Standard Commands for Programmable Instruments (SCPI)

Volume 3: Data Interchange Format

VERSION 1999.0
May, 1999
Printed in U.S.A.

Table of Contents

Chapter 6 Block Descriptions

1 Introduction

The data interchange format described in this section lets software packages and instruments share waveform and other data. It is designed to be flexible and extensible, and can accommodate a wide range of data structures and formats. In addition to describing the data itself, the data interchange format provides information regarding the data environment, structure, and other related specifications. Only a minimum of information about the data is actually required but a wide range of information can be associated with the data. Complex data dimensioned sets and simpler data are equally representable using the block structure of this data format.

Because this data format is hierarchically structured, it can be subordinate to other structures. It is suitable as a file format and its encoding formats allow it to be easily transmitted across *IEEE 488.1/488.2*, *RS-232*, *SCSI*, *IEEE 802,* and various other transmission media and protocols.

The syntax of the data interchange format is compatible with *IEEE 488.2* syntax. With minor modifications, an *IEEE 488.2* parser, or at least its lexical analyzer, can also analyze the syntax of this format. While the format imposes its own rules within *IEEE 488.2* expressions, these rules conform to the *IEEE 488.2* restrictions for expressions.

1.1 Overview

The data interchange format is block-structured. A data set generally consists of several description blocks and a data block. A block contains keywords and subordinate blocks. Within a block or sub-block are keywords with values that define characteristics of the data or the environment in which it was acquired. Many blocks accept a modifier label value and it is required for some blocks. A block modifier allows multiples of the same block type to be distinguished from one another.

Each block addresses a different aspect of data description. The DATA block contains the data. The IDENtify block describes the manner and environment in which the data was obtained. The DIMension and ENCode blocks describe how the data is physically represented and logically organized. The TRACe and VIEW blocks provide semantic information about the data. A REMark block contains textual comments regarding the data. Finally, the DIF block identifies the block as a <dif_expression> and describes the version of SCPI used.

This data interchange format focuses on data as an abstraction of instrument- or algorithm-generated data. The method of data acquisition or creation is handled as descriptive information that is not required for proper interpretation of the data. Information about the encoding, structure, scaling, range, etc., of the data is independent of the source of the data.

For example, instrument-dependent settings and characteristics such as sample rate are translated into acquisition independent parameters such as RESolution.

1.2 Defined Blocks

The Data Interchange Format consists of the following major blocks:

- **DIF** identifies the expression as a <dif_expression> and contains the version of the standard used to create the data set.
- **REMark** contains general comments in human-readable text regarding the data.
- **IDENtify** names the data set and describes the environment in which it was generated. Environment description includes project name, test number and series, date and time, and source.
- **ENCode** specifies the encoding format for data in the DATA and ENCode blocks. Also covered are signal values for underflow, overflow and no value, as well as the range and resolution of the data.
- **DIMension** specifies the structure and format of data in the DATA block. Provision is made for data scaling, offset, naming, and units specification.
- **ORDer** specifies ordering of the data elements in the DATA(CURVe) block for each dimension.
- **TRACe** logically groups dimensions as functions, surfaces, sets, etc. TRACe blocks provide semantic information about the data and are used to build VIEW blocks. They also provide a convenient and powerful mechanism for subsetting data.
- **VIEW** provides a second level of semantic information about the data. VIEW describes logical relationships among traces defined in the TRACe block such as envelope traces.
- **DATA** contains the actual data. Different subordinate blocks describe dimensioned data, waveform parameter measurements, and point values.

1.3 Implicit and Explicit Dimensions

There are many ways to describe the structure and order of dimensioned data, and not all terms are well defined or understood. The data interchange format treats all CURVe data as coordinates in n-dimensional space. If a coordinate is actually present in the data, its dimension is said to be explicit. If the coordinate value of a dimension is not present as a data value, but is instead derived from the location of other coordinate values, the dimension is said to be implicit.

Suppose you mark off an evenly spaced grid on the floor of a room, and then make measurements of temperature and humidity at some (varying) height above the floor. As you make these measurements, you write down the temperature and humidity measurements and the location (X, Y, and Z positions relative to one corner of the room). The result is 5-tuple. You must write each set of measurements in a consistent order (such as X, Y, Z, temperature, and humidity), but you may write the sets of measurements in any order.

A 5-tuple may be viewed, as this format does, as a point in a 5-dimensional space. All five dimensions are explicit, meaning that for each dimension, you have explicitly written down its coordinate.

Now suppose you make the measurements according to some consistent pattern. You start with the first grid line and make the measurements sequentially at each Y point along the X grid line. You then make the measurements along the second X grid line, and so on, until the last grid line is completed.

If you keep track of the order in which the measurements were made, you only need to write down two measurements: temperature and humidity. The other three measurements (X location, Y location, and height) are derivable from the order of the data. The data is still 5-dimensional, but three of the dimensions are implicit and two are explicit.

Simple waveforms acquired from instruments like oscilloscopes (when clocked internally) commonly consist of a time series of voltage (vertical axis) measurements. For the time-voltage space, only the voltage value or coordinate is provided by the oscilloscope; the time (horizontal axis) is derived from the order of the data, an initial starting time, and a scale factor. The voltage dimension is explicit and the time dimension is implicit.

This format classifies all dimensions as implicit or explicit. The order of the data, which is essential to the correct derivation of implicit dimension coordinates or values, is strictly defined by the ORDer block or its default values.

2 Style

The data interchange format uses, for the most part, standard *IEEE 488.2* syntactic elements along with the nested parentheses which together comprise an Expression type parameter ("Volume 1: Syntax and Style," Section 8).

The format also uses the concept of "friendly" listening and "precise" talking followed by *IEEE 488.2*. There are some differences, which are clarified in 3 of this section.

Keywords may take single or multiple values. If a keyword takes multiple values, all values shall be of the same type.

A few keywords have default values. If the keyword is omitted, the default value prevails. These values are specified in the block descriptions in 6 of this section.

An invalid set reports either a Command Error or an Execution Error as described in "Volume 2: Command Reference," 21.8.9 and 21.8.10. While uppercase characters are specified, lowercase characters may be accepted without error if interpretation of the data set is unambiguous.

It is not required that a data format parser accept all values of an enumerated set (3.4.10 of this volume). An error may be generated on receipt of an unimplemented value.

3 Syntax

The data format is block-structured. Each hierarchical level is introduced by a block name, with its subordinate elements enclosed in parentheses. A block may have a modifier and may contain subordinate blocks and keyword units, or both. Keyword units consist of a keyword followed by one or more values. If a keyword has more than one value, the values are separated by commas.

The following example shows a simple data set. It is formatted so that all blocks and keywords are indented to show their hierarchical relationship.


```
(DIF (VERSion 1993.0)
IDENtify (
        NAME “Data Format Example”
        TEST (
                NUMBer “7D4", ”2.4"))
ENCode (
        HRANge 75
        LRANge 25)
DIMension=X (
        TYPE IMPLicit
        SCALe 0.01
        SIZE 7
        UNITs “S”)
DIMension=Y (
        TYPE EXPLicit
        SCALe 0.02
        OFFSet 0.1
        UNITs “V”)
DATA (
        CURVe (
                VALues 49.0, 48.0, 50.2, 61.3, 68.5, 38.6, 48.0)))
```

The data interchange format overall is formatted as an *IEEE 488.2* <EXPRESSION PROGRAM DATA> element. Within this element, the various blocks, modifiers, keywords, and value types are composed of syntactic elements that, for the most part, are identical to the corresponding types specified in *IEEE 488.2* or a subset of them.

3.1 Character Set

All syntactic elements except <Block> contain only 7-bit ASCII-formatted (*ANSI X3.4-1977*) data with the high-order eighth bit always set to zero.

3.2 White Space

White space consists of the ASCII space character, 32 decimal. A space is used along with the parentheses, comma, and equal sign to separate syntactic elements.

When an instrument accepts data, redundant white space may appear without semantic effect anywhere except within: block names, block modifiers, keywords, and all value types.

Note that the characters constituting white space may be a part of the semantic content of <String> and <Block> value types.

When an instrument sends data, redundant white space is not allowed.

3.3 Blocks, Block Modifiers, and Keywords

When an instrument accepts data, block names, block modifiers, and keywords conform to the *IEEE 488.2* <CHARACTER PROGRAM DATA> syntax. Abbreviated short forms and any specified aliases shall be accepted.

When an instrument sends data, the *IEEE 488.2* <CHARACTER RESPONSE DATA> syntax is used. If a long and short form element is specified, the short form shall be sent.

As the data format is hierarchically organized, mnemonic generation rules for compound headers apply. See "Syntax and Style'" 2.2.1. Specific syntactic restrictions on the use of blocks, block modifiers, and keywords are covered in 4 of this section.

3.4 Value Types

The grammar employs the following value types.

3.4.1 <Numeric>

The <Numeric> value type is used to specify zero, positive and negative numeric values.

When an instrument accepts data, <Numeric> follows one of the following two *IEEE 488.2* program data formats:

<DECIMAL NUMERIC PROGRAM DATA>, or
<NON-DECIMAL NUMERIC PROGRAM DATA>

Contrary to *IEEE 488.2*, <Numeric> values with embedded white spaces do not have to be accepted.

When an instrument sends data, <Numeric> follows one of the following six response data formats:

<NR1 NUMERIC RESPONSE DATA>,
<NR2 NUMERIC RESPONSE DATA>,
<NR3 NUMERIC RESPONSE DATA>,
<OCTAL NUMERIC RESPONSE DATA>,
<BINARY NUMERIC RESPONSE DATA>, or
<HEXADECIMAL NUMERIC RESPONSE DATA>

3.4.2 <+Numeric>

The <+Numeric> value type is used to specify positive numeric values greater than zero. Other than this restriction, the value type is identical to <Numeric>.

3.4.3 <0+Numeric>

The <0+Numeric> value type is used to specify positive numeric values equal to or greater than zero. Other than this restriction, the value type is identical to <Numeric>.

3.4.4 <NR1>

The <NR1> value type is used to specify zero, positive and negative integer numeric values.

An instrument shall accept any <Numeric> value, rounding as necessary to obtain an integer value. When an instrument sends data, <NR1> follows the IEEE 488.2 <NR1 DECIMAL NUMERIC RESPONSE DATA> format.

Note that rounding may result in an invalid integer value.

3.4.5 <+NR1>

The <+NR1> value type is used to specify positive integer numeric values greater than zero. Other than this restriction, the value type is identical to <NR1>.

3.4.6 <0+NR1>

The <0+NR1> value type is used to specify positive numeric values equal to or greater than zero. Other than this restriction, the value type is identical to <NR1>.

3.4.7 <Block>

The <Block> value type is used to specify 8-bit, 16-bit, 32-bit, or 64-bit significant data. <Block> is only allowed for use with the DATA(CURVe(VALues)) keyword.

The FORMat keyword from the applicable ENCode block determines the interpretation of the data values. Bytes are grouped according to the number of bits required. For example, bytes are grouped by four's and interpreted as a 32-bit integer for INT32.

When an instrument accepts data, <Block> values shall follow the format of *IEEE 488.2* <ARBITRARY BLOCK PROGRAM DATA> format with the restriction that only definite length blocks are accepted.

When an instrument sends data, <Block> values shall follow the format of *IEEE 488.2* <DEFINITE LENGTH ARBITRARY BLOCK RESPONSE DATA> format.

