Modular Translation System

D*ocument* **C***oding* **L***anguage*

SPECIFICATION Version 2.0 December 21, 2008

Documentation and design by Rhea Barron, Zymurgy, New Orleans, LA
Omni Systems, Inc. • Email: info@omsys.com • Web: http://www.omsys.com

Contents

Introduction

Document Coding Language (DCL) provides programmers with a modular and extensible method for describing compound documents. Omni Systems' 25 years of practical experience building tools for filtering complex documents has led to the creation of a flexible document format that is both efficient and easy to understand.

DCL provides a common document format for programs that need to work with files from many sources. It permits such programs to operate on only one file format, DCL, regardless of the original source document format or the desired target format. The authors of DCL-aware programs can concentrate on their own products' features.

DCL is also a convenient intermediate language for programs that convert documents from one word-processor or desktop-publisher program format to another. When it is bundled with a WP or DTP application, it provides a standard interface for document exchange with other DCL-ready products. Eventually, this will permit application vendors to scale back their filter-construction work. Instead of trying to keep up with other vendors' constant format changes, the vendors who use DCL can focus on thorough support of one stable and extendable format.

Types of DCL Programs

Three types of programs use DCL for representing document form and content:

1. Source readers convert from the native source format to DCL.
2. Code processors perform operations on DCL format files.
3. Target writers convert from DCL to the native target format.

The **source readers** generate a sequence of DCL control records (CTLs) that describe the document, including its text, layout, graphics, spreadsheet data, and audio and video annotations. The source reader may write CTLs for some data (such as format definitions and images) to separate files that are incorporated by reference, in which case the source reader puts the filenames for such content in CTLs in the document's DCL file. For example, Omni Systems' **drmif** