Specifications of the geo-VLBI data format (GVF)

That what is not evolving doesn't live.
That what doesn't live dies.

Contents:

Introduction

Geodetic VLBI data go through several phases of transformations during their analysis:

1. Firstly, signal is recorded on tapes (discs). Data acquisition terminal writes the signal in its internal format: one of Mark-5, Mark-4, Mark-3, K-3, K-4 or S2 formats. Let's call it "signal format".
2. Correlator writes the output in another format.
3. Programs of fringe fitting like PIMA, AIPS, or FOURFIT add record to the correlator output. Let's call it "post-correlator format".
4. Geodetic VLBI data analysis software transforms the post-correlator output to another format. Let's call it "observations format".
5. Results of analysis are written in different formats. Let's call them "results formats".

The subject of these specifications is "observations format".

In the past Mark-3/Mark-4 VLBI data analysis software used so-called MARK-3 DBH format (or database-format) which kept information about VLBI session. Data could be transformed from MARK-3 DBH format to a NGS-format or a SUP (or superfile) format but with substantial loss of information which might be not essential for a simplified analysis.

MARK-3 DBH format was developed in middle 70-s. It has a substantial deficiencies. New format, called gvf (geo-VLBI format) suprseded the old format.

Why not old MARK-3 DBH format?

VLBI Data handler MARK-3 DBH was written in 1976. It is difficult to find any other program in the world which is intensively used for a quarter of century. However, several generations of computers changed since then; approaches to developing software also changed and MARK-3 DBH database handler is not adequate now. It has deficiencies which become more and more considerable obstacles to the proper use of VLBI data.

A. MARK-3 DBH format is not documented. Of course, it is not impossible to decipher the format, to reveal its internal structure and to compile its description but it is a tedious work. As a result we are not able to disseminate VLBI data for a scientific community. We can put and we are putting MARK-3 DBH files in Web, but only SOLVE users are able to use them.

B. MARK-3 DBH handler is terribly, **terribly**, **TERRIBLY** slow! Probably, it is the world most inefficient database handler. Each database handler has overheads which can be expressed as a ratio of physical time needed for reading the database to an amount of physical time needed to read the same amount of bytes. Usually this ratio is expressed in percents. MARK-3 DBH has this ratio about 100! I have a database which was loaded for 40 minutes.

   These overheads are so tremendous what makes it impossible to use MARK-3 DBH handler routinely. As a result, SOLVE uses another format called "superfile" for only one reason: to reduce overheads. It makes the process of data analysis rather more difficult and poses additional heavy obstacles to further development of analysis software. The first operation which non-SOLVE VLBI analysis software does is to convert data from MARK-3 DBH to another usable format.

C. MARK-3 DBH format has huge redundancy. The same item is stored many times. Redundancy is moderate for 2-, 3- stations experiments, but grows rapidly with increasing the number of stations. For example, if the session has 10 stations and therefore 45 baselines, time tag of the scan is stored 45 times, air pressure of the station A is stored 9 times and so on. Database size grows **quadratically** with increasing the number of stations instead of **linear** grow. A 17-baselines, 30000 observations pair of X- and S-band databases after compression has size more than 150Mb. To transfer such a huge amount of data through network is a challenge. Since the number of scans is increased in Mark-4 epoch we have to acknowledge that **we will lose capacity to exchange databases electronically in Mark-4 epoch unless we change database format**. We already reached this limit **today**. Disk space needed for analysis of entire data set (about 3000 24-hours experiments for 1979-1999) is 40-50 Gb. Not every analysis center is able to afford such resources. Redundancy (unnecessary!) of MARK-3 DBH format puts another strong restriction on data analysis.

D. MARK-3 DBH format treats X-band and S-band data as different experiments and the S-band data are treated as the data of the second sort. As a result of this band discrimination some S-band databases were lost. Remarkably, X-band and S-band databases contain at least 80% the same information.

E. MARK-3 DBH handler is not portable. It is a torture to install the handler at the native HP machine even for SOLVE developers. We cannot say to a colleague: get the source code, install it and read a database.