3.4.8 <String>

The <String> is used to represent quoted strings.

When an instrument accepts data, <String> data conforms to *IEEE 488.2* <STRING PROGRAM DATA> syntax.

When an instrument sends data, <String> data conforms to *IEEE 488.2* <STRING RESPONSE DATA> syntax.

3.4.9 <Label>

The <Label> value type is used to represent short, unquoted string labels for block modifier and keyword values.

When an instrument accepts data, <Label> follows the *IEEE 488.2* <CHARACTER PROGRAM DATA> syntax.

When an instrument sends data, <Label> follows the *IEEE 488.2* <CHARACTER RESPONSE DATA> syntax. A <Label> has no short form.

3.4.10 Enumerated Set

Some keywords specify a value from an completely enumerated set of values.

When an instrument accepts data, an enumerated set member follows the *IEEE 488.2* <CHARACTER PROGRAM DATA> syntax. If specified, acceptance of long and short form as well as aliases is required.

When an instrument sends data, an enumerated set member follows the *IEEE 488.2* <CHARACTER RESPONSE DATA> syntax. If both a long and short form element are specified, only the short form shall be sent.

It is not required that a data format parser accept all values of an enumerated set. An error may be generated on receipt of an unimplemented value. See 2 of this section.

Mnemonic generation rules for character data apply. See “Syntax and Style,” 2.2.1.

4 Grammar

This chapter specifies the grammar of the data format. This grammar describes the contents of a <dif_expression> that represents a Data Interchange Format.

4.1 Grammar Notation

The grammar notation consists of the non-terminal symbols, pseudo-terminal symbols, meta-symbols, and terminal symbols defined below.

4.1.1 Non-terminals

Non-terminals are represented as all lowercase words.

4.1.2 Pseudo-terminals

Pseudo-terminals stand for any from a set of terminal symbols (value types) as described in 3.4 of this section and are represented as words bracketed between < and >.

4.1.3 Meta-symbols

In the following table, "." represents an arbitrary string of terminals and non-terminals. The meta-symbols for the grammar are ->, {, }, *, +, [,], and |.

They have the following meanings:

->	Separates the left and right hand sides of a production
{ . }*	The enclosed item occurs zero or more times.
{ . }+	The enclosed item occurs one or more times.
[.]	The enclosed item occurs zero or one times.
{ . | . | . }	One and only one of the two or more enclosed items separated by | occurs

4.1.4 Terminal Symbols

Terminal symbols are represented in mixed uppercase and lowercase to indicate long and short form ("Syntax and Style," 1).

4.2 Grammar Description

```
dif ->          difid
                [remark]
                [identify]
                [encode]
                {dimension}+
                [order]
                {trace}*
                {view}*
                {data}+
```

```
difid ->     DIF(
                [NOTE          <String>]
                VERSion        <+Numeric>
                [SCOPe         PREamble | FULL]
             )

remark ->    REMark(
                [NOTE          <String> {,<String>}*]
             )

identify ->  IDENtify (
                [NOTE          <String>]
                [NAME          <String>]
                [TECHnician    <String> {,<String>}*]
                [PROJect       <String>]
                [DATE          <NR1>,<+NR1>,<+NR1>
                               [,<NR1>,<+NR1>,<+NR1>]]
                [TIME          <0+NR1>,<0+NR1>,<0+Numeric>
                               [,<0+NR1>,<0+NR1>,<0+Numeric>]]
                [UUT(
                   [NAME       <String>]
                   [ID         <String> {,<String>}*]
                   [DESign     <String> {,<String>}*]
                )]
                [TEST(
                   [NAME       <String>]
                   [SERies     <String>]
                   [NUMBer     <String> {,<String>}*]
                )]
                [HISTory       <String> {,<String>}*]
             )

encode ->    ENCode (
                [NOTE          <String>]
                [FORMat        { IFP32 | IFP64
                               | INT8 | INT16 | INT32 | INT64
                               | SFP32 | SFP64
                               | SINT16 | SINT32 | SINT64
                               | SUINT16 | SUINT32 | SUINT64
                               | UINT8 | UINT16 | UINT32 | UINT64 }]
                [NVALue        <Numeric>]
                [ORANge        <Numeric>]
                [URANge        <Numeric>]
                [HRANge        <Numeric>]
                [LRANge        <Numeric>]
```

```
                  [RESolution    <+Numeric>]
               )

dimension -> DIMension = <Label>(
                  [NOTE          <String>]
                  [NAME          <String>]
                  TYPE           {IMPLicit | EXPLicit}
                  [SCALe         <Numeric>]
                  [OFFSet        <Numeric>]
                  [SIZE          <+NR1>]
                  UNITs          <String>
                  [encode]
               )

order ->       ORDer (
                  [NOTE          <String>]
                  [BY            {TUPLe | DIMension}]
               )

trace ->       TRACe = <Label>(
                  [NOTE          <String>]
                  [NAME          <String>]
                  [SYMMetry      <String>]
                  {INDependent(
                     LABel       <Label>
                     [STARt      <+NR1>]
                     [STOP       <+NR1>]
                  )}*
                  {DEPendent(
                     LABel       <Label>
                  )}*
               )

view ->        VIEW = <Label> (
                  [NOTE          <String>]
                  [NAME          <String>]
                  {ENVelope(
                     UPPer       <Label>
                     LOWer       <Label>
                     [FUNCtion   <Label>]
                  )}*
               )
```

```
data ->      DATA [= <Label>](
                [NOTE          <String>]
                [delta]
                [curve]
                {waveform}*
                {measurement}*
             )

delta ->     DELTa (
                [NOTE          <String>]
                {DIMension =   <Label> (
                   [SCALe      <Numeric>]
                   [OFFSet     <Numeric>]
                   [SIZE       <+NR1>]
                )}+
                [DATE          <NR1>,<+NR1>,<+NR1>
                               [,<NR1>,<+NR1>,<+NR1>]]
                [TIME          <0+NR1>,<0+NR1>,<0+Numeric>
                               [,<0+NR1>,<0+NR1>,<0+Numeric>]]
             )

curve ->     CURVe (
                [NOTE   <String>]
                [NAME   <String>]
                [CTYPe  {SUM8| CRC16| CCITT | SUM16}]
                [VALues {<Block> | <Numeric>}
                        [,{<Block> | <Numeric>}]*]
                [CSUM   <+NR1>]
             )
```

NOTE: The presence of VALues is conditional on the DIF(SCOPe) keyword; refer to the Required Blocks, section 4.3.

```
waveform -> WAVeform[=       <Label>](
               [NOTE         <String>]
               [NAME         <String>]
               [TRACe        <Label>]
               [HLMethod     <String>]
               [HIGH         <Numeric>]
               [LOW          <Numeric>]
               [REFerence(
                  [HIGH      <Numeric>]
                  [LOW       <Numeric>]
                  [MID       <Numeric>]
                  [METHod    <String>]
               )]
               [AMPLitude    <Numeric>]
               [PWIDth       <Numeric>]
               [NWIDth       <Numeric>]
               [PERiod       <Numeric>]
               [FREQuency    <Numeric>]
               [PDUTycycle
               | DCYCle      <Numeric>]
               [NDUTycycle   <Numeric>]
               [RISE(
                  [TIME      <Numeric>]
                  [OVERshoot <Numeric>]
                  [PREShoot  <Numeric>]
               )]
               [FALL(
                  [TIME      <Numeric>]
                  [OVERshoot <Numeric>]
                  [PREShoot  <Numeric>]
               )]
               [MAXimum      <Numeric>]
               [MINimum      <Numeric>]
               [TMAXimum     <Numeric>]
               [TMINimum     <Numeric>]
               [MEAN         <Numeric>]
               [RMS          <Numeric>]
               [SDEViation   <0+Numeric>]
               [PTPeak       <0+Numeric>]
               [CYCLe(
                  [COUNt     <0+Numeric>]
                  [MEAN      <Numeric>]
                  [RMS       <Numeric>]
                  [SDEViation <0+Numeric>]
               )]
            )
```

```
measurement -> MEASurement [= <Label>](
                   [NOTE       <String>]
                   [NAME       <String>]
                   [UNITs      <String>]
                   [TYPE       <String>]
                   [TRACe      <Label>]
                   {LOCation (
                      LABel    <Label>
                      INDex    <+NR1>
                   )}*
                   [VALues     <Numeric> {,<Numeric>}*]
                )
```


4.3 Required Blocks

Note that a data set shall contain a DIF block and at least one DIMension and one DATA block.

All TRACe blocks referenced by keywords in the VIEW and DATA blocks shall be present. All DIMension blocks referenced by LABel keywords in the TRACe block shall be present.

If the DIF(SCOPe) keyword is not present or is set to FULL, any CURVe block shall contain a VALues keyword and parameters. If DIF(SCOPe) is present and set to PREamble, VALues keywords and parameters shall be omitted from all CURVe blocks.

4.4 Required Order

When an instrument accepts data, certain blocks shall be sent in the following order:

DIF
ENCode
DIMension
TRACe
VIEW
DATA

In addition, the ORDer must appear before the DATA block. The REMark and IDENtify blocks may appear anywhere after the DIF block.

Within the DATA block, the DELTa sub-block shall precede all other sub-blocks.

Within a block, individual keywords may also be accepted in any order with the exception that the CTYPe keyword must come before the VALues keyword.

When an instrument sends data, the block and keyword order specified in the grammar is required.

4.5 Unrecognized Keywords and Blocks

A listener is not required to interpret all keywords and blocks present in a data set, but only those required for the type of data of interest. For example, all listeners must interpret most keywords and sub-blocks in the DIF, DIMension, ENCode, and ORDer blocks since

information contained in these blocks is required for the correct interpretation of data in the DATA block. Exceptions are keywords such as NOTE, RESolution, HRANge, etc. Other keywords and blocks may or may not be of interest to a listener. For example, a listener that seeks only WAVeform data will ignore CURVe data, and vice versa.

This is consistent with the philosophy that different listeners may accept the same data set but use it for different purposes. In addition, the data format provides an extension mechanism that is designed to allow older listeners to accept newer data sets that may contain new keywords and sub-blocks. See 5 of this section for further details on extension mechanisms.

Therefore, listeners must be able to ignore unrecognized (but syntactically correct) keywords and blocks. The extent, if any, to which a listener notifies a user that an unrecognized keyword or block was encountered; and, the extent, if any, of required user action is implementation dependent.

Note that a block name is always followed by an open parenthesis; or, an equal sign, modifier, and open parenthesis sequence. This following structure may be ignored by searching forward to the matching closing parenthesis. A keyword is always followed by one or more comma separated values.

5 Data Format Extensions

This format may be extended and enhanced with one of the following two methods.

5.1 Extensions

Keywords, blocks (sub-blocks) may be created to accomodate information that is not otherwise provided for by existing keywords or blocks. This method of extension can be done without conflict because parsers ignore unknown keywords and blocks. Name aliases may be specifed for the new keywords or blocks. However, the use of aliases is strongly discouraged and should only be provided if there is widespread industry usage of more than one name.

5.2 Enhancements

A second method of extension involves replacing an existing keyword with a sub-block of the same name. This method is used when the semantic information represented by an existing keyword is insufficient. The replacement sub-block shall include a keyword that represents the exact same semantic information (including the same value type and range) of the original keyword. This keyword is distinguished from all other keywords in the new sub-block by appending an underscore to it. No alias names may be added for the new sub-block.

