



Streams

Version 3.03

User's Guide:
Tablets

by Dr. Roger Nokes

Streams

Version 3.03

User's Guide: Tablets



© Dr Roger Nokes
February 2021

Department of Civil and Natural Resources Engineering
University of Canterbury
Christchurch, NZ
roger.nokes@canterbury.ac.nz

Design and layout: Melody Callahan

I	Introduction.....	1
1.1	Variables and Data types	1
1.2	Commands	3
1.3	Summary.....	4
2	Command Expressions.....	5
2.1	Description.....	5
2.2	Named variables	5
2.3	Constants	6
2.4	Ranges and sampled ranges	6
2.5	Unary and binary operators	7
2.6	Mathematical functions.....	8
2.7	Independent variables.....	9
2.8	Vector field components.....	9
2.9	Brackets.....	10
2.10	Specialised functions.....	11
2.11	Custom functions.....	12
2.12	Summary.....	13
3	Field Functions.....	15
3.1	Description.....	15
3.2	Function Library.....	17
3.2.1	Autocorrelate1DScalarField(a,b,c,d)	17
3.2.2	Autocorrelate2DScalarField (a,b,c,d,e)	18
3.2.3	AverageField(a,b,c,d)	19
3.2.4	AverageFieldInBlocks(a,b,c)	19
3.2.5	BandPassFilter(a,b,c,d).....	20
3.2.6	CreateScalarField(a,b)	21
3.2.7	CreateVectorField(a,b,c,d)	21
3.2.8	Crosscorrelate1DScalarFields (a,b,c,d,e)	22
3.2.9	Crosscorrelate2DScalarFields (a,b,c,d,e,f).....	22

3.2.10	Differentiate(a,b,c)	23
3.2.11	EvaluateFieldOnSurface(a,b,c)	24
3.2.12	ExtrapolateScalarField(a,b,c,d,e)	24
3.2.13	ExtrapolateVectorField(a,b,c,d,e)	25
3.2.14	FillScalarField(a,b,c,d,e)	25
3.2.15	FindContour(a,b,c,d,e,f)	26
3.2.16	FindMax(a,b)	27
3.2.17	FindMaxLocation(a,b)	27
3.2.18	FindMin(a,b)	28
3.2.19	FindMinLocation(a,b)	28
3.2.20	GradientScalarField(a,b)	28
3.2.21	HighPassFilter(a,b,c)	29
3.2.22	IntegrateDef(a,b,c,d)	30
3.2.23	IntegrateIndef(a,b,c)	31
3.2.24	LimitScalarField(a,b,c)	31
3.2.25	LowPassFilter(a,b,c)	31
3.2.26	LTScalarField(a,b,c,d,e)	32
3.2.27	LTVectorField(a,b,c,d,e)	33
3.2.28	MergeFields(a,b)	34
3.2.29	MovingAverageField(a,b,c,d,e)	34
3.2.30	Regrid(a,b)	35
3.2.31	StdDevField(a,b,c,d)	35
3.2.32	StretchField(a,b,c)	36
3.2.33	SumField (a,b,c,d)	37
3.2.34	SwapScalarFieldDimension(a,b,c)	37
3.2.35	SwapScalarField(a,b)	37
3.2.36	SwapVectorField(a,b)	38
3.2.37	TranslateField(a,b,c)	38
3.3	Summary	39
4	Grid Functions	41
4.1	Description	41
4.2	FunctionLibrary	41
4.2.1	CreateGrid (a,b,c,d,e,f,g,h,i,j,k,l,m,o,p)	41
4.2.2	ExtractGrid(a,b)	42
4.2.3	LTGrid(a,b,c)	42
4.2.4	StretchGrid(a,b,c)	42
4.3	Summary	43

5	Matrix Functions.....	45
5.1	Description.....	45
5.2	Function Library.....	45
5.2.1	AverageMatrix(a,b,c).....	45
5.2.2	ExtractMatrix(a).....	46
5.2.3	LTMatrix(a,b,c).....	46
5.2.4	StdDevMatrix(a,b,c).....	46
5.2.5	StretchMatrix(a,b,c).....	47
5.3	Summary.....	47
6	Miscellaneous Functions.....	49
6.1	Description.....	49
6.2	Function Library.....	49
6.2.1	DimLabel(a,b).....	49
6.2.2	DimOrigin(a,b).....	50
6.2.3	DimSize(a,b).....	50
6.2.4	DimStep(a,b).....	51
6.2.5	EvaluateLeastSquaresSurfaceFit (a,b,c,d).....	51
6.2.6	LeastSquaresSurfaceFit(a,b,c,d).....	51
6.2.7	LeastSquaresSurfaceFitFunction (a,b,c,d).....	52
6.2.8	NumDims(a).....	53
6.3	Summary.....	53
7	Tablets.....	55
7.1	Description.....	55
7.2	Views.....	55
7.2.1	Details view.....	55
7.2.2	Analysis view.....	56
7.3	Popup menu.....	61
8	Fields.....	63
8.1	Description.....	63
8.2	Views.....	63
8.2.1	Details view.....	63

- 8.3 Popup menu..... 64
- 9 Grids..... 65**
 - 9.1 Description..... 65
 - 9.2 Views..... 65
 - 9.2.1 Details view..... 65
 - 9.3 Popup menu..... 66
- 10 Matrices..... 67**
 - 10.1 Description..... 67
 - 10.2 Views..... 67
 - 10.2.1 Details view..... 67
 - 10.3 Popup menu..... 68
- 11 Scalars..... 69**
 - 11.1 Description..... 69
 - 11.2 Views..... 69
 - 11.2.1 Details view..... 69
 - 11.3 Popup menu..... 70
- 12 Strings..... 71**
 - 12.1 Description..... 71
 - 12.2 Views..... 71
 - 12.2.1 Details view..... 71
 - 12.3 Popup menu..... 71
- 13 Custom functions 73**
 - 13.1 Description..... 73
 - 13.2 Views..... 73
 - 13.2.1 Details view..... 73
 - 13.3 Popup menu..... 74

I Introduction

The primary outputs of the *Streams* system are the fields generated by Eulerian and Lagrangian field time series calculators. These scalar and vector fields correspond to the physical quantities of interest to the experimenter such as velocity, density and strain. *Streams* provides a complete set of field data types (see the *Streams 3.03 – User's Guide: Field Objects* manual) and an extensive set of views for these fields that deliver powerful tools for field visualization. However, the raw field data are frequently not in a final form that is most suitable for the experimenter. Post-processing of some sort is almost always required and often the user might extract the field data from *Streams* and import that data into an appropriate analysis package such as MATLAB®.

Tablets provide a modest analysis capability so that a range of analyses can be undertaken within *Streams* itself. This has a number of advantages. Firstly, it avoids the messy export/import process to another application. Secondly, it means some analysis tasks can be performed very rapidly. And finally, it enables the user to take advantage of the *Streams* visualization capability.

A tablet can also be viewed as a data repository. Related fields can be gathered together and stored in a single tablet, allowing the task of exchanging data between different users to be more efficient.

This manual aims to provide detailed coverage of tablets and their associated objects. Chapter 2 provides a description of the anatomy of tablet commands. Chapters 3-6 introduce the library of functions that can be utilized by the user in manipulating their data. Chapter 7 provides a more formal introduction to tablets and their views, while chapters 8-13 cover the data types described below in section 1.1.

1.1 Variables and Data types

In tablets, as in most programming, or scripting, languages, the concepts of variables and data types play an important role. **Variables** are named quantities that store data of some sort within a tablet. Each variable's name must be unique. A variable can be manipulated in tablet commands through the explicit use of the variable's name.

All variables have an associated **data type** that describes the sort of data that the variable stores. A variable's data type places constraints on how that variable can be manipulated, where it can appear in function parameter lists, and how it can be combined with other variables in various expressions.

The name and data type of a variable are immutable.

Tablets support six different data types. These are described below and details of their popup menus and views are provided in later chapters.

Fields:

The most common objects manipulated within a tablet are fields – both scalar and vector fields are supported. However fields within tablets have one important difference to those elsewhere in *Streams*– they must have a unique name allocated to them to enable them to be referenced within tablet commands. Fields are immutable - once created they cannot be changed. Of course, new fields can be created through the use of tablet commands, but each field, once created, remains unchanged from its original form.

Grids:

A grid represents a 1D, 2D, 3D or 4D regular rectangular mesh upon which a field could be defined. For each dimension the grid stores the grid size (number of mesh points), the origin, the step between grid points, and an associated label. Grids are not immutable in that the user can edit the origin, step and label. However, the dimension sizes cannot be changed.

Matrices:

A matrix is a 0D, 1D, 2D, 3D or 4D array of numbers. It could be most easily created by extracting the field data from a field but without references to the grid upon which the field is based. However matrices can be created in other ways, as described below, and they play a role as parameters in a number of tablet functions. One could think of a field as a combination of a matrix (the field data) and a grid (the mesh upon which the data is defined). Matrices are immutable.

Scalars:

Simple scalar quantities – real, integer and Boolean values – can be created and manipulated. Scalars can be changed by assigning new values to them.

Strings:

Strings store textual information, and are useful in a number of tablet functions.

Custom functions:

The user is permitted to create their own custom functions. These functions generate scalar or vector fields and are defined by appropriate mathematical formulae. The formulae can include the variables x , y , z and t representing the four independent variables for multi-dimensional fields. For example, to create a custom function that produces a two-dimensional scalar field the user would use x and y as the two independent variables in their formula. If the field was to be three-dimensional the independent variables would be x , y and z .

1.2 Commands

Tablets provide a command-line interface, similar to MATLAB[®], for manipulating fields, matrices, grids, scalars, strings and custom functions. These commands are created and executed in the Analysis view described in section 7.3.

Commands have the following format:

$$\text{new_entity} = \text{expression} \quad (1.1)$$

White space, such as tabs, spaces, new line characters etc., are removed before the command is processed.

Each command is checked for errors before executing. The command is parsed in three separate steps, and each of these steps can generate errors. The steps are as follows:

1. The command parser first separates the command into tokens and checks that these tokens are valid. For example, names of matrices,

fields and so on must be valid, and brackets must appear in appropriate matching pairs.

2. In the second step the parser checks that the command is correct in both type and dimension. For example, a function that expects its first parameter to be a scalar field will generate an error if it is passed a matrix, grid, or vector field. A binary operator combining two fields of different dimensions, or defined on different grids, would generate an error, and so on.
3. Finally, the *new_entity* in equation 1.1 is checked to see whether the specified name is already allocated to an entity in the tablet. If so the user is asked whether they wish to overwrite the original entity (this feature can be suppressed – see section 7.3). If the user chooses not to do so the command is not executed. A check is also performed to ensure that the *new_entity* does not correspond to a reserved word, such as the name of a function, a predefined constant such as pi or something similar, or includes a reserved character such as the mathematical operators, brackets and so on.

