INTREPID expressions and functions (R12)

Top

You can use expressions for specifying new field values and conditions as required in INTREPID tools. INTREPID can parse any standard expression[15] that you might use in a spreadsheet or programming language. INTREPID has an extensive range of functions (the INTREPID **`Calc`** class) for including in expressions. You can create user defined functions in dynamic linked library / shared object files, or user defined function programs with built-in access to INTREPID datasets.

INTREPID tools using expressions

INTREPID uses expressions for

- Searching, replacing and creating new fields in the INTREPID Spreadsheet Editor tool (See Spreadsheet Editor (T15)).
- Defining criteria for extracting subsections of a dataset in the Subsection tool (See Subsections of datasets (T21)).
- Enabling and disabling crossovers in a tie line levelling task (See "Excluding crossovers from the levelling calculation" in Line correction and tie levelling (T30)).

Expression value data types

The values of INTREPID expressions can have the following data types:

Numeric INTREPID will freely and automatically convert amongst the Byte, Integer (2 byte), Integer (4 byte), Real (4 byte) and Real (8 byte) data types.

Logical Logical expressions have the values **`1`** for 'true' and **`0`** for 'false'. The Spreadsheet Editor tool shows these values as **`YES`** and **`NO`**.

Character INTREPID has no character operators or functions that yield character data. The only character type expression is a literal string, e.g., **`"Zone A"`**)

Object Data INTREPID types maybe created and queried in the spread sheet using expressions and special functions. Tensor, Date, Complex, and all the variations on a 3D vector are involved here.

See "Data Types in INTREPID datasets" in INTREPID database, file and data structures (R05) for further details, including a discussion of the value of ***null***.

Variables, constants, functions, parentheses

Variables

You can use the following as variables in INTREPID expressions

- Vector dataset field names (e.g., **`magnetic`**),
- Vector dataset field band specifications (e.g., **`magnetic:1`**),
- Grid dataset names (e.g., **`radio426`**),
- Grid dataset band specifications (e.g., **`radio426:1`**),
- The inbuilt variables **`%GROUP`** and **`%SAMP`** or **`%ROW`**.

1.[5] INTREPID has a full range of operators for numeric data, and can compare character data for equality.

INTREPID inbuilt variables

You can use the following inbuilt variables in INTREPID expressions.

Note: In earlier releases of INTREPID, the prefix for these variables was **$**. We have changed it to **%**. **$** will still work in interactive mode, but we recommend that you change to **%**.

Standard Variable	Refers to (grid datasets)	Refers to (vector datasets)
%GROUP	The current row of cells.	The current group (usually the current line)
%SAMP or **%ROW**	The current cell	The current data point (set of readings)

Function calls

You can include INTREPID function calls in expressions, using the format *function_name(argument1, argument2, ...)*

Examples

```
correlation(mag,adj_mag)
```

```
cos(X + sqrt(Y))
```

Note: User defined API function calls must be prefixed by an **@** symbol and may only have fields, field bands or grid dataset bands as arguments (i.e., arguments may not be other expressions). **Example**

```
@adj_mag(magnetic)
```

Numeric constants

You can include numeric constants using normal decimal notation or scientific notation using **E**:

Example

```
4  45.32   .9432   -.54   .3215E4
```

Character (string) constants

You can include string constants using double quotes

Example

```
"Zone A"
```

Logical constants

Use the numbers **0** and **1** for logical constants.

For logical constant 'true' use **1**For logical constant 'false' use **0**

For example, if you have a logical field to which you wish to assign an initial value of 'true', specify its initial value as **1**.

The Spreadsheet Editor tool displays the value 'true' as **YES** and the value 'false' as **NO**.

INTREPID *null*

Use the word **null** in an expression for the INTREPID ***null*** constant.

Parentheses

INTREPID expressions may contain nested parentheses using standard spreadsheet and programming language format.

Operators in INTREPID expressions

INTREPID expressions use operators as shown in the following table.

