Recommended Practices for Exploration and Production Data Digital Interchange

API RECOMMENDED PRACTICE 66, V2
SECOND EDITION, JUNE 1996

American
Petroleum
Institute

Recommended Practices for Exploration and Production Data Digital Interchange

API RECOMMENDED PRACTICE 66, V2
SECOND EDITION, JUNE 1996

American
Petroleum
Institute

SPECIAL NOTES

API publications necessarily address problems of a general nature. With respect to particular circumstances, local, state, and federal laws and regulations should be reviewed.

API is not undertaking to meet the duties of employers, manufacturers, or suppliers to warn and properly train and equip their employees, and others exposed, concerning health and safety risks and precautions, nor undertaking their obligations under local, state, or federal laws.

Information concerning safety and health risks and proper precautions with respect to particular materials and conditions should be obtained from the employer, the manufacturer or supplier of that material, or the material safety data sheet.

Nothing contained in any API publication is to be construed as granting any right, by implication or otherwise, for the manufacture, sale, or use of any method, apparatus, or product covered by letters patent. Neither should anything contained in the publication be construed as insuring anyone against liability for infringement of letters patent.

Generally, API standards are reviewed and revised, reaffirmed, or withdrawn at least every five years. Sometimes a one-time extension of up to two years will be added to this review cycle. This publication will no longer be in effect five years after its publication date as an operative API standard or, where an extension has been granted, upon republication. Status of the publication can be ascertained from the API Authoring Department [telephone (202) 682-8000]. A catalog of API publications and materials is published annually and updated quarterly by API, 1220 L Street, N.W., Washington, D.C. 20005.

This document was produced under API standardization procedures that ensure appropriate notification and participation in the developmental process and is designated as an API standard. Questions concerning the interpretation of the content of this standard or comments and questions concerning the procedures under which this standard was developed should be directed in writing to the director of the Authoring Department (shown on the title page of this document), American Petroleum Institute, 1220 L Street, N.W., Washington, D.C. 20005. Requests for permission to reproduce or translate all or any part of the material published herein should also be addressed to the director.

API publications may be used by anyone desiring to do so. Every effort has been made by the Institute to assure the accuracy and reliability of the data contained in them; however, the Institute makes no representation, warranty or guarantee in connection with this publication and hereby expressly disclaims any liability or responsibility for loss or damage resulting from its use or for the violation of any federal, state, or municipal regulation with which this publication may conflict.

API standards are published to facilitate the broad availability of proven, sound engineering and operating practices. These standards are not intended to obviate the need for applying sound engineering judgment regarding when and where these standards should be utilized. The formulation and publication of API standards is not intended in any way to inhibit anyone from using any other practices.

Any manufacturer marking equipment or materials in conformance with the marking requirements of an API standard is solely responsible for complying with all the applicable requirements of that standard. API does not represent, warrant, or guarantee that such products do in fact conform to the applicable API standard.

FOREWORD

This recommended practice was prepared by the Subcommittee on Standard Format for Digital Well Data. This standard is under the administration of the American Petroleum Institute Exploration and Production Department's Executive Committee on Drilling and Production Practices.

Background and Purpose

A wide variety of digital data is acquired, exchanged, and stored by petroleum industry businesses using an equally wide variety of data formats. Most of these formats have been around for many years and were designed for very specific needs. As a result, they reflect the limitations of early computing hardware and are typically oriented to one type of data. Although relatively simple to implement, many formats lack the flexibility or sufficient self-descriptive features to accommodate evolving data exchange needs.

Recommended Practice 66, Version 1 (RP66, V1) was introduced by the API in 1991 as an enhanced exchange format for well data. It incorporated the useful features of earlier formats and extended them by removing some of their historical limitations. The primary features of API Recommended Practice 66, Version 1 were machine independence, self-description, semantic extensibility, and efficient handling of bulk data. It has been implemented and is used widely for exchange of well data.

Following the introduction of API Recommended Practice 66 by the API, a number of efforts by other oil industry groups to develop data exchange formats began to mature and converge toward the adoption of API Recommended Practice 66, creating a major opportunity to have just one standard exchange format for all oilfield data. However, the strong bias of API Recommended Practice 66 toward well data was a stumbling block. In addition, introduction of new high-capacity storage devices created a need to expand the API Recommended Practice 66 physical binding mechanism, and experience in using API Recommended Practice 66 generated ideas for improvements. Working closely with other industry groups, the API Subcommittee on Recommended Format for Well Data developed a set of modifications to API Recommended Practice 66, which have resulted in the current API Recommended Practice 66, Version 2 (RP66, V2), which is described in this document.

API Recommended Practice 66, Version 2 retains the essential features of API Recommended Practice 66, Version 1 and is completely upward compatible. That is, any data recorded under API Recommended Practice 66, Version 1 can be translated into API Recommended Practice 66, Version 2 format.

The following list summarizes the differences between API Recommended Practice 66, Version 1 and API Recommended Practice 66, Version 2.

1. **Separate well data object types from neutral data object types.**
   Public object types from API Recommended Practice 66, Version 1 are separated into two groups, both administered by the API Subcommittee on Recommended Format for Well Data. One group consists of basic object types having no well data bias. This is referred to as the basic schema, described in Part 6, and is administered under organization code 0 (zero). The second group consists of all the rest and is administered under organization code 66. This group, described in Part 8, is referred to as the DLIS schema.

2. **Add new attributes and symbolic codes.**
The following are added: DIMENSION and AXIS attributes to CALIBRATION-COEFFICIENT, DATE attribute to CALIBRATION, KIND attribute to CHANNEL, PARAMETER, and COMPUTATION, DESCRIPTION attribute to all object types, EXTENDED-ATTRIBUTES attribute to all object types except FILE-HEADER and ORIGIN, new codes for TYPE attribute of EQUIPMENT, and new codes for PROPERTIES attribute (many object types).

3. **Allow COORDINATES attribute of AXIS to reference a CHANNEL object.**
This permits dynamically changing coordinates.

4. **Add new representation codes and representation code classes.**
The following codes are new: RNORM, RLONG, ISNORM, ISLONG, IUNORM, IULONG, IRNORM, IRLONG, TIDENT, TUNORM, TASCII, LOGICL, BINARY, FRATIO, DRATIO. All representation codes are grouped into classes of related representations. A restriction in the definition of an attribute to any member of a class allows use of any other member of the class.

5. **Extend ASCII to include ISO 8859-1, and allow string padding.**
The set of allowable characters in an ASCII string is enlarged. In addition, character subfields in all representations may be padded by using a terminating null (zero) character. The string count may include characters past the null, but such characters are not considered part of the string data. This supports in-place editing of API Recommended Practice 66 data.

6. **Change copy number (in OBNAME) from USHORT to UVARI.**
A maximum of 256 instances of a named object was seen as potentially limiting for some applications.

7. **Extend unit expression syntax and add new unit symbols.**
The unit expression syntax supports additional types of units, for example units with fractional exponents, and use of multiple sets of parentheses to improve meaning. A more comprehensive set of SI unit symbols and a set of monetary unit symbols are added to the unit dictionary.

8. **Allow use of multiple unit models.**
In addition to the unit model described by API as part of API Recommended Practice 66, provision is made to identify and use other unit models having different sets of unit symbols.

9. **Uncontrol EQUIPMENT identifiers.**
These identifiers are no longer required to be in a dictionary. Attributes TYPE and TRADEMARK-NAME suffice to identify equipment.

10. **Require distinct object names in a logical file.**
This rule, that no two object names can match all three subfields (origin, copy number, identifier), makes an object name a unique reference. It is no longer necessary for a reader to know the object type to resolve the reference.

11. **Revise ORIGIN.**
Well data attributes are removed and put into a new DLIS-CONTEXT object type in the DLIS schema. New attributes are added to identify schema and unit model.

12. **Support use of multiple schemas.**
The terms Private and Public are removed. All schemas are implemented equally with the only exception being mandatory (and exclusive) use of FILE-HEADER and ORIGIN object types from the basic schema. Mechanics to make this work include TIDENT representation of set types and attribute enumerations, and removal of logical record type numbers.

13. **Remove rule restricting ordering of EFLRs and IFLRs.**
An attribute in one set (EFLR) may reference an object in another set even if the two sets are separated by an IFLR.

14. **Remove special constraints on representation of FILE-HEADER attributes.**
The attributes of a FILE-HEADER object are no longer constrained regarding length and order.

15. **Neutralize FRAME and CHANNEL and support multiple frames per record.**
Attributes of these objects are revised to remove the well data bias. In addition, multiple frames per record are supported. This allows some applications to increase performance for reading and writing bulk data. In addition to dimensioned arrays, channel values may now also be described as aggregates (similar to "ragged arrays").

16. **Replace channel dimension updates with explicitly-sized channels.**
Provisions are made to write explicitly-sized channels by optionally recording channel dimensions directly in frames. Correspondingly, the DIMENSION attribute of CHANNEL is no longer updatable.

17. **Add new information to storage unit label.**
The following fields are added: binding version, producer code, creation date, and serial number. The size of maximum record length is increased.

18. **Expand visible record header and logical record segment header.**
The maximum length of a visible record is increased from a 16-bit quantity to a 32-bit quantity, and similarly for lengths in the logical record segment. This allows binding onto very large tape blocks (megabyte or more). New fields file sequence number and file section number are added to the visible record header. The visible record now also has a trailing length. A new structure, FIXREC, is added to indicate fixed-length visible records.

19. **Fix encryption mechanism to handle blind passthrough.**
The logical record segment encryption mechanism is modified to allow readers to copy and re-segment data for which the encryption method is unknown.

20. **Define a physical binding for files on random access disks.**
A file on a random access disk is defined to be a storage unit if the file has a byte stream structure consisting of a storage unit label followed by a sequence of visible records.

Organization of API Recommended Practice 66, Version 2

API Recommended Practice 66, Version 2 consists of Parts 1 through 9, plus Appendix A. These parts are briefly described below:

PART 1: MODEL AND METHODOLOGY.
This part describes the data model upon which the format is based and the

methodology for specifying schemas–namely, object types, attributes, and the rules about them.

PART 2: LOGICAL FORMAT.
This part describes the logical organization and representation of data, including the definitions and uses of storage sets, logical files, visible records, logical records, sets, objects, attributes, and values. It also describes naming and referencing rules as well as bindings between logical records and visible records and includes specifications of all representation codes used in the format.

PART 3: PHYSICAL BINDINGS.
This part describes how storage unit labels and visible records are recorded on various common medium types, including 9-track magnetic tape and random access disk files. It also describes the use of filemarks and partitions where applicable.

PART 4: THE API-SI UNIT MODEL.
This part describes a unit model based on le Système International d'Unités (SI) from which a wide variety of units can be expressed. The unit model supports a general method for computing unit conversion coefficients for dimensionally-related units.

PART 5: THE API-SI UNIT SYMBOLS.
This part lists and defines the unit symbols recognized under the unit model described in Part 4.

PART 6: BASIC SCHEMA.
This part specifies the object types administered by the API Subcommittee on Recommended Format for Digital Well Data using organization code 0 (zero). This schema includes certain basic object types such as FILE-HEADER and ORIGIN, which are required by all implementations, as well as other object types such as FRAME and CHANNEL, which are standard mechanisms for describing the storage of dynamic (or bulk) data.

PART 7: BASIC SCHEMA DICTIONARY.
This part lists and describes reference values for attributes belonging to the basic schema.

PART 8: DLIS SCHEMA.
This part specifies the Digital Log Interchange Standard (DLIS) schema object types administered by the API Subcommittee On Recommended Format For Digital Well Data using organization code 66. This schema includes object types particularly oriented toward recording well log data.

PART 9: DLIS SCHEMA DICTIONARY.
This part lists and describes reference values for attributes belonging to the DLIS schema.

APPENDIX A: ORGANIZATION CODES.
This appendix lists currently-assigned organization codes.

This standard shall become effective on the date printed on the cover, but may be used voluntarily from the date of distribution.

CONTENTS

Recommended Practices for Exploration and Production Data Digital Interchange

Part 1: Model and Methodology

Exploration and Production Department

API RECOMMENDED PRACTICE 66, V2
SECOND EDITION, JUNE 1996

American
Petroleum
Institute

CONTENTS

Recommended Practice for Exploration and Production Data Digital Interchange Part 1: Model and Methodology

0 Introduction

0.1 This standard contains introductory, normative, and annotative information. Introductory information comprises everything that precedes Section 1, Scope. The introductory clauses describe how the publication is organized and provide commentary on the history and purpose of this standard. Normative information includes the actual requirements that must be satisfied by any data conforming to the standard. Annotative information is provided as rationale for and illustrations about the normative information and does not constitute part of the standard. The standard is completely stated even if all introductory and annotative information are removed. Annotative information may refer to terms introduced in later paragraphs or parts in order to tie together concepts, i.e., from time to time its value may increase after the reader has made a complete pass over the standard.

0.2 Different styles are used to distinguish between normative and annotative information.

0.3 All normative information is written using this same font, and is contained in either numbered paragraphs or numbered tables. Normative text is aligned with the left margin of the page.

0.4 All figures are annotative, and all annotative text is written in unnumbered paragraphs in Helvetica italic font, indented, as shown here:

> *This is a paragraph of annotative text. Its purpose is to include informal commentary on the normative information in immediately preceding or following paragraphs and may include references to or usage of normative information in other parts of the standard.*

1 Scope

1.1 This part specifies a methodology for managing (generating and maintaining) a schema specification for data to be recorded under the API Recommended Practice 66, Version 2 format.

1.2 This document is intended to:

a. Facilitate the development of exchange standards based on the API Recommended Practice 66, Version 2 format.
b. Facilitate the development of schema-neutral software products and services, by promoting uniformity between API Recommended Practice 66, Version 2-based exchange standards.
c. Promote compatibility between editions of a schema.

2 Definitions

2.1 **attribute:** A named item of information or data pertaining to an object type.

2.2 **attribute count:** The number of elements in an attribute value.

2.3 attribute label: The name of an attribute.

2.4 attribute representation code: A code that identifies the recorded representation of each element of an attribute value.

2.5 attribute units: An expression that represents the units of measurement of each element of an attribute value.

2.6 attribute value: The value of an attribute. It may be present or absent. When present, it consists of zero or more elements, each having the same units and the same representation.

2.7 characteristic: A subitem of a component that contains one piece of information for a data item. Characteristics of an attribute component, for example, are its label, count, units, representation code, and value.

2.8 data model: A description of a specification and representation paradigm for data.

2.9 dictionary: A database in which identifiers and reference values used under API Recommended Practice 66, Version 2 are maintained and administered.

2.10 identifier: One of the three parts of an object name. It is a character string used to distinguish the object from other objects of the same type. For some designated object types, the identifier conveys meaning of the nature of the object, and the identifier and its meaning are maintained in a dictionary.

2.11 logical file: The main unit of data exchange. It consists of a sequence of one or more logical records, beginning with a record containing a single FILE-HEADER object.

2.12 logical model: A conceptual organization of a domain of knowledge.

2.13 object: A recorded instance of an object type.

2.14 object type: A logical entity of a schema that has a unique type name and one or more defined attributes. Instances of an object type are written in explicitly formatted logical records.

2.15 organization code: A number assigned by API to an organization that identifies the organization and represents schemas and dictionaries defined and administered by the organization. See Appendix A, "Organization Codes."

2.16 representation code: A unique number that identifies a standard encoding for a value as a sequence of one or more contiguous bytes.

2.17 schema: A formalized description of the encoding of information defined by a logical model, typically in terms of a data model.

2.18 schema code: A numeric code found in Appendix A used to identify the organization responsible for defining and administering a schema.

3 Concepts

3.1 A logical model is the conceptual organization of a domain of knowledge: What classes of things are of interest, and what are their characteristics and the relationships between them which are of interest? A logical model is a framework for common understanding of the nature of classes of things, but is not sufficient for communication about individual members of the classes.

3.2 An implementation model, also called a schema, is a formalized description of the encoding of information defined by the logical model, typically in terms of a representational model. How are general representational constructs used to encapsulate information about the elements defined in the logical model? A schema must be sufficiently dynamic to meet new requirements, and yet must be sufficiently stable that change requirements are not unnecessarily imposed on the systems which use it.

3.3 A representational (or data) model is a description of the specification and representation paradigm to be used. What are the fundamental representational constructs and their organization? How is the schema specified? How is data organized, encoded, stored, and decoded?

3.4 The purpose of a schema specification for an exchange format is to enable the exchange of information in some domain of knowledge.

4 Audience

4.1 This document is intended for:

a. Original authors and maintainers of schema specifications.
b. Readers of schema specifications.
c. Developers of schema-neutral API Recommended Practice 66, Version 2 software layers and systems.

4.2 It is assumed that the reader has a basic understanding of logical models, and has a particular logical model in mind. No particular modeling formalism is assumed; indeed, the schema specification may itself be the only formal expression of the logical model.

4.3 API Recommended Practice 66, Version 2 is a general-purpose data exchange mechanism which is not bound to a specific logical model. The purpose for using a standard methodology for schema specification is to enable the implementation of data exchange using model-driven software layers.

5 Schema Categories

5.1 Schemas may be categorized as managed, derived, and local.

5.2 A managed schema exists from the definition of its initial edition, through subsequent editions, until all data recorded under it are no longer of interest (i.e., indefinitely). During this life cycle each edition of a schema is expected to meet certain requirements for compatibility or continuity with previous editions. As the logical model and usage requirements change, new editions of the schema are developed, subject to these constraints.

5.3 A derived schema is one which is systematically derived or inferred from a formalized logical model according to a deterministic methodology. In this case a human-readable schema specification might not be produced. Furthermore, the schema edition reflects the version of the derivation methodology and the versioning policy under which it is governed. The logical model may be governed under a different versioning policy. Organizations that define derivation methodologies should provide mechanisms to record and identify the author, name, and edition of the logical model from which the schema is derived.

5.4 A local schema is one that is neither managed nor derived but is used for local exchange or storage of data using the data model.

5.5 This document is normatively applicable to managed schemas only.

6 Schema Administration

6.1 An organization must have been issued an organization code to publish a managed schema or a schema derivation methodology. Code 999 is reserved for local schema. An organization code is assigned on request by the Exploration and Production Department, American Petroleum Institute, which also maintains the most current list of organization codes. Its address is listed at the end of the Foreword.

6.2 The holder of an organization code may publish (editions of) a schema or a schema derivation methodology under that code. An organization that wishes to manage multiple schemas may obtain multiple organization codes. No other constraints apply among schemas, but certain constraints apply between editions of a managed schema. If a schema element is defined under two editions, then it has the same meaning and use in both editions. Elements may be dropped from one edition to the next and new elements may be added, but elements may not be redefined.

6.3 Alternatively, an organization may manage multiple schemas by defining a derivation methodology under an organization code for mapping an appropriate class of logical models into derived schemas.

6.4 In general, it is expected that an organization that publishes a schema or a schema derivation methodology will also publish the specific versioning policy governing compatibility and constraints between editions. Derived schemas editions are determined indirectly by reflecting the logical model edition along with the name and edition of the derivation methodology applied to the logical model.

7 Data Model

7.1 API Recommended Practice 66, Version 2 schema specifications are defined in terms of object types, attributes, and the rules about them. These data model elements are described in detail in Part 2, "Logical Format."

7.2 The specification of an object type includes its name, its semantics, its attributes, and whether its instances have controlled names. No relation shall be assumed between object types in different schemas having the same type names.

7.3 The specification of an attribute includes its label, its semantics, and any restrictions on its count, representation code, units, or value. Unless explicitly stated, no relation shall be assumed between attributes in different object types having the same labels.

7.4 An instance of an object type is constrained by its object type specification. An object may have all, some, or none of the attributes defined for it, and shall not have attributes not defined for it or duplicates of the attributes defined for it.

7.5 An instance of an attribute is constrained by its specification in the object type of the object in which it appears. An attribute has exactly one occurrence of each type of characteristic: label, count, representation code, units, and value. The label identifies the attribute in terms of its definition in the object type. The value is composed of zero or more elements, or occurrences of the data representation identified by its representation code. It has a single count, which is the number of elements in its value. It has a single units value, which applies to each element of the value.

7.6 The logical file is the logical unit of data exchange, and is a collection of objects and other data described by its objects. The object is the unit of data which may be uniquely referenced within a logical file. An object is a collection of the attributes defined for its type, with assigned values. A logical file may contain objects from any number of schemas, and objects may refer to objects in different schemas in the same logical file.

8 Schema Specification

8.1 Preliminary Part

8.1.1 The preliminary part of a schema specification may include the following:

a. Title page.
b. Table of contents.
c. Foreword.
d. Introduction.

8.2 Normative Part

8.2.1 The normative part of a schema specification may include the following:

a. Context.
   1. Title.
   2. Scope.
   3. Normative references.
b. Authority.
   1. Sanctioning organization.
   2. Document preparation, revision, and review committees.
   3. Proprietary considerations.
   4. Revision and versioning policies.
c. Edition.
   1. Edition level.
   2. Summary of this and recent editions.
   3. Document status (draft or final).
d. Definitions.
   1. Define terminology introduced.
   2. Normative references to other defining documents.
e. Model.
   Definition, presentation, or description of the logical model, or reference to equivalent documentation.
f. Schema.
   1. Definition of object types and their attributes.
   2. Constraints on objects and attributes.

3. Relationships to corresponding elements of the logical model.

g. Dictionary.
Tables of reference values, to which attributes they apply, and their meanings.

h. Units.
1. Specification of unit model(s) and unit dictionary(ies) used in the schema (if not API-SI units).
2. Identification of units model in attribute restriction specifications.

i. Normative appendices.

8.3 Supplementary Part

8.3.1 The supplementary part of a schema specification may include the following:

a. Informative appendices.
b. Footnotes.

8.4 Object Type Specification

8.4.1 An object type is specified by providing a unique type identifier, dictionary constraints on the identifier subfield of the name, a set of attributes, and the intended use of the object. An object type specification typically has the following parts:

a. Type name.
b. Context–a description of the semantics of the object type or reference to a logical model.
c. Attribute specifications.

8.5 Attribute Specification

8.5.1 The characteristics of an attribute are its label, count, representation code, units, and value. The formal specification of an attribute includes its label, and may include constraints on the remaining characteristics.

8.5.2 The formal specification of the set of attributes defined for an object type is presented in a single table, followed immediately by a set of numbered notes. The columns of the attribute specification table are:

a. Attribute Label: a valid IDENT value, unique in the object type.
b. Restrictions: a set of restrictions on the count, representation code, units, or value characteristics of an attribute. Restriction specifications are of the form 'q1=r1, q2=r2, ... qn=rn', where 'qn' is the characteristic identifier and 'rn' is a restriction specification. The characteristic identifier is lower case. When no restrictions apply, the identifier is omitted. Restriction specifications are stated in the order 'c=..., r=..., u=..., v=...'.
1. A count restriction is specified as 'c={[min1:max1], [min2:max2], ...,[minn:maxn]}', where mink and maxk represent bounds, or '?' if the interval lacks an upper or lower bound. If a single boundary set is specified, the curly braces ('{', '}') are omitted. If the minimum and the maximum are equal for any boundary range, the square brackets ('[', ']'), colon (':'), and second range limit are omitted.
2. The representation code restriction is specified as 'r=XXX | YYY | ZZZ', where XXX, YYY, and ZZZ are alternative representation codes. If a single representation code is specified, the vertical bar '|' is omitted. For each representation code, an alternative representation in the same representation code class may be used, but the value shall be representable in one of the specified representation codes.
3. The units restriction is specified as 'u=um:uexp', where 'um' identifies the units model and its dictionary, and 'uexp' is a units expression in that units model. If the

'um:' prefix is omitted, then a default unit model shall apply if declared in the general context part of the schema specification. The specification 'u=' indicates the attribute value is unitless. When the unit model is API-SI, then restriction to a unit represents a typical or preferred use, although another unit having the same canonical form may be used. A unit having a different canonical form shall not be used.

4. The value restriction is specified as 'v=ref', where 'ref' may refer to the note or to a table of reference values for the attribute. Reference values are dictionary-controlled values belonging to an enumerated set. A schema may declare a value to consist of reference values and defer actual definition of the reference values to other schemas that may use the attribute's object type. A schema may provide an actual table of reference values or a normative reference to them.

c. Notes: Numbered references to notes that immediately follow the table. Notes are free-format descriptions of semantics and other rules about the attribute and its relation to other attributes.

9 Notation: How to Refer to an Attribute

9.1 The notation TYPE:LABEL is used to refer to an attribute in a schema, where TYPE is the object type and LABEL is the attribute label. TYPE may be omitted if understood from context. If the schema is not clear from the context of the referral, then the additional notation n/TYPE:ATTRIBUTE is used, where 'n' is the schema code of the object type.

10 Dictionary Specification

10.1 Preliminary Part

10.1.1 The preliminary part of the dictionary may include the following:

a. Title page.
b. Table of contents.
c. Foreword.
d. Introduction.

10.2 Normative Part

10.2.1 The normative part of the dictionary may include the following:

a. Context.
   1. Title.
   2. Scope.
   3. Normative references.

b. Schema: a reference to the schema (schema code, schema name, schema edition) and dictionary specification within the schema for which the dictionary is applicable.

c. Authority.
   1. Sanctioning organization.
   2. Document preparation/revision/review committees.
   3. Proprietary considerations.
   4. Revision and versioning policies.

d. Edition.
   1. Edition level.
   2. Summary of this and recent editions.
   3. Document status (draft or final).

e. Definitions.
   1. Terminology.
   2. Normative references to other defining documents.

f. Description of the reference value tables.
g. Reference value tables.
   1. Title which identifies the attribute restricted to the reference values.
   2. Reference value column.
   3. Description column.
   4. Other columns as required by the schema.

10.3 Supplementary Part

10.3.1 The supplementary part of the dictionary may include the following:

a. Informative appendices.
b. Footnotes.

Recommended Practices for Exploration and Production Data Digital Interchange

Part 2: Logical Format

Exploration and Production Department

API RECOMMENDED PRACTICE 66, V2
SECOND EDITION, JUNE 1996

American
Petroleum
Institute

CONTENTS

Recommended Practice for Exploration and Production Data Digital Interchange Part 2: Logical Format

0 Introduction

0.1 This standard contains introductory, normative, and annotative information. Introductory information comprises everything that precedes Section 1, Scope. The introductory clauses describe how the publication is organized and provide commentary on the history and purpose of this standard. Normative information includes the actual requirements that must be satisfied by any data conforming to the standard. Annotative information is provided as rationale for and illustrations about the normative information and does not constitute part of the standard. The standard is completely stated even if all introductory and annotative information are removed. Annotative information may refer to terms introduced in later paragraphs or parts in order to tie together concepts, i.e., from time to time its value may increase after the reader has made a complete pass over the standard.

0.2 Different styles are used to distinguish between normative and annotative information.

0.3 All normative information is written using this same font, and is contained in either numbered paragraphs or numbered tables. Normative text is aligned with the left margin of the page.

0.4 All figures are annotative, and all annotative text is written in unnumbered paragraphs in Helvetica italic font, indented, as shown here:

> *This is a paragraph of annotative text. Its purpose is to include informal commentary on the normative information in immediately preceding or following paragraphs and may include references to or usage of normative information in other parts of the standard.*

1 Scope

This part describes the logical format of data recorded under API Recommended Practice 66. The logical format is a sequential, media-independent organization of 8-bit bytes used to represent data.

2 References

Unless otherwise specified, the most recent editions or revisions of the following standards, codes, and specifications shall, to the extent specified herein, form a part of this standard.

ANSI[1]

STD 754-1985	*IEEE Standard for Binary Floating Point Arithmetic*
X3.4	*American Standard Code for Information Interchange (ASCII)*

ISO[2]

8859-1	*Information processing–8-bit single-byte coded graphic character sets–Part 1: Latin alphabet No. 1*

[1] American National Standards Institute, 11 West 42nd Street, New York, New York 10036.
[2] International Organization for Standardization. ISO publications are available from ANSI.

3 Definitions

3.1 absent value: A placeholder that represents no value where a value is normally expected to be. For attributes, this may be an absent attribute component or a zero count. For channel samples, this is a zero count of the channel's dimension.

3.2 attribute: A named item of information or data pertaining to an object type.

3.3 attribute count: The number of elements in an attribute value.

3.4 attribute label: The name of an attribute.

3.5 attribute representation code: A code that identifies the recorded representation of each element of an attribute value.

3.6 attribute units: An expression that represents the units of measurement of each element of an attribute value.

3.7 attribute value: The value of an attribute. It may be present or absent. When present, it consists of zero or more elements, each having the same units and the same representation.

3.8 characteristic: A subitem of a component that contains one piece of information for a data item. Characteristics of an attribute component, for example, are its label, count, units, representation code, and value.

3.9 checksum: The integer result of a 16-bit cyclic redundancy computation on the bytes of a logical record segment, excluding the checksum result itself and any bytes (e.g., the logical record trailing length) that follow it. It is used to verify possible physical recording errors in the logical record segment.

3.10 component: A construct used to implement recording the parts of a set. It contains a descriptor that specifies which kind of data item the component represents and contains subitems called characteristics that describe and contain the value of the data item. A set consists of a sequence of components.

3.11 component descriptor: The part of a component that describes its role and which characteristics are present.

3.12 component format: A value in the component descriptor that indicates which characteristics are present.

3.13 component role: A value in the component descriptor that identifies what kind of data item the component represents, which include set, object, attribute, absent attribute, replacement set, redundant set.

3.14 compound representation code: A representation code that has two or more subfields.

3.15 consumer: The system or application program or company that reads or uses API Recommended Practice 66 information.

3.16 copy number: One of three parts of an object name. It is used to distinguish two objects of the same type in the same logical file that have the same origin and identifier.

3.17 dictionary: A database in which identifiers and reference values used under API Recommended Practice 66 are maintained and administered.

3.18 EFLR: Explicitly formatted logical record. The content of an EFLR, which consists of a set of objects, is determined from an analysis of the record itself.

3.19 element: One of a list of homogeneous quantities that make up the value of an attribute or channel. Every element of a value has the same units and representation.

3.20 explicitly formatted logical record: See EFLR.

3.21 format version: A two-byte field in the visible record header immediately following the length field that identifies the API Recommended Practice 66 version of the data in the record. The version is a one-byte unsigned integer, preceded by a byte containing the value FF_{16}.

3.22 identifier: One of the three parts of an object name. It is a character string used to distinguish the object from other objects of the same type. For some designated object types, the identifier conveys meaning of the nature of the object, and the identifier and its meaning are maintained in a dictionary.

3.23 IFLR: Indirectly formatted logical record. The content of an IFLR is determined from an analysis of related EFLRs.

3.24 indirectly formatted logical record: See IFLR.

3.25 logical file: The main unit of data exchange. It consists of a sequence of one or more logical records, beginning with a record containing a single FILE-HEADER object.

3.26 logical format: A description of how to encode data in a media-independent sequence of 8-bit bytes. This is the view of API Recommended Practice 66 that is independent of any physical binding.

3.27 logical record: An organization of data values into coherent, semantically-related packets of information. A logical record may have any length greater than sixteen bytes and is organized in one of two syntactic forms: explicitly-formatted logical record (EFLR) or indirectly-formatted logical record (IFLR).

3.28 logical record body: An ordered sequence of bytes representing the primary data of a logical record.

3.29 logical record segment: A construct that contains the structure necessary to describe and support the physical implementation of a logical record. A logical record is implemented as one or more logical record segments. A segment is wholly contained in a visible record, but two segments of the same logical record may be in different visible records.

3.30 logical record segment attributes: The 16 bits of binary data that describe the attributes of a logical record segment.

3.31 logical record segment body: One part of an ordered partition of a logical record body into disjoint parts. A logical record body can be reconstructed by concatenating all the segment bodies of the record.

3.32 logical record segment encryption packet: The part of a logical record header that contains information required to decrypt the segment.

3.33 logical record segment header: The part of a logical record segment that describes its length, attributes, and may contain an encryption packet.

3.34 logical record segment trailer: The part of a logical record segment that contains optional padding, checksum, and segment trailing length.

3.35 logical record structure: A logical record segment attribute. It specifies whether the logical record is an EFLR or IFLR.

3.36 object: A recorded instance of an object type.

3.37 object component: A component that delimits an object and contains its name.

3.38 object name: A three-part unique reference to an object consisting of an origin, a copy number, and an identifier.

3.39 object type: A logical entity of a schema that has a unique type name and one or more defined attributes. Instances of an object type are written in explicitly formatted logical records.

3.40 organization code: A number assigned by API to an organization that identifies the organization and represents schemas and dictionaries defined and administered by the organization.

3.41 origin: One of three parts of an object name. It is a number referring to a distinct ORIGIN object that contains context information for the objects that reference it.

3.42 pad count: The part of the padding (written at the end) that specifies how many padding bytes there are.

3.43 padding: Optional extra bytes following the logical record segment body used to ensure an even length for a segment and also to extend a segment when necessary to fill out a fixed-length visible record.

3.44 physical format: The medium-specific organization of data bytes on a storage unit.

3.45 producer: The system or application program or company that recorded information under API Recommended Practice 66.

3.46 redundant set: A verbatim copy of a set written previously in the same logical file.

3.47 replacement set: An updated copy of a set written previously in the same logical file. It includes any updates made to objects in the set since the original was written.

3.48 representation code: A unique number that identifies a standard encoding for a value as a sequence of one or more contiguous bytes.

3.49 set: A collection of one or more objects of the same object type. A set is recorded in an EFLR, and each EFLR has exactly one set.

3.50 set type: The type of objects in a set.