When errors are generated by the command parser a dialog box is displayed listing the errors. Once all errors have been corrected the command can execute. It is worth noting that often a single mistake in a command will generate multiple error messages.

1.3 Summary

This chapter has provided a brief introduction to tablets and the data types that they support. In later chapters detailed information will be provided about the operators and functions that can be included in a tablet command.

2 Command Expressions

2.1 Description

This chapter is dedicated to a detailed exposition of the structure of tablet commands. In particular the focus is on the format of the expressions that appear on the right hand side of the assignment in equation 1.1. These expressions can include a range of entities that includes the following:

- *Named variables – fields, grids, matrices, scalars and strings.*
- *Constants*
- *Ranges*
- *Unary and binary operators*
- *Mathematical functions*
- *Independent variables*
- *Vector field component variables*
- *Brackets*
- *Custom functions*
- *Field functions*
- *Grid functions*
- *Matrix functions*
- *Miscellaneous functions*

These are discussed in the following sections and chapters.

2.2 Named variables

All named variables, fields, grids, matrices, scalars, strings, custom functions, are referred to by their unique designated names. These names are case-sensitive. These names cannot be the same as any of the reserved strings used by tablets but they can include any character that can be typed (not including white space such as spaces and tabs, nor mathematical operators, brackets and other symbols that have specific meanings within a tablet). These reserved strings include all function names as well as the names of the independent variables (see below) and velocity field component variables (see below).

2.3 Constants

Scalar constants, real, integer, Boolean and strings are specified in the normal way. Exponential notation can be used to define real variables – e.g. 2.3E7 or -1.6e-2. The only two Boolean constants are *true* and *false*. Strings are specified by enclosing the string in double quotes – e.g. “x”. The constant *pi*, or *PI*, is also predefined and can be used in mathematical expressions.

Note that wherever a real number is expected in an expression an integer value can be used instead. The opposite is not true.

Examples:

The following expressions are valid:

A = 6	A is an integer scalar with value 6
A = true	A is a Boolean scalar with a value of true
A = 3.4e-2	A is a real scalar with a value of 0.034
A = "Hello"	A is a string containing the text "Hello".

2.4 Ranges and sampled ranges

In some circumstances a subset of a field or matrix is required and this subset is designated by a range variable, or sampled range variable.

A range variable specifies the start index and the end index for one dimension of a field or matrix, separated by a colon. The resulting subset will include all field, or matrix, elements in that dimension including those at the start and end indices. Note that all indices start at 0.

A sampled range variable specifies the start index and the increment of that index for one dimension of a field or matrix, separated by a hash. The resulting subset will include all field, or matrix, elements in that dimension starting at the start index and then skipping every (increment – 1) elements. Note that the increment must be positive.

Example:

0:50	all elements between, and including, 0 and 50.
10#3	extracts elements 10, 13, 16, 19 etc.

2.5 Unary and binary operators

Tablets support two unary operators (+ and -) and five binary operators (+, -, *, /, ^) and these all carry their normal meaning. However, not all operators can be used with all data types.

Fields, matrices and scalars (excluding Boolean) can be used with these operators while grids, strings, and Boolean scalars cannot. In practice functions that generate fields, matrices and real and integer values, including custom functions, can also be used with these operands.

The power operator (^) must have an integer or real value as its right hand operand. The left hand operand can be a field or matrix and the result is the same type as the left hand operand where each element in the matrix or field has been raised to the specified power. For a vector field each component of the vector is raised to the specified power.

Fields and matrices used with other binary operators must possess the same number of dimensions and the same dimension sizes. In the case of fields they must also have matching grid origins and steps – in other words their grids must match.

Binary operators can include fields of different types, or fields and matrices but these combinations must make mathematical sense. Thus it is valid to multiply a scalar field and a vector field (provided their grids match) but scalar and vector fields cannot be added or subtracted together, and vector fields cannot be multiplied together.

Examples:

Suppose U is a scalar field of size (100x50), V is a 2D vector field of the same size (and both have the same grid) and M is a matrix of size 100x50. The following expressions are valid:

$A = U * V$	A is a 100x50 2D vector field
$A = M * V$	A is a 100x50 vector field
$A = M * U$	A is a 100x50 scalar field
$A = 2 * U$	A is a 100x50 scalar field
$A = V / 5$	A is a 100x50 2D vector field
$A = U + 2$	A is a 100x50 scalar field
$A = V ^ 2$	A is a 100x50 2D vector field.

The following expressions are not valid:

$A = U+V$
 $A = V*V$
 $A = V-2$
 $A = V^M$
 $A = 2^M$

2.6 Mathematical functions

Tablets provide a range of standard mathematical functions. These functions take a single argument that may be a real scalar, a matrix, or a scalar field. In each case the function is computed for each element in the argument. The following mathematical functions are available:

abs(), sin(), cos(), tan(), asin(), acos(), atan(), exp(), log(), sinh(), cosh(), tanh(), sqrt(), cbrt(), h(), random().

While most of these have their usual mathematical meaning a number of them may benefit from a brief explanation.

cbrt(x)	cube root of x
h(x)	the Heaviside function of x (1 if the argument is greater than 0 and 0 otherwise).
random(x)	a random number between 0 and x.

Examples:

Suppose U is a scalar field of size (100x50), V is a 2D vector field of the same size (and both have the same grid) and M is a matrix of size 100x50. The following expressions are valid:

$A = \sin(5)$	A is a real scalar
$A = h(M)$	A is a 100x50 matrix whose elements are 1 if the equivalent M element is greater than 0, or 0 otherwise.
$A = \text{sqrt}(U)$	A is a 100x50 scalar field whose elements are the square roots of the equivalent elements in U.

The following expressions are not valid:

$A = \text{abs}(V)$

2.7 Independent variables

Scalar and vector fields may have up to four independent variables. The values of these variables can be accessed through the variables x_1 , x_2 , x_3 and x_4 . Independent variables are treated in the same way as mathematical functions with a single parameter – the field or grid for which the values of the independent variable is sought.

Examples:

Suppose that U is a scalar field of dimension $20 \times 100 \times 50$. Then $x_2(U)$ will generate a scalar field of dimension $20 \times 100 \times 50$ where the field values correspond to the values of x_2 at each of U 's grid points. Thus the following would be valid expressions:

$$A = x_2(U) * x_3(U)$$

$$A = x_2(U) * U$$

$$A = x_3(U) + U$$

The following would be invalid:

$A = x_4(U) * U$	as U does not have a 4 th dimension
$A = x_1 * U$	x_1 needs a parameter.

2.8 Vector field components

The three components of a vector field are denoted by the variables u_1 , u_2 , and u_3 . These variables behave in much the same way as the independent variables. They are treated as functions that take a single argument, the vector field from which the component is to be extracted, and return a scalar field of the corresponding size.

Examples:

Suppose that UV is a 2D vector field of dimension $100 \times 50 \times 1000$. Then the following would be valid expressions:

$A = u_2(UV)$	is a scalar field of size $100 \times 50 \times 1000$
---------------	---

$$A = 6 * u_1(UV)$$

The following would be invalid:

$A = u_3(UV)$	as UV doesn't have a 3 rd vector component
---------------	---

2.9 Brackets

Tablets support three types of brackets, each with its own particular meaning. These are described below:

Round brackets: ()

Round brackets are used in two contexts. Firstly, in the usual manner to dictate the order of operations. Thus

$$A = 2*(U + V)$$

They are also used to encapsulate function call parameter lists. Thus

$$A = \text{sqrt}(U)$$

Square brackets: []

Square brackets are used to create a subset of a field or matrix. They behave similarly to a function call where the number of parameters, separated by commas, must correspond to the number of dimensions of the field or matrix. Each of the parameters can be one of three types:

integer	specifies the specific index for that dimension
range	specifies a range or sampled range for that dimension
null	leaves the dimension unchanged. A null parameter is specified by including no parameter.

Each integer parameter in the parameter list reduces the dimension of the resulting field or matrix by 1.

Examples:

Suppose that U is a 3D scalar field of dimension 100x200x50. Then the following would be valid expressions:

$A = U[,5,10]$	1D scalar field of size 100
$A = U[0:20,,0:10]$	3D scalar field of size 21x200x11
$A = U[1\#2,50,20:40]$	2D scalar field of size 50,21

The following would be invalid:

$A = U[,5,10,]$	U does not have a 4 th dimension
$A = U[0:100,,0:10]$	the x1 range is invalid. It cannot be larger than 0:99 because of the zero based indices.
$A = U[5\#-2,,]$	The increment in the sampled range must be positive.

Curly brackets: { }

Curly brackets are used to initialize a 1D matrix. Matrix elements are listed, separated by commas, and the size of the resulting matrix equals the number of elements listed.

Example:

$M = \{1,2,3,4,5,6\}$ produces a 1D matrix of size 6.

2.10 Specialised functions

The power of tablets lies in the specialized functions that they provide. The majority of these specialised functions are associated with fields, grids, or matrices and are categorized as such. Chapter 3 is dedicated to field functions, chapter 4, considers grid functions, chapter 5 matrix functions, and, finally, chapter 6 introduces miscellaneous functions not specifically linked to fields, grids or matrices.

Specialised functions are called in the same way as the standard mathematical functions. However, it is important to note that, in general, these functions take more than a single parameter. The rules regarding the order and type of each function's parameter set is strictly enforced. The description of these functions, provided in subsequent chapters, includes a specification of the function's parameters as well as its output type.

Examples:

Suppose that U is a 2D scalar field of dimension 100x200. Then the following are examples of valid specialized function calls:

$A = \text{AverageField}(U,1)$	A is a 1D scalar field of size 200 obtained by averaging U over its first dimension.
$A = \text{CreateGrid}(100,0,0.1,"x")$	A is a 1D grid comprising 100

Grid points. The grid starts at 0 with an increment of 0.1 between grid points. The grid is labelled "x".

2.1.1 Custom functions

The final group of functions is that comprising the functions defined by the user themselves. These functions are specifically designed to compute a user-defined function, scalar or vector, at a set of points. The function is expressed as a mathematical formula (or more than one in the case of a vector function) that can include some, or all, of the variables, x , y , z and t – representing the coordinates of a point in 4D space. In practice x , y and z tend to correspond to spatial coordinates and t corresponds to time, but this need not be the case.