A parser which was implemented before the extension was made recognizes this special keyword by the presence of the underscore and then takes the value associated with this flagged keyword and assigns it to the old keyword. The remainder of the sub-block is discarded.

5.3 Extension Example

Suppose sometime in the future a need for a sub-block under RANGe is recognized. In this version the syntax for RANGe is:

```
RANGe       <+Numeric>
```

The extended syntax might appear as:

```
RANGe (
        HIGH_       <+Numeric>
        [LOW        <+Numeric>]
        [OVER       <+Numeric>]
)
```

A newer implementation would understand the meaning for all three values under RANGe. An older parser takes the values associated with the flagged keyword in the sub-block, in this case, HIGH_, and uses its value as the value for RANGe. Since the older parser has no need for the LOW and OVER values, they are discarded.

This extension can be continued. At some even later date, more sub-blocks are needed. The syntax might now become:

```
RANGe (
        HIGH_ (
                EXPected_    <+Numeric>
                [ALLowed     <+Numeric>]
        )
        LOW  (
                EXPected_    <+Numeric>
                [ALLowed     <+Numeric>]
        )
        OVER (
                EXPected_    <+Numeric>
                [ALLowed     <+Numeric>]
        )
)
```

The original parser could still successfully determine which value to associate with the old keyword RANGe and safely discard the rest. Both the intermediate and newest parsers can handle this syntax without conflict.

6 Block Descriptions

This chapter details all of the block descriptions for the data interface format.

6.1 Top Level Block Organization

The entire data set is composed of the following sequence of mandatory and optional blocks.

```
dif ->          difid
                [remark]
                [identify]
                [encode]
                {dimension}+
                [order]
                {trace}*
                {view}*
                {data}+
```


A data set is built from the blocks described in 6.2 through 6.10 of this section.

6.2 DATA

A DATA block contains the data of primary interest. It may contain dimensioned data such as DATA(CURVe), or specific sets of data (WAVeform, etc.). At least one DATA block is required, and multiple DATA blocks are allowed. A DATA block is defined as:

```
data ->      DATA [=    <Label>](
                [NOTE   <String>]
                [delta]
                [curve]
                {waveform}*
                {measurement}*
             )
```

<Label> is optional and allows for the unique identification of the DATA block; no two DATA blocks may have the same <Label> value within a data set.

The subordinate blocks within a particular DATA block are assumed to be related. That is, if CURVe and WAVeform measurements are present in the same DATA block, the WAVeform measurements are assumed to have been derived from the CURVe data. If that is not the case, different DATA blocks should be used for the CURVe and WAVeform measurements.

6.2.1 NOTE

This is arbitrary text.

Value type is <String>. Text should be human-readable. Only one occurrence of the NOTE keyword is allowed and only one value is allowed.

Example: NOTE “Humidity was 45%”

6.2.2 DELTa

A DELta sub-block provides for sequences of closely related data. When a data set contains multiple occurrences of DATA blocks, the DELta block differentiates the characteristics of each of the DATA blocks. Note that the DELta block is limited to redefining descriptive information, encoding information, and dimension size, such as SCALe, DATE, and OFFSet. A DELta block cannot change the organization of the data, such as the number of dimensions or the data ordering.

The values specified in a DELta block override those specified at a higher hierarchical level. For example, the value for SCALe in the DATA(DELta(DIMension)) block overrides the value for SCALe in the higher level DIMension block, but only for data within the DATA block that contains the DELta block. If there are multiple DATA blocks with subordinate DELta blocks, the DELta blocks have no effect on each other; that is, they each individually specify a modification of the same (higher level) IDENtify, DIMension, or ENCode block.

The DELta sub-block may occur only once and, if present, shall be the first block in the DATA block. The DELta sub-block is defined as:

```
delta ->     DELTa(
                [NOTE          <String>]
                {DIMension =   <Label> (
                   [SCALe      <Numeric>]
                   [OFFSet     <Numeric>]
                   [SIZE       <+NR1>]
                )}+
                [DATE          <NR1>,<+NR1>,<+NR1>
                               [,<NR1>,<+NR1>,<+NR1>]]
                [TIME          <0+NR1>,<0+NR1>,<0+Numeric>
                               [,<0+NR1>,<0+NR1>,<0+Numeric>]]
             )
```

6.2.2.1 NOTE

This is arbitrary text.

Value type is <String>. Text should be human-readable. Only one occurrence of the NOTE keyword is allowed and only one value is allowed.

Example: NOTE “Sequence 43”

6.2.2.2 DIMension

The DIMension sub-block is a limited subset of the higher level DIMension block (see 6.3 in this section). The <Label> modifier is required and is restricted to a value defined by the <Label> of a DIMension block at the same hierarchical level as the DATA block.

6.2.2.2.1 SCALe

This is the scale value for the dimension specified by the DIMension sub-block <Label>.

Value type is <Numeric>. SCALe may occur only once. See SCALe in 6.3 in this section for additional definition and restrictions.

Example: SCALe 1E-2

6.2.2.2.2 OFFSet

This is the offset value for the dimension specified by the DIMension sub-block <Label>.

Value type is <Numeric>. OFFSet may occur only once. See OFFSet in 6.3 in this section for additional definition and restrictions.

Example: OFFSet 6.0E-1

6.2.2.2.3 SIZE

The size value for the dimension specified by the DIMension sub-block <Label>.

Value type is <+NR1>. SIZE may occur only once. See SIZE in 6.3 in this section for additional definition and restrictions.

Example: SIZE 2048

6.2.2.3 DATE

This is the date of data creation or acquisition.

This keyword takes a set of three comma separated value types to specify the date corresponding to year, month, and day.

The years parameter is type <NR1>. It always includes century and millennium information.

The months parameter is type <+NR1>. Its range is 0 to 12.

The days parameter is type <+NR1>. Its range is 0 to 31 (depending on the month).

If two sets of values are specified, they define an inclusive range of dates.

Example: DATE 1990,01,01,1992,10,11 indicates a range of dates from January 1, 1990 through October 11, 1992.

6.2.2.4 TIME

This is the time of data creation or acquisition.

This keyword takes a set of three comma separated value types to specify the time corresponding to hour, minute, and second.

The hours parameter is type <0+NR1>. Its range is 0 to 23. This parameter is always expressed in 24 hour format.

The minutes parameter is type <0+NR1>. Its range is 0 to 59.

The seconds parameter is type <0+Numeric>. Its range is 0 to 60 for instruments receiving data and 0 to 59.999... for instruments sending data.

If two sets of values are specified, they define an inclusive range of time.

Example: TIME 9,0,59.095,13,1,0 indicates a range of time from 59.095 seconds after 9 AM to one minute after 1 PM.

6.2.3 CURVe

Curve data is dimensioned. Values may be organized as n-tuples or by dimension. This is determined by the ORDer block or, in its absence, the defaults for the BY keyword in the ORDer block.

Only one occurrence of CURVe is allowed in a DATA block. The CURVe block is defined as:

```
curve ->     CURVe(
                [NOTE   <String>]
                [NAME   <String>]
                [CTYPe  {SUM8| CRC16| CCITT | SUM16}]
                [VALues {<Block> | <Numeric>}
                        [,{<Block> | <Numeric>}]*]
                [CSUM   <+NR1>]
             )
```

The presence of VALues is conditional on the DIF(SCOPe) keyword; refer to the Required Blocks, section 4.3.

6.2.3.1 NOTE

This is arbitrary text.

Value type is <String>. Text should be human-readable. Only one occurrence of the NOTE keyword is allowed and only one value is allowed.

Example: NOTE “Data taken with low line voltage”

6.2.3.2 NAME

This is an arbitrary name or description of the curve.

Value type is <String>. It is recommended that the string length be 32 characters or less. NAME may occur only once and only one value is allowed.

Example: NAME “DS1C”

6.2.3.3 CTYPe

This indicates the checksum type or the absence of a checksum.

CTYPe takes a specified enumerated set of values. CTYPe may occur only once and requires one and only one of the following enumerated set values:

CRC16	16-bit cyclic redundancy check code. $x^{16} + x^{15} + x^2 + 1$. (default)
CCITT	16-bit cyclic redundancy check code. $x^{16} + x^{12} + x^5 + 1$.
SUM8	8-bit longitudinal redundancy check code (8 bit sum). $x^8 + 1$. (SUM8 is provided for compatibility and its use is discouraged).
SUM16	16-bit longitudinal redundancy check code (16 bit sum). $x^{16} + 1$.
NONE	indicates there is no checksum.

See “IEEE Micro”, Vol 8, No 4, August 1988, pp 62-76.

Note that CTYPe must preceed the VALues keyword.

Example: CTYPe SUM8

6.2.3.4 VALues

This specifies the data values.

Value type is <Block> or <Numeric>. Multiple values are allowed.

VALues shall occur exactly once if the DIF(SCOPe) keyword is not present or is set to FULL. If DIF(SCOPe) is present and set to PREamble, VALues shall be omitted.

The physical format of each value is determined by the FORMat keyword of the ENCode block that applies. If the dimension has a subordinate ENCode block, the subordinate ENCode block prevails; otherwise the ENCode block at the same hierarchical level as the DIMension blocks prevails.

Example: VALues 1.2, 3.3, 4.5, 2.7, 0.4

6.2.3.5 CSUM

This indicates the checksum for DATA(CURVe(VALues)) data

Value type is <+NR1>.

The checksum is computed according to the rules specified by CTYPe. If data is formatted in <Numeric> value type, the checksum is computed using the 8-bit ASCII decimal equivalent of each character of the number. This includes all signs, decimal points, and exponent characters but not white space.

CSUM may occur only once and takes only one value.

Example: CSUM 27

6.2.4 WAVeform

This block describes waveform parameter measurements of a two dimensional waveform. The descriptions assume a single-valued function of time in which the independent dimension is the time dimension. However, the actual units are specified in the DIMension block and it is possible to represent other quantities which are not functions of time.

Multiple occurrences of WAVeform are allowed. The WAVeform block is defined as:

```
waveform -> WAVeform[= <Label>](
               [NOTE          <String>]
               [NAME          <String>]
               [TRACe         <Label>]
               [HLMethod      <String>]
               [HIGH          <Numeric>]
               [LOW           <Numeric>]
               [REFerence(
                  [HIGH       <Numeric>]
                  [LOW        <Numeric>]
                  [MID        <Numeric>]
                  [METHod     <String>]
               )]
               [AMPLitude     <Numeric>]
               [PWIDth        <Numeric>]
               [NWIDth        <Numeric>]
               [PERiod        <Numeric>]
               [FREQuency     <Numeric>]
               [PDUTycycle
               | DCYCle       <Numeric>]
               [NDUTycycle    <Numeric>]
               [RISE(
                  [TIME       <Numeric>]
                  [OVERshoot  <Numeric>]
                  [PREShoot   <Numeric>]
               )]
               [FALL(
                  [TIME       <Numeric>]
                  [OVERshoot  <Numeric>]
                  [PREShoot   <Numeric>]
               )]
               [MAXimum       <Numeric>]
               [MINimum       <Numeric>]
               [TMAXimum      <Numeric>]
               [TMINimum      <Numeric>]
               [MEAN          <Numeric>]
               [RMS           <Numeric>]
               [SDEViation    <0+Numeric>]
               [PTPeak        <0+Numeric>]
               [CYCLe(
                  [COUNt      <0+Numeric>]
                  [MEAN       <Numeric>]
                  [RMS        <Numeric>]
                  [SDEViation <0+Numeric>]
               )]
            )
```

In the absence of a TRACe block, the following rules are used to determine the independent and dependent dimensions:

1. The independent dimension is the first implicit dimension or, if no implicit dimensions are present, then the first explicit dimension

2. The dependent dimension is the first explicit dimension if there is at least one implicit dimension. Otherwise it is the second explicit dimension.

