Designation: G 135 – 95 (Reapproved 2001)

Standard Guide for
Computerized Exchange of Corrosion Data for Metals[1]

This standard is issued under the fixed designation G 135; the number immediately following the designation indicates the year of original adoption or, in the case of revision, the year of last revision. A number in parentheses indicates the year of last reapproval. A superscript epsilon (ε) indicates an editorial change since the last revision or reapproval.

1. Scope

1.1 This guide defines the techniques used to encode corrosion of metals test results for exchange between computer systems.

1.2 Guidelines are given for creating a data exchange appendix for each ASTM corrosion of metals standard.

1.3 Instructions are given for creating data translation software from the contents of the data exchange appendix.

2. Referenced Documents

2.1 *ASTM Standards:*
G 106 Practice for Verification of Algorithm and Equipment for Electrochemical Impedance Measurements[2]
G 107 Guide for Formats for Collection and Compilation of Corrosion Data for Metals for Computerized Database Input[2]

2.2 *ANSI Standards:*[3]
ANSI/ISO 9899: 1990 [1992] Programming Language C
ANSI X3.4-1986: Coded Character Set 7 Bit ASCII

3. Terminology

3.1 *Definitions:*

3.1.1 *datatype*—a group of rules specifying the format of an object.

3.1.2 *global data*—information shared among several standards.

3.1.3 *local data*—information specific to a certain standard.

3.1.4 *semantics*—information meaning.

3.1.5 *syntax*—information format.

3.1.6 *tagged object*—a named block of information.

3.1.7 *translator*—a computer routine which writes or reads data files.

4. Significance and Use

4.1 This guide establishes a formalism for transferring corrosion test data between computer systems in different laboratories. It will be used by standards developers to specify the format of files containing test results.

4.2 This guide defines a generic approach to structuring data files. It will be used by software developers to create programs which read and write these files.

4.3 Each standard test procedure will define a unique data file derived from this guide. Each time a standard test is performed, the results can be summarized in a data file specific to that test.

4.4 Some experimental information will be global, that is, common to several standards, and will be contained in Guide G 107 and other global data dictionaries. Other information will be local, that is, unique to a given standard, and will be defined in that standard.

5. Guide for Standards Authors

5.1 *Local and Global Data*:

5.1.1 Some information may be used across several corrosion standards, that is, global. Global data is defined in Guide G 107 and other global standards.

5.1.2 Some information may be local to a particular corrosion standard. Local data is defined in the standard's data exchange appendix.

5.2 *Data File*:

5.2.1 Each test will generate a single test data file. File name formats are not specified.

5.2.2 The data file is arranged as a set of named or tagged objects. Each time a standard test is performed a set of objects is obtained. The data file can be thought of as a permanent repository for this set of objects.

5.2.3 Each tagged object will take two or more lines in the data file. Lines are strings of ASCII (ANSI X3.4-1986) characters terminated with a carriage return/linefeed character pair or a single linefeed character.

5.2.4 Lines are further subdivided into tab delimited ASCII fields that are particularly suitable for manipulation by spreadsheet and scientific charting programs. For example, Fig. 1 shows how a section of a data file would show up on printed output.

5.3 *Tagged Object*:

5.3.1 A tagged object is a repository for an individual block of information. It may be a simple piece of data, the test date for example, or it may be complex, such as a current/voltage/ time curve. A tagged object contains three subordinate areas: (*1*) the tag, (*2*) the datatype, and (*3*) the actual data. The tag and datatype are the first two fields of the first line while the actual data is contained in subsequent lines. Data lines are always indented one tab space. This is illustrated in Fig. 2.

5.3.2 *Tag*:

5.3.2.1 The object's tag is a simple string that uniquely identifies it among other objects in a tagged object set.

5.3.2.2 When implementing a translator for a given standard, the implementation is free to define other tagged object names so long as they don't clash with those defined in the standard. It is suggested that additional names be prefixed with some unlikely and unique combination of alphanumeric characters so that name conflicts do not arise in future versions of the standard. For example; NewTest_Apex Potential.

5.3.2.3 Tags are made up of one or more character strings separated by periods. The first character in each string must be alphabetic (including the underscore). Subsequent characters may be alphanumeric.

