

datacountN is the number of data points in the N direction.

DataJKL is the data at coordinate point (J,K,L)

... ... ellipses indicate extended lines, which may be continued over multiple lines.

Example:

```
{ this is an example table }
x 6
-0.01 2 4 6 8 10.01
y 6
-0.01 2 4 6 8 10.01
data
1000 1      1000 1      1000 1
1      1000 1      1000 1      1000
1000 1      1000 1      1000 1
1      1000 1      1000 1      1000
1000 1      1000 1      1000 1
1      1000 1      1000 1      1000
```

4.6.5.5. The TRANSFER input statement

The **TRANSFER** input statement allows the transfer of data between FlexPDE runs, maintaining the full information content of the original computation.

TRANSFER ('filename', name1 [,name2, ...])

The file specified in the transfer input function must have been written by FlexPDE using the **TRANSFER** output function. The names listed in the input function will become defined as if they had appeared in a "**name=**" definition statement. The names will be positionally correlated with the data fields in the referenced output file.

Examples:

Samples | Misc | Import-Export | Transfer_Out.pde
 Samples | Misc | Import-Export | Transfer_In.pde

4.6.5.6. The TRANSFERMESH input statement

The **TRANSFERMESH** input statement has the same form as the **TRANSFER** statement:

TRANSFERMESH ('filename', name1 [,name2,..])

The **TRANSFERMESH** input statement, however, not only imports data definitions stored on disk, but also **IMPOSES THE FINITE ELEMENT MESH STRUCTURE** of the imported file onto the current problem, bypassing the normal mesh generation process.

In order for this imposition to work, the importing descriptor file must have **EXACTLY** the same domain definition statements as the exporting file.

The **TRANSFERMESH** file does not contain a complete description of the boundaries and boundary conditions of the exporting problem, but only the mesh layout. Be sure to use a copy of the exporting domain definition in your importing descriptor. You may change the boundary conditions, but not the boundary positions and ordering.

TRANSFERMESH can read only files created by a **TRANSFER** output statement. Other file formats cannot be read.

Examples:

Samples | Misc | Import-Export | Mesh_out.pde
 Samples | Misc | Import-Export | Mesh_in.pde

4.6.5.7. TRANSFER File format

The format of a **TRANSFER** file is dictated by the **TRANSFER** output format, and contains the following data.

- 1) A header containing an identifying section listing the FlexPDE version, generating problem name and run time, and plotted variable name or function equation. This header is enclosed in comment brackets, { ... }.
- 2) A file identifier "FlexPDE transfer file", and the problem title.
- 3) The number of geometric dimensions and their names.

4) The finite element basis identifier from 4 to 10, meaning:

- 4 = linear triangle (3 points per cell)
- 5 = quadratic triangle (6 points per cell)
- 6 = cubic triangle (9 points per cell)
- 7 = cubic triangle (10 points per cell)
- 8 = linear tetrahedron (4 points per cell)
- 9 = quadratic tetrahedron (10 points per cell)
- 10 = cubic tetrahedron (20 points per cell)

5) The number of degrees of freedom (points per cell as above).

6) The number of output variables and their names

Each distinct material type in the exported problem is represented by a separate section in the **TRANSFER** file. Each section consists of:

7) The number of nodes

8) The nodal data, containing one line for each node with the following format:

- two or three coordinates and as many data values as specified in (6).
- a colon (:)
- the global node index
- the node type (0=interior; 1=joint; 2=edge; 3=face; 4=exterior)
- the type qualifier (region number, joint number, edge number or face number)

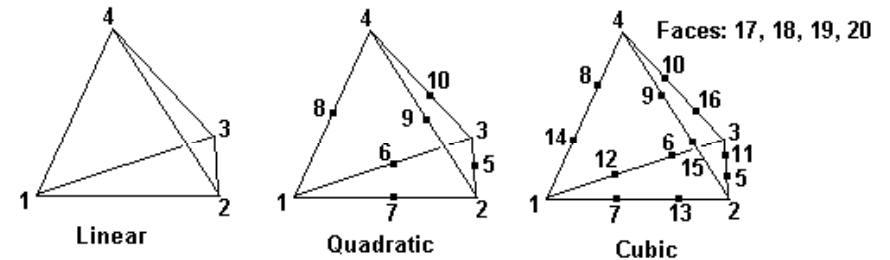
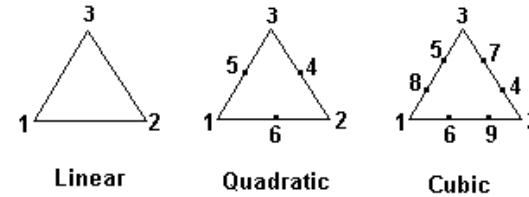
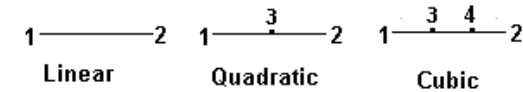
9) The number of cells.

10) The cell connectivity data, one line per cell, listing the following:

- the geometric basis (as in 4)
- the node numbers (local to the current material block) which comprise the cell. The count of these node numbers is controlled by (5).
- a colon (:)
- the global cell number
- the logical region number

- the material number

The node numbers are presented in the following order:



4.6.6. The PASSIVE Modifier

Definitions may be specified as **PASSIVE**, in which case they will be blocked from differentiation with respect to system variables in the formation of the global Jacobian matrix. In strongly nonlinear systems, this sometimes prevents pathological behavior, at the expense of slower convergence.

Example:

Viscosity = Passive(3.02*exp(-5*Temp))

The derivative of **Viscosity** with respect to **Temp** will be forced to zero, instead of the true value $(-5)*3.02*exp(-5*Temp)$.

4.6.7. Mesh Control Parameters

The names **MESH_SPACING** and **MESH_DENSITY** have special meaning in controlling the initial mesh layout. They may appear in the context of a parameter definition or redefinition (ie, in the DEFINITIONS section or in a REGION), or in the context of a boundary condition.

MESH_SPACING dictates the desired spacing between mesh nodes.

MESH_DENSITY is the reciprocal of **MESH_SPACING**, and dictates the desired number of mesh nodes per unit distance.

Appearing in the **DEFINITIONS** section, these parameters specify a global default mesh density function in the volume of the domain.

Appearing in a **REGION**, these parameters specify a mesh density function in the volume of the current region (in 3D they may be qualified by **LAYER** or **SURFACE**).

Appearing in the context of a boundary condition (ie, inside a *path*) they dictate the mesh density along the curve or sidewall surface currently being defined. In 3D they may be qualified by **LAYER** or **SURFACE** to restrict the application of the density function.

MESH_SPACING and **MESH_DENSITY** specifications may be any function of spatial coordinates (but not of **VARIABLES**).

If more than one specification is active in a part of the mesh, the control resulting in the smallest mesh cells will be used.

Examples:

MESH_DENSITY = exp(-(x^2+y^2+z^2))

This will create a Gaussian density distribution around (0,0,0), with spacing ultimately overridden by the size limit implied by NGRID.

See the User Guide section "Controlling Mesh Density" for more information.

See also

"Samples | Misc | Mesh_Control | Mesh_Density.pde"
"Samples | Misc | Mesh_Control | Mesh_Spacing.pde"
"Samples | Misc | Mesh_Control | Bdry_Density.pde"
"Samples | Misc | Mesh_Control | Bdry_Spacing.pde"

4.7. Initial Values

The **INITIAL VALUES** section is used to initialize the dependent variables.

When not specifically initialized, the dependent variables are initialized to zero.

For steady state problems the **INITIAL VALUES** section is optional.

For time dependent problems, the **INITIAL VALUES** section should include a value assignment statement for each dependent variable.

Initial value statements are formed by following the dependent variable name with the assignment operator '=' and either a constant, function, expression or previously defined definition.

Example:

INITIAL VALUES
U = 1.0-x

Setting Initial Values from an imported table:

Initial values can be set directly by an imported **TABLE**:

INITIAL VALUES
U = TABLE("initial_U.tbl")

For syntactic reasons, initial values cannot be set directly from

TRANSFERS (q.v.).

An intermediate name must be defined by the transfer command, and then assigned to the initial value:

DEFINITIONS
TRANSFER("initial_U.xfr",U0)
INITIAL VALUES

$$U = U0$$

4.8. Equations

The **EQUATIONS** section is used to list the partial differential equations that define the dependent variables of the problem.

There must be one equation for each dependent variable listed in the **VARIABLES** section.

Equations are entered into a problem descriptor in much the same way as they are written on paper. In their simplest form they can be written using the **DIV** (divergence), **GRAD** (gradient), **CURL** and **DEL2**(Laplacian) operators. FlexPDE will correctly expand these operators in the coordinate system specified in the **COORDINATES** section.

When it is necessary to enter partial differential terms, differential operators of the form **D<name>** or **D<name1><name2>** may be used. Here **<name>** represents a coordinate name, such as **X**, **Y** or **Z** (or other names chosen by the user in the **COORDINATES** section).

In the default 2D Cartesian geometry, the operators **DX**, **DY**, **DXX**, **DXY**, **DYX** and **DYY** are defined.

Similarly, in the default cylindrical geometries (**XCYLINDER** and **YCYLINDER**), the operators **DR**, **DZ**, **DRR**, **DRZ**, **DZR** and **DZZ** are defined.

In 3D Cartesian geometry, the operators **DZ**, **DZZ**, **DXZ**, and **DYZ** are also defined.

Example:

$$\text{div}(k*\text{grad}(u)) + u*\text{dx}(u) = 0$$

Third Order and Higher Order Derivatives

Equation definitions may contain only first and second order derivatives. Problems such as the biharmonic equation which require the use of higher order derivatives must be rewritten using an intermediate variable so that they contain only first and second order derivatives.

4.8.1. Association between Equations, Variables and Boundary Conditions

In problems with a single variable, there is no ambiguity about the assignment of boundary conditions to the equations.

In problems with more than one variable, FlexPDE requires that equations be explicitly associated with variables by tagging each equation with a variable name. This process also allows optimal ordering of the equations in the coupling matrix.

Example:

$$\text{U: } \text{div}(k*\text{grad}(u))+u*\text{dx}(u)= 0 \quad \{ \text{associates this equation with the variable U} \}$$

Boundary conditions are defined in the **BOUNDARIES** section, and are associated with equations by use of the variable name, which selects an equation through the association tag. **VALUE(U)=0**, for example, will cause the nodal equations for the equation tagged **U**: to be replaced by the equation **u=0** along the selected boundary .