Custom vector functions take a single argument – a grid. Calling the function will generate a vector field based on the specified grid, where each of the vector components is calculated at each grid point using the formulae provided.

In the case of a scalar function two different parameter sets are possible. The first, as for vector functions, comprises a single grid parameter. The resulting scalar field is computed by evaluating the user specified function at each grid point. The second comprises a set of 1D matrices – each of the same size. The first matrix is assumed to contain a set of x coordinates, the second matrix a set of y coordinates and so on. The function produces a matrix whose elements are computed at the points specified by the matrices.

No check is done on the variables used within the custom function and the dimensions of the points on which the function is evaluated. For example, the user could provide a 2D grid for a scalar function that included all four variables, x , y , z and t . In this case the grid points would provide the x and y values for the function and z and t would both be set to 0. On the other hand the user could specify a 3D grid as the argument for a scalar function that only refers to the coordinates x and y . In this case the z coordinate of each grid point would be ignored by the function.

Example:

Suppose the user creates a custom scalar function, called *paraboloid*, and defined as follows

$$x^2 + y^2$$

To generate a scalar field, P, representing a paraboloid of revolution, defined on the spatial domain

$$x \in [-4, 4] \quad y \in [-4, 4]$$

the following tablet commands could be used

```
G = CreateGrid(401,-4,0.02,"x",401,-4,0.02,"y")
P = paraboloid(G)
```

2.12 Summary

This chapter has laid out the rules for expressing tablet commands. These rules are generally intuitive and, as much as is possible, mimic standard rules for mathematical expressions.

The real power of tablets lies in their repository of specialized functions that provide the user with consider power to manipulate scalar and vector fields, and to extract information from them. These functions are described in detail in the following chapters.

3 Field Functions

3.1 Description

Most of the power in tablets comes from the set of specialised functions that work with fields, grids and matrices. All of these functions behave essentially like the simple mathematical functions except that these specialised functions typically take a number of parameters, separated by commas. A valid function call requires the correct number of parameters and the correct type of parameters.

In many of these functions one of the parameters corresponds to a user-selected dimension of a field, grid or matrix. In all cases dimensions start at 1 and can vary up to a maximum of 4. This is in contrast to elements of a matrix or field where the indexing is zero-based.

In this chapter we describe the functions associated with fields. We give each function's name, a description of its purpose, a description of each of its parameters, and an explanation of its output value. Parameters that are optional are listed with the (opt) designation. Table 3.1 provides an overview of the functions available.

Table 3.1 The field based functions.

Function name	Description
Autocorrelate1DScalarField()	Computes a 1D autocorrelation of a scalar field.
Autocorrelate2DScalarField()	Computes a 2D autocorrelation of a scalar field.
AverageField()	Averages a field (scalar or vector) in one of its dimensions.
AverageFieldInBlocks()	Averages a field in a set of blocks in one of its dimension.
BandPassFilter()	Performs a band pass filter on a 1D scalar field.
CreateScalarField()	Creates a scalar field from a grid and a matrix or real number.
CreateVectorField()	Creates a vector field from a grid and a set of matrices or real numbers.
Crosscorrelate1DScalarFields()	Computes a 1D cross-correlation of two scalar fields.

Crosscorrelate2DScalarFields()	Computes a 2D cross-correlation of two scalar fields.
Differentiate()	Differentiates a field (scalar or vector) in one of its dimensions.
EvaluateFieldOnSurface()	Evaluates a field on a surface that has a dimension one less than the field itself.
ExtrapolateScalarField()	Extrapolates a scalar field to the boundaries of its grid.
ExtrapolateVectorField()	Extrapolates a vector field to the boundaries of its grid.
FillScalarField()	Fills a scalar field over a specified region in one of its dimensions.
FindContour()	Searches a scalar field for a particular value in one of its dimensions.
FindMax()	Finds the maximum value of a scalar field in one of its dimensions.
FindMaxLocation()	Finds the location of the maximum value of a scalar field in one of its dimensions.
FindMin()	Finds the minimum value of a scalar field in one of its dimensions.
FindMinLocation()	Finds the location of the minimum value of a scalar field in one of its dimensions.
HighPassFilter()	Performs a high pass filter on a 1D scalar field.
GradientScalarField()	Computes the gradient field of a scalar field.
IntegrateDef()	Computes a definite integration of a scalar or vector field over one of its dimensions.
IntegrateIndef()	Computes an indefinite integration of a scalar or vector field over one of its dimensions.
LimitScalarField()	Limits the range of values that can appear in a scalar field.
LowPassFilter()	Performs a low pass filter on a 1D scalar field.
LTScalarField()	Performs a linear transformation on a scalar field and its underlying grid.
LTVectorField()	Performs a linear transformation on a vector field and its underlying grid.
MergeFields()	Merges two fields (scalar or vector).
MovingAverageField()	Computes a box moving average of a

	scalar or vector field.
Regrid()	Recomputes a scalar or vector field on a new grid.
StdDevField()	Computes the standard deviation of the field data in one of its dimensions.
StretchField()	Increases the number of dimensions of a field (scalar or vector) by one.
SumField()	Sums a field (scalar or vector) over one of its dimensions.
SwapScalarFieldDimension()	Exchanges two of the dimensions of a scalar field.
SwapScalarField()	Inverts one of the dimensions of a scalar field.
SwapVectorField()	Inverts one of the dimensions of a vector field.
TranslateField()	Translates a scalar or vector field on its underlying grid in an incremental manner.

3.2 Function Library

3.2.1 AutocorrelateIDScalarField(a,b,c,d)

This function computes a one dimensional autocorrelation of a scalar field over one of its dimensions. An example is given in equation 3.1.

$$auto(x_1, \Delta x_2, x_3) = \overline{u(x_1, x_2, x_3, t) u(x_1, x_2 + \Delta x_2, x_3, t)} \quad (3.1)$$

where *auto* is the autocorrelation function and the overbar indicates averaging over time, *t*. One of the dimensions of the field is designated the correlation dimension – dimension 2 in the example – and another is the averaging dimension – dimension 4 in the example. The correlation dimension and averaging dimension could coincide, in which case the autocorrelation possesses the same number of dimensions as the original field. If these dimensions do not coincide the autocorrelation has one dimension less – as in the example.

The user has the choice to normalise the auto-correlation. To illustrate this calculation using the example above, the normalised autocorrelation would be

$$auto(x_1, \Delta x_2, x_3) = \frac{u(x_1, x_2, x_3, t) u(x_1, x_2 + \Delta x_2, x_3, t)}{\sqrt{u^2(x_1, x_2, x_3, t)} \sqrt{u^2(x_1, x_2 + \Delta x_2, x_3, t)}} \quad (3.2)$$

Parameters:

- a (field) – the field to be correlated
- b (integer) – an integer specifying the correlation dimension (1-4).
- c (integer) – an integer specifying the averaging dimension (1-4).
- d (boolean) – specifies whether the correlation should be normalised.

Returns:

(1D scalar field) – the autocorrelation as illustrated in equations 3.1 or 3.2.

3.2.2 Autocorrelate2DScalarField(a,b,c,d,e)

This function computes a two dimensional autocorrelation of a scalar field over two of its dimensions. An example is provided in equation (3.3).

$$auto(x_1, \Delta x_2, x_3, \Delta t) = \frac{u(x_1, x_2, x_3, t) u(x_1, x_2 + \Delta x_2, x_3, t + \Delta t)}{\sqrt{u^2(x_1, x_2, x_3, t)} \sqrt{u^2(x_1, x_2 + \Delta x_2, x_3, t + \Delta t)}} \quad (3.3)$$

As for the one dimensional correlation the autocorrelation can be normalised if desired.

Parameters:

- a (field) – the field to be correlated
- b (integer) – an integer specifying the first correlation dimension (1-4).
- c (integer) – an integer specifying the second correlation dimension (1-4).
- d (integer) – an integer specifying the averaging dimension (1-4).
- e (boolean) – specifies whether the correlation should be normalised.

Returns:

(1D scalar field) – the autocorrelation as illustrated in equation 3.3.

3.2.3 AverageField(a,b,c,d)

This function averages a field (vector or scalar) over one of its dimensions.

Parameters:

- a (field) – the field to be averaged
- b (integer) – an integer specifying the dimension (1-4) that is being averaged over.
- c(opt) (range or real) – if only c is specified it must be a range variable that specifies a subset of the dimension over which the average takes place.
- d(opt) (real) – if both c and d are provided they are real values that indicate the start and end of the averaging range. The actual grid points used in the calculation will lie within c and d if c and d do not correspond exactly to grid points.

Returns:

(field) the average field.

3.2.4 AverageFieldInBlocks(a,b,c)

This function averages a field (vector or scalar) over one of its dimensions, but it does so in blocks. A block is a user specified number of values. For example, consider a one-dimensional scalar field that has 1000 values and a block average is calculated for that field with a block size of 50. The resulting field will have 20 values each value obtained by averaging 50 consecutive values of the field. The resulting field will have a new origin corresponding to the centre of the first block, and its step will be 50 times that of the original field. This function can be useful for computing turbulent averages of a fluid flow that is quasi-steady.

Parameters:

- a (field) – the field to be averaged

- b (integer) – an integer specifying the dimension (1-4) that is being averaged over.
- c (integer) – the block size (the number of values over which the average will be taken).

Returns:

(field) – the average field.

3.2.5 **BandPassFilter(a,b,c,d)**

This function performs a band pass filter – comprising a low pass and high pass filter - on a 1D scalar field. The user specifies the type of filter to be applied along with parameters for the low pass and high pass filters. Three filters are available at this time:

Step filter: Filter number = 1. Removes all frequencies below/above the specified cutoff frequency.

Gaussian filter: Filter number = 2. Frequencies below/above the specified cutoff frequency are weighted with a Gaussian function with a specified standard deviation.

Butterworth filter: Filter number = 3. Frequencies below/above the specified cutoff frequency are weighted with the Butterworth function of a specified order.

Parameters:

- a (scalar field) – the scalar field being filtered.
- b (int) – the filter number.
- c (1D matrix) – a matrix containing the parameters for the low pass filter. For the step filter the matrix contains the cutoff frequency, for the Gaussian filter the cutoff frequency and the standard deviation for the weighting function, for the Butterworth filter the cutoff frequency and the order of the filter.
- d (1D matrix) – a matrix containing the parameters for the high pass filter. For the step filter the matrix contains the cutoff frequency, for the Gaussian filter the cutoff frequency and the standard deviation for the weighting function, for the Butterworth filter the cutoff frequency and the order of the filter.