F. MARK-3 DBH is tied with another complicated and weird program called SOLVE catalogue. We are not able to separate SOLVE catalogue and MARK3 database handler. As a result we cannot use VLBI databases without entire sophisticated SOLVE infrastructure.

G. MARK-3 DBH handler makes database update difficult. It rewrites entire database of the experiment (which may have size up to 200Mb) regardless amount of updated data.

H. MARK-3 DBH handler makes it difficult to maintain multiple updates of the same experiment. We are able to save an update only of the last, the k-th version, but not the previous k-1, k-2 -th versions. It makes independent work of two analysts difficult. This puts an obstacle even to a work of one analyst when the different approaches of data analysis applied to the same experiment are compared.

I. At last, MARK-3 DBH catalogue system is (was?) not Y2K compliant (who thought about it in 1976?!).

I should notice that many of these setbacks stem from severe memory restrictions which we had in 70-s. MARK-3 DBH handler should have operated at the machines with 20Kb available memory and it did! A heavy price has been paid for it. Times changed. Now we can take 1Gb of operative memory with impunity and restrictions of MARK-3 DBH format can be and **must be** overcome.

Alternative format

Geo VLBI Format

What was done as an alternative

1. Specifications and documentation of a new format of geodetic VLBI observations were developed. The new format is called gvf (geo-VLBI format).

2. A set of subroutines which implement a gvf-handler has been developed.

3. a program gvf_read which reads a database in gvf format, extracts the specified item and writes it down in standard output has been developed

4. To develop a program gvf_transform which converts data
    - from binary gvf format to ---> ascii gvf format;
    - from ascii gvf format to ---> binary gvf format;
    - from MARK-III DBH format to ---> binary gvf format;

What did it bring?

1. VLBI data are open for scientific community. It is not enough to put the data in an ftp server. We should provide easy tools for using them.

2. VLBI databases became much shorter and transferable through slow networks.

3. VLBI handler became 100 times(!) faster. The objective: to read a database with 10 000 observations for less than 1 second has been met.

4. There is no necessity to support alternative VLBI data formats like superfiles and NGS-card formats.

5. Capacity to maintain simultaneously different database updates has bern provided.

Specifications of gvf format (geo-VLBI format)

Overview of general approaches

1. **Optimization criteria:** high speed (weight: 70%), small size (weight: 30%).

2. **Data representation:** binary. Binary data format provides the highest efficiency but is somehow obscure for non-professionals. As a compromise between the data transparency and the efficiency ascii version of gvf format is proposed. Ascii gvf-format and binary gvf format will be mutually convertible. It means that a user will have a capacity to convert a binary gvf-file to an ascii gvf-file, to look at it, even to edit (but it is not recommended) and then to convert it back to a binary format. Gvf-handler will be optimized for using only binary version. Some users has an inexplicable allergic to binary files. A capacity to keep the data in ascii format and convert then to binary on the fly can be provided although it is assumed that the binary version of the format will be normally used.

3. **Data files**. A database for an experiment may be split onto several files called extents (although it is not mandatory). One of the possible ways to split is:

```
-------------    -----------    -----------    -----------    -----------
| Fringe-1 |     | Calib-1 |    | Theo-1  |    | Solve-1 |    | Docs-1  |
-------------    -----------    -----------    -----------    -----------

-------------    -----------    -----------    -----------
| Fringe-2 |     | Calib-2 |    | Theo-2  |    | Solve-2 |
-------------    -----------    -----------    -----------
```

   Different files contain different type of information:
   1. **Fringe** is produced by a post-correlator software (namely, FOURFIT) and contains root information. This file is mandatory.
   2. **Calib** contains additional calibration, like system temperature, cable calibration, meteorological information and so on.
   3. **Theo** contains calc-supplied information: theoretical delays, partial derivatives, intermediary quantiles, like nutation angles.
   4. **Solve** contains solution supplied information: solution setup, ambiguities, suppression flags and so on.
   5. **Doc** contains arbitrary textual information. I proposed to put there verbose description of items kept in the database.