Natural boundary conditions must be written with a sign corresponding to the sign of the generating terms when they are moved to the left side of the equal sign. *We suggest that all second-order terms should be written on the left of the equal sign, to avoid confusion regarding the sign of the applied natural boundary condition.*

4.8.2. Modal Analysis and Associated Equations

When modal analysis is desired, it must be declared in the **SELECT** section with the selector

$$\text{MODES} = \text{integer}$$

where **integer** is the number of modes to be analyzed.

The equation should then be written in the form

$$\mathbf{F}(\mathbf{V}) + \text{LAMBDA} * \mathbf{G}(\mathbf{V}) = \mathbf{H}(\mathbf{X}, \mathbf{Y})$$

Where $\mathbf{F}(\mathbf{V})$ and $\mathbf{G}(\mathbf{V})$ are the appropriate terms containing the dependent variable, and $\mathbf{H}(\mathbf{X}, \mathbf{Y})$ is a driving source term.

The name **LAMBDA** is automatically declared by FlexPDE to mean the eigenvalue, and should not be declared in the **DEFINITIONS** section.

4.8.3. Moving Meshes

FlexPDE version 5 introduces the capability of moving meshes. Use of this capability requires:

- The assignment of a surrogate variable for each coordinate to be moved
- Definition of an EQUATION of motion for each such surrogate coordinate
- Suitable Boundary Conditions on the surrogate coordinate.

In the usual case, there will be a variable defining the mesh velocity. This may be the same as the fluid velocity, in which case the model is purely Lagrangian, or it may be some other better-behaved motion, in which case the model is mixed Lagrange/Eulerian (ALE).

FlexPDE 5 contains no provisions for re-connecting distorted meshes. Pure Lagrangian computations are therefore discouraged, as severe mesh corruption may result.

Internal Mesh Redistribution

A convenient technique is to define a variable for mesh velocity in each coordinate, and for which boundary conditions are imposed to move the boundary. Internally, the velocity can be simply diffused, according to

$$\text{DIV}(\text{GRAD}(\mathbf{x_velocity})) = 0$$

together with the equation associating the velocity to the surrogate coordinate variable

$$\text{VELOCITY}(\mathbf{x_surrogate}) = \mathbf{x_velocity}$$

Effect of mesh motion on ordinary equations

EQUATIONS are always written in the Eulerian (Laboratory) reference frame, regardless of whether the mesh moves or not. FlexPDE automatically computes the required correction terms for mesh motion.

4.9. Constraints

The **CONSTRAINTS** section, which is optional, is used to apply integral constraints to the system. These constraints can be used to eliminate ambiguities that would otherwise occur in steady state systems, such as mechanical and chemical reaction systems, or when only derivative boundary conditions are specified.

The **CONSTRAINTS** section, when used, normally contains one or more statements of the form

$$\text{INTEGRAL (argument)} = \text{expression}$$

CONSTRAINTS should not be used with steady state systems which are unambiguously defined by their boundary conditions, or in time-dependent systems.

A **CONSTRAINT** creates a new auxiliary functional which is minimized during the solution process. If there is a conflict between the requirements of the **CONSTRAINT** and those of the PDE system or boundary conditions, then the final solution will be a *compromise* between these requirements, and may not strictly satisfy either.

CONSTRAINTS can be applied to any of the INTEGRAL operators.

CONSTRAINTS *cannot* be used to enforce local requirements, such as positivity, to nodal variables.

Examples:

Samples | Misc | Constraints | Constraint.pde
 Samples | Misc | Constraints | Bdry_Constraint.pde
 Samples | Misc | Constraints | 3D_Constraint.pde
 Samples | Misc | Constraints | 3D_Surf_Constraint.pde

4.10. Extrusion

The layer structure of a three-dimensional problem is specified bottom-up to FlexPDE in the **EXTRUSION** Section:

```
EXTRUSION
  SURFACE "<Surface_name_1>" Z = expression_1
  LAYER  "<Layer_name_1>"
  SURFACE "<Surface_name_2>" Z = expression_2
  LAYER  "<Layer_name_2>"
  ...
  SURFACE "<Surface_name_n>" Z = expression_n
```

The specification must start with a **SURFACE** and end with a **SURFACE**.

LAYERS correspond to the space between the surfaces. The **LAYER** specifications may be omitted if a name is not needed to refer to them.

- Surfaces need not be planar, and they may merge, but they must not cross. **expression_1** is assumed to be everywhere less than or equal to **expression_2**, and so on. Use a **MIN** or **MAX** function when there is a possibility of crossover.
- Surface expressions can refer to regionally defined parameters, so that the surface takes on different definitions in different regions. The disjoint expressions must, however, be continuous across region interfaces. (see example "Samples | Misc | 3d_Domains | Regional_surfaces.pde")
- If surface expressions contain conditional values (**IF...THEN** or **MIN**, **MAX**, etc), then the base plane domain should include **FEATURES** to delineate the breaks, so they can be resolved by the gridder.
- Surfaces must be everywhere continuous, including across material interfaces. Use of conditionals or regional definitions must guarantee surface continuity.
- Surface expressions can refer to tabular input data (see example "Samples | Misc | 3D_Domains | Tabular_surfaces.pde").

The layer and surface names in these specifications are optional, and if layers have no names, the **LAYER** statements may be omitted.

Shorthand form

Stripped of labels, the **EXTRUSION** specification may be written:

EXTRUSION Z = expression_1, expression_2 [, ...]

In this form layers and surfaces must subsequently be referred to by numbers, with surface numbers running from 1 to n and layer numbers from 1 to (n-1). **SURFACE #1** is **Z=expression_1**, and **LAYER #1** is between **SURFACE #1** and **SURFACE #2**.

See the User Guide chapter Using FlexPDE in Three-Dimensional Problems for more information

4.11. Boundaries

The **BOUNDARIES** section is used to describe the problem domain over which the specified equation system is to be solved, and to specify boundary conditions along the outer surfaces of this domain.

Because of the history of FlexPDE, the discussion of boundaries has a strong two-dimensional orientation. Three-dimensional figures are made up by extruding a two-dimensional domain into the third dimension. One-dimensional domains are constructed by specializations of 2D techniques.

Every problem descriptor must have a **BOUNDARIES** section.

Problem **BOUNDARIES** are made up by walking the periphery of each material region on **BOUNDARY PATHS** through a 2D Cartesian space.

In this way, the physical domain is broken down into **REGION**, **FEATURE** and **EXCLUDE** subsections.

Every problem descriptor must have at least one **REGION** subsection. **FEATURE** and **EXCLUDE** subsections are optional.

For concrete examples of the constructs described here, refer to the sample problems distributed with the FlexPDE software.

4.11.1. Points

The fundamental unit used in building problem domains is the geometric **POINT**. **POINTS** in a FlexPDE script are expressed as a parenthesized list of coordinate values, as in then two dimensional point **(2.4, 3.72)**.

Since two- and three- dimensional figures both begin with a two-dimensional layout, the use for three-dimensional points is generally limited to ELEVATION PLOTS.

In one-dimensional systems, a point can degenerate to a single parenthesized coordinate, such as **(2.4)**.

4.11.2. Boundary Paths

A boundary path has the general form

START(a,b) segment TO (c,d) ...

where **(a,b)** and **(c,d)** are the physical coordinates of the ends of the segment, and **segment** is either **LINE**, **SPLINE** or **ARC**.

The path continues with a connected series of segments, each of which moves the segment to a new point. The end point of one segment becomes the start point of the next segment.

A path ends whenever the next input item cannot be construed as a segment, or when it is closed by returning to the start point. The closing segment may simply end at the start point, or it can explicitly reference **CLOSE**, which will cause the current path to be continued to meet the starting point:

... segment TO CLOSE.

or

... segment CLOSE.

[Note: In prior versions, return to start was signaled by the word **FINISH**. This form is still accepted by FlexPDE 5, but is deprecated because of ambiguous implications.]

Line Segments

Line segments take the form

LINE TO (x,y)

When successive **LINE** segments are used, the reserved word **LINE** does not have to be repeated, as in the following:

LINE TO (x1,y1) TO (x2,y2) TO (x3,y3) TO ...

Spline Segments

Spline segments are syntactically similar to Line segments

SPLINE TO (x,y) TO (x2,y2) TO (x3,y3) TO ...

A cubic spline will be fit to the listed points. The first point of the spline will be either the **START** point or the ending point of the previous segment. The last point of the spline will be the last point stated in the chain of **TO(,)** points.

The fitted spline will have zero curvature at the end points, so it is a good idea to begin and end with closely spaced points to establish the proper endpoint directions. [4.1]

Arc Segments

Arc segments create either circular or elliptical arcs, and take one of the following the forms:

ARC TO (x1,y1) to (x2,y2)

ARC (RADIUS = R) to (x,y)

ARC (CENTER = x1,y1) to (x2,y2)

ARC (CENTER = x1,y1) ANGLE=angle

Here **angle** is an angle *measured in degrees*, following the standard convention that positive angles rotate counter-clockwise and negative angles rotate clockwise. The coordinate point at the end of the arc is determined by the radius swept out by the angle. To specify the angle in radians, follow the radian value by the qualifier **RADIANS**.

When the form **ARC (CENTER=x1,y1) to (x2,y2)** is used and the center **(x1,y1)** is not equidistant from the start and end points, an elliptical arc segment is generated with major and minor axes along the X and Y coordinate directions.

Example:

START(0,0)
LINE TO (10,0) TO (10,10) TO (0,10) TO CLOSE

Named Paths

Names can be assigned to paths. When names are assigned to paths they take the form of a quoted string and must be placed immediately after the reserved word **START**:

START "namedpath" (<x> , <y>)

Assigned path names are useful when boundary or line-related integrals are desired or for establishing paths over which **ELEVATION** plots are desired.

4.11.3. Regions

A **REGION** is a portion of a two-dimensional problem domain (or of the projection of a 3D problem domain), bounded by **BOUNDARY PATHS**, that encloses an area and contains a single material (but see Regions in One Dimension for exceptions).

Each material property in the **REGION** has a single definition, although this definition may be arbitrarily complex.

A **REGION** may consist of many disjoint areas.

Example:

REGION 1 { an outer box }
START(0,0)
LINE TO (10,0) TO (10,10) TO (0,10) TO CLOSE

REGION 2 { with two embedded boxes }
START(1,1)
LINE TO (2,1) TO (2,2) TO (1,2) TO CLOSE
START(5,5)
LINE TO (6,5) TO (6,6) TO (5,6) TO CLOSE

Overlaying regions:

REGIONS DEFINED LATER OVERLAY AND OBSCURE REGIONS DEFINED EARLIER. AREAS COMMON TO TWO REGIONS EXIST ONLY IN THE LATER DEFINED REGION.