The block modifier <Label> value is optional and allows for the unique identification of the WAVeform block. No two WAVeform blocks may have the same <Label> within the same DATA block.

If WAVeform measurements are found together with a CURVe block in a DATA block, they are considered to be derived from or related to the CURVe data (or a subset of it). However, the exact relationship of waveform measurements to the data can be determined only if the TRACe keyword is present.

It is permissible for a data set to contain waveform measurements without a DATA(CURVe) block, but DIMension blocks for the independent and dependent dimensions shall still be specified.

Waveform measurements derived from DATA block data are considered to have been derived from the data after SCALe and OFFSet (specified in the DIMension block) have been applied. WAVeform measurements are derived from DIMension block units.

Note that this standard requires the syntax for WAVeform data to be correct, but makes no claim for the correctness or consistency of the data values. Keywords in the WAVeform block are representative of terms generally used in the industry. Keyword descriptions are general and, except where specifically provided (such as for HLMethod), no requirement for specific calculation methods are claimed.

The definitions for functions and settings within the waveform block are illustrated with the following figure:

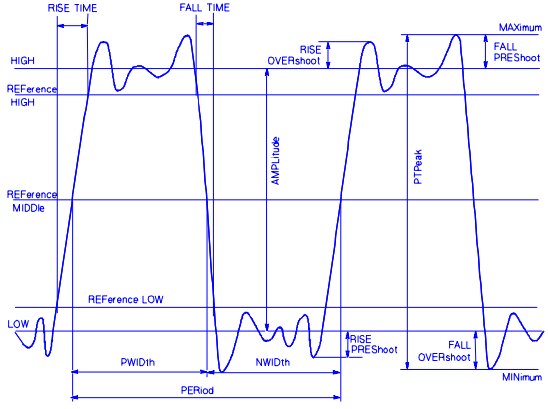

Figure 6-1 Waveform Terminology

6.2.4.1 NOTE

This is arbitrary text. Text should be human-readable.

Value type is <String>. Only one occurrence of the NOTE keyword is allowed and one value is allowed.

Example: NOTE “500 ps Reference Waveform”

6.2.4.2 NAME

This is an arbitrary name of the waveform measurement. Typically, NAME is more descriptive than the WAVeform block modifier <Label> value and it does not have to be unique.

Value type is <String>. It is recommended that the string length be 32 characters or less. NAME may occur only once and only one value is allowed.

Example: NAME “Frame bit”

6.2.4.3 TRACe

This is the TRACe from which the waveform parameters were extracted. TRACe allows a sub-set of the data to be referenced.

Value type is <Label>. Only values defined by a TRACe block modifier <Label> value are allowed.

Only one occurrence of TRACe is allowed and only one value is allowed. The TRACe referenced shall be two-dimensional.

Example: TRACe Y2

6.2.4.4 HLMethod

This notes the method used to determine the HIGH and LOW magnitudes.

For *IEEE 181* methods, HIGH becomes the more positve (less negative) and LOW becomes the more negative (less positive) of the Top Magnitude and Base Magnitude values as defined by *IEEE 194*, sections 3.2.1 and 3.2.2.

Value type is <String>. It is recommended that the string length be 32 characters or less. The following values are predefined:

UNKNOWN	Unknown or Unspecified. Default.
MEAN	Mean of Density Algorithm (*IEEE 181*, section 4.3.1.1)
MODE	Mode of Density Algorithm (*IEEE 181*, section 4.3.1.2)
PEAK	Peak Magnitude (*IEEE 181*, section 4.3.1.4)
ABSOLUTE	Absolute, user-specified levels

Example: HLMethod “MEAN”

6.2.4.5 HIGH

Indicates the high magnitude.

Value type is <Numeric>. Value is required if keyword is present. HIGH may occur only once and only one value is allowed. Units are of the dependent dimension (signal value).

Example: HIGH 4.2

6.2.4.6 LOW

This is the low magnitude.

Value type is <Numeric>. Value is required if keyword is present. LOW may occur only once and only one value is allowed. Units are of the dependent dimension (signal value).

Example: LOW -0.21

6.2.4.7 REFerence

The REFerence sub-block contains reference level information used to calculate other parameters in the WAVeform block.

6.2.4.7.1 HIGH

Indicates the high or upper reference level for measurement of rise and fall time.

Value type is <Numeric>. Value is required if keyword is present. HIGH may occur only once and only one value is allowed. Units are of the dependent dimension (signal value).

Example: HIGH 4.8

6.2.4.7.2 LOW

This is the low or lower reference level for measurement of rise and fall time.

Value type is <Numeric>. Value is required if keyword is present. LOW may occur only once and only one value is allowed. Units are of the dependent dimension (signal value).

Example: LOW 0.3

6.2.4.7.3 MID

This is the mid reference level for measurement of width, duty cycle, and period.

Value type is <Numeric>. Value is required if keyword is present. MID may occur only once and only one value is allowed. Units are of the dependent dimension (signal value).

Example: MID 3.3

6.2.4.7.4 METHod

This is the reference method. Notes the method used to determine HIGH, LOW, and MID REFerence level values.

Value type is <String>. It is recommended that the string length be 32 characters or less. The following values are predefined:

UNKNOWN	Unknown or unspecified (default).
RELATIVE	Reference levels are set according to percentages of amplitu given by:

$$high\_percentage = \frac{(WAVeform(REFerence(HIGH)) - WAVeform(LOW)) * 100}{WAVeform(AMPLitude)}$$

$$low\_percentage = \frac{(WAVeform(REFerence(LOW)) - WAVeform(LOW)) * 100}{WAVeform(AMPLitude)}$$

$$mid\_percentage = \frac{(WAVeform(REFerence(MID)) - WAVeform(LOW)) * 100}{WAVeform(AMPLitude)}$$

ABSOLUTE Reference levels set in absolute signal value units.

Example: METHod “RELATIVE”

6.2.4.8 AMPLitude

Waveform amplitude. HIGH minus LOW.

Value type is <Numeric>. Value is required if keyword is present. AMPLitude may occur only once and only one value is allowed. Units are of the dependent dimension (signal value).

Example: AMPLitude 4.03

6.2.4.9 PWIDth

Indicates positive width. Duration from a positive mid-reference crossing followed by at least one high-reference crossing to the following negative mid-reference crossing.

Value type is <Numeric>. Value is required if keyword is present. PWIDth may occur only once and only one value is allowed. Units are of the independent dimension (time).

Example: PWIDth 2.6E-7

6.2.4.10 NWIDth

Indicates negative width. Duration from a negative mid-reference crossing followed by at least one low-reference crossing to the following positive mid-reference crossing.

Value type is <Numeric>. Value is required if keyword is present. NWIDth may occur only once and only one value is allowed. Units are of the independent dimension (time).

Example: NWIDth 2.5E-7

6.2.4.11 PERiod

This is the repetition period. Duration from a mid-reference crossing (whether positive or negative) followed by at least one high-reference crossing and at least one low-reference crossing to the following mid-reference crossing in the same direction as the first mid-reference crossing.

Value type is <Numeric>. Value is required if keyword is present. PERiod may occur only once and only one value is allowed. Units are of the independent dimension (time).

Example: PERiod 1.602E-6

6.2.4.12 FREQuency

This is the frequency. FREQuency is the reciprocal of PERiod.

Value type is <Numeric>. Value is required if keyword is present. FREQuency may occur only once and only one value is allowed. Units are the reciprocal of the PERiod units.

Example: FREQuency 1.87222E+9

6.2.4.13 PDUTycycle | DCYCle

Indicates the positive dutycycle. Ratio of PWIDth to PERiod.

Value type is <Numeric>. Value is required if keyword is present. PDUTycycle may occur only once and only one value is allowed. Units are dimensionless.

Example: PDUTycycle .48

6.2.4.14 NDUTycycle

Indicates the negative dutycycle. Ratio of NWIDth to PERiod.

Value type is <Numeric>. Value is required if keyword is present. NDUTycycle may occur only once and only one value is allowed. Units are dimensionless.

Example: NDUTycycle .52

6.2.4.15 RISE

This sub-block deals with the first rising transition of the waveform where rising is defined as increasing data values.

6.2.4.15.1 TIME

This is the rise time (positive going transition). The first time interval during which the instantaneous amplitude of a waveform increases from the low reference point, REF (LOW), to the high reference point, REF (HIGH).

Value type is <Numeric>. Value is required if keyword is present. TIME may occur only once and only one value is allowed. Units are of the independent dimension (time).

6.2.4.15.2 OVERshoot

This is the rising edge overshoot. The difference between the HIGH level and the positive peak signal value to which the waveform initially rises expressed as a percentage of the waveform amplitude.

Value type is <Numeric>. OVERshoot may occur only once and only one value is allowed.

Example: OVERshoot 20.5

6.2.4.15.3 PREShoot

This is the rising edge preshoot. The difference between the LOW level and the negative peak signal value to which the waveform initially falls expressed as a percentage of the waveform amplitude.

Value type is <Numeric>. PREShoot may occur only once and only one value is allowed.

Example: PREShoot 1.2

6.2.4.16 FALL

This sub-block deals with the first falling transition of the waveform where falling is defined as decreasing data values.

6.2.4.16.1 TIME

Indicates the fall time (negative going transition). The first time interval during which the instantaneous amplitude of a waveform decreases from the high reference point, REF (HIGH), to the low reference point, REF (LOW).

Value type is <Numeric>. Value is required if keyword is present. TIME may occur only once and only one value is allowed. Units are of the independent dimension (time).

6.2.4.16.2 OVERshoot

This is the falling edge overshoot. The difference between the LOW level and the negative peak signal value to which the waveform initially falls expressed as a percentage of the waveform amplitude.

Value type is <Numeric>. OVERshoot may occur only once and only one value is allowed.

Example: OVERshoot 12.62

6.2.4.16.3 PREShoot

This is the falling edge preshoot. The difference between the HIGH level and the positive peak signal value to which the waveform initially rises expressed as a percentage of the waveform amplitude.

Value type is <Numeric>. PREShoot may occur only once and only one value is allowed.

Example: PREShoot 4.3

6.2.4.17 MAXimum

This is the maximum value of the entire waveform as specified by TRACe.

Value type is <Numeric>. MAXimum may occur only once and only one value is allowed. Units are of the dependent dimension (signal value).

Example: MAXimum 4.5903

6.2.4.18 MINimum

This is the minimum value of the entire waveform as specified by TRACe.

Value type is <Numeric>. MINimum may occur only once and only one value is allowed. Units are of the dependent dimension (signal value).

Example: MINimum -0.12

6.2.4.19 TMAXimum

This indicates the time of the first MAXimum value in the waveform as specified by TRACe.

Value type is <Numeric>. TMAXimum may occur only once and only one value is allowed. Units are of the independent dimension (time).