3.51 subfield: A part of a datum for which the representation is described by a simple (not compound) representation code. For example, the subfields of a datum having representation code OBNAME are, in order, an integer (UVARI), another integer (USHORT), and a string (IDENT).

3.52 template: An ordered group of one or more attributes that represent a default or prototype object, written at the beginning of a set.

3.53 visible record: The interface between the logical format and a medium-specific physical format. A visible record has a header, a body, and a trailing length.

4 Concepts

4.1 The API Recommended Practice 66 logical format is a description of how to encode data in a media-independent sequence of 8-bit bytes. At the highest level, the logical format is a sequence of logical files. A logical file is subdivided into a sequence of related logical records. A logical record contains a set of related values and is subdivided into a sequence of logical record segments. A logical record segment is contained in a visible record. A visible record has a header and trailer and may contain one or more segments. A visible record consists of a sequence of bytes mapped into some physical medium. The physical mapping of visible records into various media types is described in Part 3.

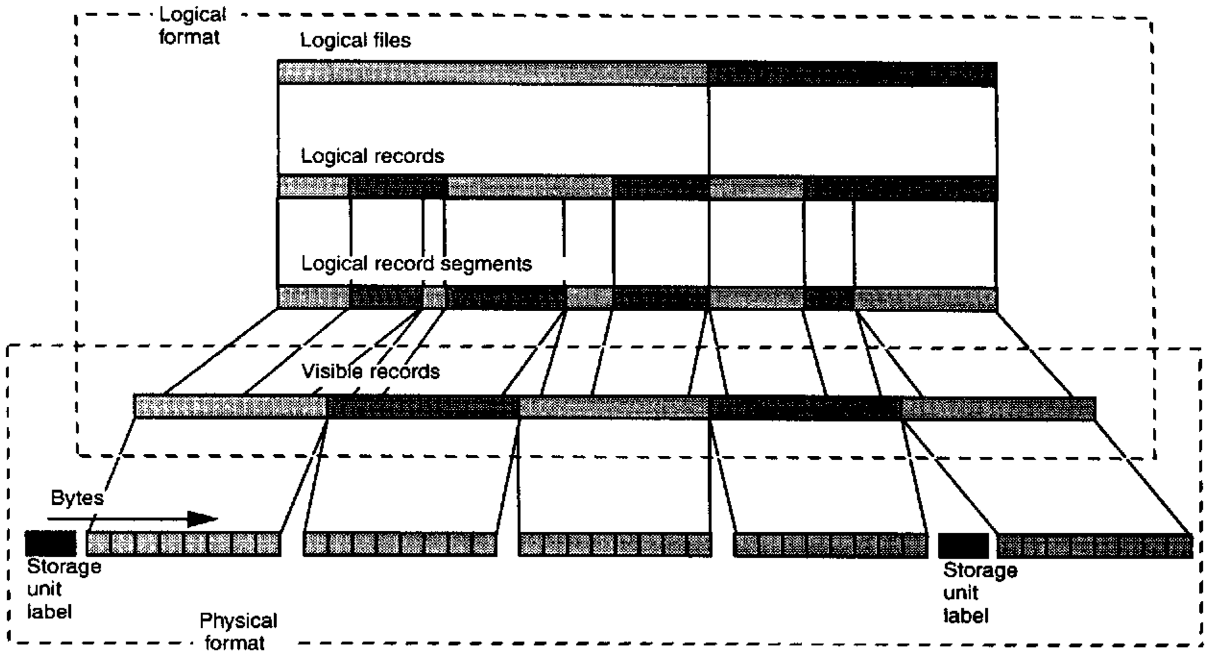


Figure 2-1—Conceptual View of a Logical Format (Bound to a Physical Format)

Figure 2-1 shows the transition from media-dependent organization of data (the physical format) to media-independent organization of data (the logical format). Notice that the two overlap at the visible record layer. Viewed from "above", the visible record layer is just a record-oriented organization of a stream of bytes, which looks the same to higher layers regardless of the medium used to record the bytes. Viewed from "below", the visible record layer is a mechanism that supports mapping logical constructs onto physical constructs such as device blocks and multiple storage units (e.g., tapes or disk files).

Logical record segmentation supports the flexible mapping of logical records (which may be of highly-varying sizes) into visible records. In particular, segments make it possible to begin a new logical record in the middle of a visible record. This results in two benefits. One, fixed-length visible records may be used. This provides a powerful recovery method in case corrupted data is encountered and also allows implementation of random access methods. Two, segmentation allows efficient packing of logical records into larger device blocks while also allowing spanning of large logical records across smaller device blocks.

4.2 A value, whether in a logical record or a visible record header or trailer, is encoded in one or more contiguous bytes according to one of a variety of standard representations. Each standard representation is identified by a unique integer code, called a representation code.

5 Representation Codes

5.1 A representation code is a unique number, specified in Table 2-12, that identifies a standard encoding for a value as a sequence of one or more contiguous bytes. Each representation code has a corresponding symbolic name used in this document to refer to the code. For example, the name UNORM refers to representation code 16, which identifies an encoding of an unsigned integer using two bytes. Only the encodings specified in this document may be used to encode values in API Recommended Practice 66. Only the representation codes specified here may be used to identify the encodings, and no representation codes may be redefined except by future editions of this document.

5.2 An encoding representation may be simple or compound. A simple representation is an encoding of a single atomic value such as an integer, decimal number, or character string. A compound representation is an encoding that is built up of previously-defined encodings and represents two or more simple values organized into a useful structure. The length of every representation is either fixed or may be determined from its content.

5.3 A subfield of a representation is any part of the representation that corresponds to a simple encoding (with an exception for representation code DTIME). Each subfield of a representation has a subfield name and subfield position, described in Table 2-13.

Representation code DTIME uses 4-bit fields to represent time zone and month.

Representation codes IDENT and ASCII include USHORT and UVARI values, respectively, to count number of characters in a string. However, there is no simple representation code corresponding to only the characters. Consequently, IDENT and ASCII are considered to be simple representations rather than compound.

5.4 The representation codes specified in Table 2-12 are divided into representation code classes. The codes in a representation code class are capable of representing the same kinds of values. Codes within a class are typically differentiated by amount of range or precision or alignment to a particular computing architecture.

For example, the codes UNORM (normal unsigned integer) and FDOUBL (double precision floating point) are in the same large class, since each is capable of representing a number. FDOUBLE has more range and precision than UNORM.

This organization of codes into classes is used when defining attributes for a schema (see Part 1). A representation code restriction may be specified for the attribute to indicate its possible range and precision, but any other code in the class may be used that supports the value being recorded.

6 Visible Record

6.1 Overview

6.1.1 Visible records are the interface between the logical format and a medium-specific physical format. They provide a structure that can be mapped into device structures and into which logical record segments may be packed.

6.1.2 A visible record is composed of three disjoint parts in the following order:

a. Visible record header.
b. Visible record body.
c. Visible record trailer.

6.1.3 The parts of a visible record are described in detail in 6.2 to 6.4.

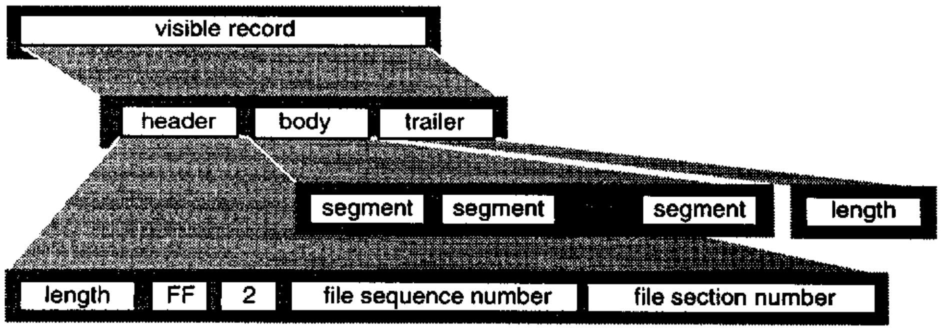


Figure 2-2—Visible Record Structure

6.2 Visible Record Header

6.2.1 The visible record header contains information about the length of the visible record, the format version being used, and the logical file in which the segments contained in the visible record belong.

6.2.2 Table 2-1 describes the fields of a visible record header in the order in which they appear.

Table 2-1 — Visible Record Header Fields

*Note	Field	Size in Bytes	Representation Code
1	Length	4	ULONG
2	The value FF_{16}	1	USHORT
3	Format version	1	USHORT
4	File sequence number	4	ULONG
5	File section number	2	UNORM

*Notes:
1. This field specifies the length in bytes of the visible record, including the header, body, and trailer. The maximum length of a visible record is limited by the largest even number having representation ULONG (see Table 2-12). This number is 4 294 967 294. The actual maximum length used for a visible record may depend on which medium the data is recorded (see Part 3). The minimum length of a visible record is the length of its header plus trailer plus the minimum length of a logical record segment (see Table 2-2).
2. This field contains the hexadecimal value FF. Its purpose is twofold: (1) to make the header length even, and (2) to provide a necessary (although not sufficient) identifier of a visible record.
3. The format version field contains the major API Recommended Practice 66 version of the data being recorded. For this edition of the standard, the format version value is 2.
4. This is the sequence number of the logical file to which the segments contained in the visible record body belong (see Part 6). Inclusion of this field implies the following requirement: All of the logical record segments contained in a visible record must belong to the same logical file.
5. This is the section number of the logical file to which the segments contained in the visible record body belong (see 10.1).

How visible records are used for various media types and various application purposes is described in Part 3. This includes how to specify variable-length or fixed-length visible records, for example.

By having a file sequence number in the visible record header, it is possible to implement rapid file access methods based on scanning only visible record headers. If visible records are fixed-length, for example, an application may compute forward or backward a given number of visible record lengths, jump to that location and determine by looking at the visible record header whether the jump landed inside or outside the current logical file.

In API Recommended Practice 66, Version 1, the visible record length was arbitrarily limited to 16,384 even though its representation (UNORM) supported lengths up to 65,534. This was done to ensure that implementations on small-memory processors would not be prevented from reading any API Recommended Practice 66 data. This was found to be a burdensome limitation and one that should be administered by the various user groups. So, in this edition of the standard, no arbitrary length limitation is imposed beyond what is supported by the ULONG representation.

6.3 Visible Record Body

The visible record body consists of one or more complete logical record segments (see 7.1.4).

6.4 Visible Record Trailer

The visible record trailer consists of a copy of the length field from the visible record header. This is always present.

7 Logical Record

7.1 Overview

7.1.1 Logical records organize values into coherent, semantically-related packets of information. A logical record may have any length greater than or equal to sixteen bytes and is organized in one of two syntactic forms: explicitly-formatted logical record (EFLR) or indirectly-formatted logical record (IFLR).

7.1.2 The content of an EFLR is determined from an analysis of the record itself. That is, an EFLR is self-descriptive in the sense that the type of, order of, and amount of information it contains may be determined by reading the record. Further information about EFLRs is given in 8.

7.1.3 The content of an IFLR is determined from an analysis of related EFLRs. The type of, order of, and amount of information in an IFLR is described by information in one or more EFLRs. A reference within each IFLR allows a reader to locate the data items within EFLRs that describe it. Many IFLRs may reference the same descriptor information. Further information about IFLRs is given in 9.

EFLRs and IFLRs are designed to handle different kinds of data, the differences resulting from either form or purpose. One of two views typically applies:

The first view is static *vs* dynamic. *Static data has a fixed or static value within the context of a logical file and is recorded in EFLRs. Dynamic data has a dynamically-changing value in a logical file, each instance of the value occurring in a different IFLR. For example, the description of a sensor is static, whereas the values acquired by the sensor over time are dynamic.*

The second view is parametric *vs* bulk. *Parametric data is complex (typically tabular) in form and of known extent, whereas bulk data is sequential in form and has an indefinite extent, i.e., the extent may not be known when data recording begins. Parametric data, e.g., computation parameters, is recorded in 4EFLRs and bulk data, e.g., seismic traces, is recorded in IFLRs.*

7.1.4 A logical record is implemented as one or more logical record segments. Segments contain the structure necessary to describe and support the physical implementation of a logical record. A segment is contained in a single visible record and is composed of up to four mutually-disjoint parts in the following order:

a. Logical record segment header.
b. Logical record segment encryption packet (only with encryption).
c. Logical record segment body.
d. Logical record segment trailer (optional).

7.1.5 The parts of a segment are described in detail in 7.2 through 7.8.

There is a very strict relation between segments and visible records. As stated earlier, a visible record body consists exactly of a whole number of logical record segments. Segments cannot span visible records, but logical records can and frequently do. It is also common for many logical records to fit into one large visible record, in which case each logical record normally has only one segment. A logical record will typically have more than one segment only when it cannot fit completely into the current visible record. The power of segments is that they allow complete separation of logical record and visible record alignment. The only exception to this alignment independence is with the first logical record of a logical file, which is described later.

Figure 2-3 and Figure 2-4 illustrate logical record segmentation.

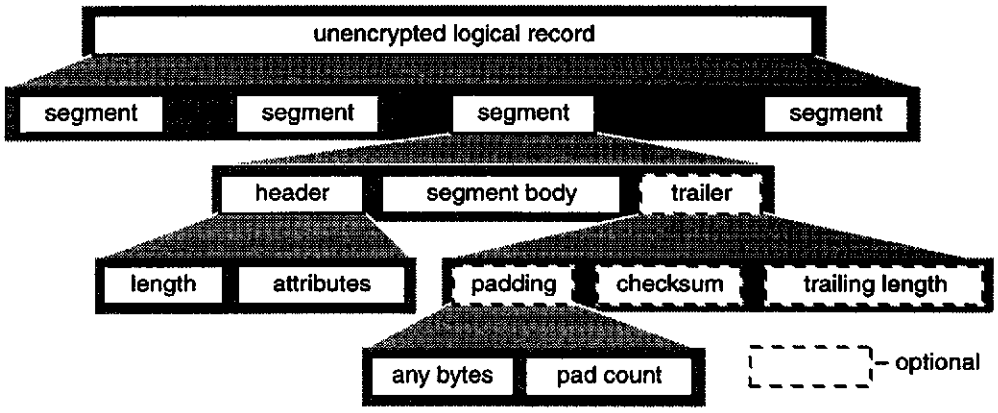


Figure 2-3—Logical Record Segmentation (Without Encryption)

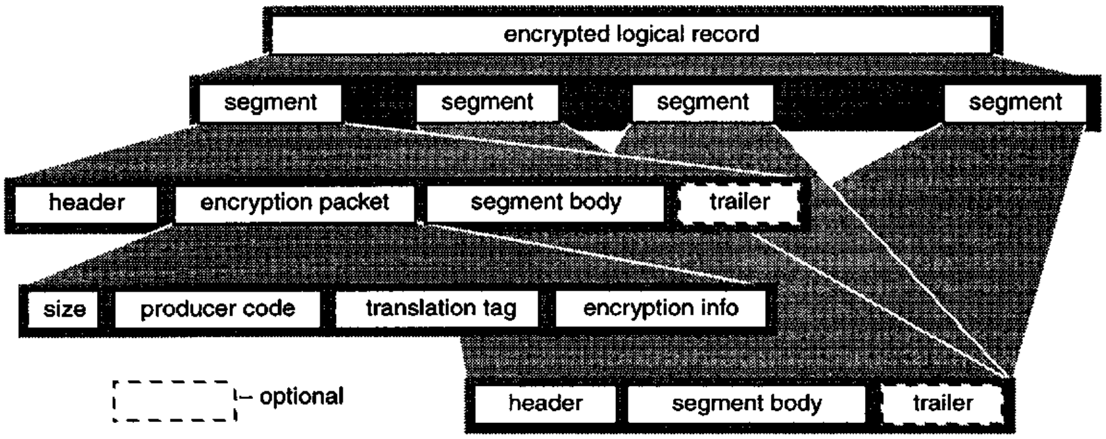


Figure 2-4—Logical Record Segmentation (With Encryption)

7.2 Logical Record Segment Header

7.2.1 The segment header describes the segment's length and attributes. The attributes indicate the kind of logical record, whether the segment is the first, middle, or last of the logical record, whether the segment is encrypted, and which optional parts of the segment are present.

7.2.2 Table 2-2 describes the fields of a logical record segment header in the order in which they appear.

Table 2-2 — Logical Record Segment Header Fields

*Note	Field	Size in Bytes	Representation Code
1	Length	4	ULONG
2	Attributes	2	(see note)

*Notes:

1. This field specifies the length in bytes of the segment, including the header, encryption packet, body, and trailer. A segment is required to have an even number of at least sixteen (16) or more bytes. The maximum length of a logical record segment is limited by the largest even number having representation ULONG (see Table 2-12). This number is 4 294 967 294.
2. Attributes are represented by bit values in a two-byte field, where bit 8 of each byte represents the highest-order bit and bit 1 represents the lowest-order bit (see Table 2-3).

The even length ensures that two-byte checksums can be computed. The minimum size, when combined with visible record overhead ensures a minimum physical block size required by some devices.

7.2.3 Table 2-3 describes the meanings of the bits in the logical record segment header attributes field. A bit is said to be set if its value = 1, and clear if its value = 0.

Table 2-3 — Logical Record Segment Header Attributes

*Note	Byte:Bit	Label	Value	Description
1	1:8	Logical record structure	0	Indirectly-formatted logical record.
			1	Explicitly-formatted logical record.
2	1:7	Predecessor	0	This is the first segment of the logical record.
			1	This is not the first segment of the logical record.
3	1:6	Successor	0	This is the last segment of the logical record.
			1	This is not the last segment of the logical record.
4	1:5	Encryption	0	The segment is not encrypted.
			1	The segment is encrypted.
5	1:4	Reserved	0	This bit is reserved for future use.
6	1:3	Checksum	0	The segment trailer does not have a checksum.
			1	The segment trailer has a checksum.
7	1:2	Trailing length	0	The segment trailer does not have a trailing length.
			1	The segment trailer has a trailing length.
8	1:1	Padding	0	The segment trailer does not have pad bytes.
			1	The segment trailer has pad bytes.
9	2:8–1	Reserved	0	These bits are reserved for future use.

*Notes:

1. All segments of a logical record must have the same value for the logical record structure bit.
2. When the predecessor bit is set, then the segment has a predecessor segment in the same logical record.
3. When the successor bit is set, then the segment has a successor segment in the same logical record.
4. All segments of a logical record must have the same value for the encryption bit. For rules on how to apply encryption, see 7.8.
5. This bit was previously used (in Version 1) to indicate presence of the encryption packet. In the current version, presence of the encryption packet is determined by the predecessor and encryption bits.
6. See Table 2-5 and 7.6.
7. See Table 2-5.
8. See Table 2-5.
9. Bits 2:1–8 must be clear. Meanings for these bits may be defined in later editions of this part.

7.3 Logical Record Segment Encryption Packet

7.3.1 The segment encryption packet immediately follows the segment header if and only if the predecessor bit is clear (no predecessor) and the encryption bit is set in the segment attributes. The packet includes its own length, who encrypted the segment, a translation tag, and may include information about the encryption method and decryption keys.

7.3.2 Table 2-4 describes the contents of an encryption packet.

Table 2-4 — Logical Record Segment Encryption Packet Fields

*Note	Field	Size in Bytes	Representation Code
1	Packet length	2	UNORM
2	Producer code	4	ULONG
3	Translation tag	V	OBNAME
4	Encryption information	V	(see note)

*Notes:

1. This is the length of the packet in bytes, including all fields. This field is required.
2. This is the organization code of the group responsible for the computer program that encrypted the record (see Appendix A). This field is required.
3. The translation tag is the name of an ORIGIN-TRANSLATION object. This field is required.
4. The encryption information is provided by the producer and has a representation known only to the producer (identified by the producer code field) of the logical record. It is used to assist producer-written computer programs in decrypting the logical record. It may consist of zero or more bytes.

The format of the encryption information is known only to the producer of the encrypted data. To the general reader, the encryption information field is just a group of bytes.

(The following discussion uses terminology from Part 6.) The purpose of the translation tag is to make it possible for "pass-through" copy-merge operations by organizations that encounter encrypted records they can't decrypt.

When data from two logical files is merged into a new logical file, some origin number collisions may occur (e.g., origin 3 is used in both input files). Such collisions are resolved by renaming the origin from one of the input files and all references to it in the output file. (This is called "origin translation".) Since origins in encrypted records can't be seen, they also cannot be translated. The translation tag in the encryption packet links this record with an optional ORIGIN-TRANSLATION object that has been recorded "in the clear" and that includes all the origins used in the encrypted record. When files are merged, the origins in the ORIGIN-TRANSLATION object, if provided, and the translation tag can be translated when copied. This makes it possible for later readers able to decrypt the merged data to figure out how to translate the origins in the decrypted record.

7.4 Logical Record Segment Body

The logical record segment body is an ordered sequence of zero or more bytes. The segment body immediately follows the encryption packet when the packet is present or immediately follows the segment header if the packet is not present. The length of the segment body is computed by subtracting the lengths of the encryption packet, the segment trailer, and the segment header from the segment length found in the header.

A zero-length segment body may be useful in the following situation:

A logical record is "edited" by replacing it with a revised version. If the replacement has less data than the original, then the replacement will require less space than the original. However, a replacement must cover the original space exactly. One way to handle this is to add padding. However, if multiple segments are required (e.g., the logical record spans visible records), then the replacement body data may be used up before writing the last segment. In this case, the last segment may have a zero-length segment body.

7.5 Logical Record Segment Trailer

7.5.1 The logical record segment trailer immediately follows the segment body. Its contents are optional and may include a segment trailing length, a checksum, and padding.

7.5.2 Table 2-5 describes the contents of the logical record segment trailer.

Table 2-5 — Logical Record Segment Trailer Fields

*Note	Field	Size in Bytes	Representation Code
1	Pad bytes	V	USHORT
1	Pad count	4	ULONG
2	Checksum	2	UNORM
3	Trailing length	4	ULONG

*Notes:

1. The pad count value is 4 (its own size) plus the number of pad bytes when padding is present (padding bit set in the segment attributes). There may be zero or more pad bytes in addition to the pad count. The pad count is present if and only if the padding bit is set in the segment attributes. Padding is used to ensure an even length for a segment and also to extend a segment when necessary to fill out a fixed-length visible record, and may be used for other reasons.
2. The checksum is a two-byte value computed according to the algorithm given in 7.6. It is present if and only if the checksum bit is set in the segment attributes. The computation applies to everything in the segment that precedes the checksum and does not include itself or the trailing length. If the segment is encrypted, the checksum is computed after encryption has been applied.
3. The trailing length is present if and only if the trailing length bit is set in the segment attributes. When present, it is a copy of the length field from the segment header. Trailing lengths must be used consistently for all segments in a logical file (see 10.2). That is, all segments in a logical file must have a trailing length, or all segments must have no trailing length.

The length of the trailer is T + C + P, where T is 4 if the trailing length is present and zero otherwise, C is 2 if the checksum is present and zero otherwise, and P is the value in the pad count if padding is present and zero otherwise.

When random access is supported by the device, and trailing lengths are present, it is possible for an implementation to read logical records from back to front.

7.6 Checksum Algorithm

The checksum, when present, is a 16-bit integer quantity computed using a cyclic-redundancy type checksum algorithm. This algorithm is described below. Note that it assumes that there are an even number of bytes in the data.

```
1) c=0                       initialize 16-bit checksum to
                             zero
2) loop i=1,n,2              loop over the data two bytes
                             at a time
3) t=byte(i+1)*256+byte(i)   compute a 16-bit addend by
                             concatenating the next two
                             bytes of data
4) c=c+t                     add the addend to the checksum
5) if carry c=c+1            add carry to checksum
6) c=c*2                     left shift checksum
7) if carry c=c+1            add carry to checksum
8) endloop
```

7.7 Logical Record Body

The logical record body is an ordered sequence of bytes representing the primary data of the logical record. The segment bodies comprise an ordered partition of the logical record body into disjoint parts. That is, the logical record body can be reconstructed by concatenating all the segment bodies of the record. The logical record body cannot be empty, i.e., at least one segment body must be non-empty. The format and rules for decoding information in the logical record body are described in 8 and 9.

7.8 Logical Record Encryption Rules

7.8.1 Encryption is applied at the logical record level. Data that would normally represent an unencrypted logical record body is encrypted by some method known by the data producer. This may involve expansion or compression of the data; that is, the number of encrypted bytes may be different from the number of unencrypted bytes. Logical record segmentation is then applied to the encrypted bytes, which constitute the recorded logical record body. No header, encryption packet, or trailer bytes (including padding) are encrypted.

7.8.2 When a logical record is encrypted, the encryption bit must be set in all segments. The first segment must have an encryption packet, and no other segments may have an encryption packet. Information required to decrypt the data (including use of any extraneous bytes to satisfy encryption blocking requirements) is placed in the encryption information field of the encryption packet.

Since encryption is applied without regard to segmentation, implementations may need to support encryption and decryption methods that require in-memory buffering of entire logical record bodies. Unencrypted records, on the other hand, can normally be accessed a segment at a time, and some implementations may wish to utilize this feature to conserve memory when processing very large logical records.

8 Explicitly-Formatted Logical Record (EFLR)

8.1 Overview

8.1.1 The data in an EFLR body (logical record body of an EFLR) is formatted using a general structure similar in nature to a table having a column heading, followed by any number of rows. The heading is called a template, and the rows are called objects. Within the template and within each object the column entries are called attributes. However, unlike a table, each object also has a name which is not considered to be the same as an attribute. The table (template plus objects) is called a set. Each EFLR body consists of exactly one set having at least one object, so the term set is a synonym for the term EFLR body.

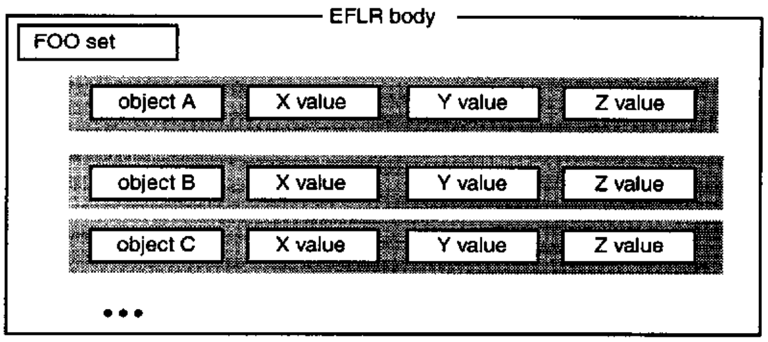


Figure 2-5—Application-Level View of a Set (Type = FOO)

API Recommended Practice 66 uses the terms set, template, object, and attribute instead of table, heading, row, and column entry to emphasize the object-oriented features of the data items in an EFLR. Although the structure *is "similar in nature to a table", there are enough differences from classical tables to warrant the newer terminology.*

8.1.2 Implementation of the component parts of a set is done using a construct called component. A component contains a descriptor that specifies which kind of data item the component represents and contains subitems called characteristics that describe and contain the value of the data item. A set, then, consists of a sequence of components.

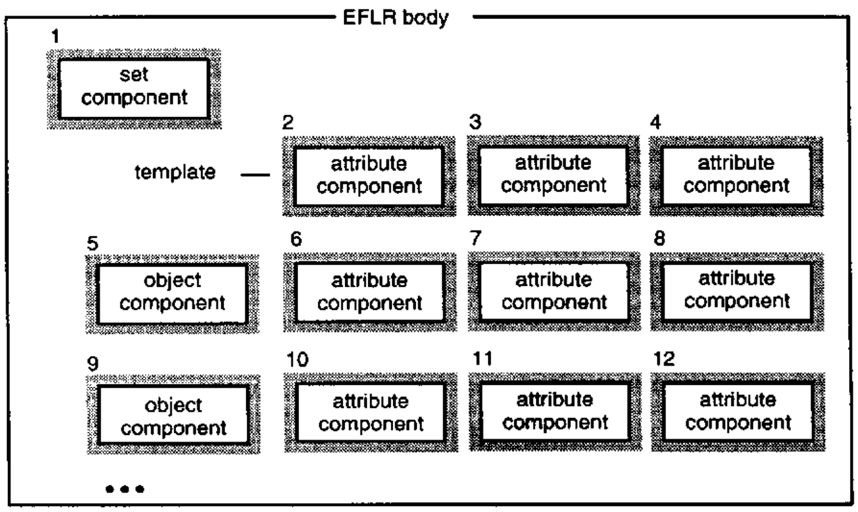


Figure 2-6—Component-Level View of a Set

8.1.3 The detailed description of set structure is given in 8.3.

8.2 Component

8.2.1 A component has a descriptor and zero or more characteristics. Table 2-6 describes the structure of a component. Bits are labeled 1 through 8, where 1 is the low-order bit.

Table 2-6 — Component Structure

*Note	Field	Size in Bytes
1	Descriptor	1
2	Characteristics	V

*Notes:
1. The descriptor of a component is subdivided into two disjoint bit fields. The first field, bits 8–6, represents the component's role (see Table 2-7). The second field, bits 5–1, represents the component's format (see 8.2.3).
2. A component has zero or more characteristics, as specified by the format subfield of the descriptor. The characteristics of a component identify, describe, and evaluate the data item represented by the component.

8.2.2 The component's role indicates what kind of data item the component represents and also determines which characteristics are represented by the format bits. Currently-defined component roles are described in Table 2-7.

Table 2-7 — Component Role

*Note	Bits 8 - 6	Symbolic Name	Type of Component
1	000	ABSATR	absent attribute
2	001	ATTRIB	attribute
3	010		
4	011	OBJECT	object
5	100		
6	101	RDSET	redundant set
7	110	RSET	replacement set
8	111	SET	normal set

*Notes:
1. See 8.6.
2. See 8.6.
3. This role is reserved and currently has no meaning. In API Recommended Practice 66, Version 1, this role represents an invariant attribute component. This component type is dropped in the current API Recommended Practice 66 version, since its use increases the complexity of implementations for very little benefit (at best one byte of storage savings per object when used).
4. See 8.5.
5. This role is reserved and currently has no meaning.
6. See 8.3.
7. See 8.3.
8. See 8.3.

8.2.3 The component's format specifies which of the characteristics for the given type of component are actually present in the component. Each component type (identified by its role) has a predefined group of characteristics that may occur in a predefined order. Each characteristic is represented by a bit in the format field of the descriptor. The characteristic is present in the component if and only if its bit is set. A global default value may be specified as part of the definition of a characteristic. This is a value that shall be assumed if the characteristic is not present. Characteristics immediately follow the descriptor in the same order as the format bits by which they are specified. There are no gaps for omitted characteristics.

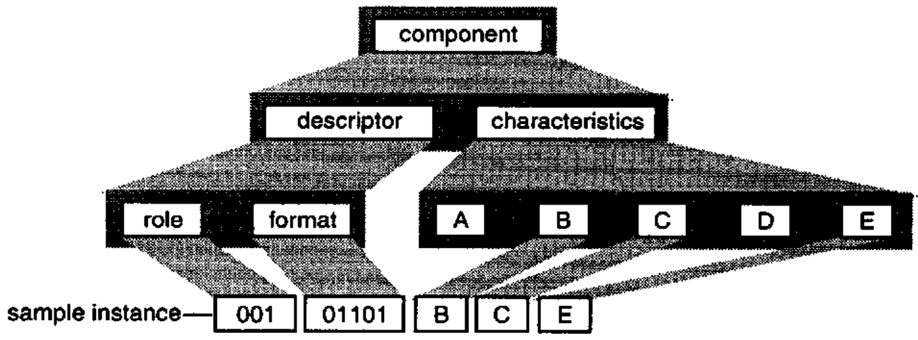


Figure 2-7—Component Structure

8.3 Sets

8.3.1 A set has a non-null type, a template, and one or more objects. It may also have a name and a count of its objects. The type identifies the type of objects in the set, and is also referred to as the object type of the set. The optional set name may be used to distinguish the set from other sets in the logical file when this is necessary and must be non-null when present.

8.3.2 Any number of sets in a logical file may have the same object type, but no two may have the same non-null set name.

8.3.3 There are three kinds of sets (see Table 2-7):

a. Normal set
b. Redundant set
c. Replacement set.

8.3.4 A normal set is as described in 8.3.1. It is the predominant kind of set.

8.3.5 A redundant set is an exact copy of a normal set written previously in the same logical file. Its purpose is to provide redundancy of information as insurance against possible data corruption (e.g., from media errors) of the normal set. The link between a redundant set and the normal set of which it is a copy is by means of the set name or type. If there is a redundant set, it and the normal set of which it is a copy must have the same non-null set name unless the normal set is the only one in the logical file having a given type. In the latter case, set name is optional. Any number of redundant sets may be written for a given normal set.

8.3.6 There is a mechanism using UPDATE objects (see Part 6) for modifying the value of an attribute previously written in a logical file. A replacement set is a near copy of a normal set previously written in the same logical file, the difference being that the replacement set reflects application of all updates that have been made between the time the normal set was recorded and the time the replacement set is recorded. The replacement set has the same objects as the normal set and the same attributes, but some attributes may have updated values. The link between a replacement set and its corresponding normal set is by means of the set name or type as described for redundant sets in 8.3.5. Any number of replacement sets may be written for a given normal set.

The purpose of replacement sets is to provide data replication when a logical file spans multiple storage units. If a normal set is used to interpret the format of an IFLR sequence (or affects computations using IFLR data), having a replacement set on each of multiple storage units ensures that loss of the initial storage unit will not prevent the ability to read data on continuation storage units.

Note that replacement sets are needed in lieu of redundant sets only when updates have been applied to data in the normal set.