Returns:

(scalar field) – the filtered scalar field.

3.2.6 CreateScalarField(a,b)

This function creates a scalar field on a grid. Either a matrix can be used to specify the field values at each grid point (the matrix and grid must match dimensionally) or a scalar value can be specified. In the latter case the field will take the same value over the entire grid.

Parameters:

- a (grid) – the grid.
- b (matrix or real) – the matrix or a scalar value.

Returns:

(field) – the new field.

3.2.7 CreateVectorField(a,b,c,d)

This function creates a vector field from a grid and a set of dimensionally matching matrices, or scalar values. For a 2D vector field 2 matrices or scalars are specified, and for a 3D vector field 3 matrices or scalars are given. Note the terms 2D and 3D refer to the number of vector components not the number of independent coordinates.

Parameters:

- a (grid) – the grid for the new 2D or 3D vector field.
- b (matrix or real) – the matrix or a real value corresponding to the first component of the vector field.
- c (matrix) – the matrix or a real value corresponding to the second component of the vector field.
- d (opt) (matrix or grid) – the matrix or a real value corresponding to the third component of the vector field for a 3D vector field. Not required for a vector field with only 2 components.

Returns:

(field) – the new field.

3.2.8 Crosscorrelate1DScalarFields (a,b,c,d,e)

This function computes a one dimensional cross-correlation of two scalar fields over one of their dimensions. An example is given in equation 3.4.

$$cor(x_1, \Delta x_2, x_3) = \overline{u(x_1, x_2, x_3, t) v(x_1, x_2 + \Delta x_2, x_3, t)} \quad (3.4)$$

where *cor* is the cross-correlation function and the overbar indicates averaging over time, *t*. One of the dimensions of the field is designated the correlation dimension – dimension 2 in the example – and another is the averaging dimension – dimension 4 in the example. The correlation dimension and averaging dimension could coincide, in which case the autocorrelation possesses the same number of dimensions as the original field. If these dimensions do not coincide the cross-correlation has one dimension less as in the example. The function computes correlations for both positive and negative values of Δx_2 .

Parameters:

- a (field) – the first scalar field in the cross-correlation
- b (field) – the second scalar field in the cross-correlation
- c (integer) – an integer specifying the correlation dimension (1-4).
- d (integer) – an integer specifying the averaging dimension (1-4).
- e (boolean) – specifies whether the correlation should be normalised.

Returns:

(field) – the correlation field.

3.2.9 Crosscorrelate2DScalarFields (a,b,c,d,e,f)

This function computes a two dimensional cross-correlation of a scalar field over two of its dimensions. An example is provided in equation (3.5).

The function computes correlations for both positive and negative values of Δx_2 and Δt .

$$\text{cor}(x_1, \Delta x_2, x_3, \Delta t) = \overline{u(x_1, x_2, x_3, t) v(x_1, x_2 + \Delta x_2, x_3, t + \Delta t)} \quad (3.5)$$

Parameters:

- a (field) – the first scalar field in the cross-correlation
- b (field) – the second scalar field in the cross-correlation
- c (integer) – an integer specifying the first correlation dimension (1-4).
- d (integer) – an integer specifying the second correlation dimension (1-4).
- e (integer) – an integer specifying the averaging dimension (1-4).
- f (boolean) – specifies whether the correlation should be normalised.

Returns:

(field) – the correlation field.

3.2.10 Differentiate(a,b,c)

This function differentiates a scalar or vector field with respect to one of its independent coordinates. By default a second order polynomial fit (i.e. a central difference is used) but the user can choose to approximate the function at any point using a polynomial, of user-specified order, along with the number of points used to fit the polynomial (in a least squares sense). If insufficient points are available the highest lower order polynomial is used.

Parameters:

- a (field) – the field being differentiated.
- b (integer) – an integer specifying the dimension being differentiated (1-4).
- c (opt) (1D matrix) – a 1D matrix with 2 elements – the polynomial order and the number of points used to fit the polynomial.

Returns:

(field) – the derivative field.

3.2.11 EvaluateFieldOnSurface(a,b,c)

This function evaluates a scalar or vector field on a surface – the surface being a scalar field of one dimension less than the scalar or vector field being evaluated. For example, this function would enable the user to evaluate a two-dimensional scalar field along a contour (the surface) in the x-y plane.

Parameters:

- a (scalar or vector field) – the field being evaluated.
- b (integer) – an integer specifying the dimension in parameter *a* that is absent from the surface.
- c (scalar field) – the surface field having one less dimension than parameter *a*.

Returns:

(field) – the surface field.

3.2.12 ExtrapolateScalarField(a,b,c,d,e)

This function extrapolates a scalar field in one dimension to the boundaries of its grid. It is not unusual for field data to not be defined to the limit of a domain due to measurement accuracy or other difficulties. This method enables careful extrapolation to be performed to “complete” the field. The user can specify boundary values if these are known and can specify the polynomial approximation used to extrapolate the field.

Parameters:

- a (scalar field) – the scalar field being extrapolated.
- b (integer) – the dimension over which the extrapolation is taking place.
- c (real or null) – the value imposed at the lower boundary if known.
- d (real or null) – the value imposed at the upper boundary if known.
- e (1D matrix) – a 1D matrix with 4 elements that provides the extrapolation polynomial information – the first two elements are the order of polynomial used at the lower boundary and the number of points used to fit the polynomial (these points being the last valid points

approaching the boundary). The last two elements are the same for the upper boundary.

Returns:

(field) – the extrapolated field.

3.2.13 ExtrapolateVectorField(a,b,c,d,e)

This function extrapolates a vector field in one dimension to the boundaries of its grid. It behaves exactly the same way as the scalar field equivalent except the extrapolation occurs for all vector components. Thus, if boundary values are specified they must be matrices of appropriate size.

Parameters:

- a (vector field) – the vector field being extrapolated.
- b (integer) – the dimension over which the extrapolation is taking place.
- c (1D matrix or null) – the value of the vector field imposed at the lower boundary if known.
- d (1D matrix or null) – the value of the vector field imposed at the upper boundary if known.
- e (1D matrix) – a 1D matrix with 4 elements that provides the extrapolation polynomial information – the first two elements are the order of polynomial used at the lower boundary and the number of points used to fit the polynomial (these points being the last valid points approaching the boundary). The last two elements are the same for the upper boundary.

Returns:

(field) – the extrapolated field.

3.2.14 FillScalarField(a,b,c,d,e)

This function fills a scalar field in one of its dimensions above or below a given threshold location. The location can be a single value or a matrix or scalar field of one dimension less than the scalar field being filled. The user can specify the fill value and whether the fill occurs below or above the location threshold. The resulting scalar field is of the same size and dimension as the original. For example a two-dimensional scalar

field could be filled with 0 values above a contour line. The fill dimension could be the y direction – dimension 2.

Parameters:

- a (scalar field) - the scalar field being filled.
- b (integer) - the dimension being filled.
- c (real, integer, scalar field or matrix) - the threshold location. If this is a scalar field or matrix it must be one dimension less than parameter a and the remaining dimensions must match in size.
- d (integer) - the fill direction (1 for above the threshold location and -1 for below).
- e (real or integer) – the fill value.

Returns:

(field) – the filled field.

3.2.15 FindContour(a,b,c,d,e,f)

This function searches a scalar field in one of its dimensions seeking a particular field value. Thus, for a 2D field the result would be a 1D scalar field containing the location of the desired contour. The resulting field will always have a dimension one less than the field on which it operates.

Parameters:

- a (scalar field) – the scalar field for which the contour is sought.
- b (integer) – the dimension over which the search is taking place. This dimension does not appear in the resulting field.
- c (real) – the field value (contour) being sought.
- d (integer) – the search direction (1 for a search in the positive direction and -1 for a search in the negative direction).
- e (boolean or null) – if the field value is not found and e is set to true then the contour is placed on a boundary. This parameter can be true/false or null.
- f (integer or null) – if e is true then this integer parameter specifies on which boundary to place the contour. If 0 it is placed on the lower boundary, if 1 on the upper boundary and if -1 (or null) it is placed on the boundary that ends the search.

Returns:

(field) – the contour field.

3.2.16 FindMax(a,b)

This function finds the maximum value of a scalar field over one of its dimensions. The result is a scalar field of one dimension less than the original.

Parameters:

- a (scalar field) – the scalar field for which the maximum is being sought.
- b (integer) – the dimension over which the search for the maximum takes place. This dimension is removed from the scalar field.

Returns:

(field) – the field of maximum values.

3.2.17 FindMaxLocation(a,b)

This function finds the location of the maximum value of a scalar field over one of its dimensions. The result is a scalar field of one dimension less than the original and contains the location of the maximum in the specified dimension.

Parameters:

- a (scalar field) – the scalar field for which the maximum location is being sought.
- b (integer) – the dimension over which the search for the maximum location takes place. This dimension is removed from the scalar field.

Returns:

(field) – the field of maximum locations.

3.2.18 FindMin(a,b)

This function finds the minimum value of a scalar field over one of its dimensions. The result is a scalar field of one dimension less than the original which holds the minimum values.

Parameters:

- a (scalar field) – the scalar field for which the minimum is being sought.
- b (integer) – the dimension over which the search for the minimum takes place. This dimension is removed from the scalar field.

Returns:

(field) – the field of minimum values.

3.2.19 FindMinLocation(a,b)

This function finds the location of the minimum value of a scalar field over one of its dimensions. The result is a scalar field of one dimension less than the original and contains the location of the minimum in the specified dimension.

Parameters:

- a (scalar field) – the scalar field for which the minimum location is being sought.
- b (integer) – the dimension over which the search for the minimum location takes place. This dimension is removed from the scalar field.

Returns:

(field) – the field of minimum locations.

3.2.20 GradientScalarField(a,b)

This function computes the gradient field of a scalar field of maximum dimension 3. The result is a vector 2D field (for a scalar field in two dimensions) or a vector 3D field (for a scalar field in three dimensions). The user can choose to specify a polynomial approximation for computing the scalar field derivatives. They must specify the order of the polynomial used and the number of points used to deduce the

polynomial. If more points are specified than required then the polynomial is computed in a least squares sense. If no polynomial fit is specified, by default, a second order central difference approximation is used where possible.

Parameters:

- a (scalar field) – the scalar field for which the gradient field is being computed.
- b(opt) (matrix) – this matrix provides the polynomial approximation information. It is one-dimensional and must contain two elements – the first being the polynomial order and the second the number of points to use in computing the polynomial.

Returns:

(vector field) – the gradient field.