So, in the example above, the two smaller boxes overlay the large box. The material parameters assigned to the large box pertain only to the part of the large box not overlaid by the small boxes.

It is customary to make the first region define the entire outer boundary of the problem domain, and then to overlay the parts of the domain which differ in parameters from this default region. If you overlay all parts of the outer domain with subregions, then the outer region definition becomes invisible. It may be useful to do this in some cases, since it allows a localization of boundary condition specifications. Nevertheless, one of the subregions is superfluous, because it could be the default.

4.11.3.1. Reassigning Regional Parameters

Names previously defined in the **DEFINITIONS** section can be assigned a new value within a **REGION** by adding one or more assignments of the form

name = new_expression

immediately following the reserved word **REGION**.

When definitions are reassigned new values in this manner, the new value applies only to the region in which the reassignment occurs.

Example:

DEFINITIONS
K = 1 { the default value }
REGION 1 { assumes default, since no override is given }
START(0,0) LINE TO (10,0) TO (10,10) TO (0,10) TO CLOSE
REGION 2
K = 2 { both sub-boxes are assigned K=2 }
START(1,1) LINE TO (2,1) TO (2,2) TO (1,2) TO CLOSE
START(5,5) LINE TO (6,5) TO (6,6) TO (5,6) TO CLOSE


```
REGION 3 { again assumes the default }
START(3,3) LINE TO (4,3) TO (4,4) TO (3,4) TO CLOSE
```

4.11.3.2. Regions in One Dimension

In one-dimensional domains, the concept that a **REGION** bounds a finite area by closing on itself is no longer true. In one dimension, it is sufficient to define a path from the start of a material region to its finish. (Referencing **CLOSE** in a 1D bounding path will cause serious troubles, because the path will retrace itself.)

For example, the statements

```
REGION 1
START(0) LINE TO (5)
```

are sufficient to define a region of material extending from location 0 to location 5 in the 1D coordinate system.

In order to maintain grammatical consistency with two- and three-dimensional constructs, omitting the parentheses is *not* permitted.

Other general characteristics of **REGIONS** remain in force in one-dimensional domains:

Later **REGIONS** overlay earlier **REGIONS**, material properties are defined following the **REGION** keyword, and so forth.

4.11.3.3. Regions in Three Dimensions

The concept of a **REGION** in 3D domains retains the same character as for 2D domains.

The **REGION** is a partition of the 2D projection of the figure, and is extruded into the third dimension according to the **EXTRUSION** specification.

A material compartment in 3D is uniquely defined by the **REGION** of the projection which bounds it, and the **LAYER** of the extrusion in which it resides.

For further discussion of the 3D extensions of the **BOUNDARIES** section, see the User Guide chapter Using FlexPDE in Three-Dimensional Problems.

4.11.3.4. Regional Parameter Values in 3D

In three-dimensional problems, a redefinition of a parameter inside a **REGION** causes the parameter to be redefined in all layers of the layer stack above the region. To cause the parameter to be redefined only in a selected layer, use the **LAYER** qualifier, as in

```
LAYER number name = new_expression
LAYER "layer_name" name = new_expression
```

The **LAYER** qualifier acts on all subsequent parameter redefinitions, until a new **LAYER** qualifier or a functionally distinct clause breaks the group of redefinitions.

Example:

The following descriptor fragment shows the redefinition of a parameter **K** in various contexts:

```
DEFINITIONS
K=1 { 1 }

BOUNDARIES
LAYER 1 K=2 { 2 }
REGION 1
K=3 { 3 }
LAYER 2 K=4 { 4 }
START(0,0) LINE TO ....
```

Line { 1 } defines the default value.

Line { 2 } (valid only in 3D) defines the value *in layer 1 of all regions*.

Line { 3 } redefines the value *in region 1 only*, in all layers of a 3D domain.

Line { 4 } (valid only in 3D) defines the value *in layer 2 of region 1 only*.

4.11.3.5. Limited Regions in 3D

In three dimensional problems, many figures do not fit readily into the extrusion model. In particular, there are frequently features that in reality exist only at very restricted positions in the extrusion dimension, and which create poor meshes when extruded throughout the domain.

FlexPDE implements the concept of **LIMITED REGIONS** to accommodate this situation.

A **LIMITED REGION** is defined as one that is considered to exist only in specified layers or surfaces of the domain, and is absent in all other layers and surfaces.

The **LIMITED REGION** will be constructed only in layers and surfaces specifically stated in the body of the REGION definition.

An example of this type of structure might be a transistor, where the junction structure of the device is present only in a very thin layer of the domain, while the substrate occupies the majority of the volume.

In earlier versions of FlexPDE, the shape of the junction structure was propagated and meshed throughout the extrusion dimension. In version 4, the structure can be restricted, or **LIMITED**, to a single layer or a few layers.

For example, the following descriptor fragment defines a 3-unit cube with a 0.2-unit cubical structure in the center. The small structure is present in the layer 2 mesh only.

```
EXTRUSION Z=0, 1.4, 1.6, 3
```

```
BOUNDARIES
```

```
REGION 1
```

```
START(0,0) LINE TO (3,0) TO (3,3) TO (3,0) TO CLOSE
```

```
LIMITED REGION 2
```

```
LAYER 2 K=9
```

```
START(1.4,1.4)
```

```
LINE TO (1.6,1.4) TO (1.6,1.6) TO (1.4,1.4) TO CLOSE
```

See the User Guide section "Limited Regions" for a graphical example of this facility.

Examples:

Samples | Misc | 3D_Domains | 3D_Limited_Region.pde

4.11.3.6. Empty Layers in 3D

In three dimensional problems, it is sometimes necessary to define holes or excluded regions in the extruded domain. This may be done using the **VOID** qualifier. **VOID** has the syntax of a parameter redefinition.

For example, the following descriptor fragment defines a 3-unit cube with a 1-unit cubical hole in the center:

```
EXTRUSION Z=0,1,2,3
```

```
BOUNDARIES
```

```
REGION 1
```

```
START(0,0) LINE TO (3,0) TO (3,3) TO (3,0) TO CLOSE
```

```
REGION 2
```

```
LAYER 2 VOID
```

```
START(1,1) LINE TO (2,1) TO (2,2) TO (1,2) TO CLOSE
```

Examples:

Samples | Misc | 3D_Domains | 3D_Void.pde

4.11.4. Excludes

EXCLUDE subsections are used to describe closed domains which overlay parts of one or more **REGION** subsections. The domain described by an exclude subsection is excluded from the system.

EXCLUDE subsections must follow the **REGION** subsections which they overlay

EXCLUDE subsections are formed in the same manner as **REGION** subsections and can use all the same **LINE** and **ARC** segments.

4.11.5. Features

FEATURE subsections are used to describe non-closed entities which do not enclose a subdomain with definable material parameters.

FEATURE subsections are formed in the same manner as **REGION** subsections and can use all the same **LINE** and **ARC** segments. **FEATURE** subsections do not end with the reserve word **CLOSE**.

A FEATURE will be explicitly represented by nodes and cell sides.

FEATURE subsections are used when a problem has internal line sources; when it is desirable to calculate integrals along an irregular path; or when explicit control of the grid is required.

In 3D problems, **FEATURES** should be used to delineate any sharp breaks in the slope of extrusion surfaces. Unless mesh lines lie along the surface breaks, the surface modeling will be crude.

Example:

```
REGION 1 { an outer box }  
START(0,0) LINE TO (10,0) TO (10,10) TO (0,10) TO CLOSE
```

```
FEATURE { with a diagonal gridding line }  
START(0,0) LINE TO (10,10)
```

4.11.6. Ordering Regions

While not strictly enforced, it is recommended that all **REGION** subsections be listed before any **EXCLUDE** or **FEATURE** subsections and that all **EXCLUDE** subsections be listed before any **FEATURE** subsections.

It is further recommended that the first **REGION** subsection be formed by walking the outside boundary of the problem thereby enclosing the entire domain of the problem.

REGIONS defined later are assumed to overlay any previously listed **REGIONS**, and any properties assigned to a **REGION** will override properties previously assigned to the domains they overlay.

4.11.7. Numbering Regions

REGION, **EXCLUDE** and **FEATURE** subsections can be assigned numbers and/or names.

When numbers are assigned they should be in ascending sequential order beginning with one. It is recommended that numbers always be assigned.

When names are assigned they must take the form of a quoted string and must be placed immediately after either the reserved word **REGION**, **EXCLUDE**, or **FEATURE** or any number assigned to the **REGION**, **EXCLUDE**, or **FEATURE**. Assigned names must be unique to the **REGION**, **EXCLUDE** or **FEATURE** that they name.

Assigned region names are useful when region-restricted plots or volume integrals are desired.

Example:

```
REGION 2 'Thing'  
{...}
```

```
PLOTS  
contour(u) on 'Thing'
```

4.11.8. Fillets and Bevels

Any point in a path may be followed by the specification **FILLET(radius)** or **BEVEL(length)**. The point will be replaced by a circular arc of the specified radius, or by a bevel of the specified length. **FILLETS** and

BEVELS should not be applied to points which are the intersection of several segments, or confusion may ensue.

Example:

LINE TO (1,1) FILLET(0.01)

Example problem:

"Samples | Misc | Fillet.pde"

4.11.9. Boundary Conditions

The following forms of boundary condition specification may be applied to boundary segments:

VALUE (variable) = Expression

NATURAL (variable) = Expression

LOAD (variable) = Expression

CONTACT (variable) = Expression

VELOCITY (variable) = Expression

NOBC (variable)

The **variable** designated in the boundary condition specification identifies (by explicit association) the equation to which this boundary condition is to be applied.

VALUE (Dirichlet) Boundary Conditions

A **VALUE** segment boundary condition forces the solution of the equation for the associated variable to the value of **Expression** on a continuous series of one or more boundary segments. The

Expression may be an explicit specification of value, involving only constants and coordinates, or it may be an implicit relation involving values and derivatives of system variables.

NATURAL (Generalized Flux) Boundary Conditions

NATURAL and **LOAD** segment boundary conditions are synonymous. They represent a generalized flux boundary condition derived from the divergence theorem. The **Expression** may be an explicit

specification, involving only constants and coordinates, or it may be an implicit relation involving values and derivatives of system variables. The Natural boundary condition reduces to the Neumann boundary condition in the special case of the Poisson equation. See the User Guide chapter Natural Boundary Conditions for information on the implementation of Natural boundary conditions.