5.3.2.4 Periods should only be used to associate different objects together. For example, Matl.Class, Matl.SubClass, Matl.TradeName, are all aspects of Material. In future specifications it is suggested that this be done using complex, multifield datatypes.

5.3.2.5 Periods should not be used to separate multiple word individual concepts. Instead use capitalization or underscore. For example; ControlMode or Control_Mode.

5.3.2.6 Tags are case insensitive although mixed case is suggested for readability.

5.3.3 *Datatype*:

5.3.3.1 Each object has a datatype which specifies the format of the object's data.

5.3.3.2 Global datatypes are defined in a global data exchange standard such as Guide G 107 and are repeated here for reference, as follows:
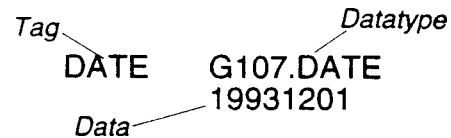

FIG. 2 The Elements of a Tagged Object

(a) *String (STRING)*—Strings contain purely character information. Strings may be further encoded depending on the semantic description of the object.

(b) *Quantity (QUANT)*—Quantities represent numeric values along with their units. Units may be further encoded depending on the semantic description of the object.

(c) *Date (DATE)*—Dates are simple day specifiers.

(d) *Time (TIME)*—Times are simple time of day specifiers.

(e) *Category Set (SET)*—Category sets are used to represent choices. The actual meaning of each value is given in the semantic description of the object.

(f) *Tabular (TABLE)*—Tables are used to hold arrays of records. The datatype, units, and name of each column is also encoded.

5.3.3.3 A particular implementation of a test is free to define local datatypes as long as they don't clash with those defined in global standards. These local datatypes are defined in the standard's data exchange appendix.

5.3.3.4 The datatype has a unique identifier made up of a standard number and a name separated by a period; for example, G107.SET. Each time an object is recorded in the data file, the datatype identifier is recorded with the object. That identifier specifies to the translator (either computer or human) what data format to use in reading the data from or writing the data to the file.

5.3.3.5 In cases where the reading translator is unable to find a datatype in its internal table, that object will be marked as untranslated. The translator is free to take the appropriate action depending on the importance of the object.

5.3.3.6 It is important to note that the datatype doesn't completely specify the meaning of the data, only its format. For example, a value of one for the tag "Surface.Condition" has a very different meaning than the value of one for the tag "Potentiostat.ControlMode" even though they are both of type G107.SET. Those meanings are construed from the tag.

5.3.4 *Data*:

5.3.4.1 The object's data is arranged in a format defined by the datatype. Data starts in the second line of the data object.

Standard	G107.STRING				
	ASTM G106				
Date	G107.DATE				
	19921103				
ControlMode	G107.SET				
	1				
Spectrum	G107.TABLE				
	QUANT	QUANT	QUANT	QUANT	QUANT
	Freq	Signal	ZReal	ZImag	StdDev
	Hz	V	Ohm	Ohm	None
	0.10	0.10	0.1	0.0	0.99
	0.20	0.10	0.12	0.1	0.99

FIG. 1 Data File Sample

There may be multiple lines and multiple fields associated with a data object. Each data line is indented by one tab space to distinguish it from the tag/datatype lines.

5.4 *Data Exchange Appendix*:

5.4.1 Standard tests that use this guide will contain a data exchange appendix. This appendix contains the data and format information required to define test data files. For an example see Appendix X1.

5.4.1.1 The data exchange appendix should have three parts, the local datatype definitions section, the object definition table, and a sample data file.

5.4.2 The local datatype definitions section gives a description of and formal syntax for each local datatype. This gives the rules of translation to programmers who are creating translators for the standard.

5.4.2.1 The rules should be written using the formal language described in Section 7. The translation rules for several data types are given in Section 6. The QUANT type is reproduced in Fig. 3 as an example:

5.4.3 The object definition table is a tabular listing of all the objects in the file. For example, consider Table 1. There are four objects in this table: Standard, Date, ControlMode, and Spectrum. In an actual standard there may be many more.

5.4.3.1 Each row of the table defines a data object. These objects may be copied from global standards such as Guide G 107 or may be locally defined. An object definition should not refer to another standard test since a revision of that test may change the object definition without warning.

5.4.3.2 *Column Definitions* (as illustrated in Table 1):

NOTE 1—Columns 1, 2, and 3 are required for global objects. Columns 1 to 6 are required for local objects.