Example: TMAXimum 9.51

6.2.4.20 TMINimum

This indicates the time of the first MINimum value in the waveform as specified by TRACe.

Value type is <Numeric>. TMINimum may occur only once and only one value is allowed. Units are of the independent dimension (time).

Example: TMINinum 3.876

6.2.4.21 MEAN

This is the arithmetic mean. Mean value of the entire waveform as specified by TRACe.

Value type is <Numeric>. MEAN may occur only once and only one value is allowed. Units are of the dependent dimension (signal value).

Example: MEAN 2.025

6.2.4.22 RMS

This is the root mean square. Root mean square value of the entire waveform as specified by TRACe.

Value type is <Numeric>. RMS may occur only once and only one value is allowed. Units are of the dependent dimension (signal value).

Example: RMS 12.3

6.2.4.23 SDEViation

This is the standard deviation. The standard deviation of the entire waveform as specified by TRACe.

Value type is <0+Numeric>. SDEV may occur only once and only one value is allowed. Units are of the dependent dimension (signal value).

Example: SDEVeviation 45.68

6.2.4.24 PTPeak

This is the waveform Peak-to-peak value. Peak-to-peak value of the entire waveform as specified by TRACe. PTPeak is equivalent to the MAXimum minus the MINimum values.

Value type is <0+Numeric>. PTPeak may occur only once and only one value is allowed. Units are of the dependent dimension (signal value).

Example: PTPeak 4.5

6.2.4.25 CYCLe

The CYCLe sub-block specifies parameters dealing with a specified number of complete cycles in the waveform as specified by the COUNt field.

6.2.4.25.1 COUNt

This is the the cycle count representing the number of cycles used for MEAN, RMS, and SDEViation in the CYCLe sub-block. Cycle count evaluates to an integer representing the number of cycles over which the CYCLe parameters apply. Default value for COUNt is 1.

If one complete cycle is not present, and COUNt is specified then the value should be zero. In this case, any related CYCLe parameters are meaningless.

Value type is <0+Numeric>. COUNt may occur only once and only one value is allowed.

Example: COUNt 10

6.2.4.25.2 MEAN

This is the cycle arithmetic mean. This indicates the mean value of the waveform as specified by COUNt.

Value type is <Numeric>. MEAN may occur only once and only one value is allowed. Units are of the dependent dimension (signal value).

Example: MEAN 2.025

6.2.4.25.3 RMS

This is the cycle root mean square. It indicates the root mean square value of the waveform as specified by COUNt.

Value type is <Numeric>. RMS may occur only once and only one value is allowed. Units are of the dependent dimension (signal value).

Example: RMS 12.3

6.2.4.25.4 SDEViation

This is the cycle standard deviation. This indicates the standard deviation of the entire waveform as specified by COUNt.

Value type is <0+Numeric>. SDEV may occur only once and only one value is allowed. Units are of the dependent dimension (signal value).

Example: SDEViation 45.68

6.2.5 MEASurement

This is the measurement of one or more specified data points. This block contains data that does not fit in other sub-blocks of the DATA block. A MEASurement block may occur multiple times, but is not required.

The MEASurement sub-block is defined as:

```
measurement -> MEASurement [= <Label>](
                  [NOTE       <String>]
                  [NAME       <String>]
                  [UNITs      <String>]
                  [TYPE       <String>]
                  [TRACe      <Label>]
                  {LOCation (
                     LABel    <Label>
                     INDex    <+NR1>
                  )}*
                  [VALues     <Numeric> {,<Numeric>}*]
               )
```

The block modifier <Label> value is optional and allows for the unique identification of the MEASurement sub-block. As all <Label> values are unique, no two MEASurement sub-blocks may have the same <Label> within the same DATA block.

The TRACe keyword allows association of the measurement or point with an arbitrary subset of DATA(CURVe) data.

Measurements derived from DATA block data are considered to have been derived from the data after SCALe and OFFSet (specified in the DIMension block) have been applied.

6.2.5.1 NOTE

This is arbitrary text.

Value type is <String>. Text should be human-readable. Only one occurrence of the NOTE keyword is allowed and only one value is allowed.

Example: NOTE “This data is retest of test 14”

6.2.5.2 NAME

This is an arbitrary name of the measurement or point. NAME is typically more descriptive than the MEASurement block modifier <Label> and it does not have to be unique.

Value type is <String>. It is recommended that the string length be 32 characters or less. NAME may occur only once and only one value is allowed.

Example: NAME “Trigger pattern 4”

6.2.5.3 UNITs

This indicates units, such as Volts, Amps, Hz, etc without multipliers.

Value type is <String>. It is recommended that the string length be 32 characters or less. These units may differ from those specified in other DIMension blocks since the values represented herein may be derived or ancillary values.

Recommended units are those specified by *IEEE 488.2* <SUFFIX PROGRAM DATA>, the null string, and UNKNOWN. The null string implies that the data has no dimensional units and UNKNOWN indicates that the units are unknown.

If the TRACe keyword is present, the UNITs value may differ from that of the trace. Only one occurrence of UNITs is allowed and only one value is allowed. If omitted, the units are as specified in the DIMension block.

Example: UNITs A

6.2.5.4 TYPE

This indicates the type of measurement or point.

Value type is <String>. It is recommended that the string length be 32 characters or less. Only one occurrence of TYPE is allowed and only one value is allowed.

The one defined type is:

UNKNOWN Unknown (default)

Example: TYPE “UNKNOWN”

6.2.5.5 TRACe

This is the trace associated with the point(s). TRACe allows a sub-set of the data to be referenced.

Value type is <Label>. Only values defined by the TRACe block modifier <Label> value are allowed. Only one occurrence of TRACe is allowed and only one value is allowed. If this keyword is present, it determines which dimensions shall be included in the LOCation sub-block for MEASurement.

Example: TRACe ADDR

6.2.5.6 LOCation

This specifies the label and index for one independent dimension. This sub-block is optional and may occur multiple times. There shall be a LOCation sub-block with a label value for each independent dimension present, or if a TRACe keyword is specified, then for each independent dimension in the specified trace.

6.2.5.6.1 LABel

This is the label of the independent dimension.

Value type is <Label>. Only values previously defined by a DIMension block modifier <Label> value are allowed and the dimension shall be implicit. LABel is required and may occur only once. Only one value is allowed.

Example: LABel A1

6.2.5.6.2 INDex

This specifies the index for the point.

Value type is <+NR1>. INDex is required and only one occurrence of INDex is allowed.

Example: INDex 274

6.2.5.7 VALues

This indicates the data value(s).

Value type is <Numeric>. VALues is optional and may occur only once. Multiple values are allowed.

Example: VALues 25.3, 93.7

6.2.6 DATA Block Example

```
DATA (
    NOTE “File 1A-23”
    DELTa (
        DIMension=Y(
            SCALe 0.1
            OFFSet 12.3
            SIZE 4096)
        DATE 1988,9,2
        TIME 23,3,0.25)
    CURVe (
        NAME “”
        CTYPe CRC16
        VALue 23.2,14.5,16.02,12.0
        CSUM 45)
    WAVeform (
        NAME “19th Pulse”
        TRACe Z
        HLMethod “MODE”
        AMPLitude 3.902
        PWIDth 2.6E-7
        PDUTycycle 32.4
        RISE (
            TIME 2.8E-8)
        FALL (
            TIME 2.703E-8)
        REFerence (
            HIGH 5.85E-5
            LOW 5.14E-5
            MID 5.32E-5)
        RMS 6.4
        PTPeak 12.5)))
```

6.3 DIMension

Each DIMension block describes one dimension of DATA(CURVe) data. A dimension may be either explicit, in which case values are present in the DATA(CURVe) block, or implicit, in which case the values are not present, but are instead generated by a function. See 1.3 in this section. The order in which dimensions appear determines the data ordering in the DATA block.

The DIMension and ORDer blocks describe the logical structure and organization of data in the DATA(CURVe) block. The ENCode block describes the physical representation or encoding of the data. The ORDer block describes the ordering of the dimensional data elements in the DATA(CURVe) block. Together they provide the sufficient and necessary information for correct extraction of the data.

Whether a dimension is to be considered an independent or dependent dimension is determined by the TRACe block. In the absence of a TRACe block, the following simple rule is applied: if one or more implicit dimensions are present, they are considered to be independent, and all explicit dimensions are considered to be dependent.

A DIMension block is required for each dimension of the data (implicit and explicit). The DIMension block is defined as:

```
dimension -> DIMension = <Label>(
                [NOTE   <String>]
                [NAME   <String>]
                TYPE    {IMPLicit | EXPLicit}
                [SCALe  <Numeric>]
                [OFFSet <Numeric>]
                [SIZE   <+NR1>]
                UNITs   <String>
                [encode]
              )
```


Each dimension is uniquely distinguished by its required block modifier <Label> value. No two DIMension blocks may have the same <Label> value. Dimension labels should be kept short, yet meaningful.

6.3.1 NOTE

This is arbitrary text.

Value type is <String>. Text should be human-readable. Only one occurrence of the NOTE keyword is allowed and only one value is allowed.

Example: NOTE “DS1C at test point 12”

6.3.2 NAME

This is an arbitrary name or description of the dimension. NAME is typically more descriptive than the dimension block label.

Value type is <String>. It is recommended that the string length be 32 characters or less. Only one occurrence of NAME is allowed and only one value is allowed.

Example: NAME “DS1C Voltage/Amplitude”

6.3.3 TYPE

This indicates if the dimension is implicit or explicit.

TYPE takes a specified enumerated set of values. TYPE is required, may occur only once, and takes one of the two following enumerated set values:

IMPLicit	Values for this dimension are not in DATA(CURVe), but are derived from a linear function, y = mx + b.
EXPLicit	Values for this dimension are present in DATA(CURVe).

Example: TYPE EXPL

6.3.4 SCALe

Scaling factor to be applied to the dimension's values in DATA(CURVe(VALues)).

Value type is <Numeric>. Default value is 1.0. Only one occurrence of SCALe is allowed and only one value is allowed. The SCALe value may be overridden by DATA(DELTa(DIMension(SCALe))).

SCALe is always applied to DATA(CURVe) data, but not to any other data, such as DATA(WAVeform).

For implicit dimensions:

X' = (SCALe * *i*) + OFFSet *i* = 1 to SIZE

For explicit dimensions:

X' = (SCALe * X) + OFFSet

where X is a value from DATA(CURVe(VALues)). Also see Section 5.6, ORDer.

Example: SCALe 0.5E-2

6.3.5 OFFSet

Offset factor to be applied to the dimension's values in DATA(CURVe(VALues)).

Value type is <Numeric>. Default value is zero. Only one occurrence of OFFSet is allowed and only one value is allowed. The OFFSet value may be overridden by DATA(DELTa(DIMension(OFFSet))).

OFFSet is always applied to DATA(CURVe) data, but not to any other data, such as DATA(WAVeform).

For implicit dimensions:

X' = (SCALe * *i*) + OFFSet *i* = 1 to SIZE

For explicit dimensions:

X' = (SCALe * X) + OFFSet

X is a value from DATA(CURVe(VALues)).

Example: OFFSet -1.0E-2

6.3.6 SIZE

This is the number of points or values in the dimension.

Value is type <+NR1>. Only one occurrence of SIZE is allowed and only one value is allowed. The SIZE value may be overridden by DATA(DELTa(DIMension(SIZE))).