CONTACT (Discontinuous Variable) Boundary Conditions

See "Jump Boundaries" in the next section.

VELOCITY (Time Derivative) Boundary Conditions

This boundary condition imposes a specified time derivative on a boundary value (time-dependent problems only). This condition is especially useful in specifying moving boundaries, by applying it to the surrogate coordinate variable. If you have declared a velocity variable which is applied to a coordinate, then you should lock the surrogate coordinate to the mesh velocity variable at the boundary using a **VELOCITY()** boundary condition.

Terminating the current BC

NOBC(VARIABLE) is used to turn off a previously specified boundary condition on the current path. It is equivalent in effect to **NATURAL(VARIABLE)=0** (the default boundary condition), except that it will not lead to "Multiple Boundary Condition Specification" diagnostics.

[Note:

The **NEUMANN**, **DNORMAL** and **DTANGENTIAL** boundary conditions supported in earlier versions have been deleted due to unreliable behavior. They may be restored in later versions. In most cases, derivative boundary conditions are more appropriately applied through the **NATURAL** boundary condition facility.] [4.2.0]

4.11.9.1. Syntax of Boundary Condition Statements

Segment boundary conditions are added to the problem descriptor by placing them in the **BOUNDARIES** section.

Segment boundary conditions must immediately precede one of the reserved words **LINE** or **ARC** and cannot precede the reserved word **TO**.

A top-down system is used for applying segment boundary conditions to the equations. Following the **START** point specification in each path definition, a segment boundary condition is set up for each variable/equation. It is recommended that a boundary condition be specified for each variable/equation. If no other segment boundary condition is specified no error will occur and a **NATURAL(VARIABLE) = 0** segment boundary condition is assumed.

Under the top-down system, as boundary segments occur, the previously specified segment boundary condition will continue to hold until a new boundary condition is specified.

If the recommendation is followed that **REGION 1** be formed by walking the outside boundary of the problem, thereby enclosing the entire domain of the problem, then for most problems segment boundary conditions need only be specified for the segments in **REGION 1**.

4.11.9.2. Point Boundary Conditions

POINT VALUE boundary conditions can be added by placing

POINT VALUE (variable) = expression

following a coordinate specification. The stated value will be imposed only on the coordinate point immediately preceding the specification.

POINT LOAD boundary conditions can be added by placing

POINT LOAD (variable) = expression

following a coordinate specification. The stated load will be imposed as a lumped source on the coordinate point immediately preceding the specification.

A Caveat:

The results achieved by use of these specifications are frequently disappointing.

A diffusion equation, for example, $\text{div}(\text{grad}(u)) + s = 0$, can support solutions of the form $u = A - B \cdot r - C \cdot r^2$, where r is the distance from the point value and A , B and C are arbitrary constants. By the

superposition principle, FlexPDE is free to add such shapes to the computed solution in the vicinity of the point value, without violating the PDE. A **POINT VALUE** condition usually leads to a sharp spike in the solution, pulling the value up to that specified, but otherwise leaving the solution unmodified.

The **POINT LOAD** is not subject to this same argument, but since it is a load without scale, it will frequently produce a dense mesh refinement around the point.

A better solution is to use a distributed load or an extended value boundary segment, ring or box.

4.11.9.3. Boundary conditions in 1D

The idea that a boundary condition applies along the length of a boundary segment, while meaningful in two and three dimensions, is meaningless in one dimension, since it is the value along the segment that is the object of the computation.

In one dimensional problems, therefore, it is necessary to use the Point boundary condition described in the previous section for all boundary condition specifications.

Example:

BOUNDARIES

REGION 1

START(0) POINT VALUE(u)=1

LINE TO (5) POINT LOAD(u)=4

4.11.9.4. Boundary Conditions in 3D

In three-dimensional problems, an assignment of a segment boundary condition to a region boundary causes that boundary condition to be applied to the "side walls" of all layers of the layer stack above the region. To selectively apply a boundary condition to the "side walls" of only one layer, use the **LAYER** qualifier, as in

LAYER number VALUE(variable) = expression

LAYER "layer_name" VALUE(variable) = expression

The **LAYER** qualifier applies to all subsequent boundary condition specifications until a new **LAYER** qualifier is encountered, or the segment geometry (**LINE** or **ARC**) statements begin.

The boundary conditions on the extrusion surfaces themselves (the slicing surfaces) can be specified by the **SURFACE** qualifier preceding the boundary condition specification.

Consider a simple cube. The **EXTRUSION** and **BOUNDARIES** sections might look like this:

```
EXTRUSION  z = 0,1

BOUNDARIES
  SURFACE 1 VALUE(U)=0      { 1 }
  REGION 1
    SURFACE 2 VALUE(U)=1    { 2 }
    START(0,0)
    NATURAL(U)=0            { 3 }
    LINE TO (1,0)
    LAYER 1 NATURAL(U)=1    { 4 }
    LINE TO (1,1)
    NATURAL(U)=0            { 5 }
    LINE TO (0,1) TO CLOSE
```

Line { 1 } specifies a fixed value of 0 for the variable U over the entire surface 1 (ie. the Z=0 plane).

Line { 2 } specifies a value of 1 for the variable U on the top surface *in region 1 only*.

Line { 3 } specifies an insulating boundary on the Y=0 side wall of the cube.

Line { 4 } specifies a flux (whose meaning will depend on the PDE) on the X=1 side wall *in layer 1 only*.

Line { 5 } returns to an insulating boundary on the Y=1 and X=0 side walls.

[Of course, in this example the restriction to region 1 or layer 1 is meaningless, because there is only one of each.]

4.11.9.5. Jump Boundaries

In the default case, FlexPDE assumes that all variables are continuous across internal material interfaces. This is a consequence of the positioning of mesh nodes along the interface which are shared by the cells on both sides of the interface.

FlexPDE 5 supports the option of making variables discontinuous at material interfaces (see the "Discontinuous Variables" in the User Guide for tutorial information).

This capability can be used to model such things as contact resistance, or to completely decouple the variables in adjacent regions.

The key words in employing this facility are **CONTACT** and **JUMP**.

The conceptual model is that of contact resistance, where the difference in voltage V across the interface (the JUMP) is given by

$$V_2 - V_1 = R \cdot \text{current}$$

In the general case, the role of "current" is played by the generalized flux, or Natural boundary condition. (See the User Guide for further discussion of Natural Boundary Conditions.) The **CONTACT** boundary condition is a special form of **NATURAL**, which defines a flux but also specifies that FlexPDE should model a double-valued boundary.

So the method of specifying a discontinuity is

$$\text{CONTACT}(V) = (1/R) \cdot \text{JUMP}(V)$$

"CONTACT(V)", like "NATURAL(V)", means the outward normal component of the generalized flux as seen from any cell. So from any cell, the meaning of "JUMP(V)" is the difference between the interior and exterior values of V at a point on the boundary. Two cells sharing a boundary will then see JUMP values and outward normal fluxes of opposite sign. "Flux" is automatically conserved, since the same numeric value is used for the flux in both cells.

Specifying a **CONTACT** boundary condition at an internal boundary causes duplicate mesh nodes to be generated along the boundary, and to be coupled according to the **JUMP** boundary condition statement.

Specifying a very small (1/R) value effectively decouples the variable across the interface.

Example Problems:

```
"Samples | Misc | Discontinuous_Variables |  
Thermal_Contact_Resistance.pde"  
"Samples | Misc | Discontinuous_Variables |  
Contact_Resistance_Heating.pde"  
"Samples | Misc | Discontinuous_Variables |  
Transient_Contact_Resistance_Heating.pde"
```

4.11.9.6. Periodic Boundaries

FlexPDE supports periodic and antiperiodic boundary conditions in two or three dimensions.

Periodicity in the X-Y Plane

Periodicity in a two-dimensional problem, or in the extrusion walls of a three-dimensional problem, is invoked by the **PERIODIC** or **ANTIPERIODIC** statement.

The **PERIODIC** statement appears in the position of a boundary condition, but the syntax is slightly different, and the requirements and implications are more extensive.

The syntax is:

```
PERIODIC ( X_mapping, Y_mapping )  
ANTIPERIODIC ( X_mapping, Y_mapping )
```

The mapping expressions specify the arithmetic required to convert a point (X,Y) in the immediate boundary to a point (X',Y') on a remote boundary. The mapping expressions must result in each point on the immediate boundary being mapped to a point on the remote boundary. Segment endpoints must map to segment endpoints. The transformation must be invertible; do not specify constants as mapped coordinates, as this will create a singular transformation.

The periodic boundary statement terminates any boundary conditions in effect, and instead imposes equality of all variables on the two

boundaries. It is still possible to state a boundary condition on the remote boundary, but in most cases this would be inappropriate.

The periodic statement affects only the next following **LINE** or **ARC** path. These paths may contain more than one segment, but the next appearing **LINE** or **ARC** statement terminates the periodic condition unless the periodic statement is repeated.

Periodicity in the Z-Dimension

Periodicity in the extruded dimension is invoked by the modifier **PERIODIC** or **ANTIPERIODIC** before the **EXTRUSION** statement, for example,

PERIODIC EXTRUSION Z=0,1,2

In this case, the top and bottom extrusion surfaces are assumed to be conformable, and the values are forced equal (or sign-reversed) along these surfaces.

Restrictions

Each node in the finite element mesh can have at most one periodic image. This means that two-way or three-way periodicity cannot be directly implemented. Usually it is sufficient to introduce a small gap in the periodic boundaries, so that each corner is periodic with only one other corner of the mesh.

Example Problems:

```
"Samples | Misc | Periodicity | periodic.pde"  
"Samples | Misc | Periodicity | periodaz.pde"  
"Samples | Misc | Periodicity | antiperiodic.pde"  
"Samples | Misc | Periodicity | 3d_xperiodic.pde"  
"Samples | Misc | Periodicity | 3d_zperiodic.pde"  
"Samples | Misc | Periodicity | 3d_antiperiodic.pde"
```

4.11.10. Fixed Points

Arbitrary points within a 2D problem domain can be designated for special treatment by the statement

FIXED POINT (X,Y)

The stated point will be represented by a mesh node, and may effect the density of mesh nodes in its neighborhood.

POINT VALUE and **POINT LOAD** boundary conditions may be applied to **FIXED POINTS**

[**Note:** This facility is not yet fully implemented in 3D.]

