New Site Programming Libraries and Tools*

James Darrell McCauley
`mccauley@ecn.purdue.edu`

Bernard A. Engel
`engelb@ecn.purdue.edu`

Department of Agricultural Engineering
Purdue University

Abstract

This paper discusses a proposed format for site records in the GRASS geographic information system. New library functions for sites input/output support data in greater than 2 dimensional space. They also provide standard parsing routines for category, decimal, and string attributes by prefixing each with a unique identifier. Examples outline the use of these functions. Other site programming efforts are summarized.
Audience: *The primary audience of this paper is C programmers. However, since sites lists are in an easily-accessible ASCII format, non-programmers who create or modify data should be aware of the proposed changes.*

1 Introduction

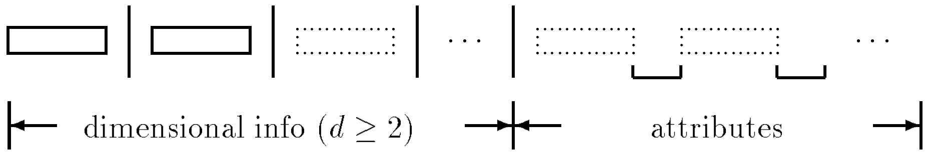
A few researchers in the GRASS community have been actively engaged in providing new programs that operate on site lists (punctual data). The current structure of a site record consists of an easting and northing plus an optional free-form label string. The lack of a standard structure of the label string and related library support has caused many problems. For example, one program may require a string comprised of an integer category value and floating-point value, while another may require only a floating-point value. Also, support for 3D and higher dimension modeling was not present and *ad hoc* extensions were necessary for some researchers. (Some examples of site programs under development are given in the Appendix.) Therefore, some site lists were valid for particular programs and invalid for others.

After discussion among programmers and users on the *GRASSHopper* electronic mailing lists, a new format was proposed that allowed for data in higher dimensions and multiple attributes with a descriptive tag differentiating integer category values, decimal numbers, and strings. Programming libraries implementing this proposed format are described along with examples.

*Originally presented at the 9th Annual GRASS GIS Users' Conference and Exhibition, 13–18 March 1994, Reston, Virginia. Revised 07 April 1994 to reflect library changes to support "quoted" strings.

2 The Format

The following format, originally proposed by Dave Gerdes (USA-CERL), emerged from the discussion:



Optional items are denoted by dotted-boxes.

The first part of the record is dimensional information describing location. The second part of the record are attributes associated with the particular location. Each of these groups of fields are described in more detail below.

2.1 Dimensional Fields

Dimensional fields (space and time) are pipe-suffixed (`|`) fields at the beginning of the record. At least two dimensions must be present, but three or more are possible. The first two must be valid eastings and northings. Recall that for latitude–longitude data, this form is:

`dd:mm:ss.ffh`

where: `dd` is degrees, 0–90 for latitude, 0–180 for longitude
`mm` is minutes, 0–59
`ss` is seconds, 0–59
`ff` is fractions of a second, ≥ 0
`h` is `n` or `s` for latitude, `e` or `w` for longitude.

The minimum amount of data constituting a valid site are two items in this portion of the record, i.e., a northing and easting. Additional dimensions, if present, must be decimal numbers.

2.2 Attribute Fields

Attribute fields follow the last pipe symbol of the record. There are three types of attribute fields defined in the new format: category, decimal, and string. Each of these are described below.

A pound sign (`#`) must precede an integer[1] field if it is to be interpreted as a GRASS category value. The field is optional.

Optional decimal attributes are preceded by a percent sign (`%`). Multiple attribute strings are allowed. These may be integers or floating-point numbers, but they are stored internally as double precision floating-point numbers.

Optional string fields are preceded by an "at sign" (`@`). Multiple string fields are allowed, but a maximum string limit of 128 characters for each field is enforced. For backward

[1]This is actually of type `CELL` rather than `int`.

compatibility, fields in the attribute portion of the record default to string attributes if none of the attribute identifiers (#, @, or %) are present. If a string contains a "space" ( ) as part of the field, it should contain double quotes (") at the beginning and end. If a string contains quotes (other than those at the beginning and end of a the field), these should be *escaped* by preceding them with a backslash (\).