8.3.7 A set's type, name, and object count are characteristics of a normal, redundant, or replacement set component. A normal, redundant, or replacement set component is the first component in the set, i.e., in the EFLR body. A set has exactly one normal, redundant, or replacement set component. Table 2-8 describes the format and characteristics of such components.

Table 2-8 — Normal, Redundant, and Replacement Set Components

*Note	FormatBit	Symbol	Characteristic	Representation Code	Global Default Value
1	5	t	type	TIDENT	(see note)
2	4	n	name	IDENT	null
3	3	c	count	ULONG	null
4	2 –1				

*Notes:

1. The type characteristic identifies the object type of the objects in the set and references the schema (see Part 1) under which the object type (i.e., its list of available attributes and their meanings) is defined. The schema is obtained from the ORIGIN:SCHEMA-CODE attribute (see Part 6) located using the tag subfield (see Table 2-13) of the characteristic. The object type is given by the identifier subfield of the characteristic. A non-null value (both subfields) of this characteristic must always be explicitly present. There is no global default value.
2. When present the name must be non-null. It is optional unless required to establish a link between redundant and normal sets (see 8.3.5) or between replacement and normal sets (see 8.3.6). No two normal sets in a logical file may have the same non-null name.
3. The count characteristic is optional. When present and non-null it specifies the number of objects in the set. A null value is valid and indicates a count has not been provided.
4. These bits are reserved. For the current format version, they are always clear (value = 0).

8.4 Templates

8.4.1 A template is an ordered group of one or more attributes that represent a default or prototype object. The template is recorded as one or more attribute components immediately following the set component. If a characteristic is omitted from an attribute component in the template, then a global default value is assumed. Table 2-10 specifies the global default values for attribute components.

8.4.2 The templates in two different sets having the same object type need not be the same. They may differ by the order in which attribute components are written and also may differ in the number and selection of attributes represented. The prototype object written in a template represents a view (or subset) of the attributes defined by a schema for a given object type. See Part 1 for a description of the methods used for defining a schema.

Whereas the object type is a name for the collection of attributes (i.e., column headings) that may apply to the objects of the set, the template contains the list of attributes actually used in the set, which may be a subset of the available attributes.

Note that a template may be empty, that is, a set may have objects with no attributes.

8.4.3 A template is terminated when the first object component is encountered. There must be at least one object component in any set.

8.5 Objects

8.5.1 An object has a name that uniquely identifies it within a logical file and has one or more attributes. The object name is a characteristic of an object component (see Table 2-9). The origin subfield of the object name identifies the schema (see Part 1) under which the object type is defined via the ORIGIN:SCHEMA-CODE attribute (see Part 6). It must identify the same schema identified by the tag subfield of the set type.

8.5.2 A schema may declare for each object type whether the identifier subfield of the object name is administered in a dictionary. If so, the ORIGIN:NAMESPACE-CODE and ORIGIN:NAMESPACE-NAME attributes identify the dictionary in which the identifier is included. The namespace assigns a permanent meaning to the identifier. When an object type has a dictionary-controlled identifier, the two-character prefix 'U-' may be used to indicate an identifier not in the dictionary, in effect overriding the dictionary control on an instance by instance basis. No dictionary-administered object identifier may begin with these characters.

8.5.3 An object is written as one object component immediately followed by zero or more attribute and/or absent attribute components. The attributes of an object correspond in order to the attributes in the template, and any characteristic omitted from an attribute component assumes the value of the corresponding characteristic from the template, or global default value if also omitted from the template. Any characteristic explicitly present in an object's attribute component overrides what is in the template. An object may not have more attributes than the template.

Although an object may be followed by zero attribute components, it still has one or more attributes due to inheritance from the template, which must have at least one attribute. It is invalid for an object to have all absent attributes, since this would violate 8.5.1.

8.5.4 Use of an absent attribute component indicates that the attribute in that position has been deleted from the object, i.e., does not even have a default value.

8.5.5 An object may have fewer attribute components than its template. In this case, the attributes for which components have been omitted assume all of the template default characteristics for the object. Since attribute order in the object must match attribute order in the template, only trailing components may be omitted, i.e., if a component is omitted for any attribute, then components must also be omitted for all subsequent attributes for the current object.

To use all template defaults, an "interior" attribute could have a component with only a descriptor, i.e., with no explicit characteristics.

Note the difference between an omitted attribute component and an absent attribute component. In the first case, the attribute exists and has value for the object, whereas in the second case, the attribute has no existence and no value for the object.

8.5.6 An object is terminated when the end of the logical record is reached or when another object component is encountered, whichever occurs first.

8.5.7 Table 2-9 describes the characteristics of an object component.

Table 2-9 — Object Component

*Note	Format Bit	Symbol	Characteristic	Representation Code	Global Default Value
1	5	n	name	OBNAME	(see note)
2	4 - 1				

*Notes:

1. The name characteristic uniquely identifies the object in a logical file and must be present. There is no global default value. To be unique in the logical file, the object name cannot match all three subfields–origin, copy number, identifier (see Table 2-13)–of any other object name in the logical file. The identifier subfield must be non-null.
2. These bits are reserved. For the current format version, they are always clear (value = 0).

8.6 Attributes

8.6.1 An attribute has five characteristics, a label, a count, a representation code, a unit (of measure), and a value. The first four characteristics identify and describe the value, which consists of count elements, each having the same unit and representation code. Attributes are represented by attribute components (see Table 2-10).

8.6.2 An absent attribute has no characteristics and is represented by an absent attribute component, which has all format bits clear. The purpose of an absent attribute component is to occupy a position for an attribute that is declared in the template but is logically omitted from its object. An absent attribute component may only appear in an object and must not appear in a template. When it appears in an object, it indicates that the attribute corresponding to its position (as declared in the template) is omitted from the object and has no value, no count, no unit, and no representation code.

8.6.3 Table 2-10 describes the characteristics of an attribute component.

Table 2-10 — Attribute Components

*Note	FormatBit	Symbol	Characteristic	RepresentationCode	Global Default Value
1	5	l	label	IDENT	(see note)
2	4	c	count	UVARI	1
2	3	r	representation code	USHORT	IDENT
2	2	u	unit	UNITS	null
3	1	v	value	(see note)	null

*Notes:

1. The label identifies an attribute. It is present if and only if the component is in the template. When present, the label must be non-null, and no two labels in the same template may be the same. There is no global default for the label.
2. The count, representation code, and unit describe the value. See note 3 for further details.
3. The value characteristic represents the value of the attribute. A value consists of zero or more ordered elements, where the number of elements is given by count. Every element has the same representation code and same unit, specified by representation code and unit. Inheritance and consistency rules for attribute characteristics are given in 8.6.4.

8.6.4 The primary consistency rule for attribute characteristics is that count and representation code must accurately describe the number and type of elements in value so that the component can be parsed correctly. When a characteristic is present in an attribute (its format bit is set), it has precedence over the corresponding characteristic in the template (or global default if the attribute is in the template). Consistency among characteristics is checked after inheritance, that is after template or global defaults are obtained for omitted characteristics. After inheritance, if count = 0, value must have zero elements. Count = 0 admits to two logical interpretations: If value is present, it is considered to be a present but empty list. If value is omitted, it is considered to be absent. After inheritance, if count > 0 value must have count elements. An inherited global default value is considered to match any count. An inherited template value must match the actual count after inheritance.

For example, if count = 3 and a global default value is inherited, then the attribute is considered to have 3 global default elements. However, it is inconsistent if 2 elements are inherited from the template when the attribute count is 3.

	template attribute	object attribute	conceptual view
label	WEIGHT		WEIGHT
count			1
representation code	FSINGL		FSINGL
unit	kg		kg
value		356.2	356.2 kg

	template attribute	object attribute	conceptual view
label	PROPERTIES		PROPERTIES
count		3	3
representation code		IDENT	IDENT
unit			''
value		'ROUND' 'TALL' 'BLUE'	'ROUND' 'TALL' 'BLUE'

	template attribute	object attribute	conceptual view
label	SIZE		SIZE
count	1	2	2
representation code	FSINGL	UNORM	UNORM
unit	in	mm	mm
value	11.6	255 184	255 mm 184 mm

In this example it would be invalid to omit value from the object attribute, since this would lead to inconsistencies between the template value and the object attribute's count, representation code and units.

	template attribute	object attribute	conceptual view
label	VALUES		VALUES
count	1	2	2
representation code		ULONG	ULONG
unit		mm	mm
value			0 mm 0 mm

In this example the template value would be interpreted as one unitless blank string (null IDENT), whereas the object attribute's value consists of 2 null ULONG elements having unit 'mm'. There is no inconsistency with the template, since the template defers to the global default for value.

Figure 2-8—Examples of Attribute Components

9 Indirectly-Formatted Logical Record (IFLR)

9.1 Overview

9.1.1 The logical record body of an IFLR is divided into three parts. The first part is a data descriptor reference (DDR) having representation code OBNAME. The second part is a modifier having representation code USHORT. The third part is a sequence of 8-bit bytes representing indirectly formatted data.

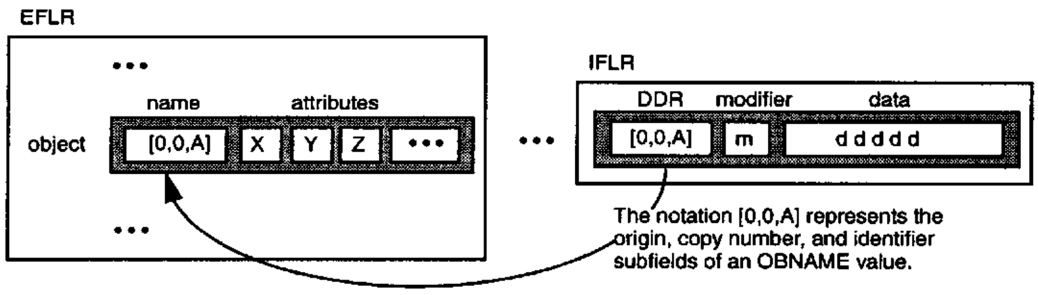

EFLR
•••
name
attributes
object
[0,0,A]
X
Y
Z
•••
•••
•••
IFLR
DDR
modifier
data
[0,0,A]
m
d d d d d
The notation [0,0,A] represents the origin, copy number, and identifier subfields of an OBNAME value.

Figure 2-9—Structure of an IFLR Body and Relation to its Data Descriptor

IFLRs are typically used to represent fixed-format records, where a potentially large number of records need to be written efficiently both in time and space. The data descriptor provides a one-time description of the fields in the record, including necessary index fields, and the DDR provides the key needed to sort out which format description goes with which record.

9.2 Data Descriptor Reference (DDR)

9.2.1 The DDR is the name of an object written previously in the same logical file. This data descriptor object contains information describing the format of the indirectly formatted data part of the IFLR. The data descriptor may reference other objects that include information about the format of the IFLR.

9.2.2 Any number of IFLRs in a logical file may have the same DDR. The sequence of all IFLRs in a logical file having the same DDR is called an IFLR type. There may be any number of IFLR types in a logical file, and there is no restriction on the order in which individual records are written. IFLRs and EFLRs may be interspersed, and records from one IFLR type may be interspersed with records of another IFLR type. The length of an unmodified IFLR depends only on the rules expressed by its data descriptor, and different records in the same IFLR type need not have the same length unless so constrained by the data descriptor.

Being writable in any order means there are no restrictions about writing EFLRs between IFLRs and vice-versa or about mixing IFLRs from different IFLR types. However, the relative sequence of records in an IFLR type typically is important and may be reflected by the required behavior of various index fields. Some descriptors may define an implicit index based on position. For example, the records of an IFLR type may represent data values on a grid. Which point of the grid is evaluated for a given record might be computed as a function of the record's sequential position in the IFLR type.

9.3 IFLR Modifier

9.3.1 The modifier of an IFLR may be used to modify its use as described in Table 2-11.

Table 2-11 — IFLR Modifier Options

Value	Description
0	The IFLR is unmodified and is used as described in 9.2.
1	The IFLR is used as an end of data marker for an IFLR type. An end of data marker has only a DDR and modifier and no data. An end of data marker indicates the end of an IFLR type. That is, no unmodified IFLRs shall follow an end of data marker having the same DDR. Using an end of data marker is optional for an IFLR type. When used, any number of them may be written.
2 - 255	Reserved.

The predecessor of end of data marker in API Recommended Practice 66, Version 1, is the type 127 EOD record. The purpose of an end of data marker is to inform the reader when no more IFLRs of a given type will be seen. This allows the reader to stop looking for the "next" such IFLR, which may be very costly if a large amount of other data remains in the logical file.

If the reader employs random access methods to skip over records when seeking an IFLR having a specified value or index, there is a chance its end of data marker may also be skipped. By writing end of data markers redundantly every so often, if one is missed another may be found before the reader has gone very far.

10 Logical File

10.1 Overview

10.1.1 A logical file is a sequence of logical records that represent a coherent dataset. Its purpose is to support the recording of a wide variety of kinds of data and to provide a context within which its data can be identified. Because it is composed of logical records, a logical file may be very small or arbitrarily large, and is not limited by its storage medium. It is possible to pack many logical files sequentially onto a single physical storage unit (see Part 3) or to span a logical file across two or more storage units.

Care is taken in this document to distinguish the term "logical file" from the term "file". The latter typically refers to a physical file on disk and sometimes to the data on a tape delimited by two tape marks. On traditional magnetic tapes logical files are typically also physical files, although there will be at least one physical file, namely the storage unit label, that is not a logical file. A disk file, on the other hand, may contain one or more logical files and will always have a storage unit label. Users may choose how to organize data, and on disk the choice may be to have one logical file per disk file.

Although putting the word "logical" in front of "file" all the time may seem awkward, the distinction is important.

See Figure 2-1 for how logical files are related to logical and visible records.

10.1.2 The complete description of logical files and the rules that govern them is distributed among Parts 1, 2, 3, and 6 of this document. This part provides information only on how logical files are related to the logical format layer of API Recommended Practice 66.

Most of the rules about a logical file concern its content, which is described in detail in Part 6. The rules discussed here pertaining to the logical format layer are relatively minimal.

10.2 Logical File Organization

10.2.1 The beginning of a logical file is determined from the content of its first logical record, which must be an EFLR. The specific object type of the set it contains is described in Part 6. Otherwise, a logical file consists of a sequence of arbitrarily many logical records until another logical file is encountered or until the sequence of logical records is exhausted.

10.2.2 Every logical record segment in a visible record must belong to the same logical file. It follows that the first logical record of a logical file begins a new visible record.

There are several reasons for this rule. One is that the inclusion of file sequence and file section numbers in the visible record header requires that all segments belong to the same logical file. Another is that it makes it possible to locate the beginning of a logical file efficiently, since only the first segment in each visible record needs to be examined. A third reason is the compatibility with traditional tape formats that is achieved by mapping visible records one-to-one onto tape blocks and then using tape marks to indicate blocks at which logical files begin.

11 Representation Code Descriptions

11.1 Introduction

Representation codes are summarized in Table 2-12. Bit-level descriptions of encodings for simple representation codes are given in 11.3. Subfield-level descriptions of compound representation codes are given in Table 2-13.

11.2 Representation Code Summary

Table 2-12 summarizes the representation codes defined for use under API Recommended Practice 66, ordered by code number. Under the Type column, S indicates a simple representation and C indicates a compound representation.

Table 2-12 — Representation Code Summary

Code	Symbolic Name	Description	Class	Type	Size in Bytes
1	FSHORT	Low precision floating point	NUMBER	S	2
2	FSINGL	IEEE single precision floating point	NUMBER	S	4
3	FSING1	Validated single precision floating point	BALANCED-INTERVAL	C	8
4	FSING2	Two-way validated single precision floating point	UNBALANCED-INTERVAL	C	12
5	ISINGL	IBM single precision floating point	NUMBER	S	4
6	VSINGL	VAX single precision floating point	NUMBER	S	4
7	FDOUBL	IEEE double precision floating point	NUMBER	S	8
8	FDOUB1	Validated double precision floating point	BALANCED-INTERVAL	C	16
9	FDOUB2	2-way validated double precision floating point	UNBALANCED-INTERVAL	C	24
10	CSINGL	Single precision complex	COMPLEX	C	8
11	CDOUBL	Double precision complex	COMPLEX	C	16
12	SSHORT	Short signed integer	NUMBER	S	1
13	SNORM	Normal signed integer	NUMBER	S	2
14	SLONG	Long signed integer	NUMBER	S	4
15	USHORT	Short unsigned integer	NUMBER	S	1
16	UNORM	Normal unsigned integer	NUMBER	S	2
17	ULONG	Long unsigned integer	NUMBER	S	4
18	UVARI	Variable-length unsigned integer	NUMBER	S	1, 2, or 4
19	IDENT	Variable-length identifier	STRING	S	V
20	ASCII	Variable-length ASCII character string	STRING	S	V
21	DTIME	Date and time	TIME	C	8
22	ORIGIN	Origin reference	ORIGIN	S	V
23	OBNAME	Object name	REFERENCE	C	V
24	OBJREF	Object reference	REFERENCE	C	V
25	ATTREF	Attribute reference	ATTRIBUTE	C	V
26	STATUS	Boolean status	STATUS	S	1
27	UNITS	Units expression	UNIT	S	V
28	RNORM	Rational	RATIO	C	4
29	RLONG	Long rational	RATIO	C	8
30	ISNORM	Inverted order normal signed integer	NUMBER	S	2
31	ISLONG	Inverted order long signed integer	NUMBER	S	4
32	IUNORM	Inverted order normal unsigned integer	NUMBER	S	2
33	IULONG	Inverted order long unsigned integer	NUMBER	S	4
34	IRNORM	Inverted order rational	RATIO	S	4
35	IRLONG	Inverted order long rational	RATIO	S	8
36	TIDENT	Tagged IDENT	TAG-STRING	C	V
37	TUNORM	Tagged UNORM	TAG-NUMBER	C	3, 4, or 6
38	TASCII	Tagged ASCII	TAG-STRING	C	V
39	LOGICL	Logical	STATUS	S	1
40	BINARY	Binary	BINARY	S	V
41	FRATIO	Floating point ratio	RATIO	C	8
42	DRATIO	Double precision ratio	RATIO	C	16

11.3 Descriptions of Simple Representation Codes

Simple representation codes are described here in terms of their bit-level structures. With each description, the null value and one or more examples are provided. Bits are labelled 1 to 8, where bit 1 represents the low-order bit of the byte, and bit 8 represents the high-order bit.

The bit-labeling convention here is the opposite of the convention used in API Recommended Practice 66, Version 1. The change was made in order to match the conventions used by cited standards.

11.3.1 ASCII

A variable-length character string consisting of N characters from the 7-bit ASCII (ANSI X3.4) or the ISO 8859-1 character sets, preceded by the number N represented as a UVARI. The number N may be any value representable as a UVARI. If a null character (0) is present, then only those characters that precede the first null are considered to be part of the actual string value.

Byte	1 to k	(k+1) to (k+N)
Field	N (UVARI)	7-bit ASCII or ISO 8859-1 characters

Null value (N = 0)

= 00000000_2

Sample value '$ / £'

```
= 00000101 00100100 00100000 00101111 00100000 10100011₂
      5        $       (sp)       /      (sp)       £
```

Note that ASCII and ISO 8859-1 characters are the same where both sets overlap. This is typically called the G0 graphic character set. ASCII includes control characters that are not part of 8859-1, and 8859-1 includes a G1 graphic character set using bit 8 that is not part of ASCII.

The use of a null character as a string delimiter allows transparent string padding, i.e., writing more characters than are in the actual string data. This makes it possible to write variable-length string data in fixed-length fields (e.g., in frames) while preserving the extent of the actual string value.

11.3.2 BINARY

A variable-length bit string. The bits are written in N – 1 bytes (when N > 1), starting at bit 8 of the first byte. The last byte contains P < 8 trailing pad bits that are not part of the bit string. The number of pad bits, P, is recorded immediately preceding the first bit string byte. The total number of bytes N used to record P and the bit string is recorded immediately preceding P. The value N = 0 corresponds to the null bit string (no bits), in which case the byte containing P is omitted. The value N = 1 is invalid, since it would represent an alternate null value. The total number of bits in the bit string is:

$\#bits = 8 * (N - 1) - P$, when $N > 1$.

Since P < 8, a bit string is written in the minimum number of bytes.

Byte	1 to k	k + 1	(k+2) to (k+N)
Field	N (UVARI)	P (USHORT)	Bit string, left justified

Null value (no bits)

= 00000000_2

Sample value 0011101011011011001

```
= 00000100 00000101 00111010 11011011 00100000₂
      4        5           bit string
```

11.3.3 FDOUBL

Double precision floating point format as defined in ANSI Document *STD 754-1985: IEEE Standard for Binary Floating Point Arithmetic.*

Byte	1								2							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	–1	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}	2^{-1}	2^{-2}	2^{-3}	2^{-4}
Field	S	E											M			
Byte	3								4							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}	2^{-16}	2^{-17}	2^{-18}	2^{-19}	2^{-20}
Field	M (cont)															
Byte	5								6							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	2^{-21}	2^{-22}	2^{-23}	2^{-24}	2^{-25}	2^{-26}	2^{-27}	2^{-28}	2^{-29}	2^{-30}	2^{-31}	2^{-32}	2^{-33}	2^{-34}	2^{-35}	2^{-36}
Field	M (cont)															
Byte	7								8							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	2^{-37}	2^{-38}	2^{-39}	2^{-40}	2^{-41}	2^{-42}	2^{-43}	2^{-44}	2^{-45}	2^{-46}	2^{-47}	2^{-48}	2^{-49}	2^{-50}	2^{-51}	2^{-52}
Field	M (cont)															

Value

= $(-1)S * (1 + M) * 2^{E-1023}$, $0 < E < 2047$ [normalized]

= $(-1)S * M * 2^{-1022}$, $E = 0$, $M \neq 0$ [denormalized]

= $(-1)S * 0$, $E = 0$, $M = 0$

= $(-1)S * \infty$, $E = 2047$, $M = 0$

= NaN (Not a Number), $E = 2047$, $M \neq 0$

Null value 0

= 00000000 00000000 00000000 00000000

$00000000\ 00000000\ 00000000\ 00000000_2$

Sample value 153

= $231_8 = (1 + .144_8) * 2^{1030-1023}$

= 01000000 01100011 00100000 00000000

$00000000\ 00000000\ 00000000\ 00000000_2$

Sample value –153

= $-231_8 = -(1 + .144_8) * 2^{1020-1023}$

= 11000000 01100011 00100000 00000000

$00000000\ 00000000\ 00000000\ 00000000_2$

11.3.4 FSHORT

2's complement 12–bit fractional mantissa with 4–bit unsigned integer exponent.

Byte	1								2							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	–1	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{3}	2^{2}	2^{1}	2^{0}
Field	M												E			

Value

$= M * 2^E$

Null value 0

$= 00000000\ 00000000_2$

Sample value 153

$= 231_8 = .462_8 * 2^8$

$= 01001100\ 10001000_2$

Sample value −153

$= -231_8 = (-1 + .316_8) * 2^8$

$= 10110011\ 10001000_2$

11.3.5 FSINGL

Single precision floating point format as defined in ANSI Document *STD 754-1985: IEEE Standard for Binary Floating Point Arithmetic.*

Byte	1								2							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	−1	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}
Field	S	E								M						
Byte	3								4							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	2^{-8}	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}	2^{-16}	2^{-17}	2^{-18}	2^{-19}	2^{-20}	2^{-21}	2^{-22}	2^{-23}
Field	M (cont)															

Value

$= (-1)^S * (1 + M) * 2^{E-127},\ 0 < E < 255$ [normalized]

$= (-1)^S * M * 2^{-126},\ E = 0,\ M \neq 0$ [denormalized]

$= (-1)^S * 0,\ E = 0,\ M = 0$

$= (-1)^S * \infty,\ E = 255,\ M = 0$

= NaN (Not a Number), $E = 255,\ M \neq 0$

Null value 0

$= 00000000\ 00000000\ 00000000\ 00000000_2$

Sample value 153

$= 231_8 = (1 + .144_8) * 2^{134-127}$

$= 01000011\ 00011001\ 00000000\ 00000000_2$

Sample value −153

$= -231_8 = -(1 + .144_8) * 2^{134-127}$

$= 11000011\ 00011001\ 00000000\ 00000000_2$

11.3.6 IDENT

A variable-length character string consisting of N characters from a subset of the 7-bit ASCII character set, immediately preceded by the number N represented as a USHORT. The number N may be any value representable as a USHORT. The valid character subset consists of null (0) plus the codes 33_{10} (!) to 96_{10} (`) and from 123_{10} ({) to 126_{10} (~) inclusive. This excludes all control characters, all "white space", and the lower-case alphabet. Two IDENT values match if and only if they have the same number of characters and the corresponding characters match. If a null character (0) is present, then only those characters that precede the first null are considered to be part of the actual string value.

The purpose for IDENT is primarily for labels and other identifiers that undergo matching. Restriction to upper case allows implementations to avoid case conversion prior to matching. Exclusion of white space helps prevent visual ambiguity.

The use of a null character as a string delimiter allows transparent string padding, i.e., writing more characters than are in the actual string data. This makes it possible to write variable-length string data in fixed-length fields (e.g., in frames) while preserving the extent of the actual string value.

Byte	1	2 to (1 + N)
Field	N (USHORT)	7-bit ASCII characters

Null value (N = 0)

```
= 00000000₂
```

Sample value 'TYPE1'

```
= 00000101 01010100 01011001 01010000 01000101 00110001₂
      5        T        Y        P        E        1
```

11.3.7 ISINGL

Single precision floating point format represented as a 24-bit fractional mantissa with a leading sign bit and a 7-bit excess 64 integer exponent, base 16. Bits 8–5 of byte 2 may not be all zero except for true zero.

Byte	1								2							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	–1	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
Field	S	E							M							
Byte	3								4							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}	2^{-16}	2^{-17}	2^{-18}	2^{-19}	2^{-20}	2^{-21}	2^{-22}	2^{-23}	2^{-24}
Field	M (cont)															

Value

$= (-1)^S * (1 + M) * 16^{E-64}$

= 0 (true zero), S = E = M = 0

Null value 0

$= 00000000\ 00000000\ 00000000\ 00000000_2$

Sample value 153

$= 231_8 = .462_8 * 16^{66-64}$

$= 01000010\ 10011001\ 00000000\ 00000000_2$

Sample value –153

$= -231_8 = -.462_8 * 16^{66-64}$

$= 11000010\ 10011001\ 00000000\ 00000000_2$

11.3.8 ISLONG

Derived from SLONG by inverting byte order (1 2 3 4) to (4 3 2 1).

Byte	4								3							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	-2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
Field	I															
Byte	2								1							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
Field	I (cont)															

Value

= I

Null value 0

= $00000000\ 00000000\ 00000000\ 00000000_2$

Sample value 153

= 231_8

= $10011001\ 00000000\ 00000000\ 00000000_2$

Sample value −153

= $-231_8 = (-2^{31} + 231_8)$

= $01100111\ 11111111\ 11111111\ 11111111_2$

11.3.9 ISNORM

Derived from SNORM by inverting byte order (1 2) to (2 1).

Byte	2								1							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	-2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
Field	I															

Value

= I

Null value 0

= $00000000\ 00000000_2$

Sample value 153

= 231_8

= $10011001\ 00000000_2$

Sample value −153

= $-231_8 = (-2^{15} + 231_8)$

= $01100111\ 11111111_2$

11.3.10 IULONG

Derived from ULONG by inverting byte order (1 2 3 4) to (4 3 2 1).

Byte	4								3							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
Field	I															
Byte	2								1							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
Field	I (cont)															

Value

= I

Null value 0

= 00000000 00000000 00000000 00000000_2

Sample value 153

= 231_8

= 10011001 00000000 00000000 00000000_2

11.3.11 IUNORM

Derived from UNORM by inverting byte order (1 2) to (2 1).

Byte	2								1							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
Field	I															

Value

= I

Null value 0

= 00000000 00000000_2

Sample value 153

= 231_8

= 10011001 00000000_2

11.3.12 LOGICL

A three-state boolean.

Byte	1
Field	S (SSHORT)

Value

= TRUE = ALLOWED = ON, if S = 1

= FALSE = DISALLOWED = OFF, if S = 0

= UNKNOWN, if S = -1

Null value FALSE

= 00000000_2

11.3.13 ORIGIN

The representation of an origin (reference) is equivalent to UVARI. The value is an integer that matches the origin subfield of the object name of an ORIGIN object (see Part 6) in the same logical file.

The representation code acts as a tag to identify values that may need origin translation when data is merged (see Table 2-4).

Byte	1 – k
Field	O (UVARI)

Value

= O

Null value 0

= 00000000_2

11.3.14 SLONG

Two's complement four-byte integer.

Byte	1								2							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	-2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
Field	I															
Byte	3								4							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
Field	I (cont)															

Value

= I

Null value 0

= $00000000\ 00000000\ 00000000\ 00000000_2$

Sample value 153

= 231_8

= $00000000\ 00000000\ 00000000\ 10011001_2$

Sample value –153

= $-231_8 = (-2^{31} + 231_8)$

= $11111111\ 11111111\ 11111111\ 01100111_2$

11.3.15 SNORM

2's complement two-byte integer.

Byte	1								2							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	-2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
Field	I															

Value

= I

Null value 0

= $00000000\ 00000000_2$

Sample value 153

= 231_8

= $00000000\ 10011001_2$

Sample value −153

= $-231_8 = (-2^{15} + 77547_8)$

= $11111111\ 01100111_2$

11.3.16 SSHORT

2's complement one-byte integer.

Byte	1							
Bit	8	7	6	5	4	3	2	1
Meaning	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Field	I							

Value

= I

Null value 0

= 00000000_2

Sample value 89

= 131_8

= 01011001_2

Sample value −89

= $-131_8 = (-2^7 + 47_8)$

= 10100111_2

11.3.17 STATUS

A two-state boolean.

Byte	1
Field	S (SSHORT)

Value

= TRUE = ALLOWED = ON, if S = 1

= FALSE = DISALLOWED = OFF, if S = 0

Null value FALSE

= 00000000_2

11.3.18 ULONG

Unsigned four-byte integer.

Byte	1								2							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	2^{31}	2^{30}	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
Field	I															
Byte	3								4							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
Field	I (cont)															

Value

= I

Null value 0

= $00000000\ 00000000\ 00000000\ 00000000_2$

Sample value 153

= 231_8

= $00000000\ 00000000\ 00000000\ 10011001_2$

11.3.19 UNITS

This representation is identical to ASCII. It is used to contain a character string representing a unit of measure. When used in the unit characteristic of an attribute, the unit model under which the unit is defined is identified in the ORIGIN object referenced by the attribute's object (see Part 6). If the attribute is in a template, the unit model is identified in the ORIGIN object referenced by the set type in the set component. When used in other values, the mechanism for locating the ORIGIN that identifies the unit model must be specified as part of the definition of the data item to which the value belongs.

11.3.20 UNORM

Unsigned two-byte integer.

Byte	1								2							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^{9}	2^{8}	2^{7}	2^{6}	2^{5}	2^{4}	2^{3}	2^{2}	2^{1}	2^{0}
Field	I															

Value

= I

Null value 0

= $00000000\ 00000000_2$

Sample value 153

= 231_8

= $00000000\ 10011001_2$

11.3.21 USHORT

Unsigned one-byte integer.

Byte	1							
Bit	8	7	6	5	4	3	2	1

Meaning	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Field	I							

Value

= I

Null value 0

= 00000000_2

Sample value 217

= 331_8

= 11011001_2

11.3.22 UVARI

An unsigned binary integer in the range 0 – (2^{30} – 1) can be represented by means of 1, 2, or 4 bytes using this representation code. Bits 8 and 7 of byte 1 indicate the number of bytes used to represent the value. A one-byte representation is indicated when bit 8 = 0. A two-byte representation is indicated when bit 8 = 1 and bit 7 = 0. A four-byte representation is indicated when bit 8 = bit 7 = 1.

Byte	1							
Bit	8	7	6	5	4	3	2	1
Meaning	0	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Field		I						

Value

= I

Range

= 0 to (2^7 - 1)

Byte	1								2							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	1	0	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Field			I													

Value

= I

Range

= 2^7 to (2^{14} - 1)

Byte	1								2							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	1	1	2^{29}	2^{28}	2^{27}	2^{26}	2^{25}	2^{24}	2^{23}	2^{22}	2^{21}	2^{20}	2^{19}	2^{18}	2^{17}	2^{16}
Field			I													
Byte	3								4							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	2^{15}	2^{14}	2^{13}	2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Field	I (cont)															

Value

```
= I
```

Range

$= 2^{14}$ to $(2^{30} - 1)$

11.3.23 VSINGL

This representation corresponds to the VAX F-floating format, with bytes ordered according to increasing internal address. That is, when loaded in memory byte 1 goes into address A, byte 2 into address A+1, byte 3 into A+2, and byte 4 into A+3. As usual, when written in API Recommended Practice 66 format, byte 1 is written first, followed in order by bytes 2, 3, and 4.

Caution: Note the unusual byte ordering in this diagram.

