# GENERAL DATA INTERCHANGE LANGUAGE

Fred C. Billingsley
Jet Propulsion Laboratory
Pasadena, California

## INTRODUCTION

With the advent of remotely sensed digital data and the development of digital information systems, the need for interchange of digital data has become a critical part of science research. Although this need has been felt for a number of years, the increasing use of diverse data sets on diverse computer systems has exacerbated the data interchange problems.

To date, most of the attempts to minimize the interchange problem revolve around the establishment of standard formats. Individual disciplines and projects have solved their data interchange problems by defining formats specialized to the applications. While these have been satisfactory within the various closed systems, they generally have proven too specialized to be adopted by later projects.

That this has not proved to be a panacea is evident from the the number of new formats which continually appear. The publication of the ISO (International Standards Organizatrion) 7-layer Open Systems Interchange model has clarified the real source of the problem: the various formats are typically defined in terms of users' applications, thus residing at the ISO Layer 7, the Applications Layer. The interchange is carried out in Layers 1-5, which define the media, transport sessions and protocols. Missing from consideration is Layer 6, the Presentation Layer. This layer arbitrates the various representations required by providing a location to define the standard representations of the logical entities, numerical forms, and the relationships between them, as these are used during data interchange.

The use of programming languages for data description during interchange has not been satisfactory, due to inabilities of the various languages to carry the required information and their overt intention of hiding the coding details from the programmer. In addition, the multitude of languages and the difficulties of translations between them prevent any one from serving all users.

Therefore, what is proposed is the development of a new language, to be used at ISO Level 6 for data description during interchange. Sufficient consideration has been given to this topic over the last 18 months that it is now felt to be an achievable task. Such a language is being developed at the Jet Propulsion Laboratory (JPL) as part of a NASA task. When complete and implemented, it will provide a discipline- and machine-independent method of describing discipline-dependent data. This will provide a method for preliminary parsing of a received data file, which in turn will allow relatively simple machine-dependent software to be coded to match the (target) machine and programming language to be used.

Ease of digital data interchange across diverse systems is influenced by several factors, resulting in an $n^2$ problem:

n = a(pplication) x m(achine) x l(anguage)

where there are n combinations at each of the generating and receiving ends. This suggests several approaches to reducing $n^2$ which may profitably be used tegether:

1) reduce the squared term by providing a single, common interface with machine readable data descriptions;

2) reduce "a" by encouraging disciplines to provide a set of format families which may minimize the proliferation of new formats;

3) reduce "m" by defining the common interface machine representations in a universally-readable form, thus minimizing cross-machine problems;

4) reduce "l" by defining the common interface in a way which will facilitate conversion at the receiving end to conform to the target language requirements.

The success of item 1) will depend upon the ability to describe the files adequately to allow machine receipt and processing with a minimum of human intervention or special logging programs (item 3), and the ability to interface the received file with the various programming languages in which the input and processing routines may be written (item 4). The proposed language attempts to solve this problem set.

Before considering the proposed development (the data definition approach, item 1), let us first consider a model of the process.

## MODELING THE PROCESS

Let us first consider the interchange process generically, with the desire to develop two concepts: 1) separation of the transfer process into layers pertaining to . generic description of whatever data is being transferred, and 2) providing data structure by reference as well as from attached data structure definition.

Figure 1 illustrates the elements of the transfer process.

```
----------------------------------------------------------------------------
!    SOURCE     !  DATA  !  DATA  ! TRANSFER !  DATA  !  DATA  !  RECEIVED   !
! INFORMATION ! FORM 1 ! FORM 2 !  PROCESS ! FORM 2 ! FORM 3 ! INFORMATION !
----------------------^-----------------------------^------------------------
                   X'form 1                       X'form 2
```

Figure 1 - Simplified Model of the Information Transfer Process

In terms of the model, the intent is to convert source data (Form 1) to a Standard Interchange Form (Form 2) which will convey the formatted data items plus format description, schema information, and topology information from a source to a target (Form 3). The source data is thus transported from its resident hardware configuration to become the target data on a potentially different computer. The transfer should result in no loss of data content or distortion of relationships between data fields.