4.12. Front

The **FRONT** section is used to define additional criteria for use by the adaptive regridding. In the normal case, FlexPDE repeatedly refines the computational mesh until the estimated error in the approximation of the PDE's is less than the declared or default value of **ERRLIM**. In some cases, where meaningful activity is confined to some kind of a propagating front, it may be desirable to enforce greater refinement near the front. In the **FRONT** section, the user may declare the parameters of such a refinement.

The **FRONT** section has the form:

FRONT (criterion, delta)

The stated **criterion** will be evaluated at each node of the mesh. Cells will be split if the values at the nodes span a range greater than $(-\text{delta}/2, \text{delta}/2)$ around zero.

That is, the grid will be forced to resolve the **criterion** to within **delta** as it passes through zero.

Example:

Samples | Misc | Front.pde

4.13. Resolve

The **RESOLVE** section is used to define additional criteria for use by the adaptive regridding. In the normal case, FlexPDE repeatedly refines the computational mesh until the estimated error in the approximation of the PDE's is less than the declared or default value of **ERRLIM**. In some cases, this can be achieved with a much less dense mesh than is necessary to make pleasing graphical presentation of derived quantities, such as derivatives of the system variables, which are much less smooth than the variables themselves. In the **RESOLVE** section, the user may declare one or more additional functions whose detailed resolution is important. The section has the form:

RESOLVE (spec1) , (spec2) , (spec3) {...}

Here, each **spec** may be either an expression, such as "(shear_stress)", or an expression followed by a weighting function, as in "(shear_stress, x^2)".

In the simplest form, only the expressions of interest need be presented. In this case, for each stated function, FlexPDE will

- form a Finite Element interpolation of the stated function over the computational mesh
- find the deviation of the interpolation from the exact function
- split any cell where this deviation exceeds **ERRLIM** times the global RMS value of the function.

In the weighted form, an importance-weighting function is defined, possibly to restrict the effective domain of resolution. The splitting operation described above is modified to multiply the deviation at each point by the weight function at that point. Areas where the weight is small are therefore subjected to a less stringent accuracy requirement.

Example:

Samples | Misc | Resolve.pde

4.14. Time

The **TIME** section is used in time dependent problem descriptors to specify a time range over which the problem is to be solved. It supports the following alternative forms:

FROM time1 TO time2
FROM time1 BY increment TO time2
FROM time1 TO time2 BY increment

Where:

time1 is the beginning time
time2 is the ending time.
increment is an optional specification of the initial time step for the solution. (the default initial time step is $1e-4 \times (\text{time2} - \text{time1})$).

All time dependent problem descriptors must include statements which define the time range.

While the problem descriptor language supports alternate methods of specifying a time range, it is recommended that all time dependent problems include the **TIME** section to specify the total time domain of the problem.

Halting Execution

The time range specification may optionally be followed by a **HALT** statement:

HALT minimum

This statement will cause the computation to halt if the automatically controlled timestep drops below **minimum**. This facility is useful when inconsistencies in data or discontinuities in parameters cause the timestep controller to become confused.

HALT condition

Here the **condition** can be any relational operation, such as **globalmax(myvariable) < 204**. If the condition is met on any timestep, the computation will be halted.

4.15. Monitors and Plots

The **MONITORS** section, which is optional, is used to list the graphic displays desired at intermediate steps while a problem is being solved.

The **PLOTS** section, which is optional, is used to list the graphic displays desired on completion of a problem or stage, or at selected problem times.

PLOTS differ from **MONITORS** in that they are written to the permanent .PG4 record for viewing after the run is completed. (For debugging purposes the global selector **HARDMONITOR** can be used to force **MONITORS** to be written to the .PG4 file.)

Plot statements and Monitor statements have the same form and function.

The basic form of a **PLOT** or **MONITOR** statement is:

display_specification (plot_data) display_modifiers

display_specification must be one of the known plot types, as described in the next section.

In some cases, multiple **plot_data** arguments may be provided. There may be any number of **display_modifiers**, with meanings determined by the **display_specification**.

The various **display_modifiers** supported by FlexPDE are listed in the "Graphic Display Modifiers" section.

An Exhortation:

The **MONITORS** facility has been provided to allow users to see immediate feedback on the progress of their computation, and to display any and all data that will help diagnose failure or misunderstanding. Please use **MONITORS** extensively, especially in the early phases of model development! Since they do not write to the .pg4 storage file, they can be used liberally without causing disk file bloat. After the model is performing successfully, you can remove them or comment them out. Many user pleas for help recieved by PDE Solutions could be avoided if the user had included enough **MONITORS** to identify the cause of trouble.

Examples:

Samples\Misc\Plottest.pde

[Note: All example problems contain PLOTS and MONITORS].

4.15.1. Graphics Display and Data Export Specifications

The **MONITORS** or **PLOTS** sections can contain one or more display specifications of the following types:

CDF (arg1 [,arg2,...])

Requests the export of the listed values in netCDF version 3 format. The output will be two or three dimensional, following the current coordinate system or subsequent **ON SURFACE** modifiers. The included domain can be zoomed. If the **FILE** modifier does not follow, then the output will be written to a file "<problem>_<sequence>.cdf". Staged, eigenvalue and time-dependent problems will stack subsequent outputs in the same file, consistent with netCDF conventions. CDF uses a regular rectangular grid, so interface definition may be ragged. Use **ZOOM** to show details.

CONTOUR (arg)

Requests a two dimensional contour map of the argument, with levels at uniform intervals of the argument.

CONTOUR (arg1, arg2)

Requests a two dimensional contour map of both **arg1** and **arg2**, each with levels at independent uniform intervals. The displayed level table pertains only to **arg1**, and contours of **arg2** are not tagged.

ELEVATION (arg1, [,arg2,...]) path

Requests a two dimensional display (some times called a line-out) which displays the value of its argument(s) vertically and the value of its path horizontally. Each **ELEVATION** listed must have at least one argument and may have multiple arguments separated by commas. **path** can be either a line segment specified using the forms **FROM (X1,Y1) TO (X2,Y2)** or **ON name**, where **name** is a literal string selecting a path named in the **BOUNDARIES** section.

GRID (arg1, arg2)

Requests a two dimensional plot of the computation grid, with nodal coordinates defined by the two arguments. Grids are especially useful for displaying material deformations. (In 3D problems, a two-argument **GRID** plot will show a cut-plane, and must be followed by an **ON** specification. 3D cut plane grid plots do not necessarily accurately represent the computational grid.)

GRID (arg1, arg2, arg3)

Requests a three dimensional plot of the computation grid, with nodal coordinates defined by the three arguments. Only the outer surface of the grid will be drawn. This plot can be interactively rotated, as with **SURFACE** plots.

SUMMARY

This plot type defines a text page on which only **REPORT** items may appear. A **SUMMARY** page can be **EXPORTed** to produce text reports of scalar values.

SUMMARY ('string')

If a string argument is given with a **SUMMARY** command, it will appear as a page header on the summary page.

SURFACE (arg)

A quasi three dimensional surface which displays its argument vertically. If no **VIEWPOINT** clause is used, the viewing azimuth defaults to 216 degrees, the distance to three times the size, and the viewing elevation to 30 degrees.

TABLE (arg1 [,arg2,...])

Requests the export of the listed values in tabular ASCII format. The output will be two or three dimensional, following the current coordinate system or subsequent **ON** modifiers. The included domain can be zoomed. If the **FILE** modifier does not follow, then the output will be written to a file "<problem>_<sequence>.tbl". Staged, eigenvalue and time-dependent problems will create separate files for each stage or mode, with additional sequencing numbers in the name. **TABLE** output uses a regular rectangular grid, so interface definition may be lost. Use **ZOOM** to show details.

TECPLOT (arg1 [,arg2,...])

Requests the export of the listed values to a file readable by the TecPlot visualization system. The output will be two or three dimensional, following the current coordinate system. The entire mesh is exported. If the **FILE** modifier does not follow, then the output will be written to a file "<problem>_<sequence>.dat". Staged, eigenvalue and time-dependent problems will stack subsequent outputs in the same file, consistent with TecPlot conventions. TecPlot uses the actual triangular or tetrahedral computation mesh (subdivided to linear basis), so material interfaces are preserved.

TRANSFER (arg1 [,arg2,...])

Requests the export of the listed values and finite element mesh data in a file readable by FlexPDE using the **TRANSFER** or **TRANSFERMESH** input command. This method of data transfer between FlexPDE problems retains the full accuracy of the computation, without the error introduced by the rectangular mesh of the **TABLE** function. The exported domain cannot be zoomed. If the **FILE** modifier does not follow, then the output will be written to a file "<problem>_<sequence>.dat". Staged, eigenvalue and time-dependent problems are not supported. This export format uses the actual computation mesh, so material interfaces are preserved. The full computation mesh is exported.

VECTOR (vector)

Requests a two dimensional display of directed arrows in which the direction and magnitude of the arrows is set by the **vector** argument. The origin of each arrow is placed at its reference point.

VECTOR (arg1, arg2)

Requests a two dimensional display of directed arrows in which the horizontal and vertical components of the arrows are given by **arg1** and **arg2**. The origin of each arrow is placed at its reference point.

VTK (arg1 [,arg2,...])

Requests the export of the listed values to a file in VTK (Visualization Tool Kit) format for display by visualization systems such as VisIt. The output will be two or three dimensional, following the current coordinate system. The entire mesh is exported. If the **FILE** modifier does not follow, then the output will be written to a file "<problem>_<sequence>.vtk". Staged, eigenvalue and time-dependent problems will produce a family of files distinguished by the sequence number. VTK format uses the actual triangular or tetrahedral computation mesh, so material interfaces are preserved. The VTK

format supports quadratic finite element basis directly, but not cubic. To export from cubic-basis computations, use VTKLIN. [4.1]

VTKLIN (arg1 [,arg2,...])

Produces a VTK format file in which the native cells of the FlexPDE computation have been converted to a set of linear-basis finite element cells. This command may be used to export to VTK visualization tools from cubic-basis FlexPDE computations, or in cases where the visualization tool does not support quadratic basis.

For all commands, the argument(s) can be any valid expression.

4.15.2. Graphic Display Modifiers

The appearance of any display can be modified by adding one or more of the following clauses:

AREA_INTEGRATE

Causes CONTOUR and SURFACE plots in cylindrical geometry to be integrated with $dr*dz$ element, rather than default $2*pi*r*dr*dz$ volume element.

AS 'string'

Changes the label on the display from the evaluated expression to **string**.

BLACK

Draws current plot in black color only.