Byte	2								1							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	−1	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
Field	S	E								M						
Byte	4								3							
Bit	8	7	6	5	4	3	2	1	8	7	6	5	4	3	2	1
Meaning	2^{-9}	2^{-10}	2^{-11}	2^{-12}	2^{-13}	2^{-14}	2^{-15}	2^{-16}	2^{-17}	2^{-18}	2^{-19}	2^{-20}	2^{-21}	2^{-22}	2^{-23}	2^{-24}
Field	M (cont)															

Value

$= (-1)^S * (0.5 + M) * 2^{E-128},\ E > 0$

$= 0,\ E = 0,\ S = 0,\ M =$ arbitrary

= undefined and invalid for $E = 0,\ S = 1$

Null value 0

$= 00000000\ 00000000\ 00000000\ 00000000_2$

Sample value 153

$= 231_8 = (0.5 + .062_8) * 2^{136-128}$

$= 00001100\ 01000100\ 00000000\ 10000000_2$

Sample value −153

$= -231_8 = -(0.5 + .062_8) * 2^{136-128}$

$= 00001100\ 11000100\ 00000000\ 10000000_2$

11.4 Descriptions of Compound Representation Codes

Table 2-13 describes compound representation codes in terms of their subfields. This table completely describes the syntactic structure of the corresponding representations in terms of simple representation codes. Unless stated otherwise below, the null value of a compound representation code is obtained by using the null value for each of its subfields. Examples and supplemental explanations for selected codes follow the table.

Table 2-13 — Compound Representation Code Descriptions

Symbolic Name	Subfield Position	Subfield Name	Subfield Code	Description
ATTREF	1	type	IDENT	Object type name
	2	origin	ORIGIN	Origin containing schema code and identifier namespace code
	3	copy	UVARI	Copy number
	4	identifier	IDENT	Object identifier
	5	label	IDENT	Attribute label
CDOUBL	1	real	FDOUBL	Real part
	2	imaginary	FDOUBL	Imaginary part
CSINGL	1	real	FSINGL	Real part
	2	imaginary	FSINGL	Imaginary part
DRATIO	1	numerator	FDOUBL	Numerator of ratio
	2	denominator	FDOUBL	Denominator of ratio (> 0)
DTIME	1	year	USHORT	Years since 1900
	2	timezone	4-bit unsigned integer	Time zone: 0=local standard time, 1=local daylight savings time, 2 = Universal Coordinated Time (Greenwich Mean Time)
	3	month	4-bit unsigned integer	Month of the year (1 to 12)
	4	day	USHORT	Day of the month (1 to 31)
	5	hour	USHORT	Hours since midnight (0 to 23)
	6	minute	USHORT	Minutes past the hour (0 to 59)
	7	second	USHORT	Seconds past the minute (0 to 59)
	8	millisecond	UNORM	Milliseconds past the second (0 to 999)
FDOUB1	1	value	FDOUBL	Nominal value V of interval [V – B, V + B]
	2	bound	FDOUBL	Interval bound, B (≥ 0)
FDOUB2	1	value	FDOUBL	Nominal value V of interval [V – A, V + B]
	2	lower	FDOUBL	Interval lower bound, A (≥ 0)
	3	upper	FDOUBL	Interval upper bound, B (≥ 0)
FRATIO	1	numerator	FSINGL	Numerator of ratio
	2	denominator	FSINGL	Denominator of ratio (> 0)
FSING1	1	value	FSINGL	Nominal value V of interval [V – B, V + B]
	2	bound	FSINGL	Interval bound, B (≥ 0)
FSING2	1	value	FSINGL	Nominal value V of interval [V – A, V + B]
	2	lower	FSINGL	Interval lower bound, A (≥ 0)
	3	upper	FSINGL	Interval upper bound, B (≥ 0)
IRLONG	1	numerator	ISLONG	Numerator of ratio
	2	denominator	IULONG	Denominator of ratio (> 0)
IRNORM	1	numerator	ISNORM	Numerator of ratio
	2	denominator	IUNORM	Denominator of ratio (> 0)
OBJREF	1	type	IDENT	Object type name
	2	origin	ORIGIN	Origin containing schema code and identifier namespace code
	3	copy	UVARI	Copy number
	4	identifier	IDENT	Object identifier
OBNAME	1	origin	ORIGIN	Origin containing identifier namespace code
	2	copy	UVARI	Copy number
	3	identifier	IDENT	Object identifier
RLONG	1	numerator	SLONG	Numerator of ratio
	2	denominator	ULONG	Denominator of ratio (> 0)
RNORM	1	numerator	SNORM	Numerator of ratio
	2	denominator	UNORM	Denominator of ratio (> 0)
TASCII	1	tag	ORIGIN	Origin reference
	2	string	ASCII	Character string value
TIDENT	1	tag	ORIGIN	Origin reference
	2	identifier	IDENT	Identifier
TUNORM	1	tag	ORIGIN	Origin reference
	2	value	UNORM	An unsigned integer value

11.4.1 ATTREF

A value acts as a reference to an attribute in an object of the given object type. The object need not be present in the same logical file. However, the ORIGIN objects referenced by the tag and origin subfields must be present.

The copy number subfield, which was USHORT in API Recommended Practice 66, Version 1, is now UVARI.

11.4.2 DTIME

Null value midnight, January 1, 1900, local standard time

$= 00000000\ 00000001\ 00000001\ 00000000$
$00000000\ 00000000\ 00000000\ 00000000_2$

11.4.3 FDOUB1

This is the representation of a balanced interval having its midpoint as a nominal value. The assignment of meaning to the interval is delegated to the producer.

11.4.4 FDOUB2

This is the representation of an unbalanced interval having a nominal double precision value. The interval is unbalanced because the nominal value may not be its midpoint.

11.4.5 FSING1

This is the representation of a balanced interval having its midpoint as a nominal single precision value. The assignment of meaning to the interval is delegated to the producer.

11.4.6 FSING2

This is the representation of an unbalanced interval having a nominal single precision value. The interval is unbalanced because the nominal value may not be its midpoint.

11.4.7 OBJREF

A value acts as a reference to an object having a given type. The object need not be present in the same logical file. However, the ORIGIN objects referenced by the origin subfield must be present.

The copy number subfield, which was USHORT in API Recommended Practice 66, Version 1, is now UVARI.

11.4.8 OBNAME

A value acts as a reference to an object. The object need not be present in the same logical file. However, the ORIGIN object referenced by the origin subfield must be present.

The copy number subfield, which was USHORT in API Recommended Practice 66, Version 1, is now UVARI.

11.4.9 TASCII

The tag subfield references an ORIGIN object in the same logical file. The use of this ORIGIN may differ from value to value and must be specified when the entity (e.g., attribute) is defined.

11.4.10 TIDENT

The tag subfield references an ORIGIN object in the same logical file. The use of this ORIGIN may differ from value to value and must be specified when the entity (e.g., object type, attribute, etc.) is defined.

11.4.11 TUNORM

The tag subfield references an ORIGIN object in the same logical file. The use of this ORIGIN may differ from value to value and must be specified when the entity is defined.

Recommended Practices for Exploration and Production Data Digital Interchange

Part 3: Physical Bindings

Exploration and Production Department

API RECOMMENDED PRACTICE 66, V2
SECOND EDITION, JUNE 1996

American
Petroleum
Institute

CONTENTS

Recommended Practice for Exploration and Production Data Digital Interchange Part 3: Physical Bindings

0 Introduction

0.1 This standard contains introductory, normative, and annotative information. Introductory information comprises everything that precedes Section 1, Scope. The introductory clauses describe how the publication is organized and provide commentary on the history and purpose of this standard. Normative information includes the actual requirements that must be satisfied by any data conforming to the standard. Annotative information is provided as rationale for and illustrations about the normative information and does not constitute part of the standard. The standard is completely stated even if all introductory and annotative information are removed. Annotative information may refer to terms introduced in later paragraphs or parts in order to tie together concepts, i.e., from time to time its value may increase after the reader has made a complete pass over the standard.

0.2 Different styles are used to distinguish between normative and annotative information.

0.3 All normative information is written using this same font, and is contained in either numbered paragraphs or numbered tables. Normative text is aligned with the left margin of the page.

0.4 All figures are annotative, and all annotative text is written in unnumbered paragraphs in Helvetica italic font, indented, as shown here:

This is a paragraph of annotative text. Its purpose is to include informal commentary on the normative information in immediately preceding or following paragraphs and may include references to or usage of normative information in other parts of the standard.

1 Scope

This part specifies physical bindings, which are descriptions of how an API Recommended Practice 66 logical format is recorded on various physical media. It defines and describes usage of terms such as storage set, storage unit, and storage unit label. A standard is defined only for the media specifically described in this part. Whenever a new physical binding is specified, a new edition of this part shall be produced.

2 Definitions

2.1 format version: A two-byte field in the visible record header immediately following the length field that identifies the API Recommended Practice 66 version of the data in the record. The version is a one-byte unsigned integer, preceded by a byte containing the value FF_{16}.

2.2 logical file: The main unit of data exchange. It consists of a sequence of one or more logical records, beginning with a record containing a single FILE-HEADER object.

2.3 logical format: A description of how to encode data in a media-independent sequence of 8-bit bytes. This is the view of API Recommended Practice 66 that is independent of any physical binding.

2.4 logical file section: The part of a logical file contained in one storage unit.

2.5 logical record: An organization of data values into coherent, semantically-related packets of information. A logical record may have any length greater than sixteen bytes and is organized in one of two syntactic forms: explicitly-formatted logical record (EFLR) or indirectly-formatted logical record (IFLR).

2.6 logical record segment: A construct that contains the structure necessary to describe and support the physical implementation of a logical record. A logical record is implemented as one or more logical record segments. A segment is wholly contained in a visible record, but two segments of the same logical record may be in different visible records.

2.7 organization code: A number assigned by API to an organization that identifies the organization and represents schemas and dictionaries defined and administered by the organization.

2.8 physical binding: A description of how API Recommended Practice 66 visible records are recorded on a specific medium type.

2.9 physical format: The medium-specific organization of data bytes on a storage unit.

2.10 producer: The system or application program or company that recorded information under API Recommended Practice 66.

2.11 record structure: A field in the storage unit label that specifies the nature of the visible records in a storage unit. Current options are RECORD and FIXREC.

2.12 storage set: A set of one or more storage units on which a sequence of one or more contiguous logical files is recorded.

2.13 storage unit: A medium-specific data container, e.g., a tape or file, that is identifiable and manageable by people who use the medium and on which API Recommended Practice 66 data is recorded.

2.14 visible record: The interface between the logical format and a medium-specific physical format. A visible record has a header, a body, and a trailing length.

3 Concepts

3.1 This part uses terminology defined in Part 2.

3.2 The visible records that make up the API Recommended Practice 66 logical format may be recorded on any number of media, such as magnetic tape, random access files, communication I/O streams, and so forth. Each medium type has a basic storage unit that is identifiable and manageable by people who use the medium. For standard magnetic tape it is the tape cassette or tape reel. For random access files it is the file. For communication streams it is the communications session.

The door is left open here on how medium types are identified. A partitionable tape, for example, may be considered a distinct medium type from a standard tape, allowing storage units to be partitions on the tape rather than the whole tape (to be determined). A database may be considered distinct from a random access file, allowing a storage unit to be a database (and its visible records the records of the database). The key requirement is that a medium type be readily identifiable by a person using an application that reads API Recommended Practice 66 data so that the type can be provided to the application before the storage unit is opened by the application.

3.3 Almost always the capacity of a storage unit and the size of a logical file are different. When the storage unit is larger, it is possible to record one or more logical files on a single storage unit. When the logical file is larger, however, more than one storage unit is needed. The one or more storage units a logical file intersects constitute part of a storage set.

3.4 To identify a storage unit, including its position in a storage set, and to identify the logical format edition of the data recorded in the storage unit, a storage unit label is recorded at the beginning of every storage unit. The storage unit label consists of fixed-format textual information.

A storage set may consist of one storage unit or many storage units. It is particularly important to have a way to identify and sequentially order multiple storage units spanned by a single logical file, and this is one of the purposes of the storage unit label.

Another purpose is to provide human-readable information about the storage unit that can be presented using common utilities having no knowledge of API Recommended Practice 66.

3.5 Each common medium type is governed by industry standards designed to support one or a few specific application data access models that can be presented by the I/O subsystems of different computers. The descriptions of physical bindings provided in this part are given in terms of the data access models rather than in terms of low-level physical implementation details. For each common medium type, a description of the selected data access model is given, followed by a description of what a storage unit is and how visible records and storage unit labels are recorded on storage units. A description of physical binding requirements that are independent of any medium type is given in 4 and 5.

4 Storage Unit Terminology and Requirements

4.1 Access Methods

4.1.1 An access method is said to be block-oriented if each read operation returns a block of bytes beginning at the current position on the medium, and the number of bytes in the block is determined by information on the medium. The reader discovers how many bytes have been read after the read request. Correspondingly, each write operation writes a single block of data. Variable-length block access occurs when the writer may specify a different number of bytes for each block. Fixed-length block access occurs when the writer specifies a fixed block size once, and every block written has the same number of bytes. Some medium standards predefine the size of the fixed block.

4.1.2 An access method is byte-oriented if each read operation returns the number of bytes specified by the reader beginning at the current position on the medium. Correspondingly, writers specify how many bytes to write for each write operation.

4.1.3 Some I/O subsystems support multiple access methods for the same medium type. The intent of this standard is that the medium type of the storage unit shall uniquely determine its physical binding. Consequently, only one physical binding per medium type shall be specified here, and a physical binding must select exactly one access method.

4.2 Content

Of the data presented to an application by the I/O subsystem, the first data shall be a storage unit label. For block-oriented access, the first block shall contain a storage unit label at its beginning and shall not contain any visible records. Any data in the first block following the storage unit label has no meaning and shall be ignored. The remaining data consists of a sequence of one or more visible records. For block-oriented access, each block coincides with one visible record. That is, if the block (as presented to the application by the I/O subsystem) is *n* bytes long, so is the visible record. All visible records must be complete except for the last, which may be incomplete. If incomplete, it shall be assumed to be a failed attempt to write a complete visible record and may be ignored.

An incomplete visible record is detected by the fact that its actual size is smaller than the size declared in its header. An incomplete visible record may occur, for example, because a transmission link failed while writing a record. This is detectable by writers, and the failed visible record should be recorded in full on the next storage unit in the storage set (e.g., in another session) if one is used.

On the other hand, actual size of a visible record may be different from the size in its header due to media defects (i.e., the header is corrupted). If not at the end of the storage unit, this is detectable as a media defect, and the application must try to recover in the best way it can. If a media defect occurs at the end, an ambiguous situation arises that the standard does not address.

4.3 End of Storage Unit Indicator

A physical binding shall specify a mechanism to indicate no more visible records in the storage unit.

For example, on standard magnetic tape the "no more visible records" mechanism is a double tape mark.

5 Storage Set Terminology and Requirements

5.1 Additional storage set requirements are stated in 6.

5.2 A storage set shall contain one or more logical files and consist of one or more storage units.

5.3 The first logical record segment in a storage set (i.e., in the first visible record of the first storage unit) shall be the first segment of a logical file (see Part 2).

That is, although a storage unit may begin in the middle of a logical file, a storage set may not.

5.4 A logical file may be contained in one or more storage units in a storage set. The part of a logical file contained in a single storage unit is called a logical file section. The storage units in a storage set are numbered sequentially (see Table 3-2). The sequential order of logical file sections shall correspond to the sequential order of the storage units containing them.

That is, if section 1 starts in storage unit 3, then section 2 is in storage unit 4, section 3 in storage unit 5, and so on. If two logical files have sections in the same storage unit, then for one logical file it must be the last section and for the other logical file it must be the first section.

Note there is no restriction on mixing medium types in a storage set. Although most storage sets will consist of the same medium types (e.g., all standard tapes or all disk files), it is legal to mix and match, provided the rules on structure hold.

6 Storage Unit Label Contents

The storage unit label consists of 128 bytes encoded using characters of the ISO 8859-1 character set. Table 3-2 describes the fields of the storage unit label.

Table 3-1 — Storage Unit Label Fields

*Note	Field	Size in Bytes
1	Storage unit sequence number	4
2	API Recommended Practice 66 version and format edition	5
3	Storage unit structure	6
4	Binding edition	4
5	Maximum visible record length	10
6	Producer organization code	10
7	Creation date	11
8	Serial number	12
9	reserved	6
10	Storage set identifier	60

*Notes:

1. Storage unit sequence number is an integer in the range 1 to 9999 that indicates the order in which the current storage unit occurs in a storage set. The first storage unit of a storage set has sequence number 1, the second 2, and so on. This number is represented using the characters 0 to 9, right justified with leading blanks if needed to fill out the field (no leading zeros). The rightmost character is in byte 4 of the label. *A valid value shall be recorded.*

2. API Recommended Practice 66 version and format edition consists of the three characters 'V2.' representing the current version of this standard, followed by the edition of the logical format (see Part 2) in the range 01 to 99. The logical format edition is represented using the characters 0 to 9, right justified with a leading zero for numbers less than 10. The character V is in byte 5 of the label. All logical files in the storage unit adhere to API Recommended Practice 66, Version 2 and to the same or earlier logical format edition. *A valid value shall be recorded.*

3. Storage unit structure is a name indicating the visible record structure of the storage unit. This name is left-justified with trailing blanks if needed to fill out the field. The leftmost character is in byte 10 of the label. All storage units in the same storage set shall have the same storage unit structure. Options are listed in Table 3-2. *A valid value shall be recorded.*

4. Binding edition is the character B in byte 16 of the label followed by a positive integer in the range 1 to 999 (no leading zeros), left justified with trailing blanks if needed to fill out the field. The integer value corresponds to the edition of this part (Part 3) of the document that describes the physical binding of the logical format to the storage unit. *A valid value shall be recorded.*

5. Maximum visible record length is an integer in the range 0 to 4 294 967 294 (2^{32}–2) indicating the maximum visible record length for the storage unit, or 0 (zero) if undeclared. This number is represented using the characters 0 to 9, right justified, with leading blanks if necessary to fill out the field (no leading zeros). The rightmost character is byte 29 of the label. *A valid value or 0 (zero) shall be recorded.*

6. Producer organization code is an integer in the range 0 to 4 294 967 295 (2^{32}–1) indicating the organization code (see Appendix A) of the storage unit producer. This number is represented using the characters 0 to 9, right justified, with leading blanks if necessary to fill out the field (no leading zeros). The rightmost character is byte 39 of the label. This field may be empty, i.e., may contain all blanks, in which case no storage unit producer is specified.

7. Creation date is the earliest date that any current information was recorded on the storage unit. The date is represented in the form dd-MMM-yyyy, where yyyy is the year (e.g., 1994), MMM is the month (one of JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC), and dd is the day of the month in the range 1 to 31. Days 1 to 9 have one leading blank. The separator is a hyphen (code 45_{10}). This field may be empty, i.e., may contain all blanks, in which case no creation date is specified.

Table 3-1 — Storage Unit Label Fields (Continued)

8. Serial number is an ID used to distinguish the storage unit from other storage units in an archive of an enterprise. The specification and management of serial numbers is delegated to organizations using this standard. This field may be empty, i.e., may contain all blanks, in which case no serial number is specified.

9. This field is reserved and should be recorded as all blanks (code 32_{10}).

10. Storage set identifier is a descriptive name for the storage set. Every storage unit in the same storage set shall have the same value for the storage set identifier in its storage unit label. A value may have embedded blanks and is non-blank if at least one character is different from blank (code 32_{10}). This field is intended to distinguish the storage set from other storage sets, but is not required to be unique. *A non-blank value shall be recorded.*

The logical format edition is recorded using the form 01, 02, etc., for consistency with its API Recommended Practice 66, Version 1 form. Otherwise, leading zeros in numeric values are generally avoided.

6.1 Storage Unit Structure Options

Although storage unit structure describes visible records, it is considered to be a storage unit attribute (hence is in the storage unit label) since it describes a feature of all visible records in the storage unit.

Table 3-2 describes the storage unit structure options currently defined.

Table 3-2 — Storage Unit Structure Options

*Note	Field
1	RECORD
2	FIXREC

*Notes:

1. Visible records may be of variable length, ranging from the minimum length required to contain one logical record segment to length specified in the maximum visible record length field of the storage unit label (if not zero). If the maximum visible record length specified is zero, then visible records may be of any length.

2. All visible records in the storage unit have the same length, namely that specified in the maximum visible record length field of the storage unit label. Although all storage units in the same storage set must have a FIXREC structure, the maximum visible record length may be different in different storage units. When the FIXREC option is used, then the maximum visible record length field shall not be 0 (zero).

7 Binding to Standard Magnetic Tapes

7.1 Medium Characteristics

7.1.1 A standard magnetic tape is a non-partitioned, block-oriented device that supports variable-length blocks. It may be written in one of several different densities. Density is detectable when reading but is not recorded in any data block. Consequently, the choice of tape density is unrestricted.

7.1.2 A standard magnetic tape has two kinds of tape mark, an indelible marks and a writable mark. There are two indelible tape marks, one called BOT (beginning of tape) and one called ETW (end of tape warning). BOT is near the physical beginning of the tape and indicates the start of the region in which recorded data is permitted. ETW is required to be a minimum distance from the physical end of the tape and serves as a warning.

With many systems, ETW can be sensed only when writing.

7.1.3 A writable tape mark (called TM) is a distinct form of recorded information that may be written or read by an application in place of a block. It contains no data.

7.2 Binding Requirements

7.2.1 For standard magnetic tapes, a single tape reel or tape cassette constitutes one storage unit.

7.2.2 On a standard magnetic tape there shall be a single TM following the BOT and immediately preceding the storage unit label, which is recorded in the first data block.

7.2.3 Each visible record shall be written in one complete tape block and shall coincide with the data of the block. A block shall have no bytes except those belonging to the visible record.

7.2.4 There shall be at least two consecutive TMs immediately following the last visible record on the storage unit. That is, two or more consecutive TMs constitute the end of storage unit indicator. The double TM may occur either before or after ETW.

7.2.5 There shall be a single TM immediately preceding the first visible record of each logical file.

The purpose of the TM before a logical file is to support fast hardware positioning to logical files.

8 Binding to Random Access Files

8.1 Medium Characteristics

8.1.1 A random access file is a named entity for which random byte access is possible. Bytes are ordered sequentially, and an application may position to any byte in the file by specifying its position and then read or write *n* bytes. All the bytes presented to an application by the I/O subsystem are data bytes. This view of a file corresponds to an implementation of ANSI C I/O accessing the file in binary mode.

8.1.2 The maximum size of a file is implementation-dependent.

Most current implementations have a maximum file size of $2^{32}-1$ bytes, the largest integer expressible as a C unsigned long. To support random access, absolute byte position in the file must be representable using a native datatype of the language.

8.2 Binding Requirements

8.2.1 A storage unit is a single complete file.

8.2.2 The first 128 bytes of a storage unit constitute a storage unit label.

8.2.3 The remaining bytes constitute a sequence of visible records. End of storage unit is indicated when there are no more bytes to read from the file.

9 Binding to Peer-to-Peer Communication Streams

9.1 Medium Characteristics

9.1.1 A peer-to-peer communication stream is a FIFO (first in first out) queue for which sequential access to an ordered sequence of bytes is possible. An application may either write to the FIFO or read from the FIFO, but may not do both.

An application using full duplex transmission facilities to read data from its remote peer and write data back to the same remote peer is reading from one FIFO and writing to a different FIFO over the same communications channel.

9.1.2 A stream is created by a communications session established between two applications and ends when the session ends. A session is defined by the data exchange occurring between the open and close operations provided by the communications protocol.

9.1.3 Binding Requirements

9.1.3.1 A storage unit consists of the stream of data bytes exchanged in one direction between applications during one session.

9.1.3.2 The first 128 bytes of a storage unit constitute a storage unit label.

9.1.3.3 The remaining bytes constitute a sequence of visible records. End of storage unit is indicated when there are no more bytes to read from the stream.

Some distinction should be made between "file-less" transfer of data in a peer-to-peer exchange vs "remote file access" or "remote tape access" using communication channels. In the latter two cases, a communications session is established between a file or tape server on one end and an application on the other end. The data access services from the server should appear the same as reading a local file or a local tape. Consequently, the remote storage unit should look like one or more files or one or more tapes. This mode allows multiple storage units in one session, for example.

However, a true peer-to-peer communications binding will present only one storage unit.

Recommended Practices for Exploration and Production Data Digital Interchange

Part 4: The API-SI Unit Model

Exploration and Production Department

API RECOMMENDED PRACTICE 66, V2
SECOND EDITION, JUNE 1996

American
Petroleum
Institute

CONTENTS

Recommended Practice for Exploration and Production Data Digital Interchange Part 4: The API-SI Unit Model

0 Introduction

0.1 This standard contains introductory, normative, and annotative information. Introductory information comprises everything that precedes Section 1, Scope. The introductory clauses describe how the publication is organized and provide commentary on the history and purpose of this standard. Normative information includes the actual requirements that must be satisfied by any data conforming to the standard. Annotative information is provided as rationale for and illustrations about the normative information and does not constitute part of the standard. The standard is completely stated even if all introductory and annotative information are removed. Annotative information may refer to terms introduced in later paragraphs or parts in order to tie together concepts, i.e., from time to time its value may increase after the reader has made a complete pass over the standard.

0.2 Different styles are used to distinguish between normative and annotative information.

0.3 All normative information is written using this same font, and is contained in either numbered paragraphs or numbered tables. Normative text is aligned with the left margin of the page.

0.4 All figures are annotative, and all annotative text is written in unnumbered paragraphs in Helvetica italic font, indented, as shown here:

This is a paragraph of annotative text. Its purpose is to include informal commentary on the normative information in immediately preceding or following paragraphs and may include references to or usage of normative information in other parts of the standard.

1 Scope

This part describes the syntax of a unit model administered by the API Subcommittee On Standard Format For Digital Well Data. The name of the unit model is API-SI. This unit model is based upon le Système International d'Unités (SI), with some extensions suitable for use by oilfield businesses.

2 References

The unit model described here is based on guidelines published in the following documents. These references are provided for information only and do not form a part of this standard.

IEEE[1]
STD 260-1978 *IEEE Standard Letter Symbols for Units of Measurement.*

[1] The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017-2394.

ISO[2]

1000-1981 (E)	*SI units and recommendations for the use of their multiples and of certain other units*

SPE[3]

The SI Metric System of Units and SPE METRIC STANDARD

3 Definitions

3.1 base unit: a unit that cannot be expressed in terms of other units.

3.2 derived unit: a unit expressed as a dimensionless scaling or as an algebraic combination of other units.

3.3 unit conversion: an algebraic transformation of a value represented in one unit to an equivalent value represented in a related unit.

3.4 unit model: a grammar in which unit symbols from an administered dictionary are combined and scaled to represent expressions of desired units.

3.5 unit symbol: a dictionary-administered word, or token, that represents a unit of measure.

4 Concepts

4.1 The API-SI unit model consists of a grammar in which unit symbols from an administered dictionary are combined and scaled to produce expressions of desired units. The grammar (see 5.1) describes how to combine unit symbols using the algebraic operations multiplication, division, and exponentiation. The resulting combination may then be scaled by a numeric multiplier, which is expressed in terms of multiplication, division, and exponentiation of integer and decimal numbers. Finally, a similar numeric offset term may be applied to the scaled combination.

4.2 The unit symbol dictionary (see Part 5) contains symbols for a large number of commonly-used units. These are divided into two groups: base unit symbols and derived unit symbols. A base unit is not defined in terms of any other unit symbols. It represents the standard reference unit of a basic physical or dimensionless quantity. A derived unit symbol, on the other hand, is defined as an expression of other unit symbols or as a dimensionless scaling.

4.3 A unit symbol may be used in an API-SI unit expression only if it appears in the API-SI unit dictionary listed in Part 5 of this standard. API-SI unit symbols match corresponding SI unit symbols when such units are defined under SI. Additional unit symbols (including base units) have been added to the API-SI unit dictionary for commonly-used oilfield and (dimensionless) monetary quantities.

> *This model is based on the SI notion that from a small handful of basic quantities all other quantities can be derived by algebraic operations. The SI basic quantities are length, mass, time, electric current, temperature, luminous intensity, and amount of substance. Two dimensionless supplemental quantities–plane angle, and solid angle–are typically added. These have*

[2] International Organization for Standardization. ISO publications are available from ANSI.
[3] Society of Petroleum Engineers, P.O. Box 833836, Richardson, TX 75083-3836.

corresponding reference units m (meter), kg (kilogram), s (second), A (ampere), K (kelvin), cd (candela), mol (mole), rad (radian), and sr (steradian).

Given length and time, for example, velocity may be derived as length/time, and a corresponding unit for velocity may be expressed as m/s.

In addition to derived units that represent new quantities, there are derived units that represent scalings of other quantities. For example, the expression 0.3048 m is also a length. The expression 0.3048 m can be assigned to a customary unit symbol, e.g., ft. The symbol ft is a derived unit symbol having definition ft = 0.3048 m. Both ft and 0.3048 m are derived units and may be used interchangeably. It happens that ft is also a unit symbol, which can be found in the unit dictionary and which may be used in the definition of other derived units.

5 Unit Expression Grammar

5.1 The following Backus-Naur Form (BNF) grammar defines the syntax for expressing an API-SI unit.

```
APIUnit ::= ' ' | Multiplier | StandardForm
StandardForm ::= [Multiplier ' '] FactorExpression
   [',' ' ' Offset]
FactorExpression ::= UnitTerm ['/' UnitTerm] |
   1 '/' UnitTerm
UnitTerm ::= '(' FactorExpression ')' | UnitFactors
UnitFactors ::= UnitFactor '.' UnitFactors | UnitFactor
UnitFactor ::= APIUnitSymbol [UnitExponent]
APIUnitSymbol ::= {UnitCharacter}
UnitCharacter ::=
   'A' | 'B' | 'C' | 'D' | 'E' | 'F' | 'G' | 'H' |
   'I' | 'J' | 'K' | 'L' | 'M' | 'N' | 'O' | 'P' |
   'Q' | 'R' | 'S' | 'T' | 'U' | 'V' | 'W' | 'X' |
   'Y' | 'Z' | 'a' | 'b' | 'c' | 'd' | 'e' | 'f' |
   'g' | 'h' | 'i' | 'j' | 'k' | 'l' | 'm' | 'n' |
   'o' | 'p' | 'q' | 'r' | 's' | 't' | 'u' | 'v' |
   'w' | 'x' | 'y' | 'z' | '%'
UnitExponent ::=
   ['-'] APINum | '(' ['-'] (APINumber | APINum
   ['/' APINum]) ')'
Multiplier ::= ComplexNum
Offset ::= ['-'] ComplexNum
ComplexNum ::= X ['E' N]['/' Y]
X ::= APINumber
N ::= ['-'] APINum Y ::= APINumber
APINumber ::= APINum | APINum '.' APINum |
   '0' '.' APINum
APINum ::= '0' APINum | APINum '0' | NonZeroDigits
NonZeroDigits ::= NonZeroDigit | NonZeroDigit
   {NonZeroDigit}
NonZeroDigit ::= '1' | '2' | '3' | '4' | '5' | '6' | '7'
   | '8' | '9'
```

Here are a few examples that illustrate the grammar:

m/smeter per second
s(0.5)root-second
1E-28 m2barn (b)
0.0174532925198 raddegree (dega)
360 degacycle (c)
(c/s)/(m/s)(cycle per second) per (meter per second)
kg.m/s2newton (N)
N.mjoule (J)
J/swatt (W)
W/Avolt (V)
A/Vsiemens (S)

Note the power of the grammar to construct an arbitrary number of unit expressions from a relatively small number of predefined unit symbols. The successive construction method illustrated here also makes the task of defining new unit symbols very simple, especially with respect to computing conversion factors (see 6.1.2).

6 Unit Conversion

6.1 Derived Units

6.1.1 Given the expression grammar described in 5.1, it is possible to derive formulas for converting values from one unit to another. This is described in 6.1.2 and 6.3.1.

6.1.2 Letting M = Multiplier, E = FactorExpression, and O = Offset, the standard form expression 'M E, O' is a new derived unit. Let U represent this derived unit, i.e., U = M E, O. Note that E is also a unit. Let X represent an amount expressed in unit U and Y represent the same amount expressed in unit E. Then the following unit conversion equations hold:

$$Y = M * (X - O) \qquad \text{(EQ 1)}$$

$$X = (1/M) * Y + O \qquad \text{(EQ 2)}$$

The unit, degree fahrenheit (°F), is derived from degree celsius (°C) using the expression °F = 5/9 °C, 32. If X and Y represent the same temperature in °F and °C, respectively, then

$Y = (5/9)(X - 32)$ and $X = (9/5)Y + 32$.

For example,

50°F = (5/9) (50 – 32)°C = 10°C.

6.1.3 Applications typically need to convert between units when one is not directly expressed in terms of the other. The procedure for doing this is described in 6.3.1.

6.2 Unit Dimension and Unit Reduction

6.2.1 Any two units U and E related by a standard form expression 'U = M E, O', are said to belong to the same unit dimension. Alternatively, any two units consisting of multiplers only are said to be dimensionless. A standard form expression can be reduced by successive substitution to a dimensionless unit or to a form in which E has only base unit symbols. This provides an algorithmic tool for detecting when two units have the same unit dimension and for computing the factors M and O needed to convert one unit to another.

6.2.2 Dimensionless units are already in reduced form. When there are no offsets, reduction is accomplished by substituting each unit symbol in E by its dimensionless or standard form, applying appropriate exponentiation and algebraic simplification, then factoring all multipliers out to create a new multiplier. If the new E reduces to 1, it is dropped out to produce a dimensionless unit. This is iterated until E has only base unit symbols or until a dimensionless unit results.