The first transformation allows conversion of the data in the first machine to the standard interchange form, if desired, which is conveyed in some manner to the second machine, where a second transform regenerates the data, in the desired form. The transformation routines will constitute a language and syntax which must be discipline and machine independent. It will be seen that the transfer form may consist merely of a proper description of the data, with little or no actual data transformations.

The conversions to and from the mutually agreed upon transport medium and the physical delivery of the medium (this includes electronic transmissions) constitute the actual transfer. This may be by magnetic tape, floppy disk, or other physical media, or by electronic transmission. Common, inverse conversion routines are normally relied upon for mutual understanding of the conveyed bit pattern. In a heterogeneous environment, and in the absence of (for example) standard bit-pattern representations of binary characters on magnetic tape or floppy disks, these quantities must be specifically defined. For purposes here, this will be considered as a media problem, to be handled by media protocols. Thus, whatever the medium receives, it delivers. This is the spirit of the ISO 7-layer model.

Note that the data transfer model does not concern information per se. The information is assumed to be already coded into the data; misrepresentations of the information vis a vis the data will not be a concern of the data transfer process, although it certainly is a major part of an information transfer process.

This model bears a close relation to the ISO Open Systems Interconnection (OSI) 7-layer model. The data transfer process, drawn in a form often used in explaining the OSI, is shown in Figure 2.

```
         LEVEL 7        (APPLICATIONS LAYER)       LEVEL 7

CONTAINS A DATA FILE INCORPORATING      UNDERSTANDS THE STRUCTURE OF THE
THE DATA STRUCTURE DESIRED              DATA FILES AND DOES APPLICATION-
                                        SPECIFIC PROCESSING
           |                                        |
           |                                        |
         LEVEL 6        (PRESENTATION LAYER)       LEVEL 6

DEFINES THE RULES FOR BUILDING          PARSES THE MESSAGE TO RECOVER THE
AN APPLICATION-INDEPENDENT              DATA STRUCTURE BY KNOWING THE LEVEL
LOGICAL AND CONCRETE DESCRIP-           6 APPLICATION-INDEPENDENT DESCRIP-
TION OF THE DATA FILE                   TION RULES
           |                                        |
           |                                        |
      LEVELS 1-5 -->--(MESSAGE TRANSMISSION)-->--LEVELS 1-5
```

Figure 2 - Data Transfer Concept Using the ISO 7-Layer Model Concepts

Conversion to and out of the Standard Interchange Form is seen as a task at the Presentation Layer (Layer 6) level. Defining the problem in this manner provides the media independence by eliminating media considerations of Levels 1-5 from the standard intermediate form, and provides discipline/mission independence by describing the data bases in standard syntax.

## OTHER INTERCHANGE SPECIFICATIONS

There are only two other known specifications for data description techniques for data interchange which operate at ISO Level 6: "Data Descriptive File for Information Interchange", ISO 8211-1985, and "Abstract Syntax Notation One, (ASN.1)", ISO 8824 and 8825.

## SHORT DESCRIPTION OF ISO 8211 - DATA DESCRIPTIVE FILE

The ISO 8211 standard is a transmittal format standard, to be used for the transmission - not processing - of any data set or structure. It is intended to apply to physical media as well as to communications media. The basic approach used is to map the sender's information, including file structures such as sequential, hierarchical, relational, and indices, to the interchange format. The user maps his data into this form for transfer, and remaps this into his new format for local use.

The standard defines a data descriptive file, DDFile, to contain a data descriptive record, DDR, and its companion data records, DR. The DDR logically precedes the data records and contains the control parameters and data definitions necessary to interpret the companion data records. The DDR is the first logical record of a file other than the file labels (if applicable). It is expected that standard ISO File Labels will precede the DDFile; the Label description is not part of this specification, as it varies with the medium.