   More than one extent of the same type can be used. It is proposed to split the extents of the same type onto extents with frequently used information and with rarely used information. It is debatable which items should be treated as "frequently used". At the beginning we can consider information to be put in superfiles now as "frequently used". It assumed that a list of the files related to the experiment is passed to the gvf-handler.

   Advantages of this scheme:

   I. We can load not all information. F.e., phase cal amplitudes are not used in routine data analysis, why to load it? We can save a lot of resources.
   II. A user may wish to get not all extents from data centers. F.e. CALC supplied information is a garbage for non-SOLVE users; SOLVE users may wish to re-CALC foreign databases, then why to browse "Theo" extents?
   III. Only the targeted section will be updated. If we made a new solution we update only relevant

extents.

IV. User-supplied information can be written in another section, f.e. Calib-3, Calib-4. Moreover, it can override items in another sections when it is desired.
V. A user "N" may have a personal copy of the extent and his/her manipulations will not affect work of a user "M" which can use either his/her own copy or a system-wide version.
VI. Files can have different protection. F.e., it is assumed that nobody will update Fringe-1 and Calib-1 extents.

4. **Data structure**. Any valid gvf-file has preamble, history, table of lcodes (or by other words items) and the data themself similar to MARK-III DBH. But the data are ordered in according with an observation basis in MARK-III DBH format: first all lcodes for the k-th observation, then all lcodes for the k+1 -th observation and so on. The data are ordered by lcodes in gvf-format: first data for all observations for the lcode j, then data for all observations for lcode j+1 and so on. It allows to exclude redundancy. The data are structured by such a manner to provide mapping the file to an address space with minimal transformation.

5. **File names**. The following file naming scheme is proposed: filename consists of 6 fields of lower case letters or digits separated by an underscore and dot like

```
europe55_b2_b03_cal1.gvf

neos-a784_w1_g08_sol2.agvf
```

Fields:

- Experiment identifier. It is granted by IVS and is used for schedules, correlation and further analysis.
- Delimiter: underscore.
- One-symbol correlation center code which has fringed the data. For example, b for GIUB, w for USNO and so on.
- One-digit fringing version. Database with different fringing version are considered as different experiments.
- Delimiter: underscore.
- One-symbol analysis center code which has created this file. For example, g for Goddard, i for IRACNR and so on.
- Two-digit version number of the file.
- Delimiter: underscore.
- Four-symbol extent code.
- Delimiter: dot
- Extension: gvf for binary files and agvf for ascii files.

6. **Data classes**. All data are considered as of a) session class, b) scan class, c) station class and d) baseline class. It allows to exclude redundancy. X-band and S-band data of the same experiment are put together. More than 2 bands and more than one recording mode (f.e. polarization) is supported.

7. **Data handler**.
   - The data handler is considered as independent on a VLBI catalogue system. It gets input files which can be obtained either by the previous calls to a catalogue system or by another way, f.e. are supplied by user.

   - The data handler first reads all data in memory, sets internal caching tables and then handles individual queries.

- The data handler consists of
    i. **Input/output segment.** It manipulates with records, opening, reading/writing and closing files, memory management. All potential system dependencies are gathered there.
    ii. **Query segment.** It contains a collection of subroutines which provide a user interface.
    iii. **Compatibility mode segment.** It contains a set of routines with the same names as routines of MARK-III DBH handler which in turn calls procedures of the query segment.

- **Catalogue system** is not a part of the handler but it provides an additional service and it is assumes that normal access to the gvf-files is done through the catalogue system, although in principle it is possible to pass it by. The catalogue system serves the following functions:

I. **Filename control.** Several files with long names are related to one experiment. User may want to use short aliases like $99JUN14XH or EUROPE50. Catalogue system maintains such aliases and prevents using duplicate file names.
II. **Backup**. Functions like maintaining tape archives, compressed archives of the files related to the experiment, data import and export.
III. **Information service.** The catalogue system reads databases, extracts some information about configuration of the experiment, saves it in internal data structures, regularly updates then and serves queries. It fulfills simple queries like to print the list of experiment names within the specified time range as well as more complicated queries like to create the list of all sessions with more than 10 good observations at the baseline NRAO20/NRAO85~3 and is able to provide statistics similar to that the program SUMRY generates.