BMP

BMP (pixels)

BMP (pixels, penwidth)

Selects automatic creation of a graphic export file in BMP format. **pixels** is the horizontal pixel count, which defaults to 1024 if omitted. **penwidth** is an integer (0,1,2 or 3) which specifies the width of the drawn lines, in thousandths of the drawing width (0 means thin). The export file name is the problem name with plot number and sequence number appended. The file name cannot be altered.

DROPOUT

Marks **EXPORT** and **TABLE** output points which fall outside the problem domain as "external", for compatibility with versions prior to 2.21. This modifier affects only **EXPORTS** and **TABLES** with **FORMAT** strings (see below).

EMF

EMF (pixels)

EMF (pixels, penwidth)

Windows version only. Produces a Microsoft Windows Enhanced Metafile output. **pixels** is the horizontal pixel count of the reference window, which defaults to 1024 if omitted. **penwidth** is an integer (0,1,2 or 3) which specifies the width of drawn lines, in thousandths of the drawing width (0 means thin). The export file name is the problem name with plot number and sequence number appended. The file name cannot be altered. (**Warning:** FlexPDE uses Windows rotated fonts to plot Y-labels and axis labels on surface plots. Microsoft Word can read and resize these pictures, but its picture editor cannot handle them, and immediately "rectifies" them to horizontal.)

EPS

Produces an Encapsulated PostScript output. The graphic is a 10x7.5 inch landscape-mode format with 7200x5400 resolution.

EXPORT

Causes a disk file to be written containing the data represented by the associated **MONITOR** or **PLOT**. A regular rectangular grid will be constructed, and the data will be printed in a format suitable for reading by the FlexPDE **TABLE** function. The dimension of the grid will be determined by the plot grid density appropriate to the type of plot. (This is a renaming of the older **PRINT** modifier) The format of **EXPORTED** data may be controlled by the **FORMAT** modifier (see below).

EXPORT (n)

Modifies the **EXPORT** command by specifying the dimension of the printed data grid. For two- or three-dimensional plots, the grid will be (n x n) or (n x n x n).

EXPORT (nx, ny)

Modifies the **EXPORT** command by specifying the dimension of the printed data grid.

EXPORT (nx, ny, nz)

Modifies the **EXPORT** command by specifying the dimension of the printed data grid.

FILE 'string'

Overrides the default naming convention for files created by the **EXPORT** or **PRINT** modifiers, and writes the file named '**string**' instead.

FIXED RANGE (arg1, arg2)

Changes the dynamically set range used for the variable axis to a minimum value of **arg1** and a maximum of **arg2**. Data outside this range is not plotted. (See also: **RANGE**, below)

FORMAT 'string'

This modifier replaces the default format of the **EXPORT** or **PRINT** modifiers, or of the **TABLE** output command. When this modifier appears, the output will consist of one line for each point in the export grid. The contents of this line will be completely controlled by the format string as follows:

- all characters except "#" will be copied literally to the output line.
- "#" will be interpreted as an escape character, and various options will be selected by the character following the "#":
 - #x, #y, #z and #t will print the value of the spatial coordinates or time of the data point;
 - #1 through #9 will print the value of the corresponding element of the plot function list;
 - #b will write a taB character;
 - #r will cause the remainder of the format string to be repeated for each plot function in the plot list;
 - #i inside a repeated string will print the value of the current element of the plot function list.

See the example problems "export_format" and "export_history".

FRAME (X, Y, Wide, High)

Forces the plot frame to the specified coordinates, regardless of the size of the problem domain. This allows the creation of consistently-sized plots in moving-mesh problems. See "Samples | Moving_Mesh | Piston.pde".

GRAY

Draws current plot with a 32-level gray scale instead of the default color palette.

INTEGRATE

Causes a report of the integral under the plotted function. For CONTOUR and SURFACE plots, this is a volume integral (with Cartesian element $dx*dy*1$ or cylindrical element $2*pi*r*dr*dz$). For ELEVATIONS, it is a surface integral (with Cartesian element $dl*1$ and cylindrical element $2*pi*r*dl$). (See also Area_Integrate, Line_integrate).

This integral differs from a REPORT(INTEGRAL(...)) in that this command will integrate on the plot grid, while the REPORT will integrate on the computation grid.

This modifier can be globally imposed by use of PLOTINTEGRATE in the SELECT section.

LINE_INTEGRATE

Causes ELEVATIONS in cylindrical geometry to be integrated with dl element, rather than default $2*pi*r*dl$ element.
(See also: Integrate, Area_Integrate)

LOG

LINLOG

LOGLIN

LOGLOG

Changes the default linear scales used to those specified by the scaling command. LOG is the same as LINLOG, and specifies logarithmic scaling in the data coordinate.

<lx><ly><lz>

Changes the default linear scales used to those specified by the scaling command. Each of <lx>, <ly> and <lz> can be either LIN or LOG, and controls the scaling in the associated dimension.

LOG (number)

...combinations as above

Limits the number of decades of data displayed to <number>. This effect can also be achieved globally by the Selector LOGLIMIT.

MERGE

Sends EXPORT output for all stages or plot times to a single output file. (This is the default for TECPLOT output). This option can be set globally by SELECT PRINTMERGE.

MESH

In SURFACE plots, causes the surface to be displayed as a hidden-line drawing of the meshed surface. This display is more suitable on some hardcopy devices.

NOHEADER

Deletes the problem-identification header from EXPORT output.

NOMERGE

Sends EXPORT output for each stage or plot time to a separate output file. (This is the default for EXPORT output).

NOMINMAX

Deletes "o" and "x" marks at min and max values on contour plot.

NORM

In VECTOR plots, causes all vectors to be drawn with the same length. Only the color identifies different magnitudes.

NOTAGS

Suppresses labelling tags on contour or elevation plot. This can be applied globally with SELECT NOTAGS.

NOTIPS

Plots VECTORS as line segments without heads. The line segment will be centered on the reference point.

ON "name"

ON LAYER number

ON LAYER "name"

ON REGION number

ON REGION "name"

ON SURFACE number

ON SURFACE "name"

ON equation

Displays will be restricted to the selected region, surface or layer. See "Controlling the Plot Domain".

PAINTED

Fills areas between contour lines with color. (This is slower than conventional contour lines.)

PAINTMATERIALS

PAINTREGIONS

Draw color-filled grid plot. These local flags are equivalent to and override the corresponding global flags set in the SELECT section. They affect only the current plot.

PNG

PNG (pixels)

PNG (pixels, penwidth)

Selects automatic creation of a graphic export file in PNG format. "pixels" is the horizontal pixel count, which defaults to 1024 if omitted. "penwidth" is an integer (0,1,2 or 3) which specifies the width of the drawn lines, in thousandths of the pixel width (0 means thin). The export file name is the problem name with plot number and sequence number appended. The file name cannot be altered.

POINTS = n

Overrides the default plot grid size and uses n instead. Two and three dimensional exports will use n in all dimensions.

POINTS = (nx , ny)

For two-dimensional export commands, the two-dimensional grid can be explicitly controlled.

POINTS = (nx, ny, nz)

For three-dimensional exports, the three-dimensional grid can be explicitly controlled.

PPM

PPM (pixels)

PPM (pixels, penwidth)

Selects automatic creation of a graphic export file in PPM format. "pixels" is the horizontal pixel count, which defaults to 1024 if omitted. "penwidth" is an integer (0,1,2 or 3) which specifies the width of the drawn lines, in thousandths of the pixel width (0 means thin). The export file name is the problem name with plot number and sequence number appended. The file name cannot be altered.

PRINT

Equivalent to EXPORT

PRINT (n)

Equivalent to EXPORT(n)

PRINT (nx, ny)

Equivalent to EXPORT(nx,ny)

PRINT (nx, ny, nz)

Equivalent to EXPORT(nx,ny,nz)

RANGE (arg1, arg2)

Changes the dynamically set range used for the variable axis to a minimum value of **arg1** and a maximum of **arg2**. If the calculated value of the variable falls outside of the range argument, the range argument is ignored and the dynamically calculated value is used. (See also: Fixed Range)

VIEWPOINT(X, Y, angle)

With SURFACE() plots, the VIEWPOINT modifier sets the viewing azimuth and perspective distance and the viewing elevation angle.

VOL_INTEGRATE

Causes CONTOURS and SURFACE plots in cylindrical geometry to be integrated with $2\pi r dr dz$ element. This is the default, and is equivalent to INTEGRATE. (See also: Integrate, Area_Integrate)

XPM

XPM (pixels)

XPM (pixels, penwidth)

Selects automatic creation of a graphic export file in XPM format. "pixels" is the horizontal pixel count, which defaults to 1024 if omitted. "penwidth" is an integer (0,1,2 or 3) which specifies the width of the drawn lines, in thousandths of the pixel width (0 means thin). The export file name is the problem name with plot number and sequence number appended. The file name cannot be altered.

ZOOM (X, Y, Wide, High)

Expands (zooms) a selected area of the display or export, with (X,Y) defining the lower left hand corner of the area and (Wide,High) defining the extent of the expanded area. In 3D cut planes, the X and Y coordinates refer to the horizontal and vertical dimensions in the cut plane.

ZOOM (X, Y, Z, Xsize, Ysize, Zsize)

Expands (zooms) a selected volume of an export, with (X,Y,Z) defining the lowest corner of the volume and (Xsize,Ysize,Zsize) defining the extent of the included volume.

4.15.3. Controlling the Plot Domain

"ON" selectors

The primary mechanism for controlling the domain over which plot data are constructed is the "ON" statement, which has many forms:

ON "name"
ON REGION "name"
ON REGION number

In three-dimensional problems, the following are also meaningful:

ON LAYER "name"
ON SURFACE "name"
ON LAYER number
ON SURFACE number
ON equation

The first form selects a boundary path, region, layer or surface depending on the definition of the "name". (It is actually redundant to specify SURFACE "name", etc, since the fact that a surface is being specified should be clear from the "name" itself. Nevertheless, the forms are acceptable.)

In many cases, particularly in 3D, more than one "ON" clause can be used for a single plot, since each "ON" clause adds a restriction to those already in effect. There is a direct correspondence between the "ON" clauses of a plot statement and the arguments of the various INTEGRAL operators, although some of the allowable integral selections do not have valid corresponding plot options.

In two dimensional geometries, area plots which are not otherwise restricted are assumed to be taken over the entire problem domain.

Contours, Surface Plots, Grid Plots and Vector Plots

Contours. "surfaces" (3D topographic displays), grid plots and vector plots must be taken on some kind of two dimensional data surface, so in 3D problems these plot commands are incomplete without at least one "ON" clause. This can be an extrusion-surface name, or a cut-plane equation (it cannot be a projection-plane boundary path). For example, in a 3D problem,

CONTOUR(...) ON SURFACE 2

requests a contour plot of data evaluated on the second extrusion surface.