Data structure definitions are contained in a combination of both a data definition record and a data record(s). Both must be present.

The record components and their uses are:

| Record | Component | Function |
|---|---|---|
| DDR | Leader | Identifies the DDR<br>Contains the entry map (sizes of the tag, length, and position fields of the corresponding directory entries in this record) |
| | Directory | Gives Tag, length, and position (relative to start of the Data Descriptive section) of each Data Descriptive field in this record |
| | Field Terminator | |
| | Data Descriptions | Multiple entries. Structure of each corresponding vble Data Field in the DR. (This is the "data" of this record.) |
| | Field Terminator | |

|  |  |  |
|---|---|---|
| DR | Leader | Identifies the DR<br>Contains the entry map (sizes of the tag, length, and position fields of the corresponding directory entries in this record) |
|  | Directory | Gives Tag, length, and position (relative to start of the Data section) of each data field in this record. |
|  | Field Terminator |  |
|  | Data Fields | Have the structure described in the correponding DDR Data Descriptive fields. |
|  | Field Terminator |  |

The format relies extensively on delimiters for separation of fields (which must be inserted during format generation). Simple fields which are text require no format control and none is permitted. Elementary fields which contain only one data type and which are delimited by standard delimiters need no format control, and use of format control for these should be avoided. ISO 6093 numeric forms are fully specified, and therefore only a field width or delimiter is used (no internal structure description of these is utilized).

The format controls define the data structure. They take the form:

```
    ({Y!mY!k(mY,...),...},...)   where

  Y    implies [Z!Z(*)!Z(n)] ,

  Z =  A  signifies character data
       I  signifies implicit-point (Integer)     ISO 6093 NR-1
       R  signifies explicit-point unscaled      ISO 6093 NR-2
       S  signifies explicit-point scaled        ISO 6093 NR-3
       C  signifies character mode representation of a bit field
       B  signifies bit field data

   n is the field width specification
```

Data fields for the I-type, R-type, and S-type specify a number as a string of ASCII decimal numbers. Bit fields are defined as positive binary, only. No other numerical or binary form is defined.

Data Records

The data records, DR, contain the same structure of leader information as the DDR. Here, the leader fields refer to the corresponding items in the DR. The DR Entry Map has the same structure as the DDR Entry Map. The DR Directory has the same structure as the DDR Directory. The DR directory contains one directory entry for each corresponding data field. The entry contains the tag, length and location of the data field. Thus, information to this point allows separation of the data records into the composite fields and identification of the structure of each field.

The user data area is comprised of User Data Fields each followed by a Field Terminator (1/14). These fields are 1) contiguous, 2) located using the field position and field length in the DR directory 3) associated with the corresponsing tag of the DR directory and 4) through this tag are associated with the proper data description in the DDR.

## ASN.1 DESCRIPTION

The ASN.1 techniques used for the Layer 6 definitions are divided into two parts, each defined in a specification. ISO 8824 specifies a notation for abstract syntax definitions of simple field types, mechanisms for constructing new types from the basic types, notations for tagging the fields, and a number of specific useful types (86 productions, 13 character set string types). For each of these types, the notation, tag, and permissible values are given. This will enable application layer (Layer 7) standards to define the types of information they need to transfer using the presentation service. In 8824, the definitions are logical, with structure and encoding left to 8825.

ISO 8825 defines a set of encoding rules that may be applied to values of types as defined in 8824. Application of these encoding rules produces a transfer syntax for such values. It is implicit that these same encoding rules will be used for decoding.

The encoding of a data value of all types except external consist of the following four components, in order:

identifier octets
length octets
contents octets
end-of-contents octets.

The identifier octets encode the ASN.1 tag (class number, as defined in 8824). This consists of one or more octets, depending on the class number. All 8 bits of the octet are used, in a specified encoding scheme.

The length octets provide the length of the contents component, in a specified manner having somewhat the structure of a linked list. The actual length value (binary) is assembled from certain bits in the sequence of octets. A special coding indicates the indefinite form, which uses the end-of-contents component to indicate the end of the encoding.

The contents consist of zero or more octets, encoded as specified for each data value type (boolean, integer, bitstring, etc).

The end-of-contents, when used for the indefinite form, consists of two zero octets.

## EVALUATION OF EXISTING TECHNIQUES

In studing the 8211 and ASN.1, it has been found that each has serious shortcomings for the data description task. Some of these are:

### ISO 8211

According to its author, 8211 was modeled after an earlier bibliographic interchange standard. It lacks easy expression of many of the concepts needed for describing science data - structures, inclusion of externally-defined format controls, binary fields for numerical representations, and multiple types of data records in a file, as examples.

Allows only one DDR per file. This makes it difficult to define more than one type of data record in each file.

Provides only unsigned binary representation for binary fields. This prevents binary fields carrying machine representations of numbers or carrying enumerated values or meanings to be defined within the message.

Has no provisions for the more complex definitions needed for commutated data or for data base transfers, nor for using prior-defined field structures.

All data field labels must be the same length. This is a severe restriction which is unacceptable to at least one science community.

Cannot define data structures by reference - the DDL must always accompany the data file. This prevents the external definition and registering of data formats for callup.

The data file cannot be used verbatim, but must be restructured within the data record.

ASN.1 (ISO 8824 and 8825)

Has gone to great lengths to serve a completely open system, which requires much more definition than the simpler data transfer description task.

As an abstract syntax notation, ASN.1 is heavily weighted toward declaration statements of the ADA or Pascal types - that is, logical constructions instead of specific data field descriptions. (This is the same reason that programming languages are unsuited for data description.)

8825 (Encoding Rules) uses a highly-encoded identifier for the contents type which cannot be "dumped" for human reading. It also uses an encoded distributed binary coding of the length field, again preventing easy dumping. Use of these encodings will prevent the transfer of ASCII files in a 7-bit mode.

Every data entry ("encoding") must consist of a type-length-value-[end] sequence. This adds tremendously to the overhead, both in file size and in time to decode. At this time, there is no provision for a single type declaration to be used with multiple data entries, such as image pixels.

Provides only unsigned binary representation or the predefined Real structure for binary fields. This prevents binary fields from carrying machine representations of numbers or carrying enumerated values or meanings.

Relies extensively on "External" data types for the more complex definitions needed for commutated data or for using prior-defined field structures. These defeat the concept of full descriptions within the specification.

These encoding rules also prevent the transmission of a data file without extensive manipulation to generate and intersperse the various type-length-[end] components.

There is no apparent way to describe commutated or scattered data.

## GDIL DESCRIPTION

The General Data Interchange Language (GDIL) concept is curently under development as a JPL task. It is being designed to avoid the problems seen with the other DDLs. Following is a brief overview of the intended GDIL capabilities. These capabilities, not found in other data descriptions, serve as the reason for developing the GDIL.

GDIL is conceived as a media-independent, content-independent tool for the transfer of information between dissimilar computer systems. It is NOT a tool for the internal processing of information. It does not require the insertion of data field terminators, or any change in a user data file, and thus may be used to describe archived files. Machine numerical forms may be used and described, without modification. It permits the sender to describe the transferred information and to send this description separately or as an integral part of the transfer file. It permits the description of both character and bit field information in fixed- (without delimiters) or variable-width (delimited) fields or subfields. It further permits the identification of fields and subfields by arbitrarily long names and labels which serve to give meaning to the data. In addition, it provides for the definition and labeling of complex structures and commutated data.