Operator	Symbol	Operandstype	Resulttype	Example
Addition	`+`	numeric	numeric	`Magnetic + .00342`
Subtraction	`-`	numeric	numeric	`Magnetic - .00342`
Multiplication	`*`	numeric	numeric	`Magnetic * 1.00562`
Division	`/`	numeric	numeric	`Magnetic / 1.00562`
Negation	`-`	numeric	numeric	`- Magnetic`
Equality	`==`	any	logical	`Latitude == null`
Greater than	`>`	any	logical	`Latitude > -23`
Less than	`<`	any	logical	`Latitude < -30`
Greater than or equal	`>=`	any	logical	`Latitude >= -23`
Less than or equal	`<=`	any	logical	`Latitude <= -30`
Not equal	`!=`	logical	logical	`Latitude != null`
And	`&&`	logical	logical	see below
Or	`||`	logical	logical	see below

Examples showing And, Or operators

```
(Latitude == null) && (Longitude > 140)
(Latitude == null) || (nan(Latitude))
```

INTREPID Functions

INTREPID functions have the form

name(argument1, argument2, ...)

Where *name* is the name of the function and *argument1, ...* are expressions

Example

dist(X-75.4,Y-.341)

The data type of the value returned by most functions is numeric. Functions returning logical type data are marked '(Logical)'

Note:

To simplify language, this table uses vector dataset terminology. In most cases in the table:

- 'Data point' refers also to all bands of a grid cell.
- 'Field' refers also to one band of a grid.

Function	Description
Scalar functions	INTREPID calculates these functions based on the argument(s)
abs(A)	absolute value of the current input value
sqr(A)	Square of the argument
sqrt(A)	Square root of the argument
abs(A)	Absolute value of the argument
negate(A)	**-1 *** the argument
modn(A)	Modulo to base *n* of the argument
log(A)	Logarithm of the value of the argument to the base e
exp(A)	e^f, where f is the value of the argument
cos(A)	Cosine of the argument in degrees
acos(A)	Angle (in degrees) whose cosine is the argument
sin(A)	Sine of the argument in degrees
asin(A)	Angle (in degrees) whose sine is the argument
tan(A)	Tangent of the argument in degrees
atan(A)	Angle (in degrees) whose tangent is the argument
atan2(Y,X)	Angle (in degrees) whose tangent is the argument

Function	Description
lookup(A,dataset)	Result of using *dataset* as a lookup table and finding the value corresponding to **A**. *dataset* is a vector dataset with fields **X** and **Y** used as the table. INTREPID uses the value **A** in the **X** field and returns the corresponding value from the **Y** field, interpolating it between data points using a spline curve if necessary. *dataset* must be a string enclosed in double quotes **" "** containing the path and name of the dataset's **..DIR** marker file (e.g., **"/disk1/surv/alt_table"**. The path can also be relative to the current directory at the time the tool being used was launched. If you are typing the formula into a **.frm** or **.job** file using a text editor, you need to • Enclose the whole formula in in double quotes **" "** • Precede each double quote character **within** the formula with a backslash: **\"**. If you are entering it interactively in the formula editor in the Spreadsheet Editor, you don't require the backslashes **** or the double quotes to enclose the whole formaula. See i**nside(X,Y,dataset)** below for an example See Creating a dataset for the lookup( ) function for further details.
inside(X,Y,dataset) (Logical)	If the point **X,Y** is inside the polygon dataset **dataset**, then **inside** = 'true'else**inside** = 'false' **dataset** must be a string containing the path and name of the dataset's **..DIR** marker file (e.g., **"disk1/surv/poly356"**. The path can also be relative to the current directory at the time the tool being used was launched. **See the important note in lookup(A,dataset) above.** Example: **Formula =** **"IF inside (East,North,\"myfence_poly..DIR\") THEN LEVMAG_MASKED=Null ELSE LEVMAG_MASKED=Levmag"**
nan(A)(Logical)	If the argument is not a numberthen**nan** = 'true'else**nan** = 'false'
int(A) or **trunc(A)**	Truncates **A**, removing its fractional part
newVector(x,y,z)	A vector from its parts or components
rotatedX(X,Y,XO,YO,A)	Gives the X by projecting the point (X,Y) onto a rotated X axis with origin (X0,Y0) and angle A anti-clockwise wrt the original X axis