Detailed specifications of binary gvf format

1. **Physical structure**
   A valid binary gvf file consists of several sections of variable length: a) preamble; b) text section; c) table of contents; d) binary data section in this order. Text section or "table of contents" and "binary data" sections may be omitted. Each section is aligned in order to start from the 256-byte page boundary. Unused space is always filled by zeroes.

2. **Section format:**

```
.--------.--------.------.--------.-------------.
| Length | Prefix | Body | Filler | Control sum |
.--------.--------.------.--------.-------------.
```

   where

Length	INTEGER*4	Length in bytes of the entire section, including body, length, prefix, control sum and filler.
Prefix	CHARACTER*4	Section identifier. One of PREA, TEXT, CONT, DATA
Body	Undefined	Section content. Treated as a byte stream.
Filler	Undefined	Unused space. May have zero length. Always binary zeroes.
Control sum	INTEGER*4	Control sum.

3. **Format of PREA** section:
   The preamble section consists of an ordered sequence of logical records of variable length. The sequence is terminated by a section terminator: CNTRL/Z (decimal code 26). Firstly, mandatory records follow, then optional records may follow.

Format of a **logical record** in preamble section:

Identifier	Sequence of letters (without delimiters!). Identifier should be unique.
Id_delimiter	two bytes: semi-column, blank (decimal codes: 58, 32)
Body	Sequence of ascii symbols in the range 32-127 decimal
Record terminator	CNTRL/J (decimal code: 10)

All mandatory records are written by handler automatically.

The list of mandatory records:

`File_format:`	Identifier of the format and its version.
`Generator:`	Name of the program to have generated the file and its version.
`Creation_UTC_date:`	Creation date.
`Binary_format:`	Identifier of binary numbers format. It is assumed that normally IEEE standard will be used, but why not to reserve the capacity to use alternative format?
`File_name:`	File name
`File_version:`	Version number of the file. Versions are numbered starting with 1 up to 99.
`Previous_filename_{ver}:`	File name of the version {ver}, where {ver} is a version number which is less than the current version number. If the current version number of greater than 2 then several Previous_filename_{ver}: records should present. For example, if the current version is 4 then records Previous_filename_03:, Previous_filename_02:, Previous_filename_01: should present.
`Experiment_id:`	Experiment identifier in lower register letters.
`Experiment_start_date:`	Start date in the format: YYYY.MM.DD , for example, 1999.08.06 The first observation which has been correlated considered as a start observation.
`Experiment_start_doy:`	Day of year of the first observation of the experiment.
`Experiment_start_UTC_time:`	Start time in the format HH:MM:SS.FFF, f.e. 08:00:52.028
`Duration:`	duration of the experiment in seconds.
`Correlation_center_code:`	One-symbol code of the correlation center.
`Correlation_center_name:`	Full name of the correlation center.
`Analysis_center_code:`	One-symbol code of the center where the file has been created.
`Analysis_center_name:`	Full analysis center name.
`analyst_name:`	Analyst name as it registered in the operating system.
`Analyst_e-mail_address:`	E-mail address of the person who created a database and to whom to send questions as a last resort. The name is generated by operating system.
`Hardware_name:`	Name and type of the computer used for creation of the file.
`OS_name:`	Name and version number of the operating system.

Records with arbitrary identifiers can be added after mandatory records.