CONTOUR(...) ON SURFACE "top"

requests a contour plot of data evaluated on the extrusion surface named "top".

CONTOUR(...) ON X=Y

requests a contour plot of data evaluated on the cut plane where $x=y$.

In addition to a basic definition of the data surface, "ON" clauses may be used to restrict the display to an arbitrary REGION or LAYER. In 2D, a REGION restriction will display only that part of the domain which is in the stated region:

CONTOUR(...) ON REGION 2

requests a contour plot of data evaluated on REGION 2.

Similarly, in 3D,

CONTOUR(...) ON SURFACE 2 ON REGION 2

requests a contour plot of data evaluated on extrusion surface 2, restricted to that part of the surface lying above REGION 2 of the baseplane projection.

CONTOUR(...) ON SURFACE 2 ON REGION 2 ON LAYER 3

requests a contour plot of data evaluated on extrusion surface 2, restricted to that part of the surface lying above REGION 2 of the baseplane projection, and with the evaluation taken in layer 3, which is assumed to be bounded by the selected surface.

Cut Planes in 3D

Contours, surface plots and vector plots can also be specified on cut planes by giving the general formula of the cutting plane:

CONTOUR(...) ON X = expression

requests a contour plot of data evaluated on the Y-Z plane where X is the specified value.

Cut planes need not be simple coordinate planes:

CONTOUR(...) ON X=Y

requests a contour plot of data evaluated on the plane containing the z-axis and the 45 degrees line in the XY plane.

The coordinates displayed in oblique cut planes have their origin at the point of closest approach to the origin of the domain coordinates. The axes are chosen to be aligned with the nearest domain coordinate axes.

Elevation Plots

Elevation plots can be specified by endpoints of a line:

ELEVATION(...) FROM (x1,y1) TO (x2,y2)

ELEVATION(...) FROM (x1,y1,z1) TO (x2,y2,z2)

The plot will be displayed on the straight line connecting the specified endpoints. These endpoints might span only a small part of the problem domain, or they might exceed the domain dimensions somewhat, in which case the plot line will be truncated to the interior portion.

In 2D geometry only, an elevation plot may be specified by the name of a boundary path, as in

ELEVATION(...) ON "outer_boundary"

These boundary-path elevations can be additionally restricted as to the region in which the evaluation is to be made:

ELEVATION(...) ON "inner_boundary" ON REGION "core"

This form requests that the evaluation of the plot function be made in the region named "core", with the assumption that "core" is one of the regions adjoining the "inner_boundary" path.

Sign of Vector Components

In many cases, boundary-path elevations present normal or tangential components of vectors. For these applications, the sense of the

direction is the same as the sense of the NATURAL boundary condition:

The positive normal is outward from the evaluation region.

The positive tangent is counter-clockwise with respect to the evaluation region.

Plots of the normal components of vectors on extrusion surfaces in 3D follows the same rule:

The positive normal is outward from the evaluation region.

4.15.4. Reports

Any display specification can be followed by one or more of the following clauses to add report quantities to the plot page:

REPORT expression

Adds to the bottom of a display the text 'text of **expression**=value of **expression**', where **expression** is any valid expression, possibly including integrals. Multiple **REPORT** clauses may be used.

REPORT is especially useful for reporting boundary and area integrals and functions thereof.

REPORT expression AS 'string'

A labeled **REPORT** of the form '**string**=value of **expression**'.

REPORT('string')

REPORT 'string'

Inserts '**string**' into the **REPORT** sequence.

4.15.5. Window Tiling

When multiple **MONITORS** or **PLOTS** are listed, FlexPDE displays each one in a separate window and automatically adjusts the window sizes to tile all the windows on the screen. Individual windows cannot be independently expanded or iconized. Any plot window can be maximized by double-clicking, or by right-clicking to bring up a menu.

In steady-state and eigenvalue problems, **MONITORS** are displayed during solution, and are replaced by **PLOTS** on completion.

In time-dependent problems, **MONITORS**, **PLOTS** and **HISTORIES** are displayed at all times.

4.15.6. Monitors in Steady State Problems

In steady state problems the listed **MONITORS** are displayed after each regrid. In addition, after each Newton-Raphson iteration of a nonlinear problem or after each residual iteration of a linear problem, if sufficient time has elapsed since the last monitor display, an interim set of monitors will be displayed.

4.15.7. Monitors and Plots in Time Dependent Problems

In time dependent problems the display specifications must be preceded by a display-time declaration statement. The display-time declaration statement may be either of the form

FOR CYCLE = number

in which case the displays will be refreshed every number time steps, or

FOR T = timeset1 [timeset2 ...]

Where each **timeset** may be either a specific time or a group specified as

t1 BY deltat TO t2

In this case the displays will be refreshed at times specified by the **timeset** values.

Any number of plot commands can follow a display-time declaration, and the specification will apply to all of them. It is not necessary to give a display-time specification for each plot.

Multiple display time declaration statements can be used. When multiple display time statements are used each applies to all subsequent display commands until a new time declaration is encountered or the **MONITORS** or **PLOTS** section ends.

Examples:

See the problem "Samples | Time_Dependent | Heatflow | Float_Zone.pde", or any of the other problems in the "Samples|Time_Dependent" folder.

4.15.8. Hardcopy

A right-click on any plot window, whether tiled or maximized, will bring up a menu from which the plot may be printed or exported (or rotated, if this is meaningful for the plot).

Text listings of plotted values can be written to disk by the plot modifier **EXPORT** (aka **PRINT**) in the descriptor.

4.15.9. Graphics Export

Bitmaps

A right-click in any displayed plot window brings up a menu, one item of which is "Export". Clicking this item brings up a dialog for exporting bitmap forms of the displayed plot. Current options are BMP, PNG, PPM and XPM. See the "Getting Started" section for more information.

All these formats can also be selected automatically as graphic display modifiers.

Retained Graphics

All displays in the **PLOTS** section are written in compressed form to a disk file with the extension ".PG5".

These files may be redisplayed at a later time by use of the "View" menu item in the "File" menu. On some systems, this may be accomplished simply by double-clicking the ".PG5" file in the system file manager.

See the "Getting Started" section for more information.

Screen Grabs

Any display may also be pasted into other windows programs by using a screen capture facility such as that provided with PaintShopPro by JASC (www.jasc.com).

Export Files

The plot types CDF, TABLE, TECPLOT and VTK can be used to export data to other applications for external processing. TRANSFER can be used to transfer data to another FlexPDE run for postprocessing.

See Graphics Display and Data Export or Technical Notes for more information.

Examples

See the following sample problems for examples of exporting plot data:

Samples\Misc\Printest.pde

Samples\Misc\Import-Export\Export.pde

Samples\Misc\Import-Export\Export_Format.pde

Samples\Misc\Import-Export\Export_History.pde

4.15.10. Examples

See the sample problem Samples | Misc | Plottest.pde for examples of **PLOTS** and **MONITORS**.

See the sample problem Samples | Misc | Printest.pde for examples of exporting plot data.

See the sample problem Samples | Misc | Import-Export | Export.pde for examples of exports without display.

4.16. Histories

The **HISTORIES** section, which is optional, specifies values for which a time history is desired. While multiple **HISTORY** statements can be listed they must all be of the form:

```
HISTORY ( arg1 [ ,arg2,...] )  
HISTORY ( arg1 [ ,arg2,...] ) AT (X1,Y1) [ (X2,Y2)...]
```

The coordinates specify locations in the problem at which the history is to be recorded. If no coordinate is given, the **arg** must evaluate to a scalar.

The modifiers and reports available to **PLOTS** and **MONITORS** may also be applied to **HISTORY** statements.

The display of **HISTORIES** is controlled by the **AUTOHIST** select switch, which defaults to **ON**. With the default setting all **HISTORIES** are automatically refreshed and displayed with the update of any **MONITORS** or **PLOTS**.

If desired, **HISTORY** statements can be included directly in the **MONITORS** section or **PLOTS** section.

Histories in Staged Problems

HISTORY statements may be used in **STAGED** problems as well as in time-dependent problems.

In this case, the default abscissa will be stage number. You can select a different value for the abscissa quantity by appending the clause

VERSUS expression

In this case, the values of the given expression in the various stages will be used as the plot axis.

4.17. End

All problem descriptors must have an **END** section.

With the exception of a numeric enabling key used in special demonstration files prepared by PDE Solutions Inc., anything appearing after the reserved word end is ignored by FlexPDE and treated as a comment.

Problem notes can be conveniently placed after the reserved word **END**.

5. Batch Processing

A special form of descriptor is used to specify a group of problems to be run in batch mode.

A single "section" introduced by the word **BATCH** identifies a descriptor as a batch control file. Following this header, a sequence of names appears, each name enclosed in quote marks. Commas may optionally be used to separate the names. Any number of names may appear on each line of the descriptor. Each name is the name of a problem descriptor to be run. Names may include directory paths, which are assumed to originate in the directory containing the batch descriptor. The ".pde" extension is not required, and will be assumed if omitted. The list should be closed with an **END** statement.

For example,

```
BATCH
{ use the correct separators for your operating system }

"misc\table", "steady_state\heat_flow\slider"
"steady_state\stress\3d_bimetal"

END
```

The entire problem list is examined immediately, and any syntax errors in the names are reported. All files named in the list are located, and missing files are reported before any processing begins.

Each problem named in the list is run to completion in sequence. As the problems run, status information is written to a log file in the directory containing the batch descriptor. This file has the same name as the batch descriptor, with the extension '.log', and all problems in the list are summarized in this single file. Graphical output from each problem is written as usual to a unique .PG4 file in the directory with the specific descriptor. After the run is completed, this graphic output may be reviewed by restarting FlexPDE and using the VIEW menu item.

Simple names may be listed without the quotes, but in this case embedded spaces, path separators, reserved words and numeric initials will all cause error diagnostics.

6. Running FlexPDE from the Command Line

When FlexPDE is run from a command line or as a subtask from another application, there are some command-line switches that can be used to control its behavior:

- R Run the file which is named on the command line.
Do not enter edit mode.
- V View the file which is named on the command line.
Do not enter edit mode.
- X Exit FlexPDE when the problem completes.
- M Run in "minimized" mode (reduced to an icon).
- Q Run "quietly". Combines -R -X -M.
- S Run "silently". -Q with all error reports suppressed.