Function	Description
rotatedY(X,Y,XO,YO,A)	Gives the Y by projecting the point (X,Y) onto a rotated X axis with origin (X0,Y0) and angle A anti-clockwise wrt the original X axis
Exotic Field Type Functions	***Tensor, vector, observed, complex etc***
newComponents(x,y,z)	A field vector from its components
newGradients(dx,dy,dz)	A gradient vector from its 3 derivative components
newStrucural (dip,strike,unit)	A geological structural observation For example, to create a Structural object from existing dip,strike and geology description data: Edit - Replace - Then "struct == newStructural(dip,strike,geology)
newTensor(xx,xy,xz,yy,yz,zz)	Create a tensor field from its components
newTensor(Auv,Ane,NULL,NULL,Bne,Buv)	Create a Falcon tensor field from its components - note use of NULL, as this is the critical factor.
IPHTTensor(g1,g2,g3,g4,g5,g6)	Create a New IPHT magnetic tensor from the measured mixed gradient fields. Uses the raw gradients to form a tensor in the Body coord system. (V5.0 only)
Body2ENU_IPHT(bodyT, roll,pitch,yaw)	Convert airborne magnetic tensor data to World Coord reference frame Tensor, ENU (V5.0 onwards)
TXX(t), TYY(t), TXY(t), TZX(t), TYZ(t), TZZ(t)	Get a component from a tensor field
AUV(t) ,ANE(t), BUV(t) , BNE(t)	extract Auv, Ane,Buv,Bne tensor components from tensor field(Falcon)
VKa(t),VKc(t)	extract VKa instrument or Tzz-Tyy from tensor field extract VKc instrument or Tzz-Txx from tensor field
ENU2NED(t)	switch the tensor coordinate type from east north up to north east down
ENU2END(t)	switch the tensor coordinate type from east north up to east north down
END2NED(t),END2ENU(t)	more switching of coord refernce frames
NED2ENU(t),NED2END(t)	
ENU2FALCON(t),END2FALCON(t), NED2FALCON(t)	create a new Falcon tensor from an existing FTG tensor. Falcon style includes A & B, which are duplicates in this case. This degrades the measured components to those captured by Falcon.

Function	Description
getTerrainDensity(FreeAirTensor, TC_1_0, Number_Good, Threshold)	estimate the density of the regolith terrain using the observed free air tensor signal, the terrain correction tensor for density 1 g/cc, the required minimum correlated gradients (example 4 of 6 gradients) the threshold of the amplitude of the trace gradients (example 3 Eotvos) this also works for Falcon tensors
EV1(t), EV2(t), EV3(t)	Get principal components of a tensor
EVolume(t)	Get unit eigenvector volume for the tensor (should be +/- 1.0)
QREAL(t)	Get real part of the quaternion for the tensor
newObserved(mag,dx,dy,dz,az)	Create an observed field from a field magnitude and gradients components in a local coordinate system that has its North axes rotated “az” degrees to the right fom the global system
igrf_field (long,lat,height,date)	Create a field with the IGRF magntic field for position, uses the decimal year returned from the date type field
igrf_incl(long,lat,ht,date)	Create a field with the IGRF magntic field inclination for position, uses the decimal year returned from the date type field
igrf_decl(long,lat,ht,date)	Create a field with the IGRF magntic field declination for position, uses the decimal year returned from the date type field
make_gradients (Vector,dx,dy,dz)	Create a gradients vector from a vector component field, when the delta distances between readings is dx,dy,dz
yearmonthday(day_seq,year)	Creates a YYMMDD fom the day in the year and the year
velocity(x,y,time)	X & Y in meters, time in seconds, output converted to knots
julian(year,month,day)	creates the day number since 1985/1/1
decimalyear(year,month,day)	Creates a double with a decimal year
newDate(y,m,d,h,m,sec)	Creates a Date from all its possible parts
completedate(y,m,d,h,m,sec)	Creates a double with a decimal date
hms2decimal(HHMMSS.SSS) or **dms2decimal(DDDMMSS.SSS)**	Converts hours or degrees, minutes, seconds to fractional hours or degrees.