4. **Format of TEXT** section:
The TEXT section consists of an ordered sequence of subsections. Each subsection is terminated by CNTRL/Z (decimal code: 26). Each subsection consists of a prefix, title, title delimiter, body and subsection terminator:

`Prefix:`	Sequence of 7 symbols: "Title: "
`Title:`	Ascii text with any symbols, except CNTRL/J and CNTRL/Z (decimal codes 10 and 26).
`Title delimiter:`	CNTRL/J (decimal codes 10).
`Body:`	Any symbols, except CNTRL/Z (decimal 26).
`Subsection terminator:`	CNTRL/Z (decimal 26)

Text section is for history records and for any textual information.

5. **Format of CONT** -- contents section:
The CONT section consists of a sequence of records of fixed length. It describes contents of binary data section:

`Lcode`	CHARACTER*8	Identifier of the item in the database.
`OFFSET`	INTEGER*4	Offset in bytes of the first byte of the first occurrence of this lcode with respect to the beginning of DATA section.
`DIM1`	INTEGER*2	First dimension of the array of values for the lcode. Lcode is considered as a three-dimensional array.
`DIM2`	INTEGER*2	Second dimension of the array of values for the lcode. Lcode is considered as a three-dimensional array.
`DIM3`	INTEGER*2	Third dimension of the array of values for the lcode. Lcode is considered as a three-dimensional array.
`Type`	Byte*1	Data type code. Supported data types: CHARACTER, BYTE*1, INTEGER*2, INTEGER*4, REAL*4, REAL*8.
`Class`	Byte*1	Code of the data class. Supported data classes: Session, Scan, Station, Baseline.
`Usage code`	Byte*1	Lcode usage code. Supported usage codes are: Primitive, Synonym, Derived. Usage code describes which additional actions are needed besides reading/writing.

6. **Format of DATA** -- section:
The DATA section consists of an ordered sequence of logical records of variable length. Each logical record corresponds to a LCODE and contains an ordered sequence of frames. Each frame contains an array of values of the lcode which corresponds to one observation. All frames have fixed length within one logical record.

Structure of a DATA-section:

```
DATA-section:
                  ~~~~~~~~~~~~~
     Record k     | LCODE-k   |
                  ~~~~~~~~~~~~~

                  ~~~~~~~~~~~~~
     Record k+1   | LCODE-k+1 |
                  ~~~~~~~~~~~~~
```

```
                    ~~~~~~~~~~~~~
        Record k+2    | LCODE-k+2   |
                    ~~~~~~~~~~~~~
        ...
```

Structure of a data record:

```
record-k:
        ~~~~~~~~~~~~~~~
        |  Frame j     |
        ~~~~~~~~~~~~~~~

        ~~~~~~~~~~~~~~~
        |  Frame j+1   |
        ~~~~~~~~~~~~~~~

        ~~~~~~~~~~~~~~~
        |  Frame j+2   |
        ~~~~~~~~~~~~~~~
    ...
```

The number of records is the same as the number of lcodes.
The number of frames depends on the class of the lcode:

- **Session**-class: 1;
- **Scan**-class: total number of scans;
- **Station**-class: sum over all stations: the number of scans with participation of the i-th station;
- **Baseline**-class: total number of observations.

7. **Lcodes**
All lcodes are split onto three groups:
   - **Primitives** which does not require any additional actions;
   - **Synonyms.** It means that a request to the data of lcode A is redirected to a request to the data of lcode B. It is done primarily for compatibility with old lcode naming scheme. For example, current lcode scheme calls formal error of group delay at X band as "DELSIGMA", but formal error of group delay at S band as "DLERR XS" what is too obscure. Synonym lcodes will allow to serve request to the old lcode names.
   - **Derived.** It means that some data transformation is to be done, for example type conversion.

   Some new lcodes will be added:

   - "NUMSCA " -- Number of scans.
   - "SCAN_ID " -- Scan identifier in according with specifications of the correlator output format.

- "SCAN_IND" -- Index of the current scan in the ordered scans table.