Example –

mi/hr^2 = (5280 ft)/(60 min)^2 = 5280/3600 ft/min^2 = 4.4/3 ft/min^2

= 4.4/3 (0.3048 m)/(60 s)^2 = (4.4)(0.3048)/(3)(3600) m/s^2

= 0.02794/225 m/s^2

6.2.3 For the simple case in which a factor expression has no denominator, and a single unit factor with exponent 1, then reduction with offset is governed by (EQ 1).

Example –

From the unit symbol dictionary,

°F = 5/9°C, 32 and
°C = K, –273.15.

If X°F, Y°C, and Z K all represent the same temperature, then from (EQ 1):

Z = Y + 273.15

= (5/9)(X – 32) + 273.15

= (5/9)(X + 459.67)

which by reverse application of (EQ 1) implies:

°F = 5/9 K, –459.67.

6.2.4 When a unit factor is in a denominator or has an exponent other than 1 or is not the only unit factor in the factor expression, then any offset in the definition of the unit factor is dropped. That is, such a unit factor is considered to represent a differential quantity.

Example –

The unit °F/ft represents a temperature difference per length. It may be reduced by ignoring the offsets in the definition of °F and °C:

°F/ft = 5/9°C/(0.3048 m)

= 0.508/3°C/m

= 0.508/3 K/m.

6.3 Unit Conversion Using Reduced Standard Forms

6.3.1 Assume units U and V have been reduced to standard form expressions 'M_U E_U, O_U' and 'M_V E_V, O_V', respectively where E_U and E_V have only base unit symbols. Then U and V belong to the same unit dimension if and only if E_U/E_V algebraically reduces to 1. Furthermore, if X represents an amount in unit U, and Y represents the same amount in unit V, then from (EQ 1):

$$M_U(X - O_U) = M_V(Y - O_V) \quad \text{(EQ 3)}$$

$$Y = (M_U/M_V)(X - O_U) + O_V \quad \text{(EQ 4)}$$

For example, to convert X mi/hr to Y km/s, first reduce both expressions:

mi/hr = 5280 ft / 3600 s

= 4.4/3 (0.3048 m)/s

= 0.44704 m/s [M_U= 0.44704, O_U= 0]

km/s = 1000 m/s [M_V= 1000, O_V= 0].

Note that m/s is the factor expression for both units, so U and V are convertible. Using (EQ 4), we get:

Y = 0.44704/1000 X

= 0.44704E−3 X.

6.3.2 The same conversions shown in (EQ 3) and (EQ 4) also hold when two units reduce to dimensionless units, since only multipliers and offset are involved.

One could incorporate the dimensionless case into the standard form case by using a pseudo base unit symbol "nodim".

Recommended Practices for Exploration and Production Data Digital Interchange

Part 5: The API-SI Unit Symbols

Exploration and Production Department

API RECOMMENDED PRACTICE 66, V2
SECOND EDITION, JUNE 1996

American
Petroleum
Institute

CONTENTS

Recommended Practice for Exploration and Production Data Digital Interchange Part 5: The API-SI Unit Symbols

0 Introduction

0.1 This standard contains introductory, normative, and annotative information. Introductory information comprises everything that precedes Section 1, Scope. The introductory clauses describe how the publication is organized and provide commentary on the history and purpose of this standard. Normative information includes the actual requirements that must be satisfied by any data conforming to the standard. Annotative information is provided as rationale for and illustrations about the normative information and does not constitute part of the standard. The standard is completely stated even if all introductory and annotative information are removed. Annotative information may refer to terms introduced in later paragraphs or parts in order to tie together concepts, i.e., from time to time its value may increase after the reader has made a complete pass over the standard.

0.2 Different styles are used to distinguish between normative and annotative information.

0.3 All normative information is written using this same font, and is contained in either numbered paragraphs or numbered tables. Normative text is aligned with the left margin of the page.

0.4 All figures are annotative, and all annotative text is written in unnumbered paragraphs in Helvetica italic font, indented, as shown here:

This is a paragraph of annotative text. Its purpose is to include informal commentary on the normative information in immediately preceding or following paragraphs and may include references to or usage of normative information in other parts of the standard.

1 Scope

This part lists and defines the unit symbols recognized under the API-SI unit model (see Part 4).

2 References

The unit symbols described here are based on guidelines published in the following standards. These references are provided for information only and do not form a part of this standard.

ISO[1]

4217	*Codes for the Representation of Currencies and Funds*
1000-1981 (E)	*SI units and recommendations for the use of their multiples and of certain other units*

[1] International Organization for Standardization. ISO documents may be obtained from GLOBAL ENGINEERING DOCUMENTS, 2805 McGraw Ave., Irvine, CA 92714. Phone (714) 261-1455.

3 Base Unit Symbols

3.1 SI Base Unit Symbols

Table 5-1 describes the base unit symbols defined under le Système International d'Unités (SI).

Table 5-1 — SI Base Unit Symbols

Unit Symbol	Unit Name
A	ampere
K	kelvin
cd	candela
kg	kilogram
m	meter
mol	mole
rad	radian
s	second
sr	steradian

3.2 API-SI Supplementary Base Units–Non-Currency

Table 5-2 describes the supplemental base unit symbols defined under API-SI, excluding currency unit symbols. These are not considered to be SI base units.

Table 5-2 — API-SI Base Unit Symbols – Non-Currency

Unit Symbol	Unit Name
dAPI	api gravity
dB	decibel
gAPI	api gamma ray
nAPI	api neutron

3.3 API-SI Currency Base Unit Symbols

Currency base units are based on the alphabetic currency codes defined in Table A.1 (excluding fund codes) of ISO 4217. Only the codes listed in Table 5-3 apply to the API-SI unit model. They are not considered to be SI base units.

Table 5-3 — API-SI Currency Base Unit Symbols

Unit Symbol	Unit Name
ADP	Andorran peseta
AED	UAE dirham
AFA	Afghanistan afghani
ALL	Albanian lek
ANG	Netherlands Antillian guilder
AOK	Angolan kwanza
ARA	Argentinian austral
ATS	Austrian schilling
AUD	Australian dollar
AWG	Aruban guilder
BBD	Barbados dollar
BDT	Bangladesh taka
BEF	Belgian franc

Table 5-3 — API-SI Currency Base Unit Symbols (Continued)

Unit Symbol	Unit Name
BGL	Bulgarian lev
BHD	Bahraini dinar
BIF	Burundi franc
BMD	Burmuda dollar
BND	Brunei dollar
BOB	Bolivian boliviano
BRC	Brazilian cruzado
BSD	Bahamian dollar
BTN	Bhutan ngultrum
BUK	Burmese kyat
BWP	Botswanian pula
BZD	Belize dollar
CAD	Canadian dollar
CHF	Swiss franc
CLP	Chilean peso
CNY	Chinese yuan renminbi
COP	Colombian peso
CRC	Costa Rican colon
CSK	Czechoslovakian koruna
CUP	Cuban peso
CVE	Cape Verde escudo
CYP	Cyprus pound
DDM	Mark der DDR
DEM	Deutsche mark
DJF	Djibouti franc
DKK	Danish krone
DOP	Dominican peso
DZD	Algerian dinar
ECS	Ecuadorian sucre
EGP	Egyptian pound
ESP	Spanish peseta
ETB	Ethiopian birr
FIM	Finnish markka
FJD	Fiji dollar
FKP	Falkland Islands pound
FRF	French franc
GBP	British pound sterling
GHC	Ghanian cedi
GIP	Gibraltar pound
GMD	Gambian dalasi
GNF	Guinea franc
GRD	Grecian drachma
GTQ	Guatemalan quetzal
GWP	Guinea-Bissau peso
GYD	Guyana dollar
HKD	Hong Kong dollar
HNL	Honduran lempira
HTG	Haitian gourde
HUF	Hungarian forint
IDR	Indonesian rupiah
IEP	Irish pound
ILS	Israeli shekel
INR	Indian rupee
IQD	Iraqi dinar
IRR	Iranian rial

Table 5-3 — API-SI Currency Base Unit Symbols (Continued)

Unit Symbol	Unit Name
ISK	Iceland krona
ITL	Italian lira
JMD	Jamaican dollar
JOD	Jordanian dinar
JPY	Japanese yen
KES	Kenyan shilling
KHR	Kampuchean riel
KMF	Comoro franc
KPW	North Korean won
KRW	South Korean won
KWD	Kuwaiti dinar
KYD	Cayman Islands dollar
LAK	Lao kip
LBP	Lebanese pound
LKR	Sri Lanka rupee
LRD	Liberian dollar
LSL	Lesothoan loti
LUF	Luxembourg franc
LYD	Libyan dinar
MAD	Moroccan dirham
MGF	Malagasy franc
MNT	Mongolian tugrik
MOP	Macauan pataca
MRO	Mauritanian ouguiya
MTL	Maltese lira
MUR	Mauritius rupee
MVR	Maldives rufiyaa
MWK	Malawian kwacha
MXP	Mexican peso
MYR	Malaysian ringgit
MZM	Mozambique metical
NGN	Nigerian naira
NIC	Nicaraguan cordoba
NLG	Netherlands guilder
NOK	Norwegian krone
NPR	Nepalese rupee
NZD	New Zealand dollar
OMR	Omani rail
PAB	Panamanian balboa
PEI	Peruvian inti
PGK	Papua New Guinean kina
PHP	Philippine peso
PKR	Pakistan rupee
PLZ	Polish zloty
PTE	Portuguese escudo
PYG	Paraguayan guarani
QAR	Qatari rial
ROL	Romanian leu
RWF	Rwanda franc
SAR	Saudi riyal
SBD	Solomon Islands dollar
SCR	Seychelles rupee
SDP	Sudanese pound
SEK	Swedish krone
SGD	Singapore dollar

Table 5-3 — API-SI Currency Base Unit Symbols (Continued)

Unit Symbol	Unit Name
SHP	St. Helena pound
SLL	Sierra Leone leone
SOS	Somali shilling
SRG	Surinam guilder
STD	Sao Tome and Principe dobra
SUR	CIS rouble
SVC	El Salvador colon
SYP	Syrian pound
SZL	Swaziland lilangeni
THB	Thai baht
TND	Tunisian dollar
TOP	Tongan pa'anga
TPE	Timor escudo
TRL	Turkish lira
TTD	Trinidad and Tobago dollar
TWD	New Taiwan dollar
TZS	Tanzanian shilling
UGS	Uganda shilling
USD	US dollar
UYP	Uruguayan peso
VEB	Venezuelan bolivar
VND	Vietnamese dong
VUV	Vanuatuan vatu
WST	Western Samoan tala
XAF	CFA franc, Central Africa
XCD	East Caribbean dollar
XDR	SDR, international monetary fund
XEU	European currency unit
XOF	CFA franc, West Africa
XPF	CFP franc
YDD	Yemeni dinar
YER	Yemeni rial
YUD	new Yugoslavian dinar
ZAR	South African rand
ZMK	Zambian kwacha
ZRZ	Zaire zaire
ZWD	Zimbabwe dollar

4 Derived Unit Symbols

4.1 Table 5-4 describes all derived unit symbols recognized under the API-SI unit model.

Table 5-4 — API-SI Derived Unit Symbols

Unit Symbol	Unit Expression	Unit Name
%	0.01	percent
B	10 dB	bel
Bq	1/s	becquerel
Btu	1055.05585262 J	British thermal unit (international)
C	A.s	coulomb
Ci	37 GBq	curie
D	0.9869233 um2	darcy

Table 5-4 — API-SI Derived Unit Symbols (Continued)

Unit Symbol	Unit Expression	Unit Name
EJ	1E18 J	exajoule
F	C/V	farad
GBq	1E9 Bq	gigabecquerel
GHz	1E9 Hz	gigahertz
GJ	1E9 J	gigajoule
GPa	1E9 Pa	gigapascal
GS	1E9 S	gigasiemens
GW	1E9 W	gigawatt
Gal	cm/s2	galileo
GeV	1E9 eV	gigaelectronvolt
Gohm	1E9 ohm	gigaohm
Grad	1E9 rad	gigaradian
Gy	J/kg	gray
H	Wb/A	henry
Hz	1/s	hertz
J	N.m	joule
L	dm3	liter
MA	1E6 A	megampere
MBq	1E6 Bq	megabecquerel
MHz	1E6 Hz	megahertz
MJ	1E6 J	megajoule
MN	1E6 N	meganewton
MPa	1E6 Pa	megapascal
MV	1E6 V	megavolt
MW	1E6 W	megawatt
Ma	1E6 a	million years
MeV	1E6 eV	megaelectronvolt
Mg	1000 kg	megagram
Mm	1E6 m	megameter
Mohm	1E6 ohm	megohm
Mpsi	1E6 psi	million pounds per square inch
Mrad	1E6 rad	megaradian
N	kg.m/s2	newton
Oe	79.57747 A/m	oersted
P	0.1 Pa.s	poise
Pa	N/m2	pascal
S	A/V	siemens
Sv	J/kg	sievert
T	Wb/m2	tesla
TBq	1E12 Bq	terabecquerel
TJ	1E12 J	terajoule
TW	1E12 W	terawatt
TeV	1E12 eV	teraelectonvolt
Tohm	1E12 ohm	teraohm
V	W/A	volt
W	J/s	watt
Wb	V.s	weber
a	3.155815E7 s	annum (sidereal year)
aJ	1E-18 J	attojoule
acre	627264E5/15499969 m2	acre
ag	1E-18 g	attogram
atm	101.325 kPa	standard atmosphere
b	1E-28 m2	barn
bar	100 kPa	bar
bbl	42 galUS	barrel (U.S.)

Table 5-4 — API-SI Derived Unit Symbols (Continued)

Unit Symbol	Unit Expression	Unit Name
c	360 dega	revolution (cycle)
cP	0.01 P	centipoise
cal	4.1868 J	calorie (international)
cm	0.01 m	centimeter
cu	0.1 1/m	capture unit
d	24 h	day
daN	10 N	decanewton
dega	0.0174532925198 rad	degree (angle)
°C	K, -273.15	degree celsius
°F	5/9°C, 32	degree fahrenheit
degR	5/9 K	degree rankine
dm	0.1 m	decimeter
ehp	746 W	electric horsepower
eV	1.60219E-19 J	electron volt
fC	1E-15 C	femtocoulomb
fm	1E-15 m	femtometer
ft	12 in	foot (international)
ftAM	0.30478897 m	foot (American modified)
ftCla	0.30479727 m	foot (Clarke)
ftUS	1200/3937 m	foot (U.S. survey)
g	0.001 kg	gram
galUK	4.54609E-3 m3	gallon (U.K.)
galUS	3.785411784E-3 m3	gallon (U.S.)
gf	0.001 kgf	gram force
h	60 min	hour
hL	100 L	hectoliter
ha	1E4 m2	hectare
hbar	100 bar	hectobar
hhp	746.043 W	hydraulic horsepower
hp	550 ft.lbf/s	horsepower
in	0.0254 m	inch (international)
inUS	1/12 ftUS	inch (U.S. survey)
kA	1000 A	kiloampere
kC	1000 C	kilocoulomb
kHz	1000 Hz	kilohertz
kJ	1000 J	kilojoule
kN	1000 N	kilonewton
kPa	1000 Pa	kilopascal
kS	1000 S	kilosiemens
kV	1000 V	kilovolt
kW	1000 W	kilowatt
kcal	1000 cal	kilocalorie
kcd	1000 cd	kilocandela
keV	1000 eV	kiloelectronvolt
kgf	9.806650 N	kilogram force
klx	1000 lx	kilolux
km	1000 m	kilometer
kmol	1000 mol	killomole
kohm	1000 ohm	kilohm
krad	1000 rad	kiloradian
lbf	4.4482216152605 N	pound force
lbm	0.45359237 kg	pound mass (avoirdupois)
lm	cd.sr	lumen
lx	cd.sr/m2	lux
mA	0.001 A	milliampere

Table 5-4 — API-SI Derived Unit Symbols (Continued)

Unit Symbol	Unit Expression	Unit Name
mC	0.001 C	millicoulomb
mCi	0.001 Ci	millicurie
mD	0.9869233E-3 um2	millidarcy
mGal	0.001 Gal	milligal
mGy	0.001 Gy	milligray
mH	0.001 H	millihenry
mHz	0.001 Hz	millihertz
mJ	0.001 J	millijoule
mL	0.001 L	milliliter
mN	0.001 N	millinewton
mPa	0.001 Pa	millipascal
mS	0.001 S	millisiemens
mSv	0.001 Sv	millisievert
mT	0.001 T	millitesla
mV	0.001 V	millivolt
mW	0.001 W	milliwatt
mWb	0.001 Wb	milliweber
mbar	0.001 bar	millibar
mg	0.001 g	milligram
mi	5280 ft	mile (statute)
miUS	5280 ftUS	mile (U.S. survey)
min	60 s	minute
mina	1/60 dega	minute (angle)
mm	0.001 m	millimeter
mmol	0.001 mol	millimole
mohm	0.001 ohm	milliohm
mrad	0.001 rad	milliradian
ms	0.001 s	millisecond
nA	1E-9 A	nanoampere
nC	1E-9 C	nanocoulomb
nCi	1E-9 Ci	nanocurie
nH	1E-9 H	nanohenry
nJ	1E-9 J	nanojoule
nT	1E-9 T	nanotesla
nW	1E-9 W	nanowatt
nm	1E-9 m	nanometer
nohm	1E-9 ohm	nanohm
ns	1E-9 s	nanosecond
ohm	V/A	ohm
ozf	1/16 lbf	ounce force
ozm	1/16 lbm	ounce mass
pA	1E-12 A	picoampere
pC	1E-12 C	picocoulomb
pCi	1E-12 Ci	picocurie
pF	1E-12 F	picofarad
pPa	1E-12 Pa	picopascal
pS	1E-12 S	picosiemens
pm	1E-12 m	picometer
ppdk	1E-4	part per ten thousand
ppk	0.001	part per thousand
ppm	1E-6	part per million
ps	1E-12 s	picosecond
psi	lbf/in2	pound per square inch
pu	0.01 m3/m3	porosity unit
seca	1/3600 dega	second (angle)

Table 5-4 — API-SI Derived Unit Symbols (Continued)

Unit Symbol	Unit Expression	Unit Name
t	1000 kg	metric ton
tonUK	2240 lbm	long ton (U.K.)
tonUS	2000 lbm	short ton (U.S.)
uA	1E-6 A	microampere
uC	1E-6 C	microcoulomb
uCi	1E-6 Ci	microcurie
uF	1E-6 F	microfarad
uH	1E-6 H	microhenry
uHz	1E-6 Hz	microhertz
uJ	1E-6 J	microjoule
uN	1E-6 N	micronewton
uPa	1E-6 Pa	micropascal
uS	1E-6 S	microsiemens
uT	1E-6 T	microtesla
uV	1E-6 V	microvolt
uW	1E-6 W	microwatt
uWb	1E-6 Wb	microweber
ubar	1E-6 bar	microbar
ug	1E-6 g	microgram
um	1E-6 m	micrometer
umol	1E-6 mole	micromole
uohm	1E-6 ohm	microhm
upsi	1E-6 psi	micropound per square inch
urad	1E-6 rad	microradian
us	1E-6 s	microsecond

Recommended Practices for Exploration and Production Data Digital Interchange

Part 6: Basic Schema

Exploration and Production Department

API RECOMMENDED PRACTICE 66, V2
SECOND EDITION, JUNE 1996

American
Petroleum
Institute

CONTENTS

Recommended Practice for Exploration and Production Data Digital Interchange Part 6: Basic Schema

0 Introduction

0.1 This standard contains introductory, normative, and annotative information. Introductory information comprises everything that precedes Section 1, Scope. The introductory clauses describe how the publication is organized and provide commentary on the history and purpose of this standard. Normative information includes the actual requirements that must be satisfied by any data conforming to the standard. Annotative information is provided as rationale for and illustrations about the normative information and does not constitute part of the standard. The standard is completely stated even if all introductory and annotative information are removed. Annotative information may refer to terms introduced in later paragraphs or parts in order to tie together concepts, i.e., from time to time its value may increase after the reader has made a complete pass over the standard.

0.2 Different styles are used to distinguish between normative and annotative information.

0.3 All normative information is written using this same font, and is contained in either numbered paragraphs or numbered tables. Normative text is aligned with the left margin of the page.

0.4 All figures are annotative, and all annotative text is written in unnumbered paragraphs in Helvetica italic font, indented, as shown here:

> *This is a paragraph of annotative text. Its purpose is to include informal commentary on the normative information in immediately preceding or following paragraphs and may include references to or usage of normative information in other parts of the standard.*

1 Scope

Using the methodology outlined in Part 1, this part describes the schema, namely the collection of object types, administered by the API Subcommittee On Standard Format For Digital Well Data using organization code 0 (zero). This schema described in this part is called the basic schema. It includes object types required by any implementation of API Recommended Practice 66, namely FILE-HEADER and ORIGIN, and object types for recording data of a general nature, for example FRAME and CHANNEL.

> *The object types of the basic schema are intended to be industry-neutral.*
>
> *Some attributes of the basic schema object types have values restricted to a dictionary of standard reference values. These are listed in Part 7.*

2 Definitions

2.1 attribute: A named item of information or data pertaining to an object type.

2.2 attribute count: The number of elements in an attribute value.

2.3 attribute label: The name of an attribute.

2.4 attribute representation code: A code that identifies the recorded representation of each element of an attribute value.

2.5 attribute units: An expression that represents the units of measurement of each element of an attribute value.

2.6 attribute value: The value of an attribute. It may be present or absent. When present, it consists of zero or more elements, each having the same units and the same representation.

2.7 channel: A measured or computed quantity that occurs as a sequence of indexed values.

2.8 consumer: The system or application program or company that reads or uses API Recommended Practice 66 information.

2.9 data descriptor reference: An object name written at the beginning of an IFLR and used to associate the IFLR with the object that describes its content.

2.10 data model: A description of a specification and representation paradigm for data.

2.11 defining origin: The first ORIGIN object in a logical file.

2.12 dictionary: A database in which identifiers and reference values used under API Recommended Practice 66 are maintained and administered.

2.13 dimension: a vector of integers that describe the dimensionality and extent of the coordinate axes of an array.

2.14 empty logical file: a logical file having only a FILE-HEADER set and for which the attribute END-OF-STORAGE-SET is both present and true (value = 1).

2.15 frame: A set of channel values, one value per channel, written in an IFLR described by a FRAME object.

2.16 frame block: The set of one or more frames written in the same IFLR.

2.17 frame number: A positive integer representing the sequential position of a frame in a frame type or (if the frame type is unordered) representing a number index on the frame. The frame numbers of all frames in a frame block are written preceding the frame block in an IFLR.

2.18 frame type: The set of frames associated with a FRAME object. A frame type is represented by the FRAME object name.

2.19 identifier: One of the three parts of an object name. It is a character string used to distinguish the object from other objects of the same type. For some designated object types, the identifier conveys meaning of the nature of the object, and the identifier and its meaning are maintained in a dictionary.

2.20 logical file: The main unit of data exchange. It consists of a sequence of one or more logical records, beginning with a record containing a single FILE-HEADER object.

2.21 logical format: A description of how to encode data in a media-independent sequence of 8-bit bytes. This is the view of API Recommended Practice 66 that is independent of any physical binding.

2.22 logical model: A conceptual organization of a domain of knowledge.

2.23 logical record: An organization of data values into coherent, semantically-related packets of information. A logical record may have any length greater than sixteen bytes and is organized in one of two syntactic forms: explicitly-formatted logical record (EFLR) or indirectly-formatted logical record (IFLR).

2.24 logical record segment: A construct that contains the structure necessary to describe and support the physical implementation of a logical record. A logical record is implemented as one or more logical record segments. A segment is wholly contained in a visible record, but two segments of the same logical record may be in different visible records.

2.25 object: A recorded instance of an object type.

2.26 object name: A three-part unique reference to an object consisting of an origin, a copy number, and an identifier.

2.27 object type: A logical entity of a schema that has a unique type name and one or more defined attributes. Instances of an object type are written in explicitly formatted logical records.

2.28 organization code: A number assigned by API to an organization that identifies the organization and represents schemas and dictionaries defined and administered by the organization.

2.29 origin: One of three parts of an object name. It is a number referring to a distinct ORIGIN object that contains context information for the objects that reference it.

2.30 parent file: For a data item, the logical file in which data item was originally created.

2.31 producer: The system or application program or company that recorded information under API Recommended Practice 66.

2.32 representation code: A unique number that identifies a standard encoding for a value as a sequence of one or more contiguous bytes.

2.33 schema: A formalized description of the encoding of information defined by a logical model, typically in terms of a data model.

2.34 schema code: A numeric code found in Appendix A used to identify the organization responsible for defining and administering a schema.

2.35 set: A collection of one or more objects of the same object type. A set is recorded in an EFLR, and each EFLR has exactly one set.

2.36 set type: The type of objects in a set.

2.37 subfield: A part of a datum for which the representation is described by a simple (not compound) representation code. For example, the subfields of a datum having representation code OBNAME are, in order, an integer (UVARI), another integer (USHORT), and a string (IDENT).

2.38 update: a change in the value of an attribute previously written in the same logical file.

2.39 storage set: A set of one or more storage units on which a sequence of one or more contiguous logical files is recorded.

2.40 storage unit: A medium-specific data container, e.g., a tape or file, that is identifiable and manageable by people who use the medium and on which API Recommended Practice 66 data is recorded.

2.41 template: An ordered group of one or more attributes that represent a default or prototype object, written at the beginning of a set.

3 Authority

Changes to the basic schema are recommended by the API Subcommittee On Recommended Format For Digital Well Data and approved by the Executive Committee of the API Exploration and Production Department. Changes may include addition of new object types, addition of new attributes to existing object types, changes to the restrictions on existing attributes, or removal of attributes or object types. An attribute or object type removed in one edition may be restored in a later edition only if restored with its previous meaning. A new edition of the basic schema occurs when approved by the Executive Committee and has an edition number obtained by adding 1 to the previous edition number.

The intent is to maintain as much consistency as possible between editions. The principle motivation for a new edition should be to add new object types or attributes. However, from time to time compelling reasons arise for removing items because they are unused or impose unreasonable burdens on users and implementations.

4 Concepts

4.1 A schema is a collection of object types specified and administered by an organization (see Part 1). The schema is identified by an organization code (see Appendix A). The object types support writing data of interest to the organization and reflect a data model adopted by the organization, whether explicitly or implied.

4.2 The data model represented by the basic schema is implied by the descriptions of the object types given in this part. The basic schema provides two kinds of object type:

a. Object types required in every API Recommended Practice 66 implementation to delimit logical files and establish the context of data included in logical files. These include:

1. FILE-HEADER, used to delimit the beginning of a logical file.
2. ORIGIN, used to identify and distinguish logical files and establish the context used to interpret other objects.

b. Object types intended to be industry-neutral that support recording data of a general nature. These include:

1. AXIS, used to describe a single coordinate axis of an array.
2. CHANNEL, used to describe the meaning and representation of an indexed sequence of data values called a (data) channel.
3. COMMENT, used to write a textual comment.
4. COMPUTATION, used to describe the source and result of a computation derived from other data.
5. FRAME, used to describe the organization of several commonly-indexed channels into a sequence of IFLRs called frame blocks, where each frame block consists of one or more frames, and each frame has one value from each channel.
6. GROUP, used to describe application-defined groupings of other objects.

7. NO-FORMAT, used to identify IFLRs that contain unformatted data, i.e., for which no format description is provided other than a name.
8. ORIGIN-TRANSLATION, used to maintain an origin translation table for encrypted logical records.
9. PARAMETER, used to write parametric data.
10. PROCESS, used to describe a process by which other data was acquired or computed, including declarations of input data, output data, and parameters.
11. UPDATE, used to indicate a change in the value of an attribute previously written in the same logical file.
12. ZONE, used to identify an interval over which a value is defined or valid.

5 Unit Model

The basic schema does not impose any unit restrictions other than 'u=' (unitless), when applicable. Unless declared unitless, an attribute may have any unit from any unit model having a valid unit model code and name.

6 Dictionary-Controlled Identifiers

The following object types are required to have dictionary-controlled identifiers (see Part 2):

a. CHANNEL.
b. COMPUTATION.
c. PARAMETER.
d. PROCESS.

7 Required vs Optional Use of Attributes

Use of any attribute is considered optional unless otherwise stated. More stringent requirements on presence of attributes is delegated to content standards, which are not part of this document.

8 Frequently-Used Attributes

The attributes listed in Table 6-1 may appear in many object types and have the same or similar meaning in each place used. The similarities are described once here and not repeated. Extended uses and whether use is mandatory or optional may be described under the various object types that have these attributes.

Table 6-1 — Frequently-Used Attributes

*Note	Attribute Label	Restrictions
1	AGGREGATE	r=ULONG, u=
2	AXIS	r=OBNAME, u=
3	DESCRIPTION	c=1, r=ASCII, u=
4	DIMENSION	r=ULONG, u=
5	EXTENDED-ATTRIBUTES	r=OBJREF, u=
6	PROPERTIES	r=IDENT I TIDENT, u=, v=(see note)

*Notes:
1. AGGREGATE is a vector of integers that describe the structure of a value interpreted to be a set of nested 1-dimensional arrays of differing lengths. The particular value typically, though not always, belongs to one or more other attributes in the same object, and is identified as part of the description of the object type. The first element of AGGREGATE is the number n_1 of first level aggregations. The next $n1$ elements contain the numbers of level 2 aggregations, and so on. The last set of elements contain the sizes of the nested 1-dimensional arrays. Examples are provided in Figure 6-1. DIMENSION and AGGREGATE are mutually exclusive. If one is present, the other must be absent.

Table 6-1 — Frequently-Used Attributes (Continued)

2. AXIS is a list of references to AXIS objects (see 10.1). A referenced AXIS object describes one coordinate axis of an array. The AXIS attribute must be used in conjunction with a DIMENSION attribute. Whereas a DIMENSION element describes the extent of a coordinate axis (number of coordinate values), the corresponding referenced AXIS object further describes a starting coordinate, spacing between coordinates, coordinate axis units, specific coordinate values, etc.

3. DESCRIPTION is a human-readable, textual description of the object and is not intended to impose or possess syntactic or semantic content. It is available in all basic schema object types except FILE-HEADER and has the same usage everywhere.

4. DIMENSION is a vector of integers that specify the dimensions of a bounded array. The particular array is often, though not always, the value of one or more other attributes in the same object, and is identified in the description of the object type. The common interpretation of a bounded array structure as described by a DIMENSION attribute is given in 8.1. This interpretation may be extended for specific object types that use DIMENSION. One kind of extension, addition of an unbounded dimension, is discussed in the description of the FRAME object type. DIMENSION and AGGREGATE are mutually exclusive. If one is present, the other must be absent.

5. EXTENDED-ATTRIBUTES is a list of references to other objects, typically in other schemas, that contain additional privately-defined attributes that apply to the given object. It is available in all basic schema objects except FILE-HEADER and ORIGIN and has the same meaning everywhere.

6. PROPERTIES is a list of dictionary-controlled reference values that indicate the general intrinsic nature of the data associated with an object and the general processing steps that have been performed to create it. Properties are not mutually exclusive and may be associated in various combinations with the data of a particular object. Properties are intended to provide broad classifications for objects. There are currently no reference values defined under the basic schema.

```
 AGGREGATE {4}
       Value {2, 3, 0, 4}
Interpretation {2, 3, 0, 4}
```

A 1-level aggregate: Simple array

```
 AGGREGATE {4, 2, 3, 0, 4}
       Value {3, 7, 1, 2, 6, 0, 1, 5, 5}
Interpretation { {3, 7}, {1, 2, 6}, {}, {0, 1, 5, 5} }
```

A 2-level aggregate: Array of arrays

```
 AGGREGATE {4, 2, 3, 0, 4, 1, 2, 3, 2, 0, 5, 4, 7, 2}
       Value {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
              14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26}
Interpretation { { {1}, {2, 3} },
              { {4, 5, 6}, {7, 8}, {} },
              { {} },
              { {9, 10, 11, 12, 13}, {14, 15, 16, 17}, {18, 19, 20, 21, 22, 23, 24}, {25, 26} } }
```

Note that the number of elements of Value is the sum of the sizes of the level *n* arrays, and the number of level *n* arrays is the sum of the numbers of the level *n*-1 arrays.

A 3-level aggregate: Array of array of arrays

Figure 6-1—Examples of Aggregates

8.1 Interpretation of a Dimensioned Array

8.1.1 DIMENSION count specifies the dimensionality of a bounded array, i.e., its number of independent, bounded coordinate axes. Each DIMENSION element specifies the extent of one coordinate axis. For example, DIMENSION = {3, 4, 128} describes a 3 by 4 by 128 array. If DIMENSION count > 0, the total number of elements of the bounded array is the product of the DIMENSION elements. The example array has $3 \times 4 \times 128 = 1{,}536$ elements. DIMENSION count shall be greater than zero (0) unless specifically allowed in special cases described for individual object types.

8.1.2 An array is written as a linear sequence of elements. The mapping of this sequence to the structure defined by DIMENSION is such that the first coordinate index changes most rapidly, and the last coordinate index changes most slowly. If the elements of the array are denoted by $A_{i,j,k}$, then the linear order of elements is $A_{1,1,1}$, $A_{2,1,1}$, $A_{3,1,1}$, $A_{1,2,1}$, $A_{2,2,1}$, ..., $A_{3,4,1}$, $A_{1,1,2}$, $A_{2,1,2}$, ..., $A_{3,4,128}$.

9 Required Object Types

9.1 File-Header

9.1.1 A single FILE-HEADER object delimits the beginning of a logical file and serves as its identifying label. Every logical file shall have exactly one FILE-HEADER object. This is required of all schemas. Furthermore, no schema other than this basic schema may define an object type called FILE-HEADER.