Function	Description
Whole dataset summary functions	INTREPID calculates these functions using all values of the input in the whole dataset.
min(A)	Minimum value of **A** in the whole dataset
max(A)	Maximum value of **A** in the whole dataset
mean(A)	Mean of **A** in the whole dataset
stddev(A)	Standard deviation of **A** in the whole dataset
sum(A)	Sum of data **A** in the whole dataset
normalise(A)	**(A – mean(A))/stddev(A)**
correlation(A,B)	The correlation between **A** and **B** over the whole dataset
diff(A)	See **diffn**(A) below
skew(A)	unbiased skew of A in the whole dataset
gaussian(x,A,P,sd)	add gaussian noise to signal . The actual Gaussian function used is : A = peak amplitude of noise = parameter 1. P = peak position = parameter 2. sd = standard deviation of peak = parameter 3. Gaussian = A*exp(-T*T/2.0), where T = (x-P)/sd
curveN(A)	Abromwitz Curvature function for gravity/magnetic gridded data N is an optional sample increment operator functions for Rows to skip. This is an order of magnitude higher than the usual operator used in Minimum Curvature gridding, thus is a good QC test of a potential field grid, created with an unknown provenance.
Current group summary functions	INTREPID calculates these functions using all values of the input in the current group.
number_samples(A)	Number of data points for field **A** in the current group
groupsum(A)	Sum of **A** in the current group
groupmin(A)	Minimum value of **A** in the current group
groupmax(A)	Maximum value of **A** in the current group
groupmean(A)	Mean of **A** in the current group
groupstddev(A)	Standard deviation of **A** in the current group
groupnormalise(A)	**(A – groupmean(A))/groupstddev(A)**
groupnonnull(A)	**number of non-null values in the current group**

Function	Description
add(A)	Sum of all values of **A** from the first data point in the current group to the current data point
area(X,Y)	Polygon specific function - The signed square area of a polygon where X and Y are geographic location fields. If +ve we traverse anticlockwise. If -ve we traverse clockwise.
groupflip()	Reverses the order of all data points in the current group. Particularly dangerous function in the hands of a novice - beware.
dist(X,Y)	Geographical distance from the first data point in the current group to the current data point (where **X** and **Y** are geographic location fields)
azimuth(X,Y)	The bearing in degrees of the current group where **X** and **Y** are geographic location fields. INTREPID calculates this value based on first and last data points in the group.
partsum(A,s,e)	sum of A in the current group from s(tart) percentage to e(nd) percentage.
partmin(A,s,e)	minimum value of A in the current group from s(tart) to e(nd) percentage.
partmax(A,s,e)	maximum value of A in the current group from s(tart) to e(nd) percentage.
partmean(A,s,e)	mean of A in the current group from s(tart) to e(nd) percentage.
partstddev(A,s,e)	standard deviation of A in the current group from s(tart) to e(nd) percentage.
lsqn(A)	INTREPID fits a least squares curve of order *n* to the values of **A** in the current group. The value of **lsqn** for each group is the value on the curve corresponding to the current row. For example, **lsq2()** will use a least squares curve of order 2.
Local comparison functions—using the surrounding *n* rows (data points)	These functions involve the adjacent values immediately before and after the current data points. To evaluate the function INTREPID performs a calculation on the set of consecutive values. The parameter *n* determines the number of adjacent data points involved in the calculation (including the current data point) For example to calculate **localmean5(A)** INTREPID finds the mean of **A** in the 5 adjacent data points surrounding and including the current data point.
localminn(A)	Minimum value of **A** for the *n* consecutive data points surrounding and including the current data point
localmaxn(A)	Maximum value of **A** for the *n* consecutive data points surrounding and including the current data point

Function	Description
localmeann(A)	Mean of A for the *n* consecutive data points surrounding and including the current data point
localstddevn(A)	stdandard deviation of A for the n consecutive data points surrounding and including the current data point
mediann(A)	Median value of **A** for the *n* consecutive data points surrounding and including the current data point
partmin(A,start,end)	Minimum value of **A** for portion of the current group where start is a number from 0 to 1 and end is also a number proportional from 0 to 1
partmax(A,start,end)	Maximum value of **A** for portion of the current group where start is a number from 0 to 1 and end is also a number proportional from 0 to 1
partmean(A,start,end)	Mean of A for portion of the current group where start is a number from 0 to 1 and end is also a number proportional from 0 to 1
partstddev(A,start,end)	Std. Dev. of A for portion of the current group where start is a number from 0 to 1 and end is also a number proportional from 0 to 1
diffn(A) **diff(A)**	*n*th successive difference between the value of **A** and those of **A** in sufficient previous rows to calculate the difference. If you omit *n*, then INTREPID calculates the first difference. See DIFF function example for an example.