For example, to store soil pH and soil type, the attribute fields of a record may look like: `#56 @"clay soil" %5.6`. The category value might be used to convert the site to a raster cell.

The ordering or arrangement of different types of attribute fields is unimportant except for the category field. If present, it must be listed first. Each attribute field is space-separated.

Currently, the maximum length for the entire record (both dimensional and attribute fields) is 1024 characters. The maximum length for individual string fields is 128 characters. Symbols `MAX_SITE_LEN` and `MAX_SITE_STRING`, respectively, are defined for programmer convenience in the event that these should be increased.

3 Implementation

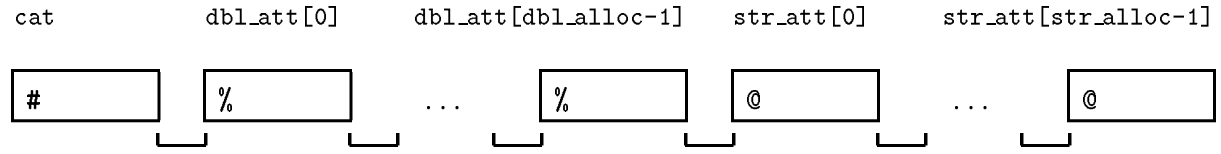
Code supporting this new format has been successfully implemented and tested. The data structure and associated library functions are described below.

3.1 Data Structure

```
typedef struct
{
  double east, north;
  double *dim;
  int dim_alloc;
  CELL cat;
  int str_alloc;
  char **str_att;
  int dbl_alloc;
  double *dbl_att;
} Site;
```

A standard site data structure was developed (left). In addition to storage for the two mandatory fields and the optional category value, pointers are included for the other types of optional fields (extra dimensions, strings, and decimal numbers). Storage for three extra integers are reserved to "remember" the amount of space allocated for these optional fields.

Recalling the discussion in §2.2, the following figure shows how tagged attribute fields are stored in this structure.



3.2 Library Functions

Library functions utilizing this new structure are presented below.

```
Site *G_new_site_struct (n_dim, n_s_att, n_d_att)
  int n_dim;                    allocate memory for a site structure with
  int n_s_att;                  specified no. of dimensions (≥ 2), string
  int n_d_att;                  attributes, and floating-point attributes.
```

Return value is a pointer to a new site structure or `NULL` *on error.*

```
G_new_get_site (ptr, s)         Reads site record from a file,
  FILE *ptr;                    parses, and fills site structure.
  Site *s;
```

Return value is zero (`0`*) on success,* `EOF` *at end-of-file, or* `-2` *on error.*

```
G_new_put_site (ptr, s, has_cat) Write site structure to
  FILE *ptr;                     output file in the new format.
  Sites s;                       has_cat indicates whether or
  int has_cat;                   not a category value is written.
```

Note that the function for memory allocation, `G_new_site_struct()`, works very much like the `Vect_new_line_struct()` function already present in GRASS [1].

4 Examples

To clarify how this proposed data structure and library functions operate, two examples are presented.

4.1 Temporal Data Support

While the new site format does not explicitly support *time* as a dimension, this kind of information can be easily dealt with. The following example shows how three spatial dimensions, a floating-point "concentration," and time (five string fields) are represented in the proposed format:

```
10.8|0|9.8|Fri Sep 13 10:00:00 1986 %31.4
11|5.5|9.9|Fri Sep 14 00:20:00 1985 %36.4
5.1|3.9|10|Fri Sep 15 00:00:30 1984 %28.4
```

Recall that the default identifier tag is `@` (string) when none are specified. Therefore, the site structure `mysite` would be initialized for 3 dimensions, 5 string attributes, and 1 decimal attribute:

```
mysite = G_new_site_struct (3, 5, 1);
```

The remaining code fragment and program output shows how straightforward handling time would be.

```
while (G_new_get_site (fp, mysite) == 0)
{
  sprintf(timestring,"%s %s %s %s\n", mysite->str_att[0],
          mysite->str_att[1], mysite->str_att[2], mysite->str_att[3],
          mysite->str_att[4]);
  if (partime(timestring,&timestruct,&zone))
    printf("At %ld seconds the concentration was %g at (%g,%g)\n",
            mktime(&timestruct), mysite->dbl_att[0],
            mysite->east, mysite->north);

  G_new_put_site(stdout,mysite,0);
}
```