9.1.2 Unless the END-OF-STORAGE-SET attribute is present and true (value = 1), the origin subfield of the FILE-HEADER name must reference the logical file's first ORIGIN object (see 9.2).

Table 6-2 — FILE-HEADER Attributes

*Note	Attribute Label	Restrictions
1	SEQUENCE-NUMBER	c=1, r=ULONG, u=
2	ID	c=1, r=ASCII, u=
3	END-OF-STORAGE-SET	c=1, r=STATUS, u=
4	CONTENT-STANDARD-CODE	r=ULONG, u=
5	CONTENT-STANDARD	r=IDENT, u=
6	CONTENT-STANDARD-EDITION	r=IDENT, u=

*Notes:

1. SEQUENCE-NUMBER is a positive integer that indicates the relative sequential position of the logical file in its storage set. It must be greater than SEQUENCE-NUMBER of the previous logical file in the same storage set and may be any positive integer for the first logical file. *This attribute is required.*
2. ID is a descriptive identification of the logical file.
3. END-OF-STORAGE-SET is used to indicate the end of a storage set (see Part 3 for a description of storage sets.) When this attribute is present and true (value = 1), then the corresponding logical file shall have no records other than the EFLR containing the FILE-HEADER set, i.e., it is an empty logical file. A logical file may be empty only when this attribute is present and true, since all other logical files are required to have at least an ORIGIN set (see 9.1.2). Furthermore, *no other logical files in the same storage set may follow an empty logical file, and every storage set is required to end with an empty logical file.*
4. CONTENT-STANDARD-CODE is a list of organization codes (see Appendix A) of the organization or organizations responsible for defining content standards to which the logical file adheres.
5. CONTENT-STANDARD is a list of content standard names corresponding in order to the elements of CONTENT-STANDARD-CODE. Each name identifies a standard for arrangement and/or quality to which the logical file contents are assumed to adhere.
6. CONTENT-STANDARD-EDITION is a list of editions of content standards corresponding in order to the elements of CONTENT-STANDARD.

Several of the constraints on FILE-HEADER that were present in API Recommended Practice 66, Version 1 have been dropped in this edition, notably the fixed-field-size and rigid organization constraints. These constraints did not accomplish the goal of supporting general file management utilities.

Note that if a logical file is copied to another storage set its SEQUENCE-NUMBER may have to change to satisfy the sequential position requirement for the new storage set.

9.2 Origin

9.2.1 ORIGIN objects contain information used to identify and distinguish logical files and establish the context for interpreting and distinguishing other objects. Every logical file shall have at least one ORIGIN object that immediately follows the FILE-HEADER and may have additional ORIGIN objects as required, not necessarily contiguous with the first. The first ORIGIN object in a logical file is called the *defining origin*.

9.2.2 Attributes FILE-ID, FILE-SET-NAME, FILE-SET-NUMBER, FILE-NUMBER, FILE-TYPE, and CREATION-TIME are used for logical file identification. With high probability (based on the effectiveness of random number generators), it should suffice to use only FILE-SET-NUMBER and FILE-NUMBER from the defining origins to distinguish two logical files having different content.

9.2.3 Every object in a logical file shall have an origin subfield in its object name that matches exactly one ORIGIN object in the same logical file. Except for FILE-HEADER and ORIGIN, any object must follow the ORIGIN object it references. This rule implies that all ORIGIN objects in the same logical file must have distinct origin subfield values.

9.2.4 An object may originate in the logical file in which it is contained, or it may represent a copy of part or all of an object that originated in another logical file. The logical file in which an object originates is called its parent file. The parent file of the defining origin is the file in which it is contained. Other ORIGIN objects may have different parent files. The parent file of an ORIGIN object is the one for which the attributes FILE-ID, FILE-SET-NAME, FILE-SET-NUMBER, FILE-NUMBER, FILE-TYPE, and CREATION-TIME match those of the defining origin. The parent file for any other object is the same as for the ORIGIN object it references.

9.2.5 When objects are copied, some origin subfield values may have to change to preserve the distinctness rule stated in 9.2.3. Thus, origin subfield values need not be preserved by copy operations.

The process of changing origin subfield values during copy or merge operations is called origin translation. The primary rule is referential consistency. Whenever a copied ORIGIN object requires origin translation, then all other objects that reference it in the source logical file must, if copied, have the same origin translation applied. Furthermore, any attributes that have origin values (e.g., OBNAME) must also be translated.

The following strategies can help reduce the need for origin translation: First, unless unavoidable, write all ORIGIN objects immediately after the FILE-HEADER. This helps edit applications, which must create a new defining origin for the edited output, to select an origin value distinct from the source origin values. An edit application, of course, must be prepared to encounter additional origins later in the file and handle possible origin translation if there is a conflict. However, if the source file has written all origins at the front, the need for translation will not arise.

Second, generate origin values randomly to reduce the probability that two logical files being merged into a third will have duplicate origin values.

9.2.6 No schema other than this basic schema may define an object type called ORIGIN.

9.2.7 The rules in 9.2.1 through 9.2.6 are required of all schemas.

9.2.8 Table 6-3 describes the attributes of ORIGIN objects.

Table 6-3 — ORIGIN Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
1	FILE-ID	c=1, r=ASCII, u=
2	FILE-SET-NAME	c=1, r=IDENT, u=
3	FILE-SET-NUMBER	c=1, r=ULONG, u=
4	FILE-NUMBER	c=1, r=ULONG, u=
5	FILE-TYPE	c=1, r=IDENT, u=
6	CREATION-TIME	c=1, r=DTIME, u=
7	SCHEMA-CODE	c=1, r=ULONG, u=
8	SCHEMA-ORGANIZATION	c=1, r=ASCII, u=
9	SCHEMA-EDITION	u=
10	SCHEMA-DICTIONARY-EDITION	u=
11	UNIT-MODEL-CODE	c=1, r=ULONG, u=
12	UNIT-MODEL-NAME	c=1, r=IDENT, u=
13	UNIT-MODEL-EDITION	u=
14	UNIT-SYMBOLS-EDITION	u=
15	NAMESPACE-CODE	c=1, r=ULONG, u=
16	NAMESPACE-NAME	c=1, r=IDENT, u=
17	NAMESPACE-ORGANIZATION	c=1, r=ASCII, u=
18	NAMESPACE-EDITION	u=
19	REMARK	c=1, r=ASCII, u=
20	CONTEXT	c=1, r=OBJREF, u=

*Notes:

1. FILE-ID is an exact copy of the FILE-HEADER:ID attribute of the parent file. *This attribute is required.*
2. FILE-SET-NAME is the name of a file set, a group of logical files to which the parent file belongs. The logical files in a file set are related according to implementation-defined criteria.
3. FILE-SET-NUMBER is a random number used to identify a file set. *This attribute is required.*
4. FILE-NUMBER identifies the parent file within a file set. It is a positive integer that represents the relative chronological order in which the logical file was created in the file set. The earliest logical file in a file set may have any positive file number. Any other logical file in the file set must have a file number that is greater than that of any earlier logical file in the same file set. Although file numbers are distinct in a file set, they are not required to be distinct in a storage set. *This attribute is required.*
5. FILE-TYPE is an implementation-defined name that identifies the general contents of the parent file or the circumstances under which the parent file was created.
6. CREATION-TIME is the date and time at which the parent file was created. This should be a time close to when the FILE-HEADER was written. *This attribute is required.*
7. SCHEMA-CODE is the organization code (see Appendix A) of the organization responsible for defining the object types, other than FILE-HEADER and ORIGIN, having this origin. *This attribute is required.*
8. SCHEMA-ORGANIZATION is the name of the organization assigned the code specified in SCHEMA-CODE.
9. SCHEMA-EDITION is the edition of the document describing the schema or schema derivation methodology corresponding to SCHEMA-CODE.
10. SCHEMA-DICTIONARY-EDITION is the edition of the dictionary of reference values used by attributes of the schema corresponding to SCHEMA-CODE.
11. UNIT-MODEL-CODE is the organization code (see Appendix A) of the organization responsible for defining the unit model used by objects having this origin.
12. UNIT-MODEL-NAME is the name of the unit model used by objects having this origin.
13. UNIT-MODEL-EDITION is the edition of the document describing the unit model used by objects having this origin.
14. UNIT-SYMBOLS-EDITION is the edition of the document describing the unit symbols used by objects having this origin.
15. NAMESPACE-CODE is the organization code (see Appendix A) of the organization responsible for administering the dictionary of object names used by objects having this origin.

Table 6-3 — ORIGIN Attributes (Continued)

16. NAMESPACE-NAME is the name of the dictionary of object names for objects having this origin.
17. NAMESPACE-ORGANIZATION is the name of the organization assigned the code specified in NAMESPACE-CODE.
18. NAMESPACE-EDITION specifies the edition of the dictionary in which dictionary-controlled object names are administered for this origin.
19. REMARK is an optional remark applicable to objects having this origin.
20. CONTEXT is a reference to an object defined under the schema specified in SCHEMA-CODE. The referenced object has attributes that establish additional context for interpreting the data having this origin. To allow multiple ORIGIN objects to share a context, the referenced object may have an origin different from this one so long as the corresponding schema code is the same.

Since FILE-HEADER and ORIGIN are required in any implementation, most logical files will have a minimum of two ORIGIN objects, one for the basic schema, and one for the industry schema that covers the data of interest. In some cases the basic schema alone may suffice to exchange meaningful data.

Declaration of required attributes is intentionally very selective. Whenever possible this decision is delegated to organizations that use API Recommended Practice 66. Nevertheless, most non-required attributes should be viewed as important to making full sense of the data.

10 Optional Object Types

10.1 Axis

10.1.1 An AXIS object describes one coordinate axis of an array. AXIS objects shall be used in conjunction with a DIMENSION attribute (see 8). The extent of the array along the coordinate axis, i.e., number of coordinate values, is determined by a DIMENSION attribute. Additional information provided by the AXIS object includes explicit coordinate values, coordinate axis units, and spacing between computed coordinate values.

10.1.2 When the array is a channel value (see 10.2), its coordinates may be modulated from frame to frame by reference to another channel value that provides dynamic coordinate values.

For example, a seismic trace may be represented as a 1-dimensional array along a time coordinate and written as a channel value described by CHANNEL object TRACE. From trace to trace (frame to frame), the initial time coordinate of the trace may change, although the trace sample interval remains fixed, e.g., at 4 ms. The initial time can be written as a companion channel in the same frame type, described by CHANNEL TRACE_START_TIME. A corresponding AXIS object may then specify SPACING = 4 ms and COORDINATES = TRACE_START_TIME.

Table 6-4 — AXIS Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	AXIS-ID	c=1, r=IDENT, u=, v=(see note)
2	COORDINATES	r=(see note)
3	SPACING	c=1, r=(see note)

Table 6-4 — AXIS Attributes (Continued)

*Notes:

1. AXIS-ID is a dictionary-controlled reference value identifying the coordinate axis. Valid identifiers are administered in the dictionary specified by the NAMESPACE-CODE and NAMESPACE-NAME attributes of the object's origin. There are currently no reference values defined under the basic schema.
2. COORDINATES specifies coordinate values along the axis, which may be either textual (for labels) or numeric. Compound representation codes are prohibited except for OBNAME. When r=OBNAME, then COORDINATES is the name of a single CHANNEL object whose value may be textual or numeric. COORDINATES may reference a CHANNEL object only if the described array is a channel value and both the array and referenced channel are in the same frame type. When the number of coordinate values specified by COORDINATES is less than the extent of the array as derived from the companion DIMENSION attribute, then subsequent coordinate values are computed in SPACING increments from the last coordinate value.
3. SPACING specifies a constant, signed, spacing along the axis between successive coordinates, beginning at the *last* coordinate value specified by COORDINATES. If COORDINATES is absent, then regularly-spaced coordinate values are implied beginning at zero (0) and separated by the declared spacing. Textual and compound representation codes are prohibited except for OBNAME. When r=OBNAME, then SPACING is the name of a single CHANNEL object whose value shall be numeric and have only one element. SPACING may reference a CHANNEL object only if the described array is a channel value and both the array and referenced channel are in the same frame type.

10.2 Channel

10.2.1 A CHANNEL object describes an instance of a channel. A channel is a named sequence of measured or computed values typically associated with an index such as depth or time. Channel values are recorded in frames (see 10.5). The CHANNEL object identifies the channel instance and specifies its representation in the frame. Channel values are similar to attribute values in that they consist of zero or more elements, all having the same units and representation code. The count, units, and representation code of a channel value are provided by its associated CHANNEL object.

Table 6-5 — CHANNEL Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	KIND	c=1, r=ASCII I TASCII, u=, v=(see note)
2	PROPERTIES	r=IDENT I TIDENT, u=, v=(see note)
3	FLAGS	r=IDENT I TIDENT, u=, v=(see note)
4	REPRESENTATION-CODE	c=1, r=USHORT, u=
5	FIXED-SIZE-IN-BYTES	c=1, r=ULONG, u=
6	UNITS	c=1, r=UNITS, u=
7	DIMENSION	r=ULONG, u=
8	DIMENSION-LIMIT	c=1, r=ULONG, u=
9	AGGREGATE	r=ULONG, u=
10	AGGREGATE-LIMIT	c=1, r=ULONG, u=
11	ELEMENT-LIMIT	c=1, r=ULONG, u=
12	SOURCE	c=1, r=OBJREF
13	AXIS	r=OBNAME, u=
14	ABSENT-ELEMENT	c=1
15	SPACING	
16	DIRECTION	c=1, r=IDENT I TIDENT, u=, v=(see note)
17	MINIMUM-VALUE	
18	MAXIMUM-VALUE	

*Notes:

1. KIND is a dictionary-controlled reference value that describes a general classification of the channel. Finer differentiation of the channel is provided by its identifier. Currently there are no reference values defined under the basic schema.
2. See Table 6-1.

Table 6-5 — CHANNEL Attributes (Continued)

3. FLAGS is a dictionary-controlled list of reference values that describe recording options for the channel. Reference values are listed in Part 7.

4. REPRESENTATION-CODE is the representation code used to write each element of a channel value. *This attribute is required.*

5. FIXED-SIZE-IN-BYTES declares a fixed number of bytes used to write each element of a channel value which has a variable-length representation code. This attribute may be used only if the representation code has an ASCII or IDENT subfield. When used, it is a guarantee that every element is written using the declared number of bytes. When the actual value of an element is shorter, one or more of the ASCII or IDENT subfields will be padded (using a null delimiter) to achieve the declared length.

6. UNITS is the unit of each element of a channel value. The unit model is obtained from the channel object's origin. If absent, the value is considered to be unitless.

7. DIMENSION describes the structure of a channel value (see Table 6-1). When both DIMENSION and AGGREGATE are absent and the channel is not explicitly-sized (see Part 7), each channel value is assumed to have one element.

8. DIMENSION-LIMIT declares the upper value of the dimension descriptor count of the channel when the channel is explicitly sized (see FLAGS). In any frame, count may either be zero, in which case the channel value is absent from the frame, or it may equal DIMENSION-LIMIT. A channel is not allowed to change dimensionality other than to become absent. DIMENSION and DIMENSION-LIMIT are mutually exclusive. If one is present, the other shall be absent.

9. AGGREGATE describes the structure of a channel value (see Table 6-1). When both DIMENSION and AGGREGATE are absent and the channel is not explicitly-sized (see Part 7), each channel value is assumed to have one element.

10. AGGREGATE-LIMIT declares a limit on the count of the aggregate descriptor of the channel when the channel is explicitly sized (see FLAGS). The count shall not exceed this limit and may or may not reach it. AGGREGATE and AGGREGATE-LIMIT are mutually exclusive. If one is present, the other shall be absent.

11. ELEMENT-LIMIT is a limit on the total number of elements per value (i.e., per frame) of the channel when explicitly sized (see FLAGS). The number of elements per value shall not exceed this limit and may or may not reach it. This attribute shall be absent if the channel is not explicitly-sized.

12. SOURCE is a reference to another object, for example a PROCESS object, that describes the immediate source of the channel.

13. AXIS is a list of references to AXIS objects that describe the channel value (see Table 6-1). When AXIS and DIMENSION are both present, they shall have the same count. That is, there shall be one AXIS object reference for each coordinate axis of the channel value. When AXIS is present and the channel is explicitly sized, then AXIS count shall equal the value of DIMENSION-LIMIT.

14. ABSENT-ELEMENT has a single element which shall have the same units and representation code as the channel value. Any channel value element that matches this attribute is considered to be absent, i.e., not valid for use.

15. SPACING declares a fixed signed spacing between channel values in successive frames for all frames of the frame type. It need not have the same units or representation code as the channel value. However, it shall have the same number of elements, and there shall be a units conversion $s -> s'$ such that $s' = v(n) - v(n-1)$ for all frame numbers $n > 1$, where s is the value of SPACING and $v(n)$ is the channel value in frame n. This attribute does not apply to explicitly-sized channels and shall not be present if the channel is explicitly-sized (see FLAGS). If the channel value has more than one element, then corresponding elements are uniformly spaced.

16. DIRECTION is a dictionary-controlled reference value that provides qualitative information about the behavior of the channel value as a function of frame number. When the channel has more than one element, the behavior applies on an element-by-element basis. Reference values are listed in Part 7.

17. MINIMUM-VALUE is the minimum channel value for all frames of the frame type. If the channel value has more than one element, it is an element-by-element minimum. That is, each element of this attribute is the minimum of the corresponding channel value element for all frames in the frame type.

18. MAXIMUM-VALUE is the maximum channel value for all frames of the frame type. If the channel value has more than one element, it is an element-by-element maximum. That is, each element of this attribute is the maximum of the corresponding channel value element for all frames in the frame type.

API Recommended Practice 66, Version 1 had a FRAME:INDEX-TYPE attribute which applied only to the index channel and has been dropped in this edition. KIND provides the same feature for any channel, regardless of use as an index.

The main usefulness of SPACING is to describe channels used as regular indexes, which have no need to change size. If it were to apply to explicitly-sized channels it would need to be updatable, which would make for very cumbersome rules on how to maintain a constant spacing when spacing can change.

In contrast to ABSENT-ELEMENT, which is essentially always used internally (i.e., by the computer), SPACING is frequently of interest to the end user, and some latitude is given to allow it to be presented in a friendlier form than the channel value, namely to have different though compatible units and representation code.

10.3 Comment

10.3.1 A COMMENT object carries arbitrary textual information considered interesting to the consumer.

Table 6-6 — COMMENT Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	TEXT	c=1, r=ASCII, u=

*Notes:
1. TEXT contains arbitrary text.

10.4 Computation

10.4.1 A COMPUTATION object is similar to a PARAMETER object except that VALUES is considered to be the result of a computation using other recorded data. Consequently, a COMPUTATION may have PROPERTIES, and the data used to compute VALUES may be identified by SOURCE.

Table 6-7 — COMPUTATION Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
	PROPERTIES	r=IDENT | TIDENT, u=, v=(see Table 6-1)
1	KIND	c=1, r=ASCII | TASCII, u=, v=(see note)
2	DIMENSION	r=ULONG, u=
3	AGGREGATE	r=ULONG, u=
4	AXIS	r=OBNAME, u=
5	ZONES	r=OBNAME, u=
6	VALUES	
7	SOURCE	c=1, r=OBJREF, u=

*Notes:
1. KIND is a dictionary-controlled reference value that describes a general classification of the computation. Finer differentiation of the computation is provided by its identifier. Currently there are no reference values defined under the basic schema.
2. DIMENSION is the same as PARAMETER:DIMENSION.
3. AGGREGATE is the same as PARAMETER:AGGREGATE.
4. AXIS is the same as PARAMETER:AXIS.
5. ZONES is the same as PARAMETER:ZONES.
6. VALUES is the same as PARAMETER:VALUES.
7. SOURCE is a reference to another object that describes the immediate computational source of data recorded in this object.

A typical SOURCE for a COMPUTATION object is a PROCESS object.

10.5 Frame

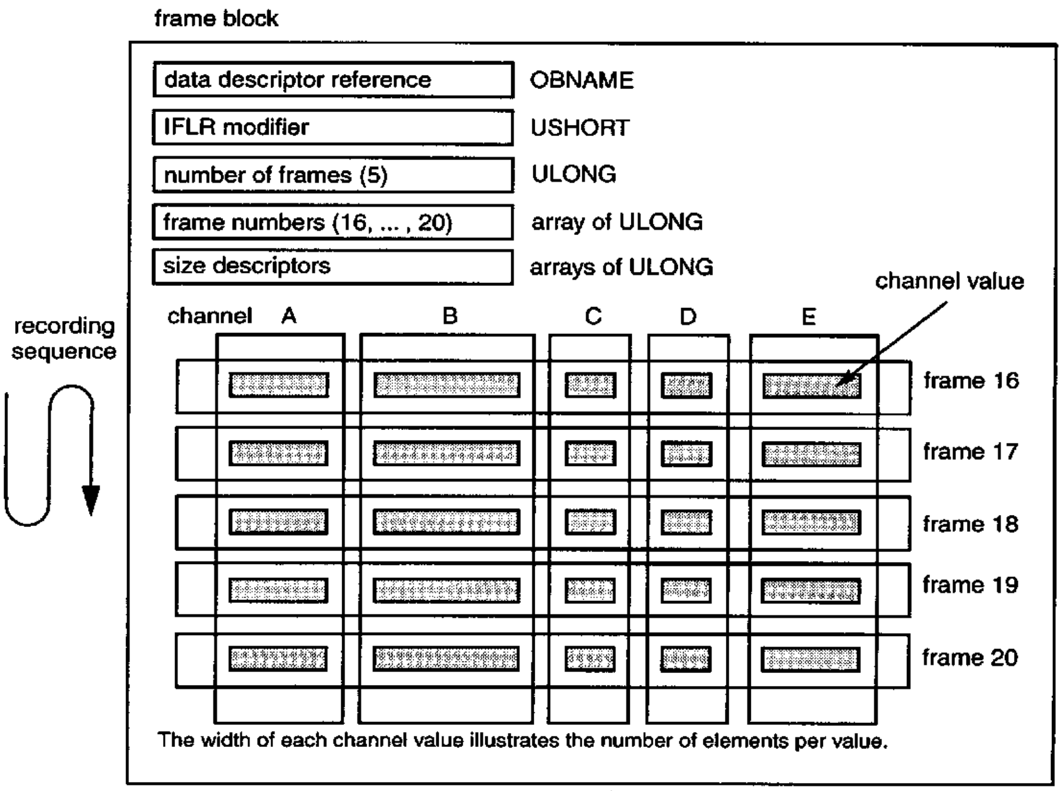


Figure 6-2—A Frame Block

10.5.1 A FRAME object describes a frame type, which is a sequence of one or more frames written in one or more frame blocks (see Table 6-2). A frame block may contain one or more frames. A frame block is an IFLR for which the data descriptor reference is the name of a FRAME object. The layout of data in a frame and its frame block is determined by information in the referenced FRAME object. The frame block IFLRs of a frame type are not required to be contiguous and may be intermixed with other logical records.

10.5.2 Conceptually, a frame is a set of channel values (see 10.2) occurring in the same order that their corresponding CHANNEL objects are listed in the CHANNELS attribute. Each channel represented in a frame block has one value per frame. Its values are written contiguously, followed by the values of the next channel, and so on. The first value of a channel belongs to the first frame in the frame block, the second value to the second frame, and so on. The maximum number of frames allowed per frame block is declared by attribute FRAMES-PER-IFLR-LIMIT or is assumed to be 1 if this attribute is absent. The actual number of frames in a frame block is written immediately following the IFLR modifier and has representation code ULONG.

10.5.3 Each frame has a frame number. The frame numbers of all the frames in a frame block are written together in order immediately following the number of frames value and have representation code ULONG.

10.5.4 A channel shall have the same number of elements per value for all frames of a frame type unless its CHANNEL object declares the value to be explicitly-sized via its

FLAGS attribute. When one or more channels in a frame type are explicitly-sized, then each frame block will have channel value size information written immediately following the frame numbers array. Channel value size information applies only to the current frame block and is written in the form of dimension or aggregate descriptors, which are arrays of ULONG values (see Part 7). Only descriptors of explicitly-sized channels are written, and they are written in the same order as the channel values.

10.5.5 The first channel value immediately follows the size descriptors. If there are no size descriptors it follows the last frame number.

10.5.6 A frame type represents a grouping of channels having the same number of values (where a value may be a simple element or an array having many elements) and sharing one or more common indexes. A frame, being a selection of the *n*th value from each channel, represents a sampling of the channels at a given index value. One common index, for example, is frame number, which corresponds to sample number. Often a more meaningful index, for example time or depth, is provided as one of the channels in the frame type. Multiple indexes are possible this way.

10.5.7 The FRAME object supports an additional implicit indexing mechanism. By means of its DIMENSION and AXIS attributes, the FRAME object may define an array representing the points of a grid having one or more coordinate axes (see Figure 6-3). The grid may be used as an indexing domain whose points are associated with frames by means of the frame number. The grid may be bounded or unbounded. If DIMENSION and AXIS have the same count, the grid is bounded. AXIS count is allowed to be one greater than DIMENSION count, provided the last referenced AXIS object has numeric COORDINATES and non-zero SPACING, in which case the grid is unbounded along the last coordinate. The mapping of frame number to grid follows the pattern for dimensioned arrays specified in 8.1.2. Note that this pattern also applies when the last coordinate is unbounded (i.e., when DIMENSION count is one less than AXIS count). DIMENSION may be absent or have count = 0 when AXIS count = 1, which represents the simplest unbounded grid: a 1-dimensional array. If the grid is bounded and frame number exceeds its extent, then mapping should wrap back to the beginning of the grid.

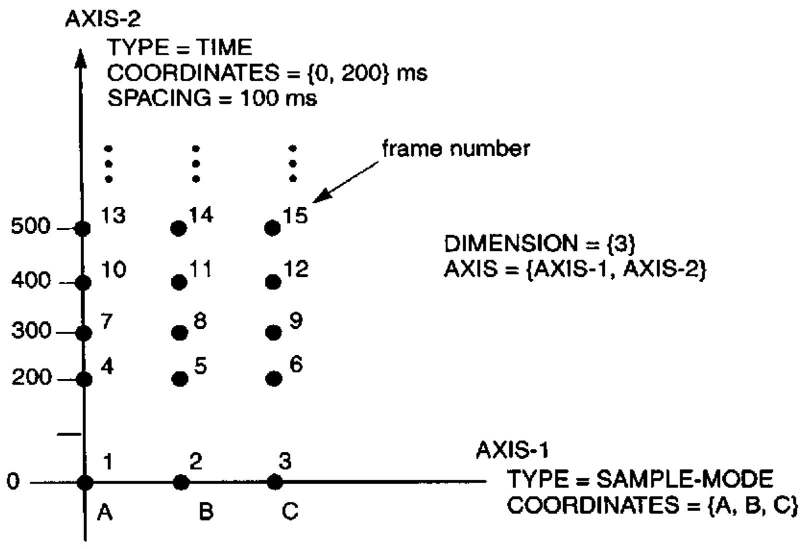


Figure 6-3—An Unbounded Grid of Frame Type Indexes

10.5.8 Table 6-8 describes the attributes of a FRAME object.

Table 6-8 — FRAME Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	FLAGS	r=IDENT I TIDENT, u=, v=(see note)
2	CHANNELS	r=OBNAME, u=
3	FRAMES-PER-IFLR-LIMIT	c=1, r=ULONG, u=
4	MIN-FRAME-NUMBER	c=1, r=ULONG, u=
5	MAX-FRAME-NUMBER	c=1, r=ULONG, u=
6	NUMBER-OF-FRAMES	c=1, r=ULONG, u=
7	DIMENSION	r=ULONG, u=
8	AXIS	r=OBNAME, u=

*Notes:

1. FLAGS is a dictionary-controlled list of reference values that describe recording options for the frame type. Reference values are listed in Part 7.
2. CHANNELS is a list of references to CHANNEL objects that describe the channel instances in the frame type. Channels occur in the frame type in the same order their corresponding CHANNEL objects are listed in this attribute. *This attribute is required.*
3. FRAMES-PER-IFLR-LIMIT is the maximum number of frames per IFLR (i.e., per frame block) for this frame type. If absent, 1 is assumed. If present, it shall be a positive integer.
4. MIN-FRAME-NUMBER is the minimum frame number for the frame type. It may be greater than 1.
5. MAX-FRAME-NUMBER is the maximum frame number for the frame type.
6. NUMBER-OF-FRAMES is the total number of frames written for the frame type.
7. DIMENSION describes the bounded portion of a grid used as an indexing domain for the frame type (see 10.5.7).
8. AXIS is a list of references to AXIS objects that describe a grid used as an indexing domain for the frame type (see 10.5.7). AXIS count must be at least as big as DIMENSION count and may be one greater. If AXIS count = DIMENSION count, the indexing grid is bounded. Otherwise, the indexing grid is unbounded.

When both MIN-FRAME-NUMBER and MAX-FRAME-NUMBER are present, then NUMBER-OF-FRAMES may be derived only if FLAGS does not declare frame numbers to be UNORDERED.

10.6 Group

10.6.1 A GROUP object describes an application-defined grouping of other objects.

Table 6-9 — GROUP Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	OBJECT-TYPE	c=1, r=IDENT I TIDENT, u=
2	OBJECT-LIST	r=OBJREF, u=
3	GROUP-LIST	r=OBNAME, u=

*Notes:

1. OBJECT-TYPE specifies the object type referenced by OBJECT-LIST when OBJECT-LIST has representation code OBNAME (which belongs to the same class as OBJREF).
2. OBJECT-LIST is a list of references to objects that belong to the group.
3. GROUP-LIST is a list of references to other GROUP objects, which are used to extend the membership of the current GROUP object.

10.7 No-Format

10.7.1 A NO-FORMAT object identifies IFLRs that contain unformatted data, i.e., for which no format description is provided other than a name.

10.7.2 Each IFLR having the name of a given NO-FORMAT object as its data descriptor reference contains a part of the unformatted data as the remainder of the logical record body following the IFLR modifier. The original data may be recovered by reading the IFLRs containing the data parts in the same order in which they were written and the bytes of each part as though they were USHORT.

Table 6-10 — NO-FORMAT Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	CONSUMER-NAME	c=1, r=ASCII, u=

*Notes:
1. CONSUMER-NAME is a user-provided name for the data, for example an external file specification.

10.8 Origin-Translation

10.8.1 An ORIGIN-TRANSLATION object is used to maintain an origin translation table for encrypted logical records (see Part 2). ORIGIN-TRANSLATION sets shall not be encrypted.

10.8.2 When a logical file has encrypted records, either EFLR or IFLR, one or more ORIGIN-TRANSLATION objects shall be written that contain the origin values used in the encrypted records. An encrypted record contains a translation tag in its first logical record segment header. This tag is the name of an ORIGIN-TRANSLATION object. The referenced object, which shall be present in the same logical file, shall contain all distinct origin values used in the encrypted record. When a data editing or merge operation requires origin translation and the encrypted record is copied without first being decrypted, the origin values it contains cannot be translated. However, if there is an ORIGIN-TRANSLATION object, its TRANSLATED-ORIGINS may be translated, and HIDDEN-ORIGINS preserved. In addition, the origin value in the translation tag may be translated if necessary to preserve the reference to its ORIGIN-TRANSLATION object. Subsequently when the edited or merged data is read by an application that can decrypt the encrypted record, its origins can be translated at that time using the appropriate ORIGIN-TRANSLATION object. At the same time, the HIDDEN-ORIGINS may be translated simply by replacing them with the corresponding TRANSLATED-ORIGINS values.

Table 6-11 — ORIGIN-TRANSLATION Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	HIDDEN-ORIGINS	r=ULONG, u=
2	TRANSLATED-ORIGINS	r=ORIGIN, u=

*Notes:
1. HIDDEN-ORIGINS is a list of untranslated origin values that may be in encrypted records.
2. TRANSLATED-ORIGINS is a list of translated origin values corresponding to HIDDEN-ORIGINS.

10.9 Parameter

10.9.1 A PARAMETER object describes a parameter that may consist of a single unzoned value or may consist of one or more zoned values. An unzoned value has no domain. It is globally defined. A zoned value is defined only in a specific domain described by a corresponding ZONE object (see Table 6-15).

Table 6-12 — PARAMETER Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	KIND	c=1, r=ASCII I TASCII, u=, v=(see note)
2	DIMENSION	r=ULONG, u=
3	AGGREGATE	r=ULONG, u=
4	AXIS	r=OBNAME, u=
5	ZONES	r=OBNAME, u=
6	VALUES	

*Notes:

1. KIND is a dictionary-controlled reference value that describes a general classification of the parameter. Finer differentiation of the parameter is provided by its identifier. Currently there are no reference values defined under the basic schema.
2. DIMENSION specifies the structure of each of one or more values in VALUES (see Table 6-1).
3. AGGREGATE specifies the structure of each of one or more values in VALUES (see Table 6-1).
4. AXIS applies to each of one or more values in VALUES (see Table 6-1).
5. ZONES is a list of references to ZONE objects that specify mutually exclusive domains in which corresponding parameter values are defined. When ZONES is absent, the parameter is unzoned. When ZONES is present its count shall be positive.
6. VALUES consists of zero or more parameter values. The extent of each value (i.e., number of elements) is determined by DIMENSION or AGGREGATE. If both are absent, each value has one element. If the parameter is zoned and VALUES count > 0, then the number of values is given by ZONES count. VALUES may have count = 0, in which case it is described by its characteristics and the other attributes (including ZONES, which may have non-zero count), but it has no values. *This attribute is required.*

10.10 Process

10.10.1 A PROCESS object describes a process by which other data was acquired or computed, including declarations of input data, output data, and parameters.