(*a*) *Reference Number (Column 1)*—The reference number is a unique number referring back to the standard and paragraph where the data object is defined. This number is made up of a Standard ID and a paragraph number separated by a period(.).

(*b*) *Tag/Column Tag (Column 2)*—This column contains the data tag. If the data object is tabular, this column will also contain sub-tags or headings for each column of the object.

(*c*) *Required (Column 3)*—This column indicates whether a particular object is required or can be safely omitted from the data file.

(*d*) *Description (Column 4)*—The description column contains free form text describing the object. Constraints, defaults, or other specifications may show up here.

(*e*) *Datatype (Column 5)*—The type gives the datatype of the object.

The data types may be global types defined in Guide G 107, or they may be local to the standard being written.

(*f*) *Category Set/Units/Column Information (Column 6)*—This column varies depending on the datatype. If the type is a SET, Column 6 contains the allowed values and meanings. If the type is a QUANT, Column 6 contains suggested units. If the type is a TABLE, Column 6 contains units or allowed values as required by the datatype of each column.

5.4.4 The last required section of the data exchange appendix is a sample data file. This should show a file as actually printed although data may be omitted for the sake of space.

6. Guide for File Translator Programmers

6.1 The following section is intended for programmers who are writing data exchange translators. A translator is a portion of a program which reads or writes a data file. Production rules are shown in bold-face Courier font.

6.1.1 *Character Set*—The data is stored in an ASCII text file which can be directly printed using most printers and manipulated using most text editors (see Fig. 4).

6.1.2 *File*—The data file is arranged as a sequence of tagged objects.

File : = TaggedObject [1 . .*]

6.1.3 *Tagged Object*—A tagged object starts with its tag line and includes all the information up to the next tag. Any other lines associated with the object must be indented one tab character. Each line is terminated with a line feed or carriage return/line feed pair.

TaggedObject : = TagLine DataBlock

6.1.3.1 *Tag Line*—The first line of a tagged object is called the Tag Line. It contains the tag or name of the object and the format specifier.

TagLine : = TagField FormatField {CommentField} NewLine

6.1.3.2 *Tag*—A tag must start with an alphabetic character or underscore ( ). Thereafter numeric characters can be used as well as alphabetic and underscore. Tags may not contain spaces. The only other punctuation allowed is a period (.) character. Any character following a period must be alphabetic or underscore. Tags are not case sensitive.

TagField : = Tag Tab
Tag : = Identifier (Period Identifier) [0 . .*]
Identifier : = AlphabeticChar AlphanumericChar [0 . .*]

```
QuantDataBlock := QuantDataLine


QuantDataLine := Tab QuantField UnitField {CommentField} NewLine


QuantField := RealNumber Tab


RealNumber := {'+'|'-'} Numeral[1..*]{'.'

    Numeral[0..*]}{('e'|'E'){'+'|'-'} Numeral[1..*]}


UnitField = StringField
```
FIG. 3 Translation Rules for the Quant Data Object

3

TABLE 1 Object Definitions

Reference Number/ Column Number	Tag/Column Tag	Required	Description/Column Description	Type/Column Type	Category Set/Suggested Units/ Column Information
G 107.5.1.4.1	Standard	Yes	Standard test specification	STRING	
G 107.5.1.4.3	Date	Yes	date test started	DATE	
G 106.X2.1	ControlMode	Yes	Circuit configuration	SET	Allowed Values
					1. Potentiostat
					2. Galvanostat
					3. ZRA
					4. V applied/No feedback
					5. I applied/No feedback
					6. Other
G 106.X2.2	Spectrum	Yes	Corrected frequency spectrum	TABLE	
Column 1	Frequency		Frequency	QUANT	Hz
Column 2	Signal		Applied Signal	QUANT	V, A
Column 3	ZReal		Z real	QUANT	ohm
Column 4	ZImag		Z imaginary	QUANT	ohm
Column 5	StdDev		Standard Deviation	QUANT	none

```
NewLine := '\n' | ('\r' '\n')

        // Handles both UNIX style and DOS style text files


Tab := '\t'


Period := '.'


AlphabeticChar := ('a'-'z')|('A'-'Z')|'_'


AlphanumericChar := AlphabeticChar | ('0'-'9')


PunctuationChar := '~'|'`'|'!'|'@'|'#'|'$'|'%'|'^'|'&'|'*'

        |'('|')'|'_'|'-'|'+'|'='|'\'|'|'|'['|'{'|']'|'}'

        |':'|'''|'"'|','|'<'|'.'|'>'|'/'|'?'|' '

        // Doesn't include semicolon


Semicolon := ';'


PrintableCharExceptSemicolon := AlphanumericChar | PunctuationChar


PrintableChar := AlphanumericChar | PunctuationChar | Semicolon
```
FIG. 4 Definition of the Character Set