- "STASCATB" -- Two-dimensional table of scan participation indices. Columns of the table are stations participated in the experiment. Rows of the table are scans. A value is an index of scan participation. It is zero if the station didn't participate in the scan (or precisely speaking, database doesn't have information about it). Participation indices are consecutively numbered from 1 till K. Example: 3 stations, 10 scans:

```
1    8    0
2    9   15
0   10   16
0   11   17
3   12   18
4    0   19
5    0   20
6    0   21
7   13   22
0   14   23
```

  Here station 1 and 3 participated in the scan #8, while station 2 -- not. Information for station-type lcodes related to the station 1 and scan #8 is in the frame with index 6; information for station-type lcodes related to the station 3 and scan #8 is in the frame with index 21. Station-dependent lcodes are written 23 times in this example; they would be written 60 times if the MARK-3 DBH format would be in use.

- "BASSCATB" -- One-dimensional table of scan participation indices. Input of the table is the index of the observation in the database. Output of the table is a scan index. Example, 13 observations, 5 scans:

```
          1
          1
          1
          2
          3
          3
          3
          3
          3
          3
4
          5
          5
```

How gvf handler should work

Reading

1. The files are consecutively read entirely to operative memory which is allocated dynamically;

2. All preamble sections are parsed. The following information is written in internal data structures:
   - The total number of preamble logical records;

  - The index of the file in the sequence;
  - Address of the identifier of the k-th logical record;
  - Length of the identifier of the k-th logical record;
  - Address of the body of the k-th logical record;
  - Length of the body of the k-th logical record.

3. All text sections are parsed. The following information is written in internal data structures:
    - The total number of text subsections;
    - The index of the file in the sequence;
    - Address of the title of the k-th subsection;
    - Length of the title of the k-th subsection;
    - Address of the body of the k-th subsection;
    - Length of the k-th subsection.
4. All contents sections are copied to the internal data structure. The format of internal data structure is similar to the format of the contents records. Fields "size" (length of the frame in bytes), "index of the file" and "address" (physical address of the first frame of the lcode) are added.
5. Lcodes are sorted in internal data structure.

Now handler is ready to fulfill a request to the data. Request to preamble is carried out by searching preamble identifiers codes. Request to the text data is carried out by searching the title. Request to the binary data is fulfilled by the following way:

1. Search in the lcode table. Find address of the first frame. Learn type of the data and their size.
2. If the lcode is of station-class then look at the table of scan participation indices which in turn is kept in memory after the first call to a station-class lcode. Find the frame index.
3. Compute frame address.
    i. **Session**-class data: frame address is the same as lcode-address;
    ii. **Scan**-class data: Lcode_address + (Scan_index-1)*Size;
    iii. **Station**-class data: Lcode_address + (frame_index-1)*Size;
    iv. **Baseline**-class data: Lcode_address + (Observation_index-1)*Size
4. Move data. If lcode usage code is "Derived" then call the procedure of transformation.

That is all.

Ascii gvf format

Ascii gvf format should satisfy two criteria:

- To be human readable;
- To be unambiguously convertible to the binary gvf format;

It consists of records of variable length terminated by CNTRL/J (decimal code: 10) symbol. Alphabetic symbols with codes 32-127 decimal are allowed.

Format of a record in the ascii gvf format:

`Prefix`	CHARACTER*2	One of "$$", " $", " ": "$$" is a section prefix; " $" is a keyword prefix, " " is continuation prefix.
`Keyword_delimiter_start`	CHARACTER*1	Double quotation sign " (decimal code 34). It is ignored if a continuation prefix was used.
`Keyword`	CHARACTER	Keyword. It is ignored if a continuation prefix was used.
`Keyword_delimiter_end`	CHARACTER*2	Double quotation sign " and blank (decimal codes 34, 32). It is ignored if a continuation prefix was used.
`Value`	CHARACTER	Value of the keyword.

The records with session prefix have the name of the section and empty value. They are the first record of the section and are inserted into the agvf-file as section delimiters. Format of a keyword and a value depends on section.