Table 6-13 — PROCESS Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
	PROPERTIES	r=IDENT I TIDENT, u=, v=(see Table 6-1)
1	KIND	c=1, r=ASCII I TASCII, u=, v=(see note)
2	TRADEMARK-NAME	c=1, r=ASCII, u=
3	VERSION	c=1, r=ASCII, u=
4	STATUS	c=1, r=IDENT, u=, v=(see note)
5	INPUT-CHANNELS	r=OBNAME, u=
6	OUTPUT-CHANNELS	r=OBNAME, u=
7	INPUT-COMPUTATIONS	r=OBNAME, u=
8	OUTPUT-COMPUTATIONS	r=OBNAME, u=
9	PARAMETERS	r=OBNAME, u=
10	COMMENTS	r=ASCII, u=

*Notes:

1. KIND is a dictionary-controlled reference value that describes a general classification of the process. Finer differentiation of the process is provided by its identifier. Currently there are no reference values defined under the basic schema.
2. TRADEMARK-NAME is the name used by the producer to refer to the process and its products.
3. VERSION is the producer's software version of the process.
4. STATUS is a reference value indicating status of the process at the time this object was written (see Part 7).
5. INPUT-CHANNELS is a list of references to CHANNEL objects that describe channels used directly by this process.
6. OUTPUT-CHANNELS is a list of references to CHANNEL objects that describe channels produced directly by this process. The same CHANNEL object shall not appear in the OUTPUT-CHANNELS attribute of more than one PROCESS object in the same logical file.

Table 6-13 — PROCESS Attributes (Continued)

7. INPUT-COMPUTATIONS is a list of references to COMPUTATION objects that describe computed results used directly by this process.
8. OUTPUT-COMPUTATIONS is a list of references to COMPUTATION objects that describe computed results produced directly by this process. The same COMPUTATION object shall not appear in the OUTPUT-COMPUTATIONS attribute of more than one PROCESS object in the same logical file.
9. PARAMETERS is a list of references to PARAMETER objects that describe parameters used directly by this process or that affect the operation of this process. PARAMETER objects may appear in the PARAMETERS attribute of more than one PROCESS object in the same logical file.
10. COMMENTS contains textual information about the process.

10.11 Update

10.11.1 An UPDATE object is used to indicate a change in the value of an attribute previously written in the same logical file. Not all attributes may be updated, since managing updates in general can be costly. Updatable attributes of the basic schema are listed in Table 6-16.

10.11.2 When present, attributes TIME, FRAME-TYPES, and FRAME-NUMBERS indicate when the change takes effect, in which case the attribute is considered to have the new value beginning at some time or point in the data and the old value prior to that point. In this case, the UPDATE object shall be written so that its new value applies only to data that follows it (but not necessarily to all data that follows it). When TIME, FRAME-TYPES, and FRAME-NUMBERS are all absent, then the meaning and use of the update is application-defined.

Table 6-14 — UPDATE Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	ATTRIBUTE	c=1, r=ATTREF, u=
2	NEW	
3	OLD	
4	TIME	c=1, r=DTIME I FDOUBL
5	FRAME-TYPES	r=OBNAME, u=
6	FRAME-NUMBERS	r=ULONG, u=
7	COMMENT	c=1, r=ASCII, u=

*Notes:
1. ATTRIBUTE is the attribute being updated. *This attribute is required.*
2. NEW is the new value of ATTRIBUTE. NEW may be absent, but OLD and NEW shall not both be absent.
3. OLD is the previous value of ATTRIBUTE. OLD may be absent, but OLD and NEW shall not both be absent.
4. TIME indicates when the change takes effect. Unless r=DTIME, this is an elapsed time from ORIGIN:CREATION-TIME. The attribute has the new value starting at the specified time and the old value prior to it.
5. FRAME-TYPES is a list of names of one or more FRAME objects representing frame types to which the update is correlated.
6. FRAME-NUMBERS is a list of one or more frame numbers indicating for each frame type in FRAME-TYPES at which frame the change takes effect. The attribute has the new value starting at the specified frame number and the old value for earlier frame numbers. Count of this attribute shall match count of FRAME-TYPES.
7. COMMENT is additional textual information associated with the update.

The UPDATE object type has been considerably simplified from API Recommended Practice 66, Version 1. All well log related attributes have been removed along with TAG-CHANNEL and TAG-VALUE. The latter were considered unnecessary, since most applications are capable of determining frame numbers from channel values and vice-versa. Note also in Table 6-16 that sev-

eral attributes previously declared updatable are no longer so. For some attributes, other mechanisms replaced the need for updates. Explicitly-sized channels and the ability to reference channels from AXIS objects removed the need to update attributes of CHANNEL and AXIS object types.

Neither OLD nor NEW is a required attribute, since it is possible that an update may be made with the intent of changing an absent attribute to a value or vice-versa.

10.12 Zone

10.12.1 A ZONE object is used to identify an interval over which a value is defined or valid.

Table 6-15 — ZONE Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	DOMAIN	c=1, r=IDENT I TIDENT, u=, v=(see note)
2	MAXIMUM	c=1
3	MINIMUM	c=1

*Notes:
1. DOMAIN is a reference value that indicates the type of zone interval. There are currently no reference values defined under the basic schema. DOMAIN shall be consistent with the units and representation code of MAXIMUM and MINIMUM. DOMAIN may be absent, in which case the units of MAXIMUM or MINIMUM imply a generic domain.
2. MAXIMUM is the maximum value of the zone. This value is *not* included in the zone. When absent, the zone is considered to extend indefinitely in the increasing direction.
3. MINIMUM is the minimum value of the zone. This value is included in the zone. When absent, the zone is considered to extend indefinitely in the decreasing direction.

10.13 Updatable Attributes

10.13.1 Table 6-16 lists attributes of the basic schema that may be updated using UPDATE objects.

Table 6-16 — Updatable Attributes

*Note	Object Type	Attribute Label
1	PARAMETER	VALUES
2	PROCESS	STATUS

*Notes:
1. The VALUES attribute may be updated if and only if the parameter is unzoned, i.e., if and only if the ZONES attribute is absent.
2. STATUS of a process may change during the course of acquiring or computing data written to a logical file.

Recommended Practices for Exploration and Production Data Digital Interchange

Part 7: Basic Schema Dictionary

Exploration and Production Department

API RECOMMENDED PRACTICE 66, V2
SECOND EDITION, JUNE 1996

American
Petroleum
Institute

CONTENTS

Recommended Practice for Exploration and Production Data Digital Interchange Part 7: Basic Schema Dictionary

0 Introduction

0.1 This standard contains introductory, normative, and annotative information. Introductory information comprises everything that precedes Section 1, Scope. The introductory clauses describe how the publication is organized and provide commentary on the history and purpose of this standard. Normative information includes the actual requirements that must be satisfied by any data conforming to the standard. Annotative information is provided as rationale for and illustrations about the normative information and does not constitute part of the standard. The standard is completely stated even if all introductory and annotative information are removed. Annotative information may refer to terms introduced in later paragraphs or parts in order to tie together concepts, i.e., from time to time its value may increase after the reader has made a complete pass over the standard.

0.2 Different styles are used to distinguish between normative and annotative information.

0.3 All normative information is written using this same font, and is contained in either numbered paragraphs or numbered tables. Normative text is aligned with the left margin of the page.

0.4 All figures are annotative, and all annotative text is written in unnumbered paragraphs in Helvetica italic font, indented, as shown here:

> *This is a paragraph of annotative text. Its purpose is to include informal commentary on the normative information in immediately preceding or following paragraphs and may include references to or usage of normative information in other parts of the standard.*

1 Scope

This part lists and defines a dictionary of reference values for attributes belonging to the basic schema (see Part 6).

2 Authority

Changes to the basic schema dictionary are recommended by the API Subcommittee On Standard Format For Digital Well Data and approved by the Executive Committee on Drilling and Production Practice of the API Exploration and Production Department. Changes may include addition of new terms or removal of obsolete terms. A term removed in one edition may be restored in a later edition only if restored with its previous meaning. A new edition of the basic schema dictionary occurs when approved by the Executive Committee and has an edition number obtained by adding 1 to the previous edition number.

3 Concepts

3.1 The tables presented here contain the various reference values defined for attributes of the basic schema that have controlled values. There is one table per controlled attribute. Each attribute is identified by its label preceded by the object type to which it belongs and separated by a colon (:). Reference values are listed alphabetically by object type first and by attribute second.

3.2 All basic schema reference values use representation code IDENT.

4 Channel Reference Values

4.1 Direction

Table 7-1 — CHANNEL:DIRECTION Reference Values

Reference Value	Description
DECREASING	Channel value strictly decreases as frame number increases.
INCREASING	Channel value strictly increases as frame number increases.
NON-DECREASING	Channel value does not decrease as frame number increases.
NON-INCREASING	Channel value does not increase as frame number increases.

4.2 Flags

Table 7-2 — CHANNEL:FLAGS Reference Values

Reference Value	Description
EXPLICIT-SIZE	When present, the channel is explicitly sized. In this case a dimension or aggregate descriptor for the channel will be found at the beginning of each frame block in which the channel value occurs. A dimension or aggregate descriptor consists of a count followed by a standard DIMENSION or AGGREGATE value, all written using representation code ULONG only. The descriptor is recorded at the beginning of the frame block immediately following the last frame number. If more than one channel is explicitly sized, then descriptors are written in the same order as the explicitlysized channels in the frames. When a channel is explicitly-sized, then DIMENSION and AGGREGATE attributes shall be absent from the CHANNEL object. Either DIMENSION-LIMIT or AGGREGATE-LIMIT but not both shall be present to indicate the the kind of descriptor and its maximum count for any frame in the frame type. ELEMENT-LIMIT shall be present to indicate the maximum number of elements of the channel's value for any frame in the frame type.

5 Frame Reference Values

5.1 Flags

Table 7-3 — FRAME:FLAGS Reference Values

Reference Value	Description
ENCRYPTED	IFLRs associated with the frame type are encrypted if and only if this value is present.
INCREASING	When present, frame numbers shall increase in the order in which frames occur, but there may be gaps. The frame number of the first frame may be any positive integer.
SEQUENTIAL	When present, frame numbers shall increase sequentially (no gaps) in the order in which frames are written. The frame number of the first frame may be any positive integer. The frame number of any other frame shall be one greater than the frame number of the previous frame.
UNORDERED	When present, frame numbers may occur in any order.

5.1.1 Flag values INCREASING, SEQUENTIAL, and UNORDERED are mutually exclusive. If none are present, then SEQUENTIAL is assumed by default.

There are two reasons for having sequentially-ordered frame numbers. First, it provides an indicator of when records have been lost–a frame number will be observed missing. Second, it allows seeking frames by position without having to count all frames in between.

On the other hand, it can be useful to have unordered frames when using an indexing hypergrid for the frame type (see Part 6). Since location on the hypergrid is a function of frame number, unordered frame numbers allow representation of different kinds of "cuts" through the hypergrid, as well as dropouts and other irregularities.

6 Process Reference Values

6.1 Status

Table 7-4 — PROCESS:STATUS Reference Values

Reference Value	Description
ABORTED	The process was aborted.
COMPLETE	The process completed.
IN-PROGRESS	The process began but did not complete.

Recommended Practices for Exploration and Production Data Digital Interchange

Part 8: DLIS Schema

Exploration and Production Department

API RECOMMENDED PRACTICE 66, V2
SECOND EDITION, JUNE 1996

American
Petroleum
Institute

CONTENTS

Recommended Practice for Exploration and Production Data Digital Interchange Part 8: DLIS Schema

0 Introduction

0.1 This standard contains introductory, normative, and annotative information. Introductory information comprises everything that precedes Section 1, Scope. The introductory clauses describe how the publication is organized and provide commentary on the history and purpose of this standard. Normative information includes the actual requirements that shall be satisfied by any data conforming to the standard. Annotative information is provided as rationale for and illustrations about the normative information and does not constitute part of the standard. The standard is completely stated even if all introductory and annotative information are removed. Annotative information may refer to terms introduced in later paragraphs or parts in order to tie together concepts, i.e., from time to time its value may increase after the reader has made a complete pass over the standard.

0.2 Different styles are used to distinguish between normative and annotative information.

0.3 All normative information is written using this same font, and is contained in either numbered paragraphs or numbered tables. Normative text is aligned with the left margin of the page.

0.4 All figures are annotative, and all annotative text is written in unnumbered paragraphs in Helvetica italic font, indented, as shown here:

> *This is a paragraph of annotative text. Its purpose is to include informal commentary on the normative information in immediately preceding or following paragraphs and may include references to or usage of normative information in other parts of the standard.*

1 Scope

Using the methodology outlined in Part 1, this part describes the *Digital Log Interchange Standard* (DLIS) schema, namely the collection of object types administered by the API Subcommittee On Standard Format For Digital Well Data using organization code 66. It includes object types useful for recording well log data.

2 Definitions

2.1 absent value: A placeholder that represents no value where a value is normally expected to be. For attributes, this may be an absent attribute component or a zero count. For channel samples, this is a zero count of the channel's dimension.

2.2 angular drift: One of the coordinates of the spatial coordinate system of a wellbore path. It is the angle measured eastwardly (clockwise when viewed from above) about the vertical generatrix from North (see Figure 8-1).

2.3 attribute: A named item of information or data pertaining to an object type.

2.4 attribute count: The number of elements in an attribute value.

2.5 attribute label: The name of an attribute.

2.6 attribute representation code: A code that identifies the recorded representation of each element of an attribute value.

2.7 attribute units: An expression that represents the units of measurement of each element of an attribute value.

2.8 attribute value: The value of an attribute. It may be present or absent. When present, it consists of zero or more elements, each having the same units and the same representation.

2.9 borehole: The physical hole created by boring or drilling. The term borehole is used in a descriptive sense, as in borehole axis, borehole diameter, borehole effect, etc.

2.10 channel: A measured or computed quantity that occurs as a sequence of indexed values.

2.11 consumer: The system or application program or company that reads or uses API Recommended Practice 66 information.

2.12 copy number: One of three parts of an object name. It is used to distinguish two objects of the same type in the same logical file that have the same origin and identifier.

2.13 data model: A description of a specification and representation paradigm for data.

2.14 dictionary: A database in which identifiers and reference values used under API Recommended Practice 66 are maintained and administered.

2.15 dimension: a vector of integers that describe the dimensionality and extent of the coordinate axes of an array.

2.16 element: One of a list of homogeneous quantities that make up the value of an attribute or channel. Every element of a value has the same units and representation.

2.17 identifier: One of the three parts of an object name. It is a character string used to distinguish the object from other objects of the same type. For some designated object types, the identifier conveys meaning of the nature of the object, and the identifier and its meaning are maintained in a dictionary.

2.18 locus: A sequence of distinct points in space and time, each of which has a three-dimensional position coordinate and a time coordinate.

2.19 logical file: The main unit of data exchange. It consists of a sequence of one or more logical records, beginning with a record containing a single FILE-HEADER object.

2.20 measured depth: The distance into a wellbore measured along a wellbore path from its wellbore path datum to a wellbore point (see Figure 8-1).

2.21 object: A recorded instance of an object type.

2.22 object name: A three-part unique reference to an object consisting of an origin, a copy number, and an identifier.

2.23 object type: A logical entity of a schema that has a unique type name and one or more defined attributes. Instances of an object type are written in explicitly formatted logical records.

2.24 organization code: A number assigned by API to an organization that identifies the organization and represents schemas and dictionaries defined and administered by the organization.

2.25 origin: One of three parts of an object name. It is a number referring to a distinct ORIGIN object that contains context information for the objects that reference it.

2.26 path: A sequence of space-time coordinates.

2.27 producer: The system or application program or company that recorded information under API Recommended Practice 66.

2.28 radial drift: The perpendicular distance of a point from the vertical generatrix (see Figure 8-1).

2.29 representation code: A unique number that identifies a standard encoding for a value as a sequence of one or more contiguous bytes.

2.30 schema: A formalized description of the encoding of information defined by a logical model, typically in terms of a data model.

2.31 schema code: A numeric code found in Appendix A used to identify the organization responsible for defining and administering a schema.

2.32 set: A collection of one or more objects of the same object type. A set is recorded in an EFLR, and each EFLR has exactly one set.

2.33 set type: The type of objects in a set.

2.34 update: a change in the value of an attribute previously written in the same logical file.

2.35 template: An ordered group of one or more attributes that represent a default or prototype object, written at the beginning of a set.

2.36 vertical depth: Distance measured along the vertical generatrix from the wellbore path datum.

2.37 vertical generatrix: A vertical line that passes through the wellbore path datum (see Figure 8-1).

2.38 wellbore: A connected network of borehole within the earth. A wellbore has a minimum of one wellbore origin and one wellbore terminus (see Figure 8-1).

2.39 wellbore path: A unique, non-overlapping path within a wellbore from a specific point of inception at the earth's surface to a specific point of ultimate extent in the subsurface. The wellbore path nominally follows the central axis of the physical borehole created by drilling between these two points (see Figure 8-1).

2.40 wellbore path datum: The origin of the coordinate system or zero point of reference for measuring along a wellbore path to a wellbore point. Ground level, derrick floor, and kelly bushing are typical zero point references for linear measurements along a wellbore path (see Figure 8-1).

2.41 wellbore point: A point position within a wellbore (see Figure 8-1).

Figure 8-1 and the terminology related to it correspond with well terminology found in the Petroleum Industry Data Dictionary (PIDD). Some changes were made to terms introduced in API Recommended Practice 66, Version 1 to achieve this correspondence. In particular, well reference point was changed to wellbore path datum and borehole depth to measured depth and their definitions appropriately modified.

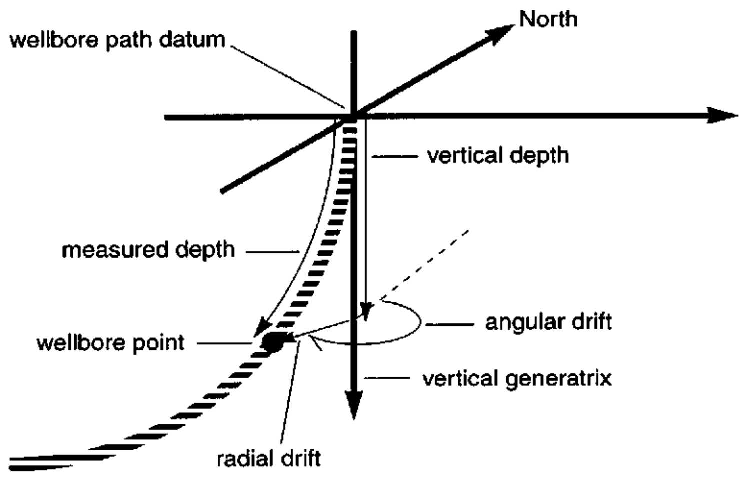


Figure 8-1—Spatial Coordinate System for a Wellbore Path

3 Authority

Changes to the DLIS schema are recommended by the API Subcommittee On Recommended Format For Digital Well Data and approved by the Executive Committee of the API Exploration and Production Department. Changes may include addition of new object types, addition of new attributes to existing object types, changes to the restrictions on existing attributes, or removal of attributes or object types. An attribute or object type removed in one edition may be restored in a later edition only if restored with its previous meaning. A new edition of the basic schema occurs when approved by the Executive Committee and has an edition number obtained by adding 1 to the previous edition number.

The intent is to maintain as much consistency as possible between editions. The principle motivation for a new edition should be to add new object types or attributes. However, from time to time compelling reasons arise for removing items because they are unused or impose unreasonable burdens on users and implementations.

4 Concepts

4.1 As stated in Part 1, a schema is a collection of object types specified and administered by an organization. The schema is identified by an organization code (see Appendix A). The object types support writing data of interest to the organization and reflect a data model adopted by the organization, whether explicitly or implied.

4.2 The data model represented by the DLIS schema is implied by the descriptions of the object types given in this part. The DLIS schema provides the following object types:

a. CALIBRATION, used to identify the collection of measurements and coefficients that participate in the calibration of a channel.
b. CALIBRATION-COEFFICIENT, used to record coefficients, their references and tolerances used in the calibration of channels.
c. CALIBRATION-MEASUREMENT, used to record measurements, references, and tolerances with which calibration coefficients are computed.
d. EQUIPMENT, used to describe an item of surface or downhole equipment used in the acquisition of data.
e. DLIS-CONTEXT, used to establish a context for the data related to a particular origin.
f. MESSAGE, used to write a textual message tied to other data by means of a time stamp and other indexing attributes.
g. PATH, used to identify channels that represent the coordinates of a path along which a logging tool string may pass and to specify certain geometric features of the tool string relative to the path.
h. SPLICE, used to identify the component channels of a splice and the splice points.
i. TOOL, used to record information about a logging tool and its component equipment parts.
j. WELLBORE-PATH-DATUM, used to record the location of a wellbore path datum (see 2.40).

5 Unit Model

In this part, any unit expressions used in unit restrictions belong to the API-SI unit model corresponding to organization code 0 (zero).

6 Dictionary-Controlled Identifiers

The following object types are required to have dictionary-controlled identifiers (see Part 2):

a. TOOL

7 Required vs Optional Use of Attributes

Use of any attribute is considered optional unless otherwise stated. More stringent requirements on presence of attributes is delegated to content standards, which are not part of this document.

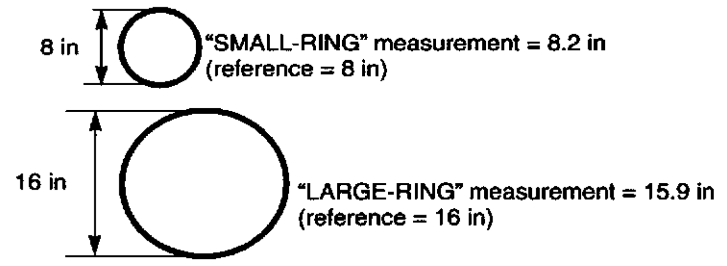
8 Frequently-Used Attributes

Attributes DESCRIPTION and EXTENDED-ATTRIBUTES are used in all object types as specified in the API Recommended Practice 66 basic schema (see Part 6).

9 DLIS Object Types

9.1 Calibration

9.1.1 A CALIBRATION object identifies the collection of measurements and coefficients that participate in the calibration of a channel.


8 in
"SMALL-RING" measurement = 8.2 in
(reference = 8 in)
16 in
"LARGE-RING" measurement = 15.9 in
(reference = 16 in)

calibrated = gain × measured + offset
8 = gain × 8.2 + offset
16 = gain × 15.9 + offset

gain = 8 / 7.7 = 1.038 ...
offset = –4 / 7.7 = –0.5194 ...

A simple application of calibration is the linear two-point method. Two measurements are taken and compared against two references. The comparison yields a gain coefficient and an offset coefficient, which are then used to compute calibrated values from uncalibrated measurements.

Figure 8-2—Illustration of Simple Two-point Linear Calibration

Table 8-1 — CALIBRATION Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	CALIBRATED-CHANNELS	r=OBNAME, u=
2	UNCALIBRATED-CHANNELS	r=OBNAME, u=
3	COEFFICIENTS	r=OBNAME
4	DATE	c=1, r=DTIME, u=
5	MEASUREMENTS	r=OBNAME, u=
6	PARAMETERS	r=OBNAME, u=
7	METHOD	c=1, r=IDENT | TIDENT, u=, v=(see note)

*Notes:

1. CALIBRATED-CHANNELS is a list of names of CHANNEL objects. The corresponding channels (typically just one) are declared to be calibrated using the coefficients and measurements identified by the COEFFICIENTS and MEASUREMENTS attributes.

2. UNCALIBRATED-CHANNELS is a list of names of CHANNEL objects. The corresponding channels (typically just one) are used, along with coefficients and according to the computational method, to compute the channels identified by the CALIBRATED-CHANNELS attribute.

3. COEFFICIENTS is a list of names of CALIBRATION-COEFFICIENT objects. The coefficients, references, and tolerances collectively defined by these objects are used to compute the channels identified by the CALIBRATED-CHANNELS attribute.

4. DATE is the date the coefficients were computed.

5. MEASUREMENTS is a list of names of CALIBRATION-MEASUREMENT objects. The measurements collectively defined by these objects are used to derive the coefficients that are used to calibrate the channels identified by the CALIBRATED-CHANNELS attribute.

6. PARAMETERS is a list of names of PARAMETER objects. The referenced objects provide information directly associated with the calibration process, for example statistics, quality control indicators, parameters entered by the operator, vendor-supplied coefficients, and other information (numeric or textual) that is potentially of interest to the consumer.

7. METHOD is a reference value that defines the computational method used to calibrate the channels identified by the CALIBRATED-CHANNELS attribute. There are currently no reference values defined under the DLIS schema.

9.2 Calibration-Coefficient

9.2.1 A CALIBRATION-COEFFICIENT object is used to record coefficients, their references and tolerances used in the calibration of channels.

Table 8-2 — CALIBRATION-COEFICIENT Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	LABEL	c=1, r=IDENT I TIDENT, u=, v=(see note)
2	COEFFICIENTS	
3	DIMENSION	r=ULONG, u=
4	AXIS	r=OBNAME, u=
5	REFERENCES	
6	PLUS-TOLERANCES	
7	MINUS-TOLERANCES	

*Notes:

1. LABEL is a reference value that identifies the role of COEFFICIENTS in the calibration process. There are currently no reference values defined under the DLIS schema.
2. COEFFICIENTS is an array of coefficients corresponding to LABEL.
3. DIMENSION applies to COEFFICIENTS, REFERENCES, PLUS-TOLERANCES, and MINUS-TOLERANCES (For DIMENSION usage, see Part 6, Basic Schema.)
4. AXIS applies to COEFFICIENTS, REFERENCES, PLUS-TOLERANCES, and MINUS-TOLERANCES (For AXIS usage, see Part 6, Basic Schema.)
5. REFERENCES is an array of references corresponding to COEFFICIENTS. Each REFERENCES element represents in some sense the nominal value of the corresponding COEFFICIENTS element.
6. PLUS-TOLERANCES is an array of tolerances corresponding to COEFFICIENTS. Each PLUS-TOLERANCE element indicates by how much the corresponding COEFFICIENTS element may exceed its reference and still be "within tolerance". Elements shall be non-negative numbers. A coefficient is within tolerance if it is less than or equal to its reference plus its plus tolerance. If this attribute is absent, then infinite plus tolerance is assumed.
7. MINUS-TOLERANCES is an array of tolerances corresponding to COEFFICIENTS. Each MINUS-TOLERANCE element indicates by how much the corresponding COEFFICIENTS element may fall short of its reference and still be "within tolerance". Elements shall be non-negative numbers. A coefficient is within tolerance if it is greater than or equal to its reference minus its minus tolerance. If this attribute is absent, then infinite minus tolerance is assumed.

9.3 Calibration-Measurement

9.3.1 A CALIBRATION-MEASUREMENT object is used to record measurements, references, and tolerances with which calibration coefficients are computed.

Table 8-3 — CALIBRATION-MEASUREMENT Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	PHASE	c=1, r=IDENT I TIDENT, u=, v=(see note)
2	MEASUREMENT-SOURCE	c=1, r=OBJREF, u=
3	TYPE	c=1, r=IDENT I TIDENT, u=, v=(see note)
4	DIMENSION	r=ULONG, u=
5	AXIS	r=OBNAME, u=
6	MEASUREMENT	
7	SAMPLE-COUNT	c=1
8	MAXIMUM-DEVIATION	
9	STANDARD-DEVIATION	
10	BEGIN-TIME	c=1

Table 8-3 — CALIBRATION-MEASUREMENT Attributes (Continued)

*Note	Attribute Label	Restrictions
11	DURATION	c=1
12	REFERENCE	
13	STANDARD	
14	PLUS-TOLERANCE	
15	MINUS-TOLERANCE	

*Notes:

1. PHASE is a reference value indicating what phase in the overall job sequence is represented by the current measurement (see Part 9).

2. MEASUREMENT-SOURCE references an object that specifies the source of the data recorded in MEASUREMENT.

3. TYPE is a reference value that specifies the type of measurement taken. Currently there are no reference values defined under the DLIS schema.

4. DIMENSION applies to MEASUREMENT, MAXIMUM-DEVIATION, STANDARD-DEVIATION, REFERENCE, PLUS-TOLERANCE, and MINUS-TOLERANCE. (For DIMENSION usage, see Part 6, Basic Schema.)

5. AXIS applies to MEASUREMENT, MAXIMUM-DEVIATION, STANDARD-DEVIATION, REFERENCE, PLUS-TOLERANCE, and MINUS-TOLERANCE. When present, its count shall match DIMENSION count. (For AXIS usage, see Part 6, Basic Schema.)

6. MEASUREMENT consists of one or more values, each described by DIMENSION, and each representing one measurement sample related to the uncalibrated data and described by TYPE. The number of values in MEASUREMENT is its count divided by the size in elements of a value described by DIMENSION. The measurement may represent values of, an average of, or some other function of the uncalibrated data.

7. SAMPLE-COUNT is the number of measurement values used to compute MAXIMUM-DEVIATION and STANDARD-DEVIATION.

8. MAXIMUM-DEVIATION is meaningful only when MEASUREMENT contains a single sample value. In this case, the measurement is considered to be a mean, and MAXIMUM-DEVIATION represents the maximum deviation from this mean of any sample used to compute the mean. For arrays, the mean and maximum deviation are computed independently for each element. The deviation for any element from the mean is computed as an absolute value.

9. STANDARD-DEVIATION is meaningful only when MEASUREMENT contains a single sample value. In this case, the measurement is considered to be a mean, and STANDARD-DEVIATION represents the statistical standard deviation of the samples used to compute the mean. For arrays, the mean and standard deviation are computed independently for each sample element.

10. BEGIN-TIME is the time at which acquisition of the measurement in MEASUREMENT began. BEGIN-TIME represents either an absolute date and time (if r=DTIME) or an elapsed time from ORIGIN:CREATION-TIME otherwise.

11. DURATION is a time interval representing the acquisition duration of the measurement in MEASUREMENT.

12. REFERENCE is the expected nominal value of a single sample value of the measurement represented in MEASUREMENT.

13. STANDARD is the measurable quantity of the calibration standard used to produce the MEASUREMENT. For example, a standard used to calibrate a caliper is a steel ring. Its measurable quantity is its inside diameter, e.g., 8 inches. The MEASUREMENT and REFERENCE may represent the same physical quantity as the calibration standard, e.g., length. In this case, STANDARD provides the same information as REFERENCE and is normally absent to avoid redundancy. It is possible, however, for MEASUREMENT and REFERENCE to represent a different physical quantity, e.g., voltage. In this case, STANDARD is needed to describe the transformation from the physical quantity represented by the MEASUREMENT and REFERENCE to the physical quantity of the calibration standard, e.g., from millivolts to inches. Deriving this transformation may require using STANDARD from more than one CALIBRATION-MEASUREMENT object.

14. PLUS-TOLERANCE indicates by how much each measurement sample value may exceed a reference and still be "within tolerance". Elements shall be non-negative numbers. If a measurement sample value is an array, then so is its reference and plus tolerance. A measurement sample value is within tolerance if each of its elements is less than or equal to the sum of the corresponding reference and plus tolerance elements. Plus tolerance represents in some sense the maximum acceptable drift of each recorded measurement sample *above* the value of the recorded reference. If PLUS-TOLERANCE is absent, then the plus tolerance is implicitly infinite.

15. MINUS-TOLERANCE indicates by how much each measurement sample value may fall short of a reference and still be "within tolerance". Elements shall be non-negative numbers. If a measurement sample value is an array, then so is its reference and minus tolerance. A measurement sample value is within tolerance if each of its elements is greater than or equal to the difference of the corresponding reference and minus tolerance elements. Minus tolerance represents in some sense the maximum acceptable drift of each recorded measurement sample *below* the value of the recorded reference. If MINUS-TOLERANCE is absent, then the minus tolerance is implicitly infinite.

9.4 DLIS-Context

9.4.1 A DLIS-CONTEXT object is used to establish a context for the data related to a particular origin. It is the object referenced from the ORIGIN:CONTEXT attribute for data using the DLIS schema.

Table 8-4 — DLIS-CONTEXT Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	PRODUCT	c=1, r=ASCII, u=
2	VERSION	c=1, r=ASCII, u=
3	PROGRAMS	r=ASCII, u=
4	ORDER-NUMBER	c=1, r=ASCII, u=
5	DESCENT-NUMBER	
6	RUN-NUMBER	
7	WELL-ID	c=1, u=
8	WELL-NAME	c=1, r=ASCII, u=
9	FIELD-NAME	c=1, r=ASCII, u=
10	COMPANY	r=ASCII, u=
11	PRODUCER-CODE	c=1, r=UNORM, u=
12	PRODUCER-NAME	c=1, r=ASCII, u=

*Notes:

1. PRODUCT is the name of the software product (e.g., the trademarked acquisition or interpretation software system) that produced the data associated with this object.
2. VERSION is the version of the product specified by PRODUCT.
3. PROGRAMS is a list of the names of specific programs or services, operating as part of the software specified by PRODUCT, that were used to generate the data associated with this object.
4. ORDER-NUMBER is a unique accounting number associated with the acquisition or creation of the data associated with this object. It is typically known as the service order number.
5. DESCENT-NUMBER is meaningful to the producer. The meaning of this attribute is specified by the producer to the consumer by means external to DLIS.
6. RUN-NUMBER is meaningful to the company or companies specified in COMPANY. The meaning of this attribute is specified to the producer by means external to DLIS.
7. WELL-ID is a codified identifier of the well in or about which measurements were taken. Whenever applicable, the API Well Number should be used. This is a unique, permanent, numeric identifier assigned to a well in accordance with the American Petroleum Institute Bulletin D12A, January, 1979.
8. WELL-NAME is the name of the well.
9. FIELD-NAME is the name of the field to which the well belongs. If there is no field, then the value shall be WILDCAT.
10. COMPANY is a list of names of the client company or companies for which the data was acquired or computed, typically the operator of the well and partners.
11. PRODUCER-CODE is the producer's API Recommended Practice 66 organization code as found in Appendix A. The producer is the organization whose authorized agent generated the logical file containing this object using software programs developed under the sponsorship of the organization. This code is assigned on request by the Production Department of the API at 1220 L Street, N.W., Washington, D.C. 20005. *This attribute is required.*
12. PRODUCER-NAME is the producer's business or organization name.