6.1.3.3 *Format*—The second field in the Tag Line is a format specifier, either by paragraph reference; for example, Guide G 107.7.1.4.1, or by shorthand name; for example, STRING. The format specifier specifies how the rest of the tagged object is to be translated.

FormatField : = {Organization Period} Standard Period Identifier Tab

6.1.3.4 *Data Lines*—Any lines following the tag line up to the next tag line are data lines associated with the tagged object. As previously stated, data lines must begin with a tab so that they may be distinguished from tag lines. The data lines are subdivided into tab delimited fields, that is, each data field is followed by a tab.

DataBlock : = DataLine [0 . .*]
DataLine : = Tab DataField [0 . .*] {CommentField} NewLine

6.1.3.5 *Data Fields*—Data fields may not start with semi-colons. Data fields must contain only printable characters. Data fields must not contain any internal tab characters since a tab indicates the end of the field. The format and meaning of characters in a data field are defined in the object format.

```
DataField : = PrintableCharExceptSemicolon PrintableChar [0 . .*]
                Tab
```

6.1.4 *Comments*:

6.1.4.1 Comment lines may be interspersed in among data lines. Comment lines must start with a tab and then a semicolon (;). The line is not translated or counted in any way. Any computer program reading the data file should discard the comment line before the tagged object is translated.

6.1.4.2 Comments may also be added to the end of a line. These start with a semicolon and continue to the end of the line. A semicolon may be safely embedded within a piece of string data but must not start the field. If a semicolon does start the field, that field and the rest of the line will be assumed to be a comment and discarded in translation.

```
CommentLine : =  Tab CommentField NewLine // i.e. a lone
                    comment
CommentField : =  Semicolon String// String includes all characters
                    up to new line
```

6.1.5 *Datatypes*:

6.1.5.1 *String (STRING)*—A string is an undifferentiated series of characters. Strings may contain printable punctuation characters and simple blanks. They may not begin with a semicolon since this is the directive for an untranslated comment.

```
StringDatablock : = StringDataLine
StringDataLine : = Tab StringField {CommentField} NewLine
StringField : = PrintableCharExceptSemicolon PrintableChar [0 . .*]
                Tab
```

6.1.5.2 *Quantity (QUANT)*:

(a) A quantity is made up of a real number and a unit. The table or reference paragraph gives the suggested units for the field. Real numbers are printed in a U.S. format; for example, the decimal point is a period. The characters *E* or *e* are used to indicate the start of an exponent although the exponent is not required. Both mantissa and exponent can be signed although a positive sign is not required.

(b) The unit is a simple string. Some translators may attempt to parse this string to adjust for a system of measurement or a magnitude prefix.

```
QuantDataBlock : = QuantDataLine
QuantDataLine : =  Tab QuantField UnitField {CommentField}
                    NewLine
QuantField : = RealNumber Tab
RealNumber : = {′ + ′|′−′} Numeral [1 . .*]{′.′
   Numeral [0 . .*]}{(′e′|′E′){′ + ′|′−′} Numeral [1 . .*]}
UnitField : = StringField
```

6.1.5.3 *Date (DATE)*—A date is a string of eight numeric characters encoding year, month, and day in the order YYYYMMDD.

```
DateDataBlock : = DateDataLine
DateDataLine : = Tab DateField {CommentField} NewLine
DateField : = Year Month Day
Year : = Numeral [4]
Month : = Numeral [2]
Day : = Numeral [2]
```

6.1.5.4 *Time (TIME)*—A time is a string of six numeric characters encoding hour, minute, and second in the order HHMMSS.

```
TimeDataBlock : = TimeDataLine
TimeDataLine : = Tab TimeField {CommentField} NewLine
DateField : = Hour Minute Second
Hour : = Numeral [2] // 24 hour clock, local time.
Minute : = Numeral [2]
Second : = Numeral [2]
```