### GDIL Structure

Punctuation symbols used in this document are as follows:

|  |  |
|---|---|
| < > | indicates a logical entity |
| [ ] | indicates optionally present |
| { } | indicates optional repetition |
| ( ) | indicates grouping |

The GDIL Module consists of Core, Extension, and Data records. The Core contains information about the module and data as a whole, and the Extension carries the desriptions of the data fields and their interrelationships.

```
<Module> ::= <Core> [<Extension>] [{<Data1>}] ... [{<Datan>}]
```

This may also be expressed as:

```
                          Terminated with:
[Core Record                  IS2]
Last Core Record              IS3
[Extension Record             IS2]
[Last Extension Record]       IS3
[Data Record                  IS2]
      ...
[Last Data Record]            IS4
```

where IS2, IS3 and IS4 are the ISO Information Separators.

```
              ______________________________________________________
THE GDIL     ¦          ¦           ¦              ¦               ¦
MODULE       ¦ CORE IS3 ¦ [EXT] IS3 ¦ {[DATA1 IS2]} ¦ {[DATAn]} IS4 ¦
STRUCTURE    ¦__________¦___________¦______________¦_______________¦
```

The Core and Extension records each consist of a seies of segments having a single Backus-Nauer (BNF) form:

```
     <Core>      ::= {<Segment>}

     <Extension> ::= {<Segment>}.
```

Each segment consists of a Length-Type-Value series of fields:

```
 <Segment> ::= <Length> <Tag> IS1 <SegValue> ISn    where

 Tag is the segment Type (Name)
 SegValue is the Segment data contents
```

```
                             _________________________________________
                            ¦          ¦     ¦    ¦         ¦      ¦
CORE (OR EXTENSION)         ¦ LENGTH   ¦ TAG ¦ IS1¦ SEGVALUE ¦IS2(3)¦
SEGMENT STRUCTURE           ¦ 2 bytes  ¦vble ¦1 by¦   vble   ¦ 1 by ¦
                            ¦__________¦_____¦____¦__________¦______¦
                                 ¦    \_____________  _______________/
                                 ¦                  \/
                                 ----defines-------
```

The Length field (2 bytes) is the length of the Segment, from the [Tag] to the IS, inclusive.

The GDIL may be considered as Keyword-driven, where the GDIL keywords are the segment tags. A standard, recognizable, group of segment tags is specified in the GDIL, from which a given instance may be assembled. This allows the building of a GDIL Module from a relatively small group of specification-defined Tags plus user-defined Tags. Similarly, the user may define keywords (Labels) for the data fields of the user records. These fields are described by the GDIL and may be located by application software using the labels as keywords. Thus, only those Tags and Labels necessary for the instance need be included. This approach provides a more flexible and extensible descriptive form than pre-defined descriptive formats.

## Data Field Structure Description

The philosophy behind the structure definitions is that in a transmitted series of bytes, there is no inherent logical structure, either to the grouping of the bytes or to any numerical or logical forms recognized by computers. Therefore, everything must be defined.

The data field structures are described in a series of entries called format controls which are related to the corresponding data fields through labels as follows:

The structures of externally-defined fields may be referenced using the EXAF (External Authority and Format) segment:

```
EXAF IS1 {<Label> , <Authority> , <Format ID>}    where
```

Label is the user-defined label for this instance,
Authority is the external authority being referenced, and
Format ID is its format reference.

New variables, arrays and complex data field structures may be defined once in a structure segment and subsequently used:

```
STRUCT IS1 <Label> , [<Type>] : {<Format Control>}
```

Format Controls describe the Integer, Real, Character, or other form of the data field. They are based on the set from 8211 plus others which have been found to be necessary. Format controls are recursive in the sense that format control of previously-defined structures or fields may be included by reference, using a preceding asterisk, in the format controls:

```
STRUCT IS1 <Label> , : [{*<Label1>}] [{<Format Control>}]
```

where Label1 is a previously-defined Label.