The following two invariants hold:

1. All explicit dimensions must have the same SIZE value.
2. The product of the SIZE values of the implicit dimensions must equal the SIZE value of any explicit dimension.

If SIZE is not specified for a dimension, it is assumed to be consistent with the above invariants. It is an error if there is insufficient information to determine a dimension's SIZE. For example, SIZE cannot be determined if two or more implicit dimensions do not specify a

SIZE, nor can it be determined if two or more explicit dimensions specify different SIZE values.

It is strongly recommended that SIZE be specified for all dimensions whenever possible.

Example: SIZE 512

6.3.7 UNITs

This indicates units, such as Volts, Amps, Hz, etc. Multipliers are not allowed.

Value type is <String>. It is recommended that the string length be 32 characters or less. Recommended units are those specified by *IEEE 488.2* <SUFFIX PROGRAM DATA>, the null string, and UNKNOWN. The null string implies that the data has no dimensional units and UNKNOWN indicates that the units are unknown or unspecified. UNITs is required and only one occurrence of UNITs is allowed.

Example: UNITs “V”

6.3.8 ENCode

An ENCode sub-block may appear subordinate to a DIMension block. The DIMension(ENCode) block keywords and sub-blocks are the same as the major ENCode block described in 6.4 in this section. They define data resolution, special range values, and encoding format for <Block> value type data.

The scope of a DIMension(ENCode) block includes only the DATA(CURVe) block and the DIMension(ENCode) block itself. The scope of the major ENCode block includes all other values in the DATA block.

Each keyword specified in a DIMension(ENCode) block overrides the corresponding keyword in the major ENCode block,but only for the values in the DATA(CURVe) block for the specific dimension.

Only one ENCode block can be specified subordinate to a DIMension block. It is not required.

6.3.9 DIMension Block Example

```
DIM=X (
        NOTE “Procedure 12.8”
        NAME “DS3”
        TYPE EXPLicit
        UNITs “S”
        SCALe 1.0
        OFFSet 0.0
        SIZE 1024
        ENCode (
                NOTE “This ENCode block affects only X dimension values in the
                  DATA(CURVe) block”
                FORMat INT16
                NVALue -32768
```

ORANge 32767
URANge -32767
HRANge 32766
LRANge -32766
RESolution 1))

6.4 ENCode

The ENCode block addresses data resolution, special range values, and encoding format <Block> value type data.

The scope of an ENCode block is determined by its hierarchical level in the data format structure. If the ENCode block is at the same level as DIMension blocks, it applies globally to all data in the DATA blocks. A global encoding specification can, however, be overridden on a dimension-by-dimension basis with an ENCode block that is subordinate to the particular dimension. A keyword specification at the higher level rules unless explicitly superceded by the same keyword at the subordinate level.

Only one ENCode block can be specified at the same hierarchical level as DIMension blocks, and only one ENCode block can be specified subordinate to a DIMension block.

The ENCode block is not required at any hierarchical level. However, the specification of HRANge and LRANge for all explicit dimensions is strongly recommended. The ENCode block is defined as:

```
encode ->    ENCode (
                [NOTE          <String>]
                [FORMat        { IFP32 | IFP64
                               | INT8 | INT16 | INT32 | INT64
                               | SFP32 | SFP64
                               | SINT16 | SINT32 | SINT64
                               | SUINT16 | SUINT32 | SUINT64
                               | UINT8 | UINT16 | UINT32 | UINT64 }]
                [NVALue        <Numeric>]
                [ORANge        <Numeric>]
                [URANge        <Numeric>]
                [HRANge        <Numeric>]
                [LRANge        <Numeric>]
                [RESolution    <+Numeric>]
             )
```

If no ENCode block is specified, the default values for each keyword prevail. The ENCode block sub-blocks and keywords are:

6.4.1 NOTE

This is arbitrary text. Value type is <String>. Text should be human-readable. Only one occurrence of the NOTE keyword is allowed and only one value is allowed.

Example: NOTE “Not-a-number and plus and minus infinity do not follow the IEEE 488.2 recommendation”

6.4.2 FORMat

This is the data encoding format for CURVe(VALue) data. <Block> value type is introduced by a pound sign followed by one or more decimal digit characters. See section 3.4.5. ASCII representation uses comma separated <Numeric> data.

Specifying a block FORMat does not require that the values in DATA block be formatted as <Block> value type. Values may always be specified as <Numeric>.

If the ENCode block is subordinate to a DIMension block, then FORMat applies only to values in DATA(CURVe(VALues)) for the particular dimension.

FORMat takes a specified enumerated set of values. Only one occurrence of FORMat is allowed and only one of the following enumerated set values is allowed:

ASCii	Comma-separated <Numeric> data
IFP32	IEEE floating point, 32 bits.
IFP64	IEEE floating point, 64 bits.
INT8	IEEE 8-bit signed integer (default).
INT16	IEEE 16-bit signed integer.
INT32	IEEE 32-bit signed integer.
INT64	IEEE 64-bit signed integer.
SFP32	IEEE swapped floating point, 32 bits.
SFP64	IEEE swapped floating point, 64 bits.
SINT16	Swapped IEEE 16-bit signed integer.
SINT32	Swapped IEEE 32-bit signed integer.
SINT64	Swapped IEEE 64-bit signed integer.
SUINT16	Swapped IEEE 16-bit unsigned integer.
SUINT32	Swapped IEEE 32-bit unsigned integer.
SUINT64	Swapped IEEE 64-bit unsigned integer.
UINT8	IEEE 8-bit unsigned integer.
UINT16	IEEE 16-bit unsigned integerUINT16.
UINT32	IEEE 32-bit unsigned integerUINT32.
UINT64	IEEE 64-bit unsigned integerUINT64.

All signed integers are two's complement. IEEE formats are defined by *IEEE 488.2* Binary Integer and Binary Unsigned Integer codes. "Swapped" means that the byte order (from low to high address) is from least significant to most significant byte.

For example, to transfer #H1234 as an INT16 (16-bit integer) across a byte oriented transmission medium, transfer the first byte as 12 (hex) and the second byte as 34 (hex). To transfer #H1234 as an SINT16 (swapped 16-bit integer), transfer the first byte as 34 (hex) and the second byte as 12 (hex).

Similarly, to transfer #H12345678 as an INT32 (32-bit integer), transfer the bytes in the order: 12 (hex), 34 (hex), 56 (hex), and 78 (hex). To transfer #H12345678 as an SINT32

(32-bit swapped integer), transfer the bytes in the order: 78 (hex), 56 (hex), 34 (hex), and 12 (hex).

Example: FORMat INT16

6.4.3 NVALue

This is the numeric value (before application of SCALe and OFFSet) that is to be treated as implying "not a number" or "no value recorded" for values in the DATA block.

Value type is <Numeric>. Only one occurrence of NVALue is allowed and only one value is allowed. If NVALue is not present, no not-a-number value is defined for integer formats. For IEEE floating point formats, no NVALue is ever specified. In this case, NVALue is as defined by IEEE 754 for not-a-number. For ASCII, the default value is 9.91E+37.

Example: NVALue 9999

6.4.4 ORANge

This indicates the numeric value (before application of SCALe and OFFSet) that is to be treated as over range for values in the DATA block.

Value type is <Numeric>. Only one occurrence of ORANge is allowed and only one value is allowed. If ORANge is not present, no over range value is defined for integer formats. For IEEE floating point formats, no ORANge is ever specified. In this case, ORANGe is as defined by IEEE 754 for plus infinity. For ASCII, the default value is 9.9E+37.

Example: ORANge #H7FFF

6.4.5 URANge

This is the numeric value (before application of SCALe and OFFSet) that is to be treated as under range for values in the DATA block.

Value type is <Numeric>. Only one occurrence of URANge is allowed and only one value is allowed. If URANge is not present, no under range value is defined for integer formats. For IEEE floating point formats, no URANge is ever specified. In this case, URANGe is as defined by IEEE 754 for negative infinity. For ASCII, the default value is -9.9E+37.

Example: URANge -32767

6.4.6 HRANge

This is the greatest signed value (before application of SCALe and OFFSet) that may be present in the DATA(CURVe(VALUE)).

Value type is <Numeric>. Only one occurrence of HRANge is allowed and only one value is allowed. The value specified shall be at least as great as the greatest value present in the data but may be greater.

NVALue, ORANge, and URANge values are not considered when determining range. Units are the measurement curve units after SCALe and OFFSet are applied to the data.

Example: HRANge 32766

6.4.7 LRANge

This is the least value (before application of SCALe and OFFSet) that may be present in DATA(CURVe(VALUE)).

Value type is <Numeric>. Only one occurrence of LRANge is allowed and only one value is allowed. The value specified shall be at least as small as the least value present in the data but may be less.

NVALue, ORANge, and URANge values are not considered when determining range. Units are the measurement curve units after SCALe and OFFSet are applied to the data.

Example: LRANge -32766

6.4.8 RESolution

This is the minimum resolution of data values present in the DATA block.

Value type is <+Numeric>. Only one occurrence of RESolution is allowed and only one value is allowed.

Units are the measurement curve units after SCALe and OFFSet are applied to the data.

Example: RESolution 1

6.4.9 ENCode Block Example

```
ENCode (
        NOTE “Acquisition resolution was 10 bits”
        FORMat INT16
        NVALue -32768
        ORANge 32767
        URANge -32767
        HRANge 32766
        LRANge -32766
        RESolution 1)
```

6.5 IDENtify

This is the IDENtify block contains fields uniquely identifying the data and describing the environment in which it was created or acquired. The precise definitions for the contents of all text string values in the IDENtify block is left to the creator of the data set. However, the general category shall be followed.

The IDENtify block may occur only once, but is not required. The IDENtify block is defined as:

```
identify ->  IDENtify (
                [NOTE          <String>]
                [NAME          <String>]
                [TECHnician    <String> {,<String>}*]
                [PROJect       <String>]
                [DATE          <NR1>,<+NR1>,<+NR1>
                               [,<NR1>,<+NR1>,<+NR1>]]
                [TIME          <0+NR1>,<0+NR1>,<0+Numeric>
                               [,<0+NR1>,<0+NR1>,<0+Numeric>]]
                [UUT(
                   [NAME       <String>]
                   [ID         <String> {,<String>}*]
                   [DESign     <String> {,<String>}*]
                )]
                [TEST(
                   [NAME       <String>]
                   [SERies     <String>]
                   [NUMBer     <String> {,<String>}*]
                )]
                [HISTory       <String> {,<String>}*]
             )
```


6.5.1 NOTE

This is arbitrary text.

Value type is <String>. Text should be human-readable. Only one occurrence of the NOTE keyword is allowed and only one value is allowed.

Example: NOTE “24 AWG may have been out of calibration”

6.5.2 NAME

This is the user-defined name identifying the data set. NAME is typically more descriptive than the block modifier.

Value type is <String>. It is recommended that the string length be 32 characters or less. This field is optional. Only one occurrence of the NAME keyword is allowed. Only one value is allowed.

Example: NAME “Spectral Power Test”

6.5.3 TECHnician

This identifies the technician(s) involved in the generation or modification of the data.

Value type is <String>. It is recommended that the string length be 32 characters or less. Only one occurrence of the TECHnician keyword is allowed. Multiple values, up to 7, are allowed.

Example: TECHnician “John H. Doe”

6.5.4 PROJect

This is the name of project, job, task, etc.