3.2.21 HighPassFilter(a,b,c)

This function performs a high pass filter on a 1D scalar field. The user specifies the type of filter to be applied along with any parameters that filter requires. Three filters are available at this time:

Step filter: Filter number = 1. Removes all frequencies below the specified cutoff frequency.

Gaussian filter: Filter number = 2. Frequencies below the specified cutoff frequency are weighted with a Gaussian function with a specified standard deviation.

Butterworth filter: Filter number = 3. Frequencies below the specified cutoff frequency are weighted with the Butterworth function of a specified order.

Parameters:

- a (scalar field) – the scalar field being filtered.
- b (int) – the filter number.
- c (1D matrix) – a matrix containing the parameters for the particular filter. For the step filter the matrix contains the cutoff frequency, for the Gaussian filter the cutoff frequency and the standard deviation for the weighting function, for

the Butterworth filter the cutoff frequency and the order of the filter.

Returns:

(scalar field) – the filtered scalar field.

3.2.22 IntegrateDef(a,b,c,d)

This function performs a definite integration of a scalar or vector field over one of its dimensions. The result is a field of the same type with one less dimension. The function can have 2, 3, or 4 parameters. If only two parameters are specified the field is integrated over the whole domain. Parameters 3 and 4 are used to specify the limits of the integration if the entire domain is not to be included.

Parameters:

- a (field) – the field being integrated.
- b (integer) – the dimension over which the integrating is taking place. This dimension does not appear in the resulting field.
- c (opt) (range or real, matrix or scalar field) – if there are only 3 parameters c must be a range parameter specifying the range of indices over which the integration takes place. If there are 4 parameters specified this parameter specifies the lower limit of the integration. This can be either a real value (that doesn't need to correspond to a node location) or a matrix or scalar field of the same dimensions as the field being integrated minus the integrated dimension. Thus the limit of integration can vary for each location in the domain. Thus if a scalar field of dimensions 10x50x100 is integrated in dimension 2 then c would have dimensions 10x100.
- d(opt) (real, matrix or scalar field) – if 4 parameters are specified then d is the upper bound of the integration. Like c it can be a real value of a scalar field or matrix of appropriate dimensions.

Returns:

(field) – the integrated field.

3.2.23 **IntegrateIndef(a,b,c)**

This function performs an indefinite integral of a field producing a field of the same type and number of dimensions. The integration always starts and ends on grid nodes.

Parameters:

- a (field) – the field being integrated.
- b (integer) – the dimension over which integration is taking place.
- c (opt) (range) – if this parameter is specified it must be a range which specifies the limits of the integration. If not specified the integration is performed over the full range of the dimension specified in b.

Returns:

(field) – the integrated field.

3.2.24 **LimitScalarField(a,b,c)**

This function limits the values that can appear in a scalar field. The user specifies the minimum and maximum allowable field values and any field values outside this range are set either to the minimum (for those less than the minimum) or maximum (for those larger than the maximum value).

Parameters:

- a (scalar field) – the scalar field being analysed.
- b (real) – the minimum value for the field.
- c (real) – the maximum value for the field.

Returns:

(scalar field) – the field that has limited values.

3.2.25 **LowPassFilter(a,b,c)**

This function performs a low pass filter on a 1D scalar field. The user specifies the type of filter to be applied along with any parameters that filter requires. Three filters are available at this time:

Step filter: Filter number = 1. Removes all frequencies above the specified cutoff frequency.

Gaussian filter: Filter number = 2. Frequencies above the specified cutoff frequency are weighted with a Gaussian function with a specified standard deviation.

Butterworth filter: Filter number = 3. Frequencies above the specified cutoff frequency are weighted with the Butterworth function of a specified order.

Parameters:

- a (scalar field) – the scalar field being filtered.
- b (int) – the filter number.
- c (1D matrix) – a matrix containing the parameters for the particular filter. For the step filter the matrix contains the cutoff frequency, for the Gaussian filter the cutoff frequency and the standard deviation for the weighting function, for the Butterworth filter the cutoff frequency and the order of the filter.

Returns:

(scalar field) – the filtered scalar field.

3.2.26 LScalarField(a,b,c,d,e)

This function performs a linear transformation on a scalar field. The field and its independent coordinates can be scaled and offset through the chosen parameters. The result is a scalar field of exactly the same size and dimensions as the original.

Parameters:

- a (scalar field) – the scalar field being transformed.
- b (real) – the scale factor for the field. All field values are multiplied by this value.
- c (real) – the additive constant for the field. This is added to the field value after it has been scaled by b.
- d (real or 1D matrix) – the scale factor(s) for the grid coordinates. This can be a real value implying that all grid coordinates are scaled by the same factor, or a 1D matrix of

- the appropriate size which specifies different scale factors for each grid coordinate.
- e (real or 1D matrix) – the additive constant(s) for the grid coordinates. This is either a real value so that the same constant is added to all scaled grid variables, or a 1D matrix which specifies different additive constants for each grid coordinate.

Returns:

(scalar field) – the transformed field.

3.2.27 **LTVectorField(a,b,c,d,e)**

This function performs a linear transformation on a vector field. The field and its independent coordinates can be scaled and offset through the chosen parameters. The result is a vector field of exactly the same size and dimensions as the original.

Parameters:

- a (vector field) – the vector field being transformed.
- b (real or 1D matrix) – the scale factor(s) for the vector field components. This can be a real value, in which case all vector components are scaled by the same amount, or it can be a 1D matrix of appropriate size so each vector component can be scaled differently.
- c (real or 1D matrix) – the additive constant(s) for the vector field components. This can be a real value or a 1D matrix. The constants are added after the vector components have been scaled.
- d (real or 1D matrix) – the scale factor(s) for the grid coordinates. This can be a real value implying that all grid coordinates are scaled by the same factor, or a 1D matrix of the appropriate size which specifies different scale factors for each grid coordinate.
- e (real or 1D matrix) – the additive constant(s) for the grid coordinates. This is either a real value so that the same constant is added to all scaled grid variables, or a 1D matrix which specifies different additive constants for each grid coordinate.

Returns:

(vector field) – the transformed field.

3.2.28 MergeFields(a,b)

This function merges two fields (vector or scalar). The fields must possess the same grids. Merging is not exactly averaging. When two fields are averaged the resultant field is only defined if both fields have well defined values. When two fields are merged the average is computed if both fields are defined at a grid point, but if only one field is defined then its value is used. Only if both fields are undefined will the resultant field be undefined. The merge capability is particularly useful when the results from two, perhaps overlapping cameras, are to be combined.

Parameters:

- a (field) – a vector or scalar field.
- b (field) – a vector or scalar field of exactly the same dimensions and sizes as a.

Returns:

(field) – the merged field.

3.2.29 MovingAverageField(a,b,c,d,e)

This function smooths a scalar or vector field in one dimension using a moving box average. For each grid point a weighted average of the field values within a box, centred on the grid point, are used to compute the averaged field value. Two weighting regimes are possible – a constant weight whereby each field value within the box contributes equally to the average, and a factor based weight where each field value is weighted by its distance from the grid point in question. This weighting factor is computed as follows:

$$w = \alpha^{|i-j|} \quad (3.6)$$

where w is the weight factor applied to the field value at grid index j when computing the averaged field value at grid index i . The parameter α is set by the user.

Parameters:

- a (scalar or vector field) – the field being smoothed.
- b (integer) – the dimension of the field over which the smoothing is occurring.

- c (integer) – the box half-width expressed in terms of number of grid points. This number of grid points on either side of the point in question, together with the grid point, are used in the averaging process. Thus the total box size is $2c + 1$.
- d (integer) – the weighting regime. 1 for a constant weight and 2 for the factor based weight as per equation 3.6.
- e(opt) (real) – the value of α if the factor based weighting regime is employed.

Returns:

(scalar or vector field) – the smoothed field.

3.2.30 Regrid(a,b)

This function takes a vector or scalar field and transforms it onto a different grid. Linear, bi-linear, tri-linear etc. interpolation is used to compute the field data on the new grid.

Parameters:

- a (field) – the scalar or vector field being re-gridded.
- b (grid) – a grid of the same dimensions as the field specified in a. Clearly the grid can have different origins, steps and number of grid points than the grid of the field.

Returns:

(field) – the regridded field.

3.2.31 StdDevField(a,b,c,d)

This function computes the standard deviation of the data in one dimension of a scalar or vector field. It complements the AverageField function.

Parameters:

- a (field) – the field for which the standard deviation is required.
- b (integer) – an integer specifying the dimension (1-4) used to compute the standard deviation.

- c(opt) (range or real) – if c only is specified it must be a range variable that specifies a subset of the dimension to be used in the computation.
- d(opt) (real) – if both c and d are provided they are real values that indicate the start and end of the range to be used. The actual grid points used in the calculation will lie within c and d if c and d do not correspond exactly to grid points.

Returns:

(field) – the field of standard deviations.

3.2.32 StretchField(a,b,c)

This function increases the number of dimensions of a field by one. Thus a 2D field will become a 3D field and so on. The user specifies which dimension is to be created and the grid to use to establish this dimension. The field values will be constant in this new dimension.

An example may help to make this process clear. Imagine a 2D velocity field that depends on x, y and t and has size 300x100x1000. The time-average velocity field can be computed using the AverageField() function. This average field will have size 300x100. If it was desirable to compute the fluctuating velocity field the average field must be subtracted from the original field. This cannot be done because the two fields have different numbers of dimensions. By stretching the average velocity field in the third dimension a velocity field with size 300x100x1000 can be created and then subtracted from the original field. The field value at each x, y point is copied through the third dimension.

Parameters:

- a (field) – the field being stretched.
- b (integer) – the dimension being “created” or stretched – (1-4).
- c (1D grid) – the 1D grid that defines the new dimension.

Returns:

(field) – the stretched field.

3.2.33 SumField (a,b,c,d)

This function sums the data in one dimension of a scalar or vector field.

Parameters:

- a (field) – the field that is summed.
- b (integer) – an integer specifying the dimension (1-4) used in the summation.
- c(opt) (range or real) – if c only is specified it must be a range variable that specifies a subset of the dimension to be used in the computation.
- d(opt) (real) – if both c and d are provided they are real values that indicate the start and end of the range to be used. The actual grid points used in the calculation will lie within c and d if c and d do not correspond exactly to grid points.

Returns:

(field) – the field summed field.

3.2.34 SwapScalarFieldDimension(a,b,c)

This function swaps two of the dimensions of a scalar field. Effectively these exchange places. This is not the same as the SwapScalarField() function which reverses the grid in one dimension of the field.