6.1.5.5 *Category Set (SET)*—A category set is a closed list of values for a particular object. The active member of a category set is represented by an integer value. Category sets should not be used for integral quantities. Use the quantity type, instead. The last column of the table gives a list of acceptable values and their meanings.

```
SetDataBlock : = SetDataLine
SetDataLine : = Tab SetField {CommentField} NewLine
SetField : = Numeral [1 . .*]
```

6.1.5.6 *Tabular (TABLE)*—A tabular field is made up of a group of values. The last column gives the title and type of each value. The first data row gives the primitive datatype of the column. The second row gives the subtag or symbolic name of the column. The third row gives the column units. The remaining rows up to the next tag contain the actual data (see Fig. 5).

7. Formal Language Specification

7.1 Computerized information exchange is often specified in terms of a formal language definition. This guide uses a definition method based on Backus-Naur syntax. This section describes this syntax.

7.1.1 *Production Rules*—Production rules consist of a production name defined, a ″: = ″, and a list of lower level productions. For example;

```
DimensionalNumber : = RealNumber UnitString
```

7.1.2 *Repeated Terms*—Square brackets are used as a shorthand notation for duplicated terms. For example the following production has 0 or 1 comments:

```
DataLine : = Tab DataField CommentField [0 . .1] NewLine
```

An asterisk is used in the second place to indicate no upper limit for the number of duplicated terms. For example;

```
TaggedData : = TagLine DataLine [0 . .*]
```

7.1.3 *Optional Terms*—The particular choice of taking an item 0 or 1 times; for example, the item may be missing, can also be represented by curly braces. For example;

```
DataLine : = Tab DataField {CommentField} NewLine
```

7.1.4 *Choice*—A vertical bar is used to indicate a choice between two alternate productions. For example;

```
Number : = RealNumber | IntegerNumber
```

7.1.5 *Range*—A dash is used to compress a range of choices. For example;

```
LowerCaseAlpha : = ′a′−′z′
//as opposed to the more cumbersome ′a′ | ′b′ | ... | ′z′
```

```
TableDataBlock := TableDatatypeLine TableSymbolLine TableUnitsLine
        TableDataLine[0..*]

TableDatatypeLine := Tab TableDatatypeField[0..*] {CommentField} NewLine
        // All column widths must be consistent within a given table

TableDatatypeField := ("STRING"│"QUANT"│"SET"│"DATE"│"TIME") Tab

TableSymbolLine := Tab StringField[0..*]

TableUnitsLine:= Tab StringField[0..*]

TableDataLine := Tab TableDataField[0..*] NewLine

TableDataField = (StringField│QuantField│SetField│DataField) Tab
        // DataField must be consistent with datatype field
```

FIG. 5 Formal Definition of the Table Data Object

7.1.6 *Characters*—Individual characters are shown in single quotes. If the character is not printable, a C language (see ANSI/ISO 9899) character constant will be used. For example;

ControlMode : = '1'
LineFeed : = 'ICs n'

7.1.7 *Strings*—Double quotes are used to indicate a sequence of characters. The rules for translating characters within a string are the same as in the ANSI standard C language. For example:

StringID : = "STRING"

7.1.8 *Grouping*—Parentheses may be used to group expressions. For example;

DimensionalList : = (Number Units)[0 . .*]

7.1.9 *Comments*—Double slashes are used to indicate comments at the end of a rule. For example:

EndOfLine : = LineFeed | (CarriageReturn LineFeed)
// handles either Unix or DOS file

8. Keywords

8.1 computerization; corrosion; data; database; data exchange

APPENDIX

(Nonmandatory Information)

X1. SAMPLE DATA EXCHANGE APPENDIX

NOTE X1.1—The following is a sample data exchange appendix created for Practice G 106.

X1.1 *Local Datatypes:*

X1.1.1 *MATERIAL*—The MATERIAL datatype identifies the metal being tested.

X1.1.1.1 *Production Rules*—See Fig. X1.1.

X1.2 Object definitions are illustrated in Table X1.1.

X1.3 Fig. X1.2 is a typical output data file produced according to this appendix.