Value type is <String>. It is recommended that the string length be 32 characters or less. Only one occurrence of PROJect is allowed and only one value is allowed.

Example: PROJect “S27A408”

6.5.5 DATE

This is the date of data creation or acquisition.

This keyword takes a set of three comma separated value types to specify the date corresponding to year, month, and day.

The years parameter is type <NR1>. It always includes century and millennium information.

The months parameter is type <+NR1>. Its range is 0 to 12.

The days parameter is type <+NR1>. Its range is 0 to 31 (depending on the month).

If two sets of values are specified, they define an inclusive range of dates.

Example: DATE 1776,07,04 indicates the 4th of July, 1776

6.5.6 TIME

This is the time of data creation or acquisition.

This keyword takes a set of three comma separated value types to specify the time corresponding to hour, minute, and second.

The hours parameter is type <0+NR1>. Its range is 0 to 23. This parameter is always expressed in 24 hour format.

The minutes parameter is type <0+NR1>. Its range is 0 to 59.

The seconds parameter is type <0+Numeric>. Its range is 0 to 60 for instruments receiving data and 0 to 59.999... for instruments sending data.

If two sets of values are specified, they define an inclusive range of time.

Example: TIME 23,59,59 indicates one second before midnight.

6.5.7 UUT

This is a description of the “unit under test”. This block may occur only once.

6.5.7.1 NAME

This is the name or type of unit under test. NAME is similar in use to ID, except that it is typically more descriptive and only one value is allowed.

Value type is <String>. It is recommended that the string length be 32 characters or less. Only one occurrence of the NAME keyword is allowed and only one value is allowed.

Example: NAME “Model 29AP402”

6.5.7.2 ID

This is the identification of the unit under test.

Value type is <String>. It is recommended that the string length be 32 characters or less. Only one occurrence of the ID keyword is allowed. Multiple values, up to 7, per keyword are allowed.

Example: ID “AA32303”,“B7”

6.5.7.3 DESign

This identifies the design specification.

Value type is <String>. It is recommended that the string length be 32 characters or less. Only one occurrence of the DESign keyword is allowed. Multiple values, up to 7, are allowed.

Example: DESign “2W32 Rev 3C”

6.5.8 TEST

This is the test description. This block may occur only once.

6.5.8.1 NAME

This is the test series name.

Value type is <String>. It is recommended that the string length be 32 characters or less. Only one occurrence of the NAME keyword is allowed and only one value is allowed.

Example: NAME “DC Parametrics”

6.5.8.2 SERies

This is the number or id of test series.

Value type is <String>. It is recommended that the string length be 32 characters or less. Only one occurrence of SERies is allowed and only one value is allowed.

Example: SERies “24B208”

6.5.8.3 NUMBer

This is the number or id of particular test.

Value type is <String>. It is recommended that the string length be 32 characters or less. Only one occurrence of the NUMBer keyword is allowed. Multiple values, up to 7, are allowed.

Example: NUMBer “A3”,“B6”,“B7”

6.5.9 HISTory

This is the derivation history of the data set, such as names of tests, instruments, instrument systems, and software packages.

Value type is <String>. It is recommended that the string length be 32 characters or less. Only one occurrence of the HISTory keyword is allowed. Multiple values, up to 7, are allowed and imply an ordering from oldest to most recent.

Example: HISTory “T24.WFM”, “CCITT DS3 Template”

6.5.10 IDENtify Block Example

```
IDENtify (
    NOTE “This IDENtify block example contains examples of all keywords”
    NAME “”
    TECHnician “Matt Wilson”
    PROJect “Acorn”
    DATE 1988,9,2
    TIME 23,3,0.25
    UUT (
        NAME “Ironman”
        ID “007”
        DESign “Rev D”)
    TEST (
        NAME “Cal Menu”
        SERies “3A”
        NUMBer “5”)
    HISTory “Tinman”, “Strawman”)
```


6.6 ORDer

The ORDer block defines the physical layout of data within the DATA(CURVe) block. Data may be ordered as n-tuples, in which the data for each explicit dimension is “inter-mixed”. This is also called row order. Data may also be ordered by dimension. When ordered by dimension, all data for each dimension is grouped consecutively, with all data for one dimension followed by all data for the next dimension. This is also called column order.

When more than one implicit dimension exists, the first implicit dimension index always increments the slowest and the last implicit dimension index always increments the fastest.

The ORDer block may occur only once, but is not required. If omitted, the default values for its keywords prevail. The ORDer block is defined as:

```
order ->    ORDer (
                [NOTE   <String>]
                [BY     {TUPLe | DIMension}]
            )
```

6.6.1 NOTE

This is arbitrary text.

Value type is <String>. Text should be human-readable. Only one occurrence of the NOTE keyword is allowed and only one value is allowed.

Example: NOTE “Most important first”

6.6.2 BY

This indicates if the ordering is by tuple or dimension.

BY takes a specified enumerated set of values. BY may occur only once, and takes one of the two following enumerated set values:

TUPLe	Data values are organized by n-tuple (default).
DIMension	Data values are organized by dimension.

6.6.3 Examples

Suppose the sample measurements in the following example were made according to the scenario of 1.3 of this section. The data has five dimensions. Assume also that the five DIMension blocks have label values corresponding to the headings in as shown below.

Sample Measurements

X	Y	Z	Temp	Hum
5	1	8.1	18.1	61
5	2	9.2	20.2	62
7	1	6.3	16.3	63
7	2	3.4	16.4	64
9	1	8.5	18.5	65
9	2	3.6	16.6	66

Example 1:
If all values are present in the data, the five DIMension blocks will have TYPE EXPLicit (see 6.3 in this section). If no ORDer block is present, the data will be ordered:

HUM, TEMP, X, Y, Z, HUM, TEMP, X, Y, Z ...

That is, the data is ordered by 5-tuple and by column order of occurrence within a 5-tuple. Note that the rows may appear in any order since there are no implicit dimensions.

The DIMension and DATA blocks for this example are:

```
DIMension=HUM (
        TYPE EXPLicit
        SIZE 6
        UNITs “PCT”
        ENCode (
                HRANge 60
                LRANge 66))
DIMension=TEMP (
        TYPE EXPLicit
        SIZE 6
        UNITs “CEL”
        ENCode (
                HRANge 25
                LRANge 15))
DIMension=X (
        TYPE EXPLicit
        SIZE 6
```

```
        UNITs “M”
        ENCode (
        HRANge 9
        LRANge 5))
    DIMension=Y (
        TYPE EXPLicit
        SIZE 6
        UNITs “M”
        ENCode (
            HRANge 2
            LRANge 1))
    DIMension=Z (
        TYPE EXPLicit
        SIZE 6
        UNITs “M”
        ENCode (
            HRANge 10
            LRANge 0))
    DATA (
        CURVe (
            VALues 61, 18.1, 5, 1, 8.1,
                       64, 16.4, 7, 2, 3.4,
                       65, 18.5, 9, 1, 8.5,
                       66, 16.6, 9, 2, 3.6,
                       62, 20.2, 5, 2, 9.2,
                       63, 16.3, 7, 1, 6.3))
```

Example 2:

If the coordinate values for the X and Y dimensions are omitted from the data, these dimensions are implicit (TYPE is IMPLicit).

The X and Y dimensions are easily derived by linear functions:

$$x = 2m + 3$$
$$y = n$$

where m and n are indices related to the ordering of the values for Z, TEMP, and HUM.

The DIMension and DATA blocks for this example are:

```
    DIMension=TEMP (
        TYPE EXPLicit
        SIZE 6
        UNITs “CEL”
        ENCode (
            HRANge 25
            LRANge 15))
        DIMension=X(
            TYPE IMPLicit
            SIZE 3
```

```
        UNITs “M”
        SCALe  2
        OFFSet 3)
    DIMension=Y(
        TYPE IMPLicit
        SIZE 2
        UNITs “M”)
    DIMension=Z (
        TYPE EXPLicit
        SIZE 6
        UNITs “M”
        ENCode (
            HRANge 10
            LRANge 0))
    DIMension=HUM (
        TYPE EXPLicit
        SIZE 6
        UNITs “PCT”
        ENCode (
            HRANge 60
            LRANge 66))
    DATA (
        CURVe (
            VALues 18.1, 8.1, 61,
                   20.2, 9.2, 62,
                   16.3, 6.3, 63,
                   16.4, 3.4, 64,
                   18.5, 8.5, 65,
                   16.6, 3.6, 66))
```

Note that the humidity DIMension now appears last and that DATA values in the DATA block are therefore in a different order. The reordering of the X and Y DIMension blocks affects only the default determination of the independent and dependent dimensions (see chapter 6.3). In this second example, the presence of implicit dimensions forces a required row order on the VALUes values.

6.7 REMark

This block is for information not fitting elsewhere.

The REMark block may occur only once, but is not required. The REMark block is defined as:

```
remark ->    REMark(
                [NOTE   <String> {,<String>}*]
             )
```

6.7.1 NOTE

This is arbitrary text.

Value type is <String>. Text should be human-readable. Only one occurrence of the NOTE keyword is allowed. Multiple values are allowed and no ordering is implied.

Example: NOTE “The REMark block is the appropriate place for observations and annotations about the data, its acquisition, creation, or modification. In general, its a place to stuff technical information.”

6.7.2 REMark Block Example

REMark (NOTE “A remarkable data structure”)

6.8 DIF

The DIF block is the first block in a dif data set. It uniquely defines the expression data as being a dif data set with the beginning DIF character sequence. The DIF block provides version information to assist the parser in determining compatibility. It further defines the scope of the data set.

The DIF block may occur only once and is required. The DIF block is defined as:

```
difid ->     DIF(
                [NOTE          <String>]
                VERSion        <+Numeric>
                [SCOPe         PREamble | FULL]
             )
```

6.8.1 NOTE

This is arbitrary text.

Value type is <String>. Text should be human-readable. Only one occurrence of the NOTE keyword is allowed and only one value is allowed.

Example: NOTE “First Version”

6.8.2 VERSion

The DIF(VERSion) number exactly tracks the SCPI SYSTem:VERSion number (Command Reference, 21.20) and correlates changes and extensions in the DIF volume with the rest of SCPI.

Note: The DIF(VERSion) keyword was new in 1994.0 and was created as a part of a one-time major restructuring and simplification of DIF. DIF parsers designed prior to 1994 will not be compatible with 1994.0 and following versions.

Value type is <+Numeric>. Only one value is allowed. Only one occurrence of VERSion is allowed.

Note that while version numbers increase in value in time sequence, lesser version numbers will not necessarily be guaranteed to be fully compatible.

Example: VERSion 1993.0

6.8.3 SCOPe

This indicates the scope of the DIF block.

SCOPe takes a specified enumerated set of values. SCOPe may occur only once and takes one of the two following enumerated set values:

When SCOPe is set to PREamble, the DIF block omits the following data:

- dif blocks of waveform, measurement, and delta.
- dif curve block keywords VALues and CSUM and their parameters.

The various notes and identification parameters may be sent if absolutely required.

When SCOPe is set to FULL, a complete dif data set as described in Section 4.3 is present (default).