- A keyword of PREA section is an identifier of the logical record and the value is its body.
- A keyword of the TEXT section is a title of a subsection and a value is its body.
- Format of the CONT section:

`Prefix`	CHARACTER*2	" $" (decimal codes: 32, 36)
`Delimiter`	CHARACTER*1	Double quotation sign " (decimal code 34).
`Lcode`	CHARACTER*8	Lcode.
`Delimiter`	CHARACTER*2	Delimiter (decimal codes: 34, 32).
`Offset`	CHARACTER*9	Offset in bytes of the first byte of a logical record in the DATA section with respect to beginning of the DATA section.
`Delimiter`	CHARACTER*1	Delimiter (decimal code: 32).
`Dim1`	CHARACTER*5	First dimension of the array of values of LCODE.
`Delimiter`	CHARACTER*1	Delimiter (decimal code: 32).
`Dim2`	CHARACTER*5	Second dimension of the array of values of LCODE.
`Delimiter`	CHARACTER*1	Delimiter (decimal code: 32).
`Dim3`	CHARACTER*5	Third dimension of the array of values of LCODE.

`Delimiter`	CHARACTER*1	Delimiter (decimal code: 32).
`Date type`	CHARACTER*2	One of CH, B1, I2, I4, R4, R8
`Delimiter`	CHARACTER*1	Delimiter (decimal code: 32).
`Date class`	CHARACTER*4	One of SESS, SCAN, STAN, BASE
`Delimiter`	CHARACTER*1	Delimiter (decimal code: 32).
`Usage code`	CHARACTER*4	One of PRIM, DERV, SYNM

- Format of the DATA section:

`Prefix`	CHARACTER*2	" $" (decimal codes: 32, 36)
`Delimiter`	CHARACTER*1	Delimiter: double quotation sign " (decimal code 34).
`Lcode`	CHARACTER*8	Lcode.
`Delimiter`	CHARACTER*1	Delimiter: (decimal code: 32)
`Object name`	CHARACTER	Experiment-code for session-class lcodes; source name for scan-class lcodes; station name for station-class lcodes and baseline name for baseline-class lcodes.
`Delimiter`	CHARACTER*1	Delimiter: (decimal code 32)
`Date`	CHARACTER*10	Date in the format yyyy.mm.dd
`Delimiter`	CHARACTER*1	Delimiter (decimal code 95)
`Time`	CHARACTER*8	UTC time tag in the format HH:MM:SS
`Delimiter`	CHARACTER*1	Delimiter: (decimal code 32)
`Index element`	CHARACTER	Index of the element in the lcode array in the format (IN1, IN2, IN3)
`Delimiter`	CHARACTER*2	Delimiter: double quotation sign " and blank (decimal codes 34, 32)
`Value`	CHARACTER	Value of the keyword.

Example:

```
$$"PREA"
 $"File_format:" gvf v 0.1  1999.08.08

...

$$"TEXT"
 $"HISTORY version 1 1999.08.04 08:34:35" Created by Dbedit HP-UX version 3.1
  Dbedit control file history entry: IRIS-S138, fort-gilc-hart-west-wett, -LP
  Directory /diskA5/is138/1513

...
```

```
$$"CONT"
 $"JUL DATE"    880256    1   1    1  R8 SCAN PRIM
 $"CALBYFRQ"   1280264    3   2   14  I2 STAN PRIM

...

$$"DATA"
 $"JUL DATE 0552+398 1999.05.03 22:12:33 (1,1,1)" .2451301500000000D+07

...

 $"CALBYFRQ HARTRAO  1999.05.03_22:12:33 (1,1,1)"    459
 $"CALBYFRQ HARTRAO  1999.05.03_22:12:33 (2,1,1)"  -1752
 $"CALBYFRQ HARTRAO  1999.05.03_22:12:33 (3,1,1)"     10

 $"GRDEL    GILCREEK/HARTRAO 1999.05.03_22:12:33 (1,1,1)" -.1162569743908074D-02
 $"GRDEL    GILCREEK/HARTRAO 1999.05.03_22:12:33 (2,1,1)" -.1162561782032324D-02

...
```

Format of other keywords is seen from the example above.

This page was prepared by ( )
Last update: 05-SEP-99 19:53:44