Parameters:

- a (scalar field) – the scalar field being swapped.
- b (integer) – the first dimension to be swapped – (1-4).
- c (integer) – the second dimension to be swapped – (1-4).

Returns:

(scalar field) – the swapped field.

3.2.35 SwapScalarField(a,b)

This function swaps one of the dimensions of a scalar field. Thus the origin of that dimension becomes the negative of the previous maximum of the dimension. The step and number of grid points remain the same.

Parameters:

- a (scalar field) – the field being swapped.
- b (integer) – the dimension being swapped – (1-4).

Returns:

(scalar field) – the swapped field.

3.2.36 SwapVectorField(a,b)

This function swaps one of the dimensions of a vector field. It behaves in the same way as the SwapScalarField() but with the addition that the component of the vector field corresponding to the dimension being swapped will reverse sign. Thus if a field is being swapped in the second dimension the second component of the vector field will be multiplied by -1.

Parameters:

- a (vector field) – the field being swapped.
- b (integer) – the dimension being swapped – (1-4).

Returns:

(vector field) – the swapped field.

3.2.37 TranslateField(a,b,c)

This function translates a scalar or vector field on its underlying grid. It is primarily intended to provide the functionality that would translate field data into a frame of reference moving with a constant velocity. Thus the translation requires one dimension to be designated as the invariant dimension. All other dimensions are then translated by a constant amount for each increment of the invariant dimension. For example, consider a 3D scalar field with dimensions x, y and t. To shift the field data into a moving frame of reference t would be selected as the invariant dimension and the user would specify constant offsets that would be applied to the x and y dimensions for each increment in t. These offsets would correspond to the products of the velocities in the x and y directions with the time step. The result is a new field that has an expanded grid where much of the data on that grid may be undefined. The new grid retains the same steps as the original grid and therefore it may be necessary to interpolate the translated data onto the new grid.

Parameters:

- a (field) – the field being translated.
- b (integer) – the invariant dimension – (1-4).
- c (matrix) – a matrix of offsets for the remaining dimensions.
The matrix will have one less dimension than the original field.

Returns:

(field) – the translated field.

3.3 Summary

In this chapter a detailed inventory of the tablet functions that operate on fields has been presented. A description of each function has been provided along with a list of its parameters and its return value. The ordering and types of the function parameters are rigidly enforced by the tablet command parser.

4 Grid Functions

4.1 Description

In this chapter we describe the functions that are applicable to grids. Table 4.1 provides a list of the grid functions with details to follow.

Table 4.1 The grid based functions.

Function name	Description
CreateGrid()	Creates a new grid of 1, 2, 3 or 4 dimensions.
ExtractGrid()	Extracts an entire grid or one of its dimensions from a grid or field.
LTGrid()	Performs a linear transformation on a grid.
StretchGrid()	Increases the number of dimensions of a grid by one.

4.2 FunctionLibrary

4.2.1 CreateGrid (a,b,c,d,e,f,g,h,i,j,k,l,m,o,p)

This function creates a grid of 1, 2, 3 or 4 dimensions. Four parameters (described below) are required for each dimension, thus the number of parameters must be 4, 8, 12, or 16.

Parameters:

- a (integer) – the number of grid points in this dimension
- b (real) – the origin of the grid in this dimension
- c (real) – the grid step between grid points in this dimension
- d (string) – the dimension label

Returns:

(grid) – the created grid.

4.2.2 **ExtractGrid(a,b)**

This function extracts either the entire grid or a 1D grid from a grid or field of the same or higher dimension.

Parameters:

- a (grid or field) – the grid or field from which the 1D grid is being extracted.
- b(opt) (integer) – an integer specifying the dimension being extracted. If not provided the entire grid is extracted.

Returns:

(grid) – the extracted grid.

4.2.3 **LTGrid(a,b,c)**

This function performs a linear transformation on a grid. The grid variables can be scaled and offset. The result is a grid of exactly the same size and dimensions as the original.

Parameters:

- a (grid) – the grid being transformed.
- b (real or 1D matrix) – the scale factor(s) for the grid coordinates. This can be a real value implying that all grid coordinates are scaled by the same factor, or a 1D matrix of the appropriate size which specifies different scale factors for each grid coordinate.
- c (real or 1D matrix) – the additive constant(s) for the grid coordinates. This is either a real value so that the same constant is added to all scaled grid variables, or a 1D matrix which specifies different additive constants for each grid coordinate.

Returns:

(grid) – the transformed grid.

4.2.4 **StretchGrid(a,b,c)**

This function behaves in the same way as the StretchField() function in that it increases the number of dimensions of a grid by one. Thus a 2D grid will become a 3D grid and so on. The user specifies which

dimension is to be created and a 1D grid to use to establish this dimension.

Parameters:

- a (grid) – the grid being stretched.
- b (integer) – the dimension being “created” or stretched – (1-4).
- c (1D grid) – the 1D grid that defines the new dimension.

Returns:

(grid) – the stretched grid.

4.3 Summary

This chapter has provided details of the grid-based functions.

5 Matrix Functions

5.1 Description

In this chapter we describe the functions that are applicable to matrices. Table 5.1 provides an overview of the matrix based functions with details provided following the table.

Table 5.1 The matrix based functions.

Function name	Description
AverageMatrix()	Averages a matrix over one of its dimensions.
ExtractMatrix()	Extracts a matrix from a field.
LTMatrix()	Performs a linear transformation on a matrix.
StdDevMatrix()	Computes the standard deviation of the data in one the dimensions of the matrix.
StretchMatrix()	Increases the number of dimensions of a matrix by one.

5.2 Function Library

5.2.1 AverageMatrix(a,b,c)

This function averages a matrix over one of its dimensions.

Parameters:

- a (matrix) – the matrix to be averaged
- b (integer) – an integer specifying the dimension (1-4) that is being averaged over.
- c (opt) (range) – if c is specified it must be a range variable that specifies a subset of the dimension over which the average takes place.

Returns:

(matrix) – the average matrix.

5.2.2 ExtractMatrix(a)

This function extracts a matrix from a scalar field. Effectively the matrix is the field without the underlying grid.

Parameters:

- a (field) – the field from which the matrix is being extracted.

Returns:

- (matrix) – the extract matrix.

5.2.3 LTMatrix(a,b,c)

This function performs a linear transformation on a matrix. The matrix can be scaled and offset through the chosen parameters. The result is a matrix of exactly the same size and dimensions as the original.

Parameters:

- a (matrix) – the matrix being transformed.
- b (real) – the scale factor for the matrix. All matrix values are multiplied by this value.
- c (real) – the additive constant for the matrix. This is added to the matrix value after it has been scaled by b.

Returns:

- (matrix) – the transformed matrix.

5.2.4 StdDevMatrix(a,b,c)

This function computes the standard deviation of the data in one dimension of a matrix. It complements the AverageMatrix() function.

Parameters:

- a (matrix) – the matrix for which the standard deviation is required.
- b (integer) – an integer specifying the dimension (1-4) used to compute the standard deviation.

- c (opt) (range) – if c is specified it must be a range variable that specifies a subset of the dimension to be used in the computation.

Returns:

(matrix) – the matrix of standard deviations.

5.2.5 **StretchMatrix(a,b,c)**

This function behaves in a similar way to the StretchField() function in that it increases the number of dimensions of a matrix by one. Thus a 2D matrix will become a 3D matrix and so on. The user specifies which dimension is to be created and the size of this dimension.

Parameters:

- a (matrix) – the matrix being stretched.
- b (integer) – the dimension being “created” or stretched – (1-4).
- c (integer) – an integer specifying the size of the new dimension.

Returns:

(matrix) – the stretched matrix.

5.3 **Summary**

This chapter has provided a detailed summary of all tablet functions associated with matrices.

6 Miscellaneous Functions

6.1 Description

In this chapter we describe a number of functions that are not specifically associated with fields, matrices or grids. Table 6.1 provides an overview of the miscellaneous functions with details provided following the table.

Table 6.1 The miscellaneous functions.

Function name	Description
DimLabel()	Extracts the label of one dimension of a field or grid.
DimOrigin()	Extracts the origin of one dimension of a field or grid.
DimSize()	Extracts the size of one dimension of a field, matrix, or grid.
DimStep()	Extracts the step size of one dimension of a field or grid.
EvaluateLeastSquaresSurfaceFit ()	Evaluates a least squares fit to a 2D surface at a set of specified (x,y) points.
LeastSquaresSurfaceFit()	Computes a least squares fit to a 2D surface using polynomial basis functions up to order 3 in each dimension.
LeastSquaresSurfaceFitFunction()	Computes a custom function based on a least squares fit to a 2D surface at a set of specified (x,y) points.
NumDims()	Extracts the number of dimensions of a field, matrix, or grid.

6.2 Function Library

6.2.1 DimLabel(a,b)

This function returns the label associated with the specified dimension of a scalar field, vector field or grid.

Parameters:

- a (scalar field, vector field or grid) – the field or grid whose label is required.
- b (integer) – an integer specifying the dimension (1-4) whose label is sought.

Returns:

(string) – the label string.

6.2.2 DimOrigin(a,b)

This function returns the origin associated with the specified dimension of a scalar field, vector field or grid.

Parameters:

- a (scalar field, vector field or grid) – the field or grid whose origin is required.
- b (integer) – an integer specifying the dimension (1-4) whose label is sought.

Returns:

(real) – the origin of the specified dimension.

6.2.3 DimSize(a,b)

This function returns the size of a specified dimension of a scalar field, vector field, matrix or grid.

Parameters:

- a (scalar field, vector field, matrix or grid) – the field or grid whose size is required.
- b (integer) – an integer specifying the dimension (1-4) whose label is sought.

Returns:

(integer) – the number of elements in the specified dimension.

6.2.4 DimStep(a,b)

This function returns the step between values of a specified dimension of a scalar field, vector field, or grid.

Parameters:

- a (scalar field, vector field, matrix or grid) – the field or grid whose step is required.
- b (integer) – an integer specifying the dimension (1-4) whose label is sought.

Returns:

(real) – the step between values in the specified dimension.

6.2.5 EvaluateLeastSquaresSurfaceFit(a,b,c,d)

This function evaluates a least squares fit to a surface at a set of (x,y) points. The result is a matrix containing the surface coordinates. The coefficients of the least squares fit to the surface are likely to have been computed using the LeastSquaresSurfaceFit() function and are ordered as given in that function's description. This function can be useful in computing errors in the least squares fit.

Parameters:

- a (integer) – the order of polynomial being used (0-3).
- b (matrix) – a 1D matrix containing the x coordinates of the points defining the surface.
- c (matrix) – a 1D matrix containing the y coordinates of the points defining the surface.
- d (matrix) – a 1D matrix containing the least squares fit coefficients.