Sites are read from a file until `EOF` or an error occurs. "Real" code should always check return values. The `partime()` function, taken from `rcs` [3], converts a string representation of time to an ANSI C time structure (`struct tm`). The function `mktime()` [2] converts the `tm` structure into the number of seconds since 00:00:00 Coordinated Universal Time (UTC), January 1, 1970. The output of the above code included in a simple program is:

```
At 526989600 seconds the concentration was 31.4 at (10.8,0)
10.8|0|%31.4 @Fri @Sep @13 @10:00:00 @1986
At 495523200 seconds the concentration was 36.4 at (11,5.5)
11|5.5|%36.4 @Fri @Sep @14 @00:20:00 @1985
At 464072430 seconds the concentration was 28.4 at (5.1,3.9)
5.1|3.9|%28.4 @Fri @Sep @15 @00:00:30 @1984
```

Note that the functions `partime()` and `mktime()` are not part of the library that was developed.

4.2 Latitude–Longitude to Raster

The following example converts longitude, latitude, and the category attribute in:

```
33.55083333|45.25138889|#1 @a %0.1
30.84722222|45.00002778|#2 "a b" %0.2
33.5|44.5|#3 @"a \"b c\"" %0.3
33|44|#4 \d %0.4
```

to a GRASS raster map. Each record is also printed to standard output.

```
mysite = G_new_site_struct (2, 2, 1);
while ((code = G_new_get_site (fp, mysite)) == 0)
{
  row = (int) G_northing_to_row (mysite->north, window);
  col = (int) G_easting_to_col (mysite->east, window);
  G_put_map_row_random (fd, &(mysite->cat), row, col, 1);
  G_new_put_site(stdout,mysite,1);
}
```

Notice that even though the extra string and decimal attributes are not required for the conversion to a raster map, memory must be allocated for reading these by `G_new_site_struct()`. If memory were not allocated, the return code from `G_new_get_site()` after the first read would have been `-2`, not `0`.

To further explain how quoted strings are interpreted, the string attribute in the second record was interpreted as `a b`. The string attribute for the third record was interpreted and stored as `a "b c"`, removing the backslashes from the quotes. For the string attribute in the fourth record, the backslash was kept.

The output from `G_new_put_site()` for this example is a properly formatted lat-long record:

```
33:33:02.999988E|45:15:05.000004N|#1 %0.1 @a
30:50:49.999992E|45:00:00.100008N|#2 %0.2 @"a b"
33:30E|44:30N|#3 %0.3 @"a \"b c\""
33E|44N|#4 %0.4 @\d
```

`G_new_get_site()` can read coordinates in this format as well. Notice that quotes are added to string attributes containing a space and that quotes contained in the actual attribute (e.g., the third record) are escaped.

5 Discussion

The most significant changes in the format of sites proposed in this paper are the addition of optional fields describing location in higher dimension space and the addition of attribute identifiers (`#`, `@`, or `%`). Support for higher dimension space keeps location and attribute information separate, which is fundamental to good GIS design. Attribute identifiers and parsing routines relieve the programmer from guessing how GRASS users will interpret different fields of sites records. Each of these were necessary to foster development of new site utilities (see Appendix).

When attribute identifiers are not included, all attributes are interpreted as string fields. In the existing format, everything past the easting and northing is simply read as one string. Backward compatibility is achieved by concatenating fields into one larger string through the use of `G_strcat()` and possibly `sprintf()`.

The proposed format is not backward compatible with records with the seldom-used `point|` prefix (a string preceding the easting).

Currently, there are only four sites programs in the main distribution of GRASS (version 4.1). The changes required to utilize this new format and library are minimal and would take a relatively short amount of time to complete. As shown in the Appendix, the number of site programs in GRASS may multiply many times in the very near future. Therefore, if the format of sites lists is standardized and changes are made to library functions, standardization should take place immediately.