Example: SCOPe PREamble

6.8.4 DIF Block Example

```
DIF (
        NOTE "Third revision of standard"
        VERSion 1993.0)
```

6.9 TRACe

TRACe blocks describe logical relationships among all or some of the dimensions. They may define functions, curves, surfaces, or arbitrary sets or data groupings. They also provide a convenient means of describing subsets of the data. The windowing effect of the TRACe blocks may, for example, be used to make two-dimensional "cuts" through three-dimensional data. Other blocks, such as the VIEW block, build on TRACe blocks. TRACe block information is built from DIMension blocks; VIEW blocks are built from TRACe block information.

If no TRACe block is present, the independence or dependence of a dimension is determined by rules described in 6.3 in this section.

The TRACe block is optional and multiple TRACe blocks are allowed. The TRACe block is defined as:

```
trace ->     TRACe = <Label>(
                [NOTE          <String>]
                [NAME          <String>]
                [SYMMetry      <String>]
                {INDependent(
                   LABel       <Label>
                   [STARt      <+NR1>]
                   [STOP       <+NR1>]
                )}*
                {DEPendent(
```

```
      LABel       <Label>
   )}*
)
```

The TRACe block modifier <Label> value is required. It provides for the unique identification of the TRACe block. As <Label> for the entire data set are unique, no two TRACe blocks may have the same <Label>.

Explicit and implicit, as applied to dimensions, are distinguished from independent and dependent as follows. Implicit and explicit are concerned with how the data is physically represented and structured. Independent and dependent relate to how a dimension is interpreted.

6.9.1 NOTE

This is arbitrary text.

Value type is <String>. Text should be human-readable. Only one occurrence of the NOTE keyword is allowed and only one value is allowed.

Example: NOTE “Values generated pursuant to 12A34D”

6.9.2 NAME

This is an arbitrary name or description of the trace.

Value type is <String>. It is recommended that the string length be 32 characters or less. Only one occurrence of NAME is allowed and only one value is allowed.

Example: NAME “Gain”

6.9.3 SYMMetry

This specifies the type of symmetry on a per dimension basis. Only one occurrence of SYMMetry is allowed.

Value type is <String>. It is recommended that the string length be 32 characters or less. The following values are predefined:

UNKNOWN	Symmetry is unknown. (default)
NONE	No symmetry.
PEVEN	Positive and even symmetry.
NEVEN	Negative and even symmetry.
PODD	Positive and odd symmetry.
NODD	Negative and odd symmetry.

Example: SYMMetry “UNKNOWN”

6.9.4 INDependent

This identifies one independent dimension that constitutes the trace. If independence and dependence are not meaningful, the trace should be defined using the INDependent block. This sub-block may occur multiple times.

STARt and STOP parameters represent integer values referring to indexes on implicit dimensions and therefore apply only to implicit dimensions

6.9.4.1 LABel

This is the label of the independent dimension.

Value is of type <Label>. Only values defined by the DIMension block label are allowed. LABel is required and may occur only once. Only one value is allowed.

Example: LABel X

6.9.4.2 STARt

This specifies the starting index (inclusive).

Value is of type <+NR1>. Only one occurrence of STARt is allowed and only one value is allowed. This keyword may be present only if the dimension is implicit.

If STARt is omitted, then the default value is 1. If the index value is greater than the dimension's SIZE value (from the applicable DIMension or DATA(DELTa(DIMension)) block), then an error is reported.

Example: STARt 12

6.9.4.3 STOP

This specifies the ending index (inclusive).

Value is of type <+NR1>. Only one occurrence of STOP is allowed and only one value is allowed. This keyword may be present only if the dimension is implicit.

If STOP is omitted then SIZE is used. If the index value is greater than the dimension's SIZE value (from the applicable DIMension or DATA(DELTa(DIMension)) block), then an error is reported.

If the stopping index value is less than the starting index value then an error is reported.

Example: STOP 982

6.9.5 DEPendent

This identifies a dependent dimension that constitutes the trace. This sub-block may occur multiple times.

6.9.5.1 LABel

This is the label of the dependent dimension.

Value is of type <Label>. Only values defined by the DIMension block label are allowed. LABel is required and may occur only once. Only one value is allowed.

Example: LABel Y

6.9.6 TRACe Block Example

```
TRACe=TEMP (
        NOTE “Missing values were interpolated”
        NAME “Transient acceleration”
        SYMMetry “UNKNOWN”
        INDependent (
                LABel X
                STARt 1
                STOP 10)
        INDependent (
                LAB Y
                STARt 100
                STOP 200)
        INDependent (
                LABel Z
                STARt 2
                STOP 4)
        DEPendent (
                LABel A))
```


6.10 VIEW

VIEW blocks provide semantic information about the data in DATA(CURVe). With a VIEW block, one can show relationships between and among subsets of the data as defined by TRACe blocks. For example, a complex waveform or surface can be easily defined, as can a waveform and its upper and lower envelopes.

The VIEW block builds on traces described in a TRACe block. A trace commonly defines a simple function, but the data format allows for the definition of sets, surfaces and other complex functions.

For example, a time varying envelope might be described by three DIMension blocks. The two explicit dimensions define the two Y-axis values for the upper and lower components of the envelope. The implicit dimension defines the common X-axis. While the organizational structure of the data is defined by the DIMension and ORDer blocks, its meaning is determined by a VIEW block. It is the view block that identifies which dimension is the upper envelope and which is the lower.

Alternatively, the same data could be viewed as a complex waveform. The DIMension blocks would be the same, but a VIEW block would identify which dimension is the real component and which is the complex component.

The VIEW block is optional and multiple VIEW blocks are allowed. The VIEW block is defined as:

```
view ->      VIEW = <Label> (
                [NOTE          <String>]
                [NAME          <String>]
                {ENVelope(
                   UPPer       <Label>
                   LOWer       <Label>
                   [FUNCtion   <Label>]
                )}*
                {RCOMplex(
                   REAL        <Label>
                   IMAGinary   <Label>
                )}
                {PCOMplex(
                   MAGNitude   <Label>
                   PHASe       <Label>
                )}
             )
```


The modifier label is required. It provides for the unique identification of the VIEW block. As all <Label> elements are unique, no two VIEW blocks may have the same label.

A VIEW block takes the following sub-blocks. Except for NAME and NOTE, only one type of the other sub-blocks may be present:

6.10.1 NOTE

This is arbitrary text.

Value type is <String>. Text should be human-readable. Only one occurrence of the NOTE keyword is allowed and only one value is allowed.

Example: NOTE “Template defines the max and min values for calibrating mil spec 97654-D”

6.10.2 NAME

This is the name of the view. NAME is typically more descriptive than the VIEW block label and does not need to be unique.

Value type is <String>. It is recommended that the string length be 32 characters or less. Only one occurrence of NAME is allowed and only one value is allowed.

Example: NAME “97654-D”

6.10.3 ENVelope

This identifies which two traces constitute the envelope for a third in DATA(CURVe) data. Multiple occurrences of ENVelope are allowed.

6.10.3.1 UPPer

This specifies the name of the upper envelope trace.

Value type is <Label> and is restricted to a value defined by a TRACe block modifier <Label> value. Only one occurrence of UPPer is allowed and only one value is allowed.

Example: UPPer YH

6.10.3.2 LOWer

This specifies the name of the lower envelope trace.

Value type is <Label> and is restricted to a value defined by a TRACe block modifier <Label> value. Only one occurrence of LOWer is allowed and only one value is allowed.

Example: LOWer YL

6.10.3.3 FUNCtion

This specifies the set of data being enveloped.

Value type is <Label> and is restricted to a value defined by a TRACe block <Label>. Only one occurrence of FUNCtion is allowed and only one value is allowed.

Example: FUNCtion Y

6.10.4 RCOMPlex

This identifies which two TRACes constitute a rectangular complex waveform. RCOMPlex may only occur once.

6.10.4.1 REAL

This specifies the name of the real trace.

Value is of type <Label>. Only values defined by the TRACe block are allowed. REAL is required and may occur only once. Only one value is allowed.

Example: REAL S11REAL

6.10.4.2 IMAGinary

This specifies the name of the imaginary trace.

Value is of type <Label>. Only values defined by the TRACe block are allowed. IMAGinary is required and may occur only once. Only one value is allowed.

Example: IMAGinary S11IMAG

6.10.5 PCOMplex

This identifies the which two TRACes constitute a polar complex waveform. PCOMplex may only occur once.

6.10.5.1 MAGNitude

This specifies the name of the magnitude trace.

Value is of type <Label>. Only values defined by the TRACe block are allowed. MAGNitude is required and may occur only once. Only one value is allowed.

Example: REAL S11MAG

6.10.5.2 PHASe

This specifies the name of the phase trace.

Value is of type <Label>. Only values defined by the TRACe block are allowed. PHASe is required and may occur only once. Only one value is allowed.

Example: PHASe S11PHASE

6.10.6 VIEW BLOCK EXAMPLE

```
VIEW=T3_J4 (
        NOTE “Mil spec 97654-D replaces 54679-D”
        NAME “Mil spec 97654-D”
        ENV (
                UPPer YH
                LOWer YL))
```

7 Data Format Example

The following data set exemplifies the use of several related blocks in the data interchange format. The example below is shown encapsulated in parentheses; that is, as a <dif_expression>.

```
( DIF (
        VERsion 1993.0)
IDENtify (
        DATE 1993,4,23
        TIME 16,4,14.23)
ENCode (
        FORMat INT8
        HRANge 127
        LRANge -128)
DIMension=YH (
        TYPE EXPLicit
        SCALe 2.000000E-002
        OFFSet -3.500000E-001
        SIZE 512
        UNITs “V”)
DIMension=YL (
        TYPE EXPLicit
        SCALe 2.000000E-002
        OFFSet -3.500000E-001
        SIZE 512
        UNITs “V”)
DIMension=X (
        TYPE IMPLicit
        SCALe 2.000000E-005
        OFFSet -1.024000E-002
        SIZE 512
        UNITs “s”)
TRACe=H (
        INDependent(
                LABel X)
        DEPendent(
                LABel YH))
TRACe=L (
        INDependent(
                LABel X)
        DEPendent (
                LABel YL))
VIEW=ENV1 (
        ENVelope (
                UPPer H
                LOWer L))
```

```
DATA (
        CURVe (
                VALue #41024{1024 8-bit significant bytes of data goes here}
                CSUM 139))
        WAVeform (
                TRACe H
                RISE (
                        TIME 1.04E-003)
                FALL (
                        TIME 0.86E-003)))
```

In this example, we have a data set that follows the 1993.0 version of SCPI and is dated at about 4 PM on April 23rd, 1993. The data is encoded as 8-bit signed integers ranging from -128 to 127 and is presented as a sequence of 512 byte pairs representing two voltage signal levels (YH and YL). The raw bit values must be multiplied by .02 with .35 subtracted to yield the measured value in Volts. The first data pair occurred at a measured time (X) of (20 - 10,240 microseconds) or -10.22 milliseconds. Subsequent data pairs were recorded at 20 microsecond intervals.

Two traces (H and L) are specified with YH and YL voltage levels as the dependent variables and time as the independent variable. The traces can be viewed as envelope-type data and referred to as ENV1. The data itself is formatted in *IEEE 488.2* block format (1024 bytes long) and its integrity can be verified by computing a 16-bit cyclic redundancy check and comparing the results to the stated (CSUM) value of 139.

Finally, the rise and fall times of the H trace have been computed and recorded in the data set as 1.04 milliseconds and 0.86 milliseconds.

Index

K

L

M

N

O

P

R

S