Returns:

(matrix) – the matrix of values computed by the least squares fit at the specified x and y coordinates.

6.2.6 LeastSquaresSurfaceFit(a,b,c,d)

This function computes a least squares fit to a 2D surface using polynomial basis functions up to order 3. This function is of value when

mapping the coordinates in a 2D particle record. By using a number of points in the plane with known coordinates and pixel locations these can be used to compute mapping functions that take the pixel coordinates as inputs and yield either the x or y physical coordinates.

The output of this function is a one dimensional matrix containing the coefficients of the least squares fit. The number of the coefficients depends on the order of the basis functions. If a full 3rd order polynomial in both x and y directions is used the coefficients are output in the following order – 1, x, y, x², xy, y², x³, x²y, xy², y³.

Parameters:

- a (integer) – the order of polynomial being used (0-3).
- b (matrix) – a 1D matrix containing the x coordinates of the points defining the surface.
- c (matrix) – a 1D matrix containing the y coordinates of the points defining the surface.
- d (matrix) – a 1D matrix containing the surface value of at the corresponding (x,y) point.

Returns:

(matrix) – the least squares best fit polynomial coefficients.

6.2.7 LeastSquaresSurfaceFitFunction (a,b,c,d)

This function is almost identical to the LeastSquaresSurfaceFit() function. The difference is that this function yields a custom function that encapsulates the fitted polynomial instead of producing a matrix of polynomial coefficients. In addition, the order of the polynomial basis functions is not restricted.

Parameters:

- a (integer) – the order of polynomial being used.
- b (matrix) – a 1D matrix containing the x coordinates of the points defining the surface.
- c (matrix) – a 1D matrix containing the y coordinates of the points defining the surface.
- d (matrix) – a 1D matrix containing the surface value of at the corresponding (x,y) point.

Returns:

(custom function) – the polynomial function fitted to the surface.

6.2.8 NumDims(a)

This function returns the number of dimensions of a field, matrix or grid.

Parameters:

- a (scalar field, vector field, matrix or grid) – the field or grid whose dimensions are sought.

Returns:

(integer) – the number of dimensions of the field, matrix or grid.

6.3 Summary

This chapter has summarised the suite of miscellaneous functions provided by tablets.

7 Tablets

7.1 Description

Tablets play two roles within the *Streams* environment. Firstly, they provide an analysis toolkit, based around a library of specialized functions, that enables the user to manipulate fields (and other objects) in a variety of ways. Secondly, they provide a repository for related fields that allows simple transfer of data from one user to another, or simply an efficient means for organizing an individual user's data.

The preceding chapters have provided an extensive introduction to the structure of tablet commands as well as a description of the specialized library functions.

Tablets are created by selecting the Create tablet... option from the Tablet menu in the application's main menubar. Once created the tablet appears in the application's object list.

7.2 Views

A tablet has two views.

- *Details view.*
- *Analysis view.*

7.2.1 Details view

Description

The details view enables the user to change the name of the tablet and to edit the tablet's description. These are provided on two tabs labelled Name and Description.

Access

The tablet details view can be opened through the popup menu of the tablet.

7.2.2 Analysis view

Description

The analysis view is the primary, default, view for a tablet, providing the user access to all data objects in the tablet and the ability to enter commands.

At the top of the view are six tabs, one for each of the different data types. Below the tabs is a list of the named variables for the selected data type. Thus, when the Fields tab is selected the list displays all scalar and vector fields currently stored within the tablet. For each field three items are displayed in the list. Firstly the icon corresponding to the field type, secondly the unique name allocated to the field, and lastly a description of the sizes of the field's dimensions – e.g. 100x50x1000 for a three dimensional field.

The grid and matrix lists display the same set of information for the currently defined grids and matrices. The scalar list includes the value of each scalar and the String list the string associated with string variable. Finally, the custom functions list includes just an icon and the names of the custom functions.

All of these lists support drag and drop. Thus fields can be dragged and dropped onto the field list, grids can be dropped onto the grid list and so on. Each list can respond in a number of ways to an object being dropped upon it. If that object comes from another tablet there are two possibilities. Firstly, if the name of the dropped object conflicts with any object currently in the tablet the user will be prompted to provide a different unique name, and then the dropped object is copied into the tablet and displayed in the appropriate list. Secondly, if the name does not conflict with any of the objects currently in the tablet the object is simply copied to the tablet and displayed in the appropriate list.

In the case of fields the source of the drag and drop could be one of a number of different field lists that exist in *Streams*. The source could be a field list in a calculator manager view, it could be the object or field list views of the application, or it could be the Field Registry list view. In all of these cases the user will be prompted to provide a unique name for the field so that it can be copied to the tablet.

It is worth reiterating that a field added to a tablet through the drag and drop mechanism will be a copy of the source field. Thus, if the original field is deleted the field within the tablet will be unaffected.

Beneath the list of objects for each of the six tabs lies a set of buttons that manage the associated list of objects. The buttons for each tab vary somewhat and are described below.

Fields

- *Create* – pressing this button will open a dialog box that enables the user to create a new field. The dialog box provides a text field in which the field name can be specified and two buttons, *Select* and *Load*, that specify the source of the field. The *Select* button allows the user to select a field from the Field Registry (a list of all fields known to *Streams*). The *Load* button allows the user to select a field from a disk file. In both cases the user must enter a name.
- *Delete* – all fields selected in the list box are deleted. The user is prompted to confirm the action.
- *Clear* – deletes all fields in the tablet. The user is prompted to confirm the action.

Grids

- *Create* – pressing this button will open a dialog box that enables the user to create a new grid based on a field already existing within *Streams*. The dialog box provides a text field in which the grid name can be specified. The *Select* button allows the user to select a field from the Field Registry from which the grid details will be extracted.
- *Load* – pressing this button will open a dialog box that enables the user to load a grid previously stored in a disk file. Grids have a .g1d, .g2d, .g3d or .g4d extension depending on the number of dimensions of the grid.
- *Delete* – all grids selected in the list box are deleted. The user is prompted to confirm the action.
- *Clear* – deletes all grids in the tablet. The user is prompted to confirm the action.

Matrices

- *Create* – pressing this button will open a dialog box that enables the user to create a new matrix based on a field already existing within *Streams*. The dialog box provides a text field in which the matrix name can be specified. The *Select* button allows the user to select a field from the Field Registry from which the matrix data will be extracted.
- *Load* – pressing this button will open a dialog box that enables the user to load a matrix previously stored in a disk file. Matrices have a .m0d, .m1d, .m2d, .m3d, .m4d extension depending on the number of dimensions of the matrix.
- *Delete* – all matrices selected in the list box are deleted. The user is prompted to confirm the action.
- *Clear* – deletes all matrices in the tablet. The user is prompted to confirm the action.

Scalars

- *Load* – pressing this button will open a dialog box that enables the user to load a scalar previously stored in a disk file. Scalars have a .slr extension.
- *Delete* – all scalars selected in the list box are deleted. The user is prompted to confirm the action.
- *Clear* – deletes all scalars in the tablet. The user is prompted to confirm the action.

Strings

- *Load* – pressing this button will open a dialog box that enables the user to load a string previously stored in a disk file. Strings have a .str extension.
- *Delete* – all strings selected in the list box are deleted. The user is prompted to confirm the action.
- *Clear* – deletes all strings in the tablet. The user is prompted to confirm the action.

Custom functions

- *Create scalar* – pressing this button will open a dialog box that enables the user to define a custom function that generates a scalar field. Two pieces of information are required and are entered into the two tabs in the dialog box. The first is the unique name given to the custom function. The second is the mathematical formula that defines the function. By right clicking in the text box in which the formula is to be entered the user can display a popup menu that lists all valid tokens that can appear in the formula. As discussed in chapter 1 the independent variables are given the symbols x , y , z and t . Note that even if the independent variables physically correspond to x , y and t for a three dimensional scalar field, the symbols x , y and z would be used in the formula.
- *Create vector 2D* – pressing this button will open a dialog box that enables the user to define a custom function that generates a 2D vector field. The discussion for the Create scalar button is relevant to this button as well but in this dialog box 3 tabs are provided. Two of the tabs are used to enter the formulae for the two vector field components.
- *Create vector 3D* – pressing this button will open a dialog box that enables the user to define a custom function that generates a 3D vector field. The discussion for the Create scalar button is relevant to this button as well but in this dialog box 4 tabs are provided. Three of the tabs are used to enter the formulae for the three vector field components.
- *Load* – pressing this button will open a dialog box that enables the user to load a custom function previously stored in a disk file. Custom functions have one of three extensions – .sfn for a custom function that produces a scalar field, .v2fn for a 2D vector field function and .v3fn for a 3D vector field function.

- *Delete* – all custom functions selected in the list box are deleted. The user is prompted to confirm the action.
- *Clear* – deletes all custom functions in the tablet. The user is prompted to confirm the action.

The bottom half of the analysis view contains the command area. At the top is the command list. This list includes all commands previously executed with the latest command the last in the list. This list supports drag and drop so that any command(s) in the list can be dragged to the command line (see below) in this view, or the analysis view of any other tablet. Doubling clicking a command in the list has the same result as dragging the command to the command line.

Groups of commands can be saved to file. Commands are selected in the list and right clicked with the mouse. The resulting popup menu has a *Save commands...* option that enables the user to save the selected commands to a file with a *.comscr* extension. These commands can then be loaded into any tablet either into the command list or directly into the command line.

Beneath the command list is a row of buttons that manage the command list.

- *Load commands* – allows the user to load commands from a script file (with *.comscr* extension). The commands in the file are added to the end of the command list.
- *Delete commands* – all selected commands in the list are deleted. The user is prompted to confirm this action.
- *Clear commands* – deletes all commands from the list. The user is prompted to confirm this action.

Beneath the command list is the command line itself. The user enters commands in this area following the syntax described in chapters 1 and 2. Commands that contain errors will cause a dialog box describing the errors to be displayed. In addition, the command will be reproduced on the command line for easy editing. Pressing the ESCAPE key when the command line has the keyboard focus will cause the command line to be cleared. Past commands can be recalled by pressing the UP and DOWN arrow keys.

When the user right clicks in the command line area a popup menu is displayed that gives the user access to all tokens that can be used in a command, including user defined variables and custom functions. This eases the burden on the user in having to remember the exact names of the many specialized functions available to them.