9.5 Equipment

9.5.1 An EQUIPMENT object describes an item of surface or downhole equipment used in the acquisition of data. The purpose of this object type is to record information about individual pieces of equipment of any sort that is used during a job. The TOOL object then collects equipment together in ensembles that are more readily recognizable to the consumer.

Table 8-5 — EQUIPMENT Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	TRADEMARK-NAME	c=1, r=ASCII, u=
2	STATUS	c=1, r=STATUS, u=
3	TYPE	c=1, r=IDENT I TIDENT, u=, v=(see note)
4	SERIAL-NUMBER	c=1, r=IDENT, u=
5	LOCATION	c=1, r=IDENT I TIDENT, u=, v=(see note)
6	HEIGHT	c=1
7	LENGTH	c=1
8	MINIMUM-DIAMETER	c=1
9	MAXIMUM-DIAMETER	c=1
10	VOLUME	c=1
11	WEIGHT	c=1
12	HOLE-SIZE	c=1
13	PRESSURE	c=1
14	TEMPERATURE	c=1
15	VERTICAL-DEPTH	c=1
16	RADIAL-DRIFT	c=1
17	ANGULAR-DRIFT	c=1

*Notes:

1. TRADEMARK-NAME is the name used by the producer to refer to the equipment.
2. STATUS indicates the operational status of the equipment.
3. TYPE is a reference value that indicates the generic type of equipment (see Part 9).
4. SERIAL-NUMBER is the serial number of the equipment instance.
5. LOCATION is a reference value that indicates the general location of the equipment (see Part 9).
6. HEIGHT applies only to equipment located in the borehole. It is the height of the bottom of the equipment above the tool zero point (see 9.7.5) when the tool string containing the equipment is vertical. This value is positive when the equipment bottom is above the tool zero point and is negative when the equipment bottom is below the tool zero point. There is normally one piece of equipment for which the height is zero.
7. LENGTH is the length of the equipment and is typically measured from bottom make up point to top make up point. It may not apply to surface equipment. The total length of the tool string may not equal the sum of the lengths of all the equipment that make up the tool string, since some equipment may slip over other equipment. Such "slip-on" equipment includes, for example, standoffs, centralizers, and excentralizers. Similarly, the height of a piece of equipment may be independent of the lengths of the equipment below it.
8. MINIMUM-DIAMETER applies to equipment used in the borehole. It is the minimum outer diameter of the equipment. This is defined to be the minimum horizontal cross-sectional diameter measured when the equipment is in a vertical configuration. For extendible or compressible equipment (e.g., caliper arms and centralizers), this measurement indicates the smallest operational diameter possible.
9. MAXIMUM-DIAMETER applies to equipment used in the borehole. It is the maximum outer diameter of the equipment. This is defined to be the maximum horizontal cross-sectional diameter measured when the equipment is in a vertical configuration. For extendible or compressible equipment (e.g., caliper arms and centralizers), this measurement indicates the largest operational diameter possible.
10. VOLUME is the volume of the equipment and is typically used to determine buoyant weight of the equipment. It may not apply to surface equipment.
11. WEIGHT is the weight of the equipment in air. It may not apply to surface equipment.
12. HOLE-SIZE applies to equipment in the borehole. It is the minimum borehole diameter for which the equipment may reasonably be used.
13. PRESSURE is the maximum operational pressure rating of the equipment, when applicable.
14. TEMPERATURE is the maximum operational temperature rating of the equipment, when applicable.
15. VERTICAL-DEPTH is the vertical depth of an equipment item that is normally stationary (see 2.36).
16. RADIAL-DRIFT is the radial drift of an equipment item that is normally stationary (see 2.28).
17. ANGULAR-DRIFT is the angular drift of an equipment item that is normally stationary (see 2.2).

9.6 Message

9.6.1 A MESSAGE object is used to write a textual message tied to other data by means of a time stamp and other indexing attributes. Such messages typically represent operator interaction with the system, e.g., a "scroll" of the logging session, or informational messages concerning events that occurred during the session.

Table 8-6 — MESSAGE Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	TYPE	c=1, r=IDENT I TIDENT, u=, v=(see note)
2	TIME	c=1
3	MEASURED-DEPTH	c=1
4	VERTICAL-DEPTH	c=1
5	RADIAL-DRIFT	c=1
6	ANGULAR-DRIFT	c=1
7	TEXT	r=ASCII, u=

*Notes:
1. TYPE is a reference value indicating the source and purpose of the message (see Part 9). *This attribute is required.*
2. TIME is the time the message was issued.
3. MEASURED-DEPTH is the measured depth of the tool zero point at the time message was issued (see 2.20).
4. VERTICAL-DEPTH is the vertical depth of the tool zero point at the time message was issued (see 2.36).
5. RADIAL-DRIFT is the radial drift of the tool zero point at the time message was issued (see 2.28).
6. ANGULAR-DRIFT is the angular drift of the tool zero point at the time message was issued (see 2.2).
7. TEXT is the text of the message. *This attribute is required.*

9.7 Path

9.7.1 A PATH object is used to identify channels that represent the coordinates of a path along which a logging tool string may pass and to specify certain geometric features of the tool string relative to the path.

9.7.2 Log data consists of a sequence of values (i.e., a channel) traversing a locus in space and time. A locus in space and time is a sequence of distinct points, each of which, in the most general case, has a three-dimensional position coordinate, and a time coordinate. The sequence $\{value_i, position_i, time_i\}$ is called a data path, and each member of the sequence is called a step on the data path. The sequence $\{position_i, time_i\}$ is the locus of the data path. Note that it is possible for two points on a locus to occupy the same position in space so long as they occupy that position at different times. In the extreme case, a locus can have a fixed position.

9.7.3 A complete position coordinate is made up of three components that correspond to the spatial coordinate system of a well (see Figure 8-1): depth (measured or vertical); radial drift; and angular drift. Occasionally, both measured and vertical depth components are known and are recorded together.

9.7.4 Data paths are represented as subgroups of channels in frames. Some or all of the components of a data path may be recorded; other components may be unknown or irrelevant. The $\{value_i\}$ sequence, known as the data path's value channel, is always recorded. The mechanism for defining data paths is the PATH object. PATH objects are not needed to decode frames, but they add informational value to the contents of frames.

9.7.5 A tool string has three points of particular interest. They are its measure point, its tool zero point, and its data reference point (see Figure 8-3). These points give rise to a depth offset and a measure point offset which are critical for interpreting the relation between index channels and data channels in a frame type.

9.7.6 A channel is sampled at its measure point, which is a fixed position relative to the tool string, whenever a data reference point (another fixed position relative to the tool string) passes certain positions along the well. If the well positions are equally spaced in depth, and if the interval between the measure point and the data reference point is not evenly divisible by the sampling interval, the channel will have a depth offset.

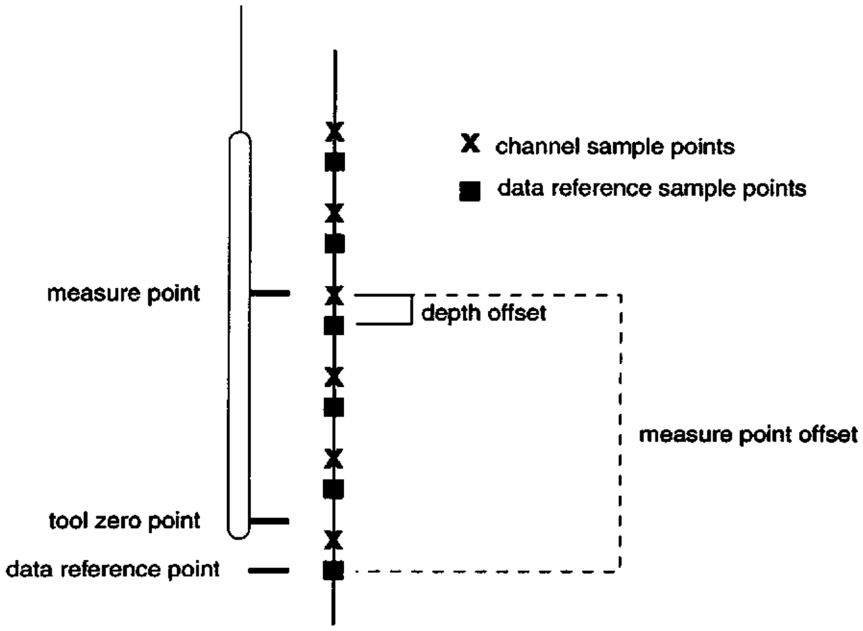


Figure 8-3—Tool String Configuration

Table 8-7 — PATH Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	FRAME-TYPE	c=1, r=OBNAME, u=
2	WELLBORE-PATH-DATUM	c=1, r=OBNAME, u=
3	VALUE	r=OBNAME, u=
4	MEASURED-DEPTH	c=1
5	VERTICAL-DEPTH	c=1
6	RADIAL-DRIFT	c=1
7	ANGULAR-DRIFT	c=1
8	TIME	c=1
9	DEPTH-OFFSET	c=1

Table 8-7 — PATH Attributes (Continued)

*Note	Attribute Label	Restrictions
10	MEASURE-POINT-OFFSET	c=1
11	TOOL-ZERO-OFFSET	c=1

*Notes:

1. FRAME-TYPE is the name of a FRAME object that describes the frame type in which the path channels are recorded.
2. WELLBORE-PATH-DATUM is the name of a WELLBORE-PATH-DATUM object that describes the wellbore path datum for this path.
3. VALUE is a list of names of CHANNEL objects that describe one or more value channels for this path. *This attribute is required.*
4. MEASURED-DEPTH is a constant measured depth coordinate for this path if r is numeric, or the name of a CHANNEL object if r=OBNAME that describes a measured depth channel for this path.
5. VERTICAL-DEPTH is a constant vertical depth coordinate for this path if r is numeric, or the name of a CHANNEL object if r=OBNAME that describes a vertical depth channel for this path.
6. RADIAL-DRIFT is a constant radial drift coordinate for this path if r is numeric, or the name of a CHANNEL object if r=OBNAME that describes a radial drift channel for this path.
7. ANGULAR-DRIFT is a constant angular drift coordinate for this path if r is numeric, or the name of a CHANNEL object if r=OBNAME that describes a angular drift channel for this path.
8. TIME is a constant time coordinate for this path if r is numeric, or the name of a CHANNEL object if r=OBNAME that describes a time channel for this path. If r is numeric and not DTIME, it represents elapsed time since ORIGIN:CREATION-TIME.
9. DEPTH-OFFSET is a depth offset, which indicates how much VALUE is "off depth". This is meaningful only when there is a measured depth channel for this path. If D is the value of the measured depth channel in a frame and D′ is the actual known measured depth at which channels in the frame were sampled, then D = D′ + depth offset.
10. MEASURE-POINT-OFFSET is a measure point offset, which indicates a fixed distance along measured depth from the value channel's measure point to a data reference point. This is a special case that depends on the data acquisition model and applies only when there is a recorded measured depth channel for this Path. If MEASURE-POINT-OFFSET is zero or absent, then the time channel for this path is explicitly related to the value channel. That is, in each frame, v_i is sampled at t_i. If the MEASURE-POINT-OFFSET is present and non-zero, the time channel is instead explicitly related to the data reference point and is implicitly related to the value channel. In each frame, t_i is the time that the data reference point was at d_i, which is the frame's measured depth. The value channel sample v_i is still considered to be sampled at d_i, but at a time different from t_i. The explicit time for the value channel can be recovered using the knowledge that at time t_i when the data reference point was at depth d_i, the value channel measure point was at depth d_i–measure point offset. Typically, only a single time channel per origin will be recorded in a frame type, the one explicitly associated with the data reference point.
11. TOOL-ZERO-OFFSET is the distance of the data reference point for this path above the tool string's tool zero point. It may be positive or negative and is frequently zero. Specifically, data reference point = tool zero point + tool zero offset.

9.8 Splice

9.8.1 A SPLICE object is used to identify the component channels of a splice and the splice points. A splice is the result of taking values of two or more distinct channels (e.g., from different runs) from mutually disjoint intervals to produce a resultant channel defined over the union of the intervals.

Table 8-8 — SPLICE Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	OUTPUT-CHANNEL	c=1, r=OBNAME, u=
2	INPUT-CHANNELS	r=OBNAME, u=
3	ZONES	r=OBNAME, u=

*Notes:

1. OUTPUT-CHANNEL is the name of a CHANNEL object that represents the spliced channel, i.e., the resultant of the splice operation. The spliced channel may be implied by the SPLICE object and need not actually exist. When the spliced channel does exist, its PROPERTIES attribute shall include the reference value SPLICED.

Table 8-8 — SPLICE Attributes (Continued)

2. INPUT-CHANNELS is a list of names of CHANNEL objects that represent the input channels of the splice operation. *This attribute is required.*

3. ZONES is a list of names of ZONE objects that describe mutually disjoint intervals in which the spliced channel is defined. When present, ZONES count matches INPUT-CHANNELS count. The spliced channel is derived from the kth input channel in the kth zone. If ZONES is absent, then INPUT-CHANNELS count shall be 1, and API basic schema UPDATE objects are used to indicate where input channels change.

9.9 Tool

9.9.1 A TOOL object is used to record information about a logging tool and its component equipment parts.

9.9.2 TOOL objects specify ensembles of equipment that work together to provide specific measurements or services. Such combinations are more recognizable to the consumer than are their individual pieces. A typical tool consists of a sonde and a cartridge and possibly some appendages such as centralizers and spacers. It is also possible to identify certain pieces or combinations of surface measuring equipment as tools.

Table 8-9 — TOOL Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	TRADEMARK-NAME	c=1, r=ASCII, u=
2	GENERIC-NAME	c=1, r=ASCII, u=
3	PARTS	r=OBNAME, u=
4	STATUS	c=1, r=STATUS, u=
5	CHANNELS	r=OBNAME, u=
6	PARAMETERS	r=OBNAME, u=

*Notes:

1. TRADEMARK-NAME is the name used by the producer to refer to the tool.
2. GENERIC-NAME is the name generally used within the industry to refer to tools of this type.
3. PARTS is a list of names of EQUIPMENT objects that represent the parts of the tool.
4. STATUS indicates whether the tool is enabled to provide information to the acquisition system or whether it has been disabled and is simply occupying space.
5. CHANNELS is a list of names of CHANNEL objects describing channels that are produced directly by this tool. A channel shall not be produced directly by more than one tool. Channels that have multiple tool sources should be associated with their indirect tool sources via API basic schema PROCESS objects.
6. PARAMETERS is a list of names of PARAMETER objects describing parameters that directly affect or reflect the operation of this tool. Parameters may be shared by different tools.

9.10 Wellbore-Path-Datum

9.10.1 A WELLBORE-PATH-DATUM object is used to record the location of a wellbore path datum.

This object type was called WELL-REFERENCE-POINT in API Recommended Practice 66, Version 1. The name change was made to correspond to standard terminology found in the Petroleum Industry Data Dictionary (PIDD).

Table 8-10 — WELLBORE-PATH-DATUM Attributes

*Note	Attribute Label	Restrictions
	DESCRIPTION	c=1, r=ASCII, u=
	EXTENDED-ATTRIBUTES	r=OBJREF, u=
1	PERMANENT-DATUM	c=1, r=ASCII, u=
2	VERTICAL-ZERO	c=1, r=ASCII, u=
3	PERMANENT-DATUM-ELEVATION	c=1
4	ABOVE-PERMANENT-DATUM	c=1
5	MAGNETIC-DECLINATION	c=1
6	COORDINATE-1-NAME	c=1, r=ASCII, u=
7	COORDINATE-1-VALUE	c=1
8	COORDINATE-2-NAME	c=1, r=ASCII, u=
9	COORDINATE-2-VALUE	c=1
10	COORDINATE-3-NAME	c=1, r=ASCII, u=
11	COORDINATE-3-VALUE	c=1

*Notes:

1. PERMANENT-DATUM is a permanent entity or structure (e.g., ground level) from which vertical distance can be measured.
2. VERTICAL-ZERO is a particular entity (e.g., kelly bushing) that corresponds to zero measured depth.
3. PERMANENT-DATUM-ELEVATION is the distance of the PERMANENT-DATUM above mean sea level. A negative value indicates the distance is below mean sea level.
4. ABOVE-PERMANENT-DATUM is the distance of VERTICAL-ZERO above PERMANENT-DATUM. If negative, then VERTICAL-ZERO is below PERMANENT-DATUM.
5. MAGNETIC-DECLINATION is the angle with vertex at the wellbore path datum determined by the line of direction to geographic north and the line of direction to magnetic north. A positive value indicates that magnetic north is east of geographic north. A negative value indicates that magnetic north is west of geographic north.
6. COORDINATE-1-NAME is the name of the first of three independent spatial coordinates, such as longitude or latitude or elevation, that can be used to locate the wellbore path datum.
7. COORDINATE-1-VALUE is the numerical value of the coordinate named by COORDINATE-1-NAME.
8. COORDINATE-2-NAME is the name of the second of three independent spatial coordinates, such as longitude or latitude or elevation, that can be used to locate the wellbore path datum.
9. COORDINATE-2-VALUE is the numerical value of the coordinate named by COORDINATE-2-NAME.
10. COORDINATE-3-NAME is the name of the third of three independent spatial coordinates, such as longitude or latitude or elevation, that can be used to locate the wellbore path datum.
11. COORDINATE-3-VALUE is the numerical value of the coordinate named by COORDINATE-3-NAME.

9.11 Updatable Attributes

9.11.1 Table 8-11 lists attributes of the DLIS schema that may be updated using UPDATE objects.

Table 8-11 — Updatable Attributes

*Note	Object Type	Attribute Label
1	CALIBRATION-COEFFICIENT	COEFFICIENTS
2	SPLICE	INPUT-CHANNELS

*Notes:

1. Update of COEFFICIENTS indicates an adjustment or correction of some kind. The updated values may not be directly derived from measurements.
2. When a splice operation begins, it may not be known what all of the input channels will be. It must be possible to provide this information on the fly. The INPUT-CHANNELS attribute may be updated if and only if the ZONES attribute is absent. In this case, the count of INPUT-CHANNELS is 1, and an update indicates a change in the input channel at the frames indicated by the UPDATE object.

Recommended Practices for Exploration and Production Data Digital Interchange

Part 9: DLIS Dictionary

Exploration and Production Department

API RECOMMENDED PRACTICE 66, V2
SECOND EDITION, JUNE 1996

American
Petroleum
Institute

CONTENTS

Recommended Practice for Exploration and Production Data Digital Interchange
Part 9: DLIS Schema Dictionary

0 Introduction

0.1 This standard contains introductory, normative, and annotative information. Introductory information comprises everything that precedes Section 1, Scope. The introductory clauses describe how the publication is organized and provide commentary on the history and purpose of this standard. Normative information includes the actual requirements that must be satisfied by any data conforming to the standard. Annotative information is provided as rationale for and illustrations about the normative information and does not constitute part of the standard. The standard is completely stated even if all introductory and annotative information are removed. Annotative information may refer to terms introduced in later paragraphs or parts in order to tie together concepts, i.e., from time to time its value may increase after the reader has made a complete pass over the standard.

0.2 Different styles are used to distinguish between normative and annotative information.

0.3 All normative information is written using this same font, and is contained in either numbered paragraphs or numbered tables. Normative text is aligned with the left margin of the page.

0.4 All figures are annotative, and all annotative text is written in unnumbered paragraphs in Helvetica italic font, indented, as shown here:

This is a paragraph of annotative text. Its purpose is to include informal commentary on the normative information in immediately preceding or following paragraphs and may include references to or usage of normative information in other parts of the standard.

1 Scope

This part lists and describes reference values for attributes belonging to the Digital Log Interchange Standard (DLIS) schema (see Part 8).

2 Authority

Changes to the DLIS schema dictionary are recommended by the API Subcommittee On Standard Format For Digital Well Data and approved by the Executive Committee on Drilling and Production Practice of the API Exploration and Production Department. Changes may include addition of new terms or removal of obsolete terms. A term removed in one edition may be restored in a later edition only if restored with its previous meaning. A new edition of the DLIS schema dictionary occurs when approved by the Executive Committee and has an edition number obtained by adding 1 to the previous edition number.

3 Concepts

3.1 The tables presented here contain the reference values defined under the DLIS schema for attributes of either the API basic schema or the DLIS schema. There is one table per attribute. Each attribute is identified by its label preceded by the object type to which it belongs and separated by a colon (:). Object types belonging to the API basic schema are identified by the prefix '0/' denoting its schema code. Tables are presented alphabetically by object type first and by attribute second.

3.2 All DLIS schema reference values use representation code IDENT if written in DLIS schema objects and TIDENT if written in API basic schema objects.

4 Calibration-Measurement Reference Values

4.1 Phase

4.1.1 Currently-defined values correspond to the phase of a calibration measurement.

Table 9-1 — CALIBRATION-MEASUREMENT:PHASE Reference Values

Reference Value	Description
AFTER	After survey calibration
BEFORE	Before survey calibration
MASTER	Master calibration

5 0/Channel Reference Values

5.1 Kind

5.1.1 Currently-defined values correspond to measurements of index channels in and about the borehole.

Table 9-2 — 0/CHANNEL:KIND Reference Values

Reference Value	Description
ANGULAR-DRIFT	Channel value represents angular drift.
MEASURED-DEPTH	Channel value represents measured depth.
NON-STANDARD	Channel value is a non-standard measurement.
RADIAL-DRIFT	Channel value represents radial drift.
TIME	Channel value represents elapsed time.
VERTICAL-DEPTH	Channel value represents vertical depth.

The reference value MEASURED-DEPTH replaces BOREHOLE-DEPTH used in API Recommended Practice 66, Version 1. The change is made to achieve conformance with terminology found in the Petroleum Industry Data Dictionary (PIDD).

5.2 Properties

Table 9-3 — 0/CHANNEL:PROPERTIES Reference Values

Reference Value	Description
AUXILIARY	Data not normally included has been provided at specific request of the consumer, possibly at some additional cost.
AVERAGED	Data is the average of two or more other sources.
CALIBRATED	A calibration process has been applied to the data.
CHANGED-INDEX	Data has been re-sampled along an index that is different from its original sampling index. For example, data that was originally sampled according to time is re-sampled according to depth, or data originally sampled according to measured depth is re-sampled according to vertical depth.

Table 9-3 — 0/CHANNEL:PROPERTIES Reference Values (Continued)

Reference Value	Description
COMPUTED	Data is the output of a transform, e.g., function former, based on input data from a single tool.
DEAD-TIME-CORRECTED	Data has been corrected for dead time.
DEPTH-MATCHED	Data has been aligned along its depth index against a reference data source.
DERIVED	Data is an output of empirical equations or is the solution of log response equations based on input data from more than one tool.
ENVIRONMENTALLY-CORRECTED	All known borehole corrections have been applied to the data.
FILTERED	A filtering process has been applied to the data.
HOLE-SIZE-CORRECTED	Data has been corrected for hole size effect based on the value of a hole size input.
INCLINOMETRY-CORRECTED	Data has been corrected to standard directional references (vertical axis and North axis).
LITHOLOGY-CORRECTED	Data has been corrected or computed based on the value of a matrix lithology parameter.
LOCAL-COMPUTATION	Data is the result of locally-defined computational expressions. Such data is normally experimental.
LOCALLY-DEFINED	Object's name has been created by the operator. This name might not be dictionary-controlled and may have no semantic association with the object's data.
MODELLED	Data is the output of a theoretical model and is not derived from any measured quantity.
MUD-CORRECTED	Data has been corrected for effects of mud.
MUDCAKE-CORRECTED	Data has been corrected for mudcake effect.
NORMALIZED	Data has been corrected so that its range corresponds to a prescribed norm.
OVER-SAMPLED	Interpolated data samples have been added to the original data to align this data with other data.
PATCHED	Original data values have been replaced at specific levels. This is normally done to remove spurious values (e.g., spikes).
PRESSURE-CORRECTED	Data has been corrected for hydrostatic pressure (mud weight).
PROPRIETARY	Data is proprietary to the producer and is not warranted for use by the consumer.
RE-SAMPLED	Data has been resampled along its original index. For example, the original index values may be wrong. Using a curve-fit, it may be possible to compute new data values that fit the original signal more accurately along the original index.
SALINITY-CORRECTED	Data has been corrected for salinity effect.
SAMPLED-DOWNWARD	Original sampling direction is downward.
SAMPLED-UPWARD	Original sampling direction is upward.
SPEED-CORRECTED	Data has been corrected for variations of the downhole tool speed.
SPLICED	Data has been obtained by concatenating two or more other data sources.
SQUARED	Data is the result of a squaring process, i.e., a process that converts a smooth function into a step function.
STACKED	Data is the sum of two or more other data sources.
STANDARD-DEVIATION	Data represents the estimated deviation, due to environmental factors or incoherence of the computational model, of another data source.
STANDOFF-CORRECTED	Data has been corrected for standoff effect.
TEMPERATURE-CORRECTED	Data has been corrected for temperature effect based on the value of a temperature source.
UNDER-SAMPLED	Some of the original data samples have been discarded to reduce the amount of data or to align this data with other data.

6 0/Computation Reference Values

6.1 Properties

See Table 9-3.

7 Equipment Reference Values

7.1 Location

Table 9-4 — EQUIPMENT:LOCATION Reference Values

Reference Value	Description
LOGGING-SYSTEM	Equipment is on or in the logging system unit.
REMOTE	Equipment is on the surface away from the rig and logging unit system.
RIG	Equipment is on the rig.
WELL	Equipment is in the borehole.

7.2 Type

Table 9-5 — EQUIPMENT:TYPE Reference Values

Reference Value	Description
ADAPTER	Adapter
BOARD	Processor board
BOTTOM-NOSE	Bottom nose
BRIDLE	Bridle
CABLE	Cable
CALIBRATOR	Calibrator
CARTRIDGE	Cartridge
CENTRALIZER	Centralizer
CHAMBER	Sample chamber
CUSHION	Water cushion
DEPTH-DEVICE	Depth measuring device
DISPLAY	Display
DRAWER	Processor drawer
EXCENTRALIZER	Excentralizer
EXPLOSIVE-SOURCE	Explosive source
FLASK	Flask
GEOPHONE	Geophone
GUN	Gun
HEAD	Head
HOUSING	Housing
JIG	Calibration jig
JOINT	Joint
NUCLEAR-DETECTOR	Nuclear detector
PACKER	Packer
PAD	Pad
PANEL	Panel
POSITIONING	Positioning device
PRESSURE-GAUGE	Pressure gauge
PRINTER	Printer
RADIOACTIVE-SOURCE	Radioactive source
SHIELD	Shield
SIMULATOR	Simulator
SKID	Skid

Table 9-5 — EQUIPMENT:TYPE Reference Values (Continued)

Reference Value	Description
SONDE	Sonde
SPACER	Spacer
STANDOFF	Standoff
SYSTEM	Processor system
TOOL	Tool
TOOL-MODULE	Tool module
TRANSDUCER	Transducer
VIBRATION-SOURCE	Vibration source

8 Message Reference Values

8.1 Type

Table 9-6 — MESSAGE:TYPE Reference Values

Reference Value	Description
COMMAND	Operator command
RESPONSE	Operator response
SYSTEM	System message

9 0/Parameter Reference Values

9.1 Properties

See Table 9-3.

10 0/Zone Reference Values

10.1 Domain

Table 9-7 — 0/ZONE:DOMAIN Reference Values

Reference Value	Description
MEASURED-DEPTH	Zone interval is measured depth.
TIME	Zone interval is elapsed time.
VERTICAL-DEPTH	Zone interval is vertical depth.

The reference value MEASURED-DEPTH replaces BOREHOLE-DEPTH used in API Recommended Practice 66, Version 1. The change is made to achieve conformance with terminology found in the Petroleum Industry Data Dictionary (PIDD).

Recommended Practices for Exploration and Production Data Digital Interchange

Appendix A: Organization Codes

Exploration and Production Department

API RECOMMENDED PRACTICE 66, V2
SECOND EDITION, JUNE 1996

American
Petroleum
Institute

CONTENTS

Recommended Practice for Exploration and Production Data Digital Interchange Appendix A: Organization Codes

0 Introduction

0.1 This standard contains introductory, normative, and annotative information. Introductory information comprises everything that precedes Section 1, Scope. The introductory clauses describe how the publication is organized and provide commentary on the history and purpose of this standard. Normative information includes the actual requirements that must be satisfied by any data conforming to the standard. Annotative information is provided as rationale for and illustrations about the normative information and does not constitute part of the standard. The standard is completely stated even if all introductory and annotative information are removed. Annotative information may refer to terms introduced in later paragraphs or parts in order to tie together concepts, i.e., from time to time its value may increase after the reader has made a complete pass over the standard.

0.2 Different styles are used to distinguish between normative and annotative information.

0.3 All normative information is written using this same font, and is contained in either numbered paragraphs or numbered tables. Normative text is aligned with the left margin of the page.

0.4 All figures are annotative, and all annotative text is written in unnumbered paragraphs in Helvetica italic font, indented, as shown here:

This is a paragraph of annotative text. Its purpose is to include informal commentary on the normative information in immediately preceding or following paragraphs and may include references to or usage of normative information in other parts of the standard.

1 Scope

Table A-1 contains a list of *organization codes* assigned by the American Petroleum Institute, Exploration and Production Department (API E&P) for use in API Recommended Practice 66.

Several of the organization codes in this appendix are historical in nature and reflect the well log origins of API Recommended Practice 66.

2 Assignment of Organization Codes

Organization codes are assigned by API Exploration and Production Department, which maintains the current list of codes. To request a new organization code, contact:

American Petroleum Institute
Exploration and Production Department
1220 L Street, N.W.
Washington, D.C. 20005
Phone: (202) 682-8000
FAX: (202) 682-8426

Table A-1 — Organization Codes

Code	Organization
0	API Subcommittee On Recommended Format For Digital Well Data, Basic Schema
1	Operator
2	Driller
3	Mud Logger
10	Analysts, The
20	Baroid
30	Birdwell
40	BPB
50	Brett Exploration
60	Cardinal
65	Center Line Data
66	API Subcommittee On Recommended Format For Digital Well Data, DLIS Schema
70	Century Geophysical
77	CGG Logging, Massey France
80	Charlene Well Surveying
90	Compagnie de Services Numerique
95	Comprobe
100	Computer Data Processors
110	Computrex
115	COPGO Wood Group
120	Core Laboratories
125	CRC Wireline, Inc.
127	Davis Great Guns Logging, Wichita, KS
129	Digicon Exploration, Ltd.
130	Digigraph
137	Digital Logging Inc., Tulsa, OK
140	Digitech
145	Deines Perforating
150	Dresser Atlas
160	Earthworm Drilling
170	Electronic Logging Company
180	Elgen
190	El Toro
200	Empire
210	Frontier
215	Geolog
217	Geoshare
220	G O International
230	Gravilog
240	Great Guns Servicing
250	Great Lakes Petroleum Services
260	GTS
268	Guardian Data Seismic Pty. Ltd.
270	Guns
280	Halliburton Logging
285	Horizon Production Logging
290	Husky
300	Jetwell
310	Lane Wells
315	Logicom Computer Services
320	Magnolia
330	McCullough Tool
335	Mincom Pty Ltd
337	MR-DPTS Ltd.
338	NRI On-Line Inc.
339	Oilware, Inc.
340	Pan Geo Atlas
345	Perfco
350	Perfojet Services
360	Perforating Guns of Canada
362	Petroleum Exploration Computer Consultants, Ltd.
366	Phillips Petroleum Company
370	Petroleum Information
380	Petrophysics

Table A-1 — Organization Codes (Continued)

Code	Organization
390	Pioneer
395	Q. C. Data Collectors
400	Ram Guns
410	Riley's Datashare
420	Roke
430	Sand Surveys
440	Schlumberger
450	Scientific Software
460	Seismograph Service
462	SEGDEF
463	SEG Technical Standards High Density Media Format Subcommittee
464	Shell Service Co.
465	Stratigraphic Systems, Inc.
470	Triangle
480	Welex
490	Well Reconnaissance
495	Wellsite Information Transfer Specification (WITS)
500	Well Surveys
510	Western
520	Westronics
525	Winters Wireline
530	Wireline Electronics
540	Worth Well
560	Z & S Consultants Limited
999	Reserved for local schemas
1000	Petrotechnical Open Software Corp.

PC-01200—06/96—1M (4E)

Additional copies available from API Publications and Distribution:
(202) 682-8375

Information about API Publications, Programs and Services is available on the World Wide Web at: http://www.api.org

American Petroleum Institute
1220 L Street, Northwest
Washington, D.C. 20005-4070
202-682-8000

Order No. G66002