Function	Description
Local comparison functions—finding contiguous blocks of data	The purpose of these functions is to determine the sizes of blocks of non-zero (or non-zero and non-***null***) values of **A**. For data points that are within such a block, the value of **contig** or **contigplus** is the number of data points in the block. See CONTIG and CONTIGPLUS function example for an illustration of **contig** and **contigplus**.
contig(A)	If argument = 0 or ***null*** then**contig** = 0else**contig** = the number of contiguous adjacent rows before, after and including the current row which contain input values <>***null*** and <>0
contigplus(A)	If argument = 0then**contig** = 0else**contig** = the number of contiguous adjacent rows before, after and including the current row which contain input values <>***null*** and <>0
Local comparison functions—interpolating missing values	
interplinear(A)	A linear interpolation between the nearest non-***null*** values on either side of **A**
interpspline(A)	An Akima spline interpolation between the nearest non-***null*** values on either side of **A**
fill_last(A)	If **A** is not null in the current row then **fill_last** = **A** Else **fill_last** = the last non-null value of **A** in previous rows
fill_next(A)	If **A** is not null in the current row then **fill_next** = **A** Else **fill_next** = the next non-null value of **A** in following rows
Row (data point) shifting functions—copying data to different rows (within the current group)	
relrown(A)	The value of **A** as it would be for the data *n* rows (data points) after the current row (data point)
relrowmn(A)	The value of **A** as it would be for the data *n* rows (data points) before the current row (data point)
Functions which do not use any row input values	INTREPID calculates the value of this function without referring to any input values. You must specify some parameter for ease of parsing, but INTREPID will ignore it.
random(A)	A random number between –1 and 1.

Function	Description
Band redistributing functions—changing the number of channels in multi-band data	These functions use statistical techniques ('re-binning') to convert multi-channel data from 512 or 1024 channels to 256, 258 or a user-specified number of channels
`to256(spectrum)`	Returns a 256-channel multiband field. **`spectrum`** is a multiband vector dataset field of 512 channels or more
`to258(spectrum)`	Returns a 258-channel multiband field, where 2 channels are for maximum and minimum energy and 256 are for signal. **`spectrum`** is a multiband vector dataset field of 512 channels or more
`bands (spectrum,start,stop,incr)`	Returns a multiband field that combines channels from the input field, **`spectrum`**. **`spectrum`** is a multiband vector dataset field of 512 channels or more. **`start`** is the first band of **`spectrum`** to be included in the output. **`stop`** is the last band to be included. **`incr`** is the number of input bands to be combined into a output band. See Band redistributing function example.
User defined functions	See The INTREPID SDK and API (R18) for full instructions and examples.
`@funct(field1,field2,...)`	An API function (only accepts fields or grid bands as arguments)
`funct(arg1,arg2,..)`	A DLL/SO function (accepts expressions as arguments)

CONTIG and CONTIGPLUS function example

This example shows

- 14 contiguous values that are non-zero and not ***null***
- 19 contiguous values that are non-zero

A	contig	contigplus
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
null	0	19
null	0	19
null	0	19
null	0	19
null	0	19
5	14	19
6	14	19
4	14	19
3	14	19
2	14	19
1	14	19
2	14	19
4	14	19
5	14	19
6	14	19
5	14	19
4	14	19
3	14	19
5	14	19
0	0	0
0	0	0
0	0	0

Band redistributing function example

Here is an example of a Spreadsheet Editor task specification file that creates a new 11-band field from an existing multi-band field.

```
Process Begin
  Name = dbedit
  Comments = "Intrepid Audit Stamp ... "
  Parameters Begin
    Action Begin
      Type = OpenField
      Name = "data/512D_SCE_063..DIR/SPC_DOWN"
    Action End
    Action Begin
      Type = CreateField
      Dtype = IEEE8ByteReal
      Width = 0
      Bands = 11
      GroupBy = No
      Initial = "bands(SPC_DOWN,1,21,2)"
      Name = "bands"
   Action End
  Parameters End
Process End
```

User defined functions

You can create your own functions for use with INTREPID.