It is possible for multiple commands to be copied to the command line together, thus effectively generating an executable script. When these are executed the tablet executes the commands in sequence starting at the top. There are some subtleties here. It is common for these scripts to create new variables that are then used in subsequent commands. To ensure that this order of execution is precisely followed the commands must be serially executed and block the execution of the rest of the application. This is the default mode of tablet command execution. It is possible to execute these commands while other parts of the application can proceed in parallel. In this case accessing newly created variables within the script is likely to fail as updating of the tablet happens asynchronously. The execution mode is set in the Settings menu (see below).

Below the command line is a row of buttons that manage the command line.

- *Execute* – pressing this button will execute the current command. Pressing the RETURN key achieves the same result.
- *Load command* – pressing this button allows the user to load commands from a script file (.comscr) directly into the command line. The command is cleared before the new commands are added.
- *Clear command line* – deletes all commands in the command line.

Access

The analysis view is the default view for a tablet. It can be opened by double clicking the tablet, or through the popup menu of the tablet object.

Menus

Settings menu

- *Execute in parallel* – when set this causes the tablet to execute the command line commands in parallel. When not set execution occurs serially. This is the preferred option when executing scripts containing many commands. See discussion above.
- *Confirm overwrite of variables* – when this menu item is selected (as it is by default) the user will be prompted if they attempt to overwrite a variable that has been previously created through the execution of a new command. If not selected the overwrite occurs without confirmation by the user.

7.3 Popup menu

The tablet provides the following options in its popup menu.

- *Save as...* – saves the tablet object to a new disk file. All tablet files have the extension *.tab*. All data objects as well all commands are stored with the tablet.
- *Save* – saves the tablet object in its current disk file. If no file exists the user will be prompted to specify a file.
- *Change name...* – this option allows the user to change the name of the tablet.
- *Change description...* – this option allows the user to edit the description of the tablet.
- *Open details view* – opens the tablet's details view. See section 7.2.1.
- *Open analysis view* – opens the tablet's analysis view. See section 7.2.2.
- *Close all views* – closes all views for this tablet that are currently open.
- *Close* – removes the tablet from the *Streams* environment. The user will be prompted to save the tablet before closing if it has been changed since last being saved.

8 Fields

8.1 Description

Field objects are normally the primary objects manipulated within a tablet. This prime position is reflected by the extensive library of specialized functions dedicated to field manipulation (see chapter 3). Fields are discussed extensively in the *Streams 3.03 – User's Guide: Field Objects* manual, and there is little reason to repeat that information here. The reader is referred to that manual for material on field views and their popup menus.

Fields are created in tablets in a number of ways. Most commonly fields are dragged to the tablet analysis view from some other source, or loaded from file. In each case the user will be asked to prompt to specify a unique name for the field. Occasionally a field might be constructed manually from a matrix and grid using one of the specialized functions designed for that purpose. Most commonly fields are produced through the execution of a tablet command as the vast majority of the functions described in chapter 3 create fields.

8.2 Views

The views available for each type of field object are described in detail in the *Streams 3.03 – User's Guide: Field Objects* manual. The only view described here is the details view as it differs from the standard details view.

8.2.1 Details view

Description

The details view of a field matches that of the equivalent dimensional grid. A tab is dedicated to each dimension of the field, listing its origin, its step, the number of grid points and its label. Unlike the grid details view the user cannot change any of these parameters. A separate tab lists the name allocated to the field on its creation, together with the number of dimensions. Neither of these can be changed by the user.

Access

The field details view can be opened through the popup menu of the field.

8.3 Popup menu

The field provides the same popup menu as that described in the *Streams 3.03* – User's Guide: Field Objects manual.

9 Grids

9.1 Description

A grid object represents a regular rectangular mesh. That mesh may be one, two, three or four dimensional. Each dimension of the grid is defined by four parameters – the origin of the dimension, the step between grid points, the number of grid points, and a textual label.

All field objects are based on an underlying grid.

Grids can be created using the `CreateGrid()` function. This function can take, 4, 8, 12 or 16 parameters depending on the number of grid dimensions. Thus the following command would create a three dimensional grid with its origin at (0,0,1), with grid spacings of 0.1, 0.2 and 0.5 respectively, and with 101, 201 and 301 grid points in the three directions.

```
G3 = CreateGrid(101,0,0.1,"x",201,0,0.2,"y",301,1,0.5,"z")
```

9.2 Views

A grid has one view.

- *Details view.*

9.2.1 Details view

Description

The details view of a grid has a tab for each of its dimensions. Each tab lists the origin, step, size and label for that dimension. The user is able to edit the origin, step and label, but not the size. An additional tab lists the name and number of dimensions of the grid. Neither can be changed.

Access

The grid details view can be opened through the popup menu of the grid.

9.3 Popup menu

The grid provides the following options in its popup menu.

- *Save as...* – saves the grid object to a new disk file. The extension used for the file depends on the grid dimensions. The extensions are *.g1d*, *.g2d*, *.g3d*, and *.g4d*.
- *Open details view* – opens the tablet's details view. See section 9.2.1.
- *Close all views* – closes all views for this tablet that are currently open.

10 Matrices

10.1 Description

Matrix objects are equivalent to their mathematical counterparts. Tablets support zero, one, two, three and four dimensional matrices.

The field data in a scalar field object can be thought of as a matrix of real numbers defined upon a rectangular grid.

Within tablets one dimensional matrices are the most common and are the most useful. They are frequently used as parameters in specialized tablet functions. One dimensional matrices are the only type of matrix that can be manually constructed through the command line. The following command creates a six element one dimensional matrix:

$$M = \{1,2,3,4,5,6\}$$

10.2 Views

Matrix objects support a wide range of views. The majority of these fields replicate the views available for the scalar field of corresponding dimension. For example, a one dimensional matrix provides a graph view, a PDF view, a spectral view and a table view. All of these views are equivalent to the corresponding views of the scalar 1D field, described in chapter 13 of the *Streams 3.03 – User's Guide: Field Objects* manual. The difference between the views for the matrix and the field is that the matrix has no underlying grid and therefore the independent variables for the matrix views are simply the indices (starting at 0) of the elements in the matrix.

Here we will discuss only the details views of the matrix objects as these differ from the equivalent views of the field.

10.2.1 Details view

Description

The details view enables the user to view the name of the matrix, the number of its dimensions and the size of each of those dimensions. None of these parameters can be changed by the user.

Access

The matrix details view can be opened through the popup menu of the matrix.

10.3 Popup menu

The matrix provides the following options in its popup menu.

- *Save as...* – saves the matrix object to a new disk file. The extension used for the file depends on the grid dimensions. The extensions are .m0d, .m1d, .m2d, .m3d, and .m4d.
- *Open details view* – opens the tablet's details view. See section 10.2.1.
- *Open ... view* – matrices of different dimensions will have their own set of views that can be opened with the menu items listed.
- *Close all views* – closes all views for this matrix that are currently open.

II Scalars

II.1 Description

Scalars correspond to variables that store a single number or Boolean value. Tablets support three different types of scalar variables

- *integers*
- *reals*
- *Booleans*

Wherever a real variable can appear in a tablet command it is legitimate to substitute an integer. However the reverse is not the case. Integers are most frequently used in specifying indices in matrices or fields and real numbers cannot play this role.

Tablets use a number of reserved tokens for use with scalar variables. The tokens *pi* and *PI* corresponds to their mathematical equivalent, and *true* and *false* are self-explanatory Boolean constants.

Scalars are created by assigning a value to a variable. For example

I = 6	creates an integer scalar variable
R = 2*pi	creates a real scalar variable
B = false	creates a Boolean scalar variable

II.2 Views

A scalar has a single view.

- *Details view.*

II.2.1 Details view

Description

The details view displays the scalar name and its associated value.

Access

The details view can be opened through the popup menu of the scalar.

11.3 Popup menu

The scalar provides the following options in its popup menu.

- *Save as...* – saves the scalar object to a new disk file. All scalar files have the extension *.slr*.
- *Open details view* – opens the string's details view. See section 11.2.1.
- *Close all views* – closes all views for this string that are currently open.

12 Strings

12.1 Description

Strings are variables that store textual information. They are created by assigning a double-quoted text string to a string variable in a tablet command. For example

```
axisLabel = "strain"
```

Within a tablet strings play a limited role, primarily used for labelling grid or field axes.

12.2 Views

A string has a single view.

- *Details view.*

12.2.1 Details view

Description

The details view displays the string name and its associated text.

Access

The details view can be opened through the popup menu of the string.

12.3 Popup menu

The string provides the following options in its popup menu.

- *Save as...* – saves the string object to a new disk file. All string files have the extension *.str*.
- *Open details view* – opens the string's details view. See section 12.2.1.
- *Close all views* – closes all views for this string that are currently open.

13 Custom functions

13.1 Description

Custom functions are bespoke functions that are intended to compute a scalar or vector field, or a matrix, at a set of points in 1D, 2D, 3D or 4D space. The functions are based on mathematical formulae that include some or all of the four independent variables – designated as x , y , z and t .

Custom functions are created via two mechanisms. The first is a manual creation process via the tablet analysis view. The user can define the function by typing its formula or formulae – depending on whether it creates a scalar or a vector field – into a text box. The second is through the execution of a specialized function. For example, the `LeastSquaresSurfaceFitFunction()` function generates a custom function based on two-dimensional polynomials that corresponds to the least squares best fit to points on a given surface.

Once created, custom functions can be used in exactly the same manner as specialized functions in tablet commands. The parameters for a custom function are either a grid at which the function is to be evaluated, or, in the case of a function that computes a scalar field, a set of arbitrary points encapsulated in a set of one-dimensional matrices. When a grid is specified a scalar or vector field is generated. If a set of points is specified through a set of matrices, a matrix, containing the function values at the associated points, is generated.

13.2 Views

A custom function has a single view.

- *Details view.*

13.2.1 Details view

Description

The details view enables the user to view the name and formula(e) for the custom function. A separate tab is provided for each formula, so for a custom function that creates a scalar field a single tab is provided, while

for a function that creates a 2D or 3D vector field, two or three tabs are provided respectively. A separate tab displays the function name. Neither the function name nor the formula(e) can be edited by the user.

Access

The custom function details view can be opened through the popup menu of the custom function.

I 3.3 Popup menu

The custom function provides the following options in its popup menu.

- *Save as...* – saves the custom function object to a new disk file. The file extension depends on the type of field the custom function produces. For a scalar field the extension is *.sfn*, for 2D and 3D vector fields the extensions are *.v2fn* and *.v3fn* respectively.
- *Open details view* – opens the custom function's details view. See section I3.2.1.
- *Close all views* – closes all views for this custom function that are currently open.