Index

"Include" Files.....	6	3D.....	86
.PDE.....	4	ANTIPERIODIC.....	88
.PGX.....	109	PERIODIC.....	88
3D coordinates.....	47	Point Load.....	84
3D PLOTS.....	94	Point Value.....	84
ABS function.....	15	syntax.....	84
Accuracy threshold.....	49	Boundary Paths and Path	
ALE.....	68	Names.....	72
alias.....	47	cartesian.....	47
ALIAS.....	44	cartesian3.....	47
Analytic Functions.....	15	Case Sensitivity.....	6
ANTIPERIODIC Boundary		CDF output.....	94
Conditions.....	88	CDFGRID.....	44
ARC.....	72	CENTER.....	72
ARCCOS function.....	15	CHANGELIM.....	39
ARCSIN function.....	15	Command-line Switches.....	114
ARCTAN function.....	15	Commas.....	12
AREA_INTEGRAL.....	29	Comments.....	10
AREA_INTEGRATE.....	97	components	
Arithmetic Operators.....	23	vector.....	25
ARRAY definitions.....	53	conditional expressions.....	33
AS 'string'.....	97	Constants.....	14
ASPECT.....	38	Constraints.....	69
AT.....	111	CONTACT Boundary Conditions	
ATAN2 function.....	15	82, 87
AUTOHIST.....	44, 111	CONTOUR plot.....	94
AUTOSTAGE.....	39	CONTOURGRID.....	44
batch runs.....	113	CONTOURS.....	44
Bessel Function.....	15	coordinate.....	47
BESSJ function.....	15	coordinates.....	47
BESSY function.....	15	COS function.....	15
Bevels.....	82	COSH function.....	15
BINTEGRAL.....	28	CROSS product.....	25
Bitmaps.....	109	CUBIC.....	39
BLACK.....	44, 97	CURL Operator.....	26
BMP.....	97, 109	CURVEGRID.....	38
BOUNDARIES.....	71	CYCLE plot interval.....	108
Boundary Conditions.....	82	cylindrical.....	47
1D.....	85	Decimal Numbers.....	14

DEFINTIONS.....	52	expressions.....	33
DEL2 - Laplacian Operator.....	26	EXTRUSION Section.....	70
dependent variables.....	49	FEATUREPLOT.....	44
Derivative operators.....	26	Features.....	80
Derivatives		File extension.....	4
high order.....	66	File name.....	4
Descriptor format.....	5	FILE 'string'.....	97
Differential operators.....	66	Fillets.....	82
Differential Operators.....	26	FINDERBINS.....	44
differentiation		FINISH.....	72
suppressing.....	63	FIRSTPARTS.....	39
Dirichlet Boundary Conditions.....	82	FIT Function.....	18
Display Modifiers.....	97	FIXDT.....	39
Display Specifications.....	94	FIXED POINT.....	90
DIV - Divergence Operator.....	26	FIXED RANGE.....	97
DOTproduct.....	25	Flux Boundary Conditions.....	82
eigenvalue.....	32	FONT.....	44
Eigenvalue.....	67	Format.....	5
ELEVATION plot.....	94	FORMAT 'string'.....	97
ELEVATIONGRID.....	44	FRONT.....	90
EMF.....	97	Function definition.....	54
Empty Layers in 3D.....	79	Functions	
END.....	111	non-analytic.....	15
ENDREPEAT.....	34	fuzzy IF.....	22
Engineering Notation.....	14	GAMMA function.....	15
EPS.....	97	GLOBAL VARIABLES.....	51
Equations.....	66	GLOBALMAX function.....	15
Equations and Variables.....	67	GLOBALMAX_X function.....	15
EQUATIONS section.....	66	GLOBALMAX_Y function.....	15
ERF function.....	15	GLOBALMAX_Z function.....	15
ERFC function.....	15	GLOBALMIN function.....	15
ERRLIM.....	39	GLOBALMIN_X function.....	15
Error Function.....	15	GLOBALMIN_Y function.....	15
Eulerian.....	68	GLOBALMIN_Z function.....	15
EVAL function.....	23	GRAD - Gradient Operator.....	26
examples		Graphic Display modifiers.....	97
graphics.....	110	Graphics Examples.....	110
Excludes.....	80	graphics export.....	109
EXP function.....	15	GRAY.....	44, 97
Exponential Integral - EXPINT		grid control features.....	80
.....	15	GRID plot.....	94
EXPORT.....	97	GRIDARC.....	38
Exporting graphics.....	109	GRIDLIMIT.....	38

HALT	92
Hardcopy	109
HARDMONITOR	44
histories	111
Histories in Staged Problems	111
history	111
HYSTERESIS	39
ICCG	39
include files	6
INITGRIDLIMIT	38
Initial values	65
input	3
INTEGRAL	29, 30
Integral Constraints	69
Integral Operators	28
Integrals	
area	29
line	28
surface	30, 31
time	28
volume	29, 30
INTEGRATE	97
INTSTRING	17
ITERATE	39
JUMP Boundary Conditions	87
Lagrange	68
LAMBDA	32
LAYER	
Extrusion	70
LIMITED REGIONS	78
LINE	72
LINE_INTEGRAL	28
LINE_INTEGRATE	97
linear basis	51
LINUPDATE	39
Literal Strings	13
LN function	15
LOAD Boundary Conditions	82
LOG	97
LOG10 function	15
logarithm	15
LOGLIMIT	44

LUMP function	19
MAGNITUDE of vector	25
Material parameters	75
MAX function	15
MERGE	44, 97
MESH	97
Mesh control features	80
Mesh Control Parameters	64
Mesh Generation Controls	38
Mesh Refinement - FRONT	90
Mesh Refinement - RESOLVE	91
MESH_DENSITY	64
MESH_SPACING	64
Metafile	97
MIN function	15
MOD function	15
Modal Analysis	67
MODES	39
MONITORS	93
Monitors in Steady State Problems	108
MOVE	50
Moving Meshes	50, 68
NATURAL Boundary Conditions	82
NEWTON	39
NGRID	38
NOBC Boundary Conditions	82
NODELIMIT	38
NOMINMAX	44
Non-Analytic Functions	15
NONLINEAR	39
NONSYSMMETRIC	39
NORM	97
NORMAL component	25
NOTAGS	44
NOTIFY_DONE	39
NOTIPS	44, 97
NRMATRIX	39
NRMINSTEP	39
NRSLOPE	39
NRUPDATE	39

NRUPFIT	39
Numbering and Naming	
Regions	81
Numeric Range	14
Numeric Reports	107
Numerical Constants	14
ON "name"	104
ON <equation>	104
ON equation	94
ON LAYER "name"	104
ON LAYER number	104
ON REGION "name"	104
ON REGION number	104
ON SURFACE	94
ON SURFACE "name"	104
ON SURFACE number	104
Operators	
Arithmetic	23
Differential	26
Integral	28
Relational	24
String	24
Vector	25
ORDER	39
Ordering Regions	81
OVERSHOOT	39
PAINTED	44, 97
PAINTGRID	44
PAINTMATERIALS	44
PAINTREGIONS	44
Parameterized Definitions	54
Parameters	
redefining	52
regional	75
PASSIVE Modifier	63
Paths and Path Names	72
PERIODIC Boundary Conditions	88
PI32	
Plot Domain	104
Plot Modifiers	97
plot POINTS	97
plot range	97

Plot time selection	108
PLOTINTEGRATE	44
PLOTS	93
PNG	97, 109
POINT	72
POINT definitions	56
POINT LOAD Boundary Conditions	84
POINT VALUE Boundary Conditions	84
PostScript	97
PPM	97, 109
PRECONDITION	39
PREFER_SPEED	39
PREFER_STABILITY	39
Preparing a Descriptor File	3
PRINT	97
PRINTMERGE	44
problem description	3
Problem Descriptor Structure	4
pulse function	17
QUADRATIC	39
quoted strings	13
R 32	
radius	32
RADIUS	72
ramp function	17
RAMP function	19
RANDOM function	15
Regional Parameter Values	75
Regions	74
excluded	80
Numbering and naming	81
Ordering	81
overlying	74
REGRID	38
REINITIALIZE	39
Relational Operators	24
REPEAT	34
Repeated Text	34
REPORT	107
Reporting numbers	107

Reserved Words and Symbols	11
Resolve	91
SAVE function	20
SCALAR VARIABLES	51
script	3
section names	4
SELECT	37
Semicolons	12
SENSITIVITY	39
Separators	12
SIGN function	15
SIMPLEX	51
SIN function	15
SINH function	15
SINTEGRAL	30, 31
SMOOTHINIT	38
Spaces	12
SPLINE	72
SPLINETABLE function	58
SQRT function	15
STAGE	32
STAGED Definitions	55
STAGED Geometry	55
STAGEGRID	38
STAGES	39
STAGES Selector	55
START	72
step function	17
String Functions	17
String Operators	24
Strings	13
SUBSPACE	39
SUM function	21
SUMMARY	94
SUMMARY plot	94
SURF_INTEGRAL	30, 31
SURFACE	
Extrusion	70
SURFACE plot	94
SURFACEGRID	44
SWAGE function	22
Switches	

Command-line	114
TABLE	94
TABLE File format	59
TABLE Input function	57
TABLE output	94
Tabledef Input	58
TAN function	15
TANGENTIAL component	25
TANH function	15
TECPLOT	94
TECPLOT putput	94
TERRLIM	39
Text Strings	13
TEXTSIZE	44
THERMAL_COLORS	44
THETA	32
THRESHOLD	49
Time	92
time range	92
TIME_INTEGRAL	28
TINTEGRAL	28
Title	37
TNORM	39
TRANSFER	94
TRANSFER File format	61
TRANSFER input	60
TRANSFER output	94
TRANSFERMESH	94
TRANSFERMESH input	61
trigonometric functions	15
Unit Functions	17
UPFACTOR	39
UPULSE function	17
UPWIND	39
URAMP function	17
USTEP function	17
VAL function	23
VALUE Boundary Conditions	82
Values	
Initial	65
VANDENBERG	39
variables	49
Variables and Equations	67

VECTOR	94
VECTOR composition	25
Vector Operators	25
VECTOR plot	94
VECTORGRID	44
VELOCITY Boundary	
Conditions	82
VERSUS	111
VIEWPOINT	44
VOID	79
VOL_INTEGRAL	29, 30
VOL_INTEGRATE	97

VTK	94
VTKLIN	94
white space	12
Window Tiling	108
XCOMP function	25
xcylinder	47
XERRLIM	39
XPM	97, 109
YCOMP function	25
ycylinder	47
ZCOMP function	25
ZOOM	97