6 Future Directions

The first priority will be to merge this library into existing GRASS code, but where do we go from here? A few possible enhancements include:

- the addition of a format description to the top of site lists (similar to the `name|` and `desc|` records) and
- the addition of other types of attributes. One suggestion has been a special string attribute acting as a pointer to a file.

Many such enhancements like this may be possible and would be much easier to implement as soon as site program begin using a standard library to parse attributes.

7 Acknowledgments

The authors are grateful for feedback on this issue from the *GRASSHopper* electronic mailing lists, including contributions from Simon Cox, Helena Mitàšová, Glenn Kroeger, Raymond Venneker, Nalneesh Gaur, Malcolm Williamson, David Gerdes, Mark Line, Jaro Hofierka, Nick Reddingius, Bill Hargrove, Mike Camann, Chris Skelly, and Philip Verhagen. The authors are also grateful to those who communicated their interests and concerns in private email.

References

[1] Michael Shapiro, Jim Westervelt, Dave Gerdes, Marjorie Larson, and Kenneth R. Brownfield. *GRASS 4.1 Programmers Manual.* U.S. Army Corps of Engineers, Construction Engineering Research Laboratories, Champaign, Illinois, 1993.

[2] Sun Microsystems, Sunnyvale, CA. *SunOS Reference Manual: C Library Functions*, SunOS 5.3 edition, 22 Jan 1993.

[3] Walter F. Tichy. RCS–a system for version control. *Software–Practice & Experience*, 15(7):637–654, July 1985.

Appendix

The examples presented in §4 show how easily sites programs can be adapted to use this new format. Currently, in version 4.1, there are only four sites programs in the main distribution of GRASS: `s.in.ascii`, `s.menu`, `s.out.ascii`, and `s.surf.idw`. However, many more are being developed for future versions. These are outlined below:

Sites Programs Under Development by McCauley[2]

Program	Description	Status
`s.bissf`	Akima's interpolation method (TOMS[3] 526)	80%
`s.c2grid`	Pruesser's interpolation method (TOMS 684)	50%
`s.delaunay`	Delaunay triangulation	100%
`s.kcv`	partitions sites for k-fold validation	100%
`s.masub`	interpolation (TOMS 677)	75%
`s.medp`	median polish for sites lists	90%
`s.nngrid`	natural neighbor interpolation	10%
`s.normal`	test for univariate normality	10%
`s.perturb`	random perturbation of site locations	95%
`s.qcount`	indices for quadrat count data	100%
`s.rand`	generate spatially random sites	100%
`s.sample`	image sampling at site locations	100%
`s.semivar`	exploratory analysis & semivariogram modeling	80%
`s.univar`	simple univariate stats	100%
`s.uvkrig`	universal kriging	50%
`s.voronoi`	Voronoi diagrams	90%
`s.windavg`	window average filter for sites	90%

Sites Programs Under Development by Others

Program	Description	Author
`d.what.site`	interactive query	Raghavan Srinivasan
`d.site.edit`	interactive update	Raghavan Srinivasan
`s.surf.krig`	ordinary kriging	Kym Lenkus
`s.filter`	hi/lo pass filtering (user sets min/max)	Dave Johnson
`s.in.table`	convert a table of specially formatted site information into one or more site lists	Dave Johnson
`s.to.labels`	creates annotation for a site list by creating paint labels file	Dave Johnson
`s.gnis`	create sites summarizing USGS Geographic Names Information System data base at quad, county, or UTM zone level	Tom D'Avello
`s.surf.tps`	thin-plate spline interpolation	Helena Mitàšová, Irina Kosinovsky, & Dave Gerdes
`s.db.rim`	RIM data base management/query interface	James Hinthorne & David Satnik
`s.to.rast`	convert sites list to raster file	Katarina Johnsson & Tom Brauer
`s.to.vect`	convert sites list to vector file	R. L. Glenn
`s.reclass`	reclassifies categories in sites list	Sue Huse

Additionally, Sue Huse may be linking an external ASCII database. There are undoubtably others that have not have been brought to our attention.

[2] Many of these are available for evaluation from `pasture.ecn.purdue.edu:pub/mccauley/grass`

[3] Association for Computing Machinery *Transactions on Mathematical Software*