There are two types of user defined function:

- API functions can directly access INTREPID datasets.
- DLL/SO functions process the data passed to them using arguments.

See The INTREPID SDK and API (R18) for full instructions and examples.

Formula files

INTREPID formula files are block structured text files containing part or all of an **`if - then - else`** specification for replacing data in a field or grid band in a dataset. Formula files have the extension **`.frm`**

You can load a formula file into an INTREPID tool for use in specifying a search or search and replace operation.

The following table sets out the format for a formula file.

Text	Description
`Formula Begin`	Block Header
`Directory = <text>`	Project directory path (optional)
`Name = <text>`	Formula title (optional)
`Description = <text>`	Formula description (optional)
`Formula = "`	Start of Formula statement
`IF <expression>`	IF section (optional)
`THEN <expression>`	THEN section (optional)
`ELSE <expression>`	ELSE section (optional)
`"`	End of formula statement
`Formula End`	End of Block

Notes:

- If you are using several lines for the Formula statement you must enclose them in **`{ }`** as shown in the examples below.
- **`<text>`** can be any text. Quotes denoting a string are not required.
- **`<expression>`** can be any INTREPID expression.
- When you load a formula file into an INTREPID dialog box that has If, Then and/or Else text boxes, INTREPID locates the corresponding sections of the formula and places them into their correct positions.

The three sections can exist without each other. It may be, for example, that a formula only has a Then section and you need to type in the If and Else sections yourself before using the dialog box.

Formula file examples

The following example shows a formula file fully annotated by a user.

```
Formula Begin
   Directory = /disk1/survey
   Name = Line type
   Description = Sets the line type field
   Formula = { "IF line_number >= 7000
    THEN line_type = 4
    ELSE line_type = 2" }
Formula End
```

The following example shows the comments and default description field values when you save the formula from INTREPID.

```
#### Audit Stamp Audit Stamp V3.2 - 10/12/1996 14:7:18
Formula Begin
   Comments = "Audit Stamp V3.2 - 4/12/1996"
   Directory = dbedit
   Name = dbedit
   Description = dbedit
   Formula = { "IF raw_mag > 1979.5
    THEN raw_mag = raw_mag * 0.998
    ELSE raw_mag = raw_mag" }
Formula End
```

The following example shows a simple formula file with the bare minimum of text.

```
Formula Begin
   Formula = " IF line_type <>2 "
Formula End
```

DIFF function example

Here is an example of the successive difference function. The column headed A represents the data whose differences the function is calculating. The columns 1st, 2nd, 3rd show the successive first, second and third difference calculations (**`diff1`**, **`diff2`**, **`diff3`**) for the data.

Data point	A	1st	2nd	3rd	In this example, For the data point 3 value (5), diff1 = –2, `diff2` = –5, and `diff3` is not shown. For the data point 4 value (8), `diff1` = 3, `diff2` = 5, and `diff3` = 10
1	4				
		3			
2	7		–5		
		–2		10	
3	5		5		
		3			
4	8				

Creating a dataset for the lookup( ) function

The **lookup(value**,*dataset*) function uses a vector dataset as a lookup table and finds or interpolates a value corresponding to the *value* argument. INTREPID uses the **X** and **Y** fields of the dataset as the table.

INTREPID finds or interpolates *value* in the **X** field and returns the corresponding value from the **Y** field, interpolating it between data points using a spline curve if necessary.

If you have the data for lookup table in an existing delimited text file, you can import it to a point dataset using a simple DDF file such as the following

```
TYPE(POINT)
     X REAL*4 IsX
     Y REAL*4 IsY
```

The example assumes space delimited data and could be used to import data such as the following

```
0     .0523
1.2  1.246
3.5  3.61324
9    8.285
15  14.543
```

See "Importing ASCII Columns data" in Importing to INTREPID datasets (T05) and The INTREPID DDF format (R08) for detailed instructions.