Format Control Segments, Long and Short (FCL and FCS) are used to describe data records, and are structured logically as:

```
FCL IS1 <Xref> , { <Label> <Width> <Offset> <Format Control> }

FCS IS1 <Xref> , { <Format Control> }
FCS IS1 <Xref> , { *<Format Control> }
```

Xref is the identifier of the data record being described.
Labels are the user-defined field or other aggregate labels.
Width is the width of a data field.
Offset is the offset of the field from the beginning of the data record.

## Datatags

The DATATAGS segment contains an optionally parenthesized list of the user data field labels, to whatever depth the user desires. The allowable set of labels are those specified in the user application specification. The same labels are used in the FCL, FCS, STRUCT, EXAF, and the logical description segments.

## Hierarchical and Network Stsuctures

The parenthesized Datatags form will describe a hierarchical or fieldsubfield structure. An alternate method of description is to provide a list of node labels in a preorder traverse sequence from a single root representing the entire section of data, plus an ordered list of the last node of preorder traverse sequences beginning at each node (including leaf nodes). These two sequences are carried in the TRAV(erse) and LASTNODE segments.

Network structures may be described by cutting the structure into a hierarchical tree, and sending this plus a list of the cut links, using the LABELPAIR Segment.

Relational Structures

Relational structures may be decomposed into a set of orthogonal relational tables. The structure of the lines of these tables may be described using the Structure segment. The column headings may be given as labels in a Datatags segment.

```
STRUCT:          <TableLabel>,Table:<FmtCtl1>,<FmtCtl2>, ... ,<FmtCtln>
DATATAGS:<Xref>,<TableLabel>(<ColHdg1>,<ColHdg2>, ... ,<ColHdgn>)
FCS:      <Xref>,*<TableLabel>                 where
```

the Datatags and FCS Xref field refers to data records by that name, and the FCS form indicates that each row has the form specified in the TableLabel Struct segment.

Machine and language independence will be accomplished by: 1) defining the transfer as a series of bytes, thus eliminating media byte-interchange problems such as the VAX vs IBM tape formats; 2) providing methods for defining binary data fields such as machine representations in such as way that suitable new target representations may be constructed; 3) defining a canonical interface as a pair of ASCII tables which describe the data records in such a manner that data fields may be located, read, and converted to the desired representations on the target machine, using the desired target programming language or DBMS.

The canonical ASCII interface tables will have the following contents:

<u>Segment Table</u> (for each segment)

|  |  |
|---|---|
| Segment Tag | Segment contents verbatim |

<u>Access Table</u> (for each data field)

| Record Label | Field Label | Structure | Length | Position | Format Controls |
|---|---|---|---|---|---|

The total set of capabilities, from the consistent segment structures, through the format control techniques, through the canonical interface, will constitute a unique and new tool for the systematic transfer of data. With it available, local software which will be needed to convert user files to and from the canonical interface will be appreciably simplified. This software on each end may be independent, one end from the other, thus reducing the $O(n^2)$ problem to an $O(2n)$ problem.

## CURRENT ACTIVITIES AND STATUS

Development of the concept has been underway at JPL for about two years, sponsored by NASA OSSA Information Systems Office (EI) and the Communications and Data Division (TS). It has developed from earlier attempts at defining a suitable format standard for two national committees: ACSM National Committee for Digital Cartographic Data Standard, and the Federal Interagency Coordinating Committee for Digital Cartography.

The Consultative Committee on Space Data Systems, which is designing a message interchange system structure called Standard Formatted Data Units, is sponsoring the developemnt of the GDIL as a candidate language. The SFDUs are intended to be used in the NASA space activities such as the space station; of particular interest here is the intended use in the ground system, in archive distributions, and between experimenters.

A simple GDIL demonstration builder (GDB) and parser (GDP) for demonstration purposes has been developed through a small contract at UCSB. This allows interactive definition of the structure of simple files and display of the resulting GDIL file. It is to be used as a test bed for developing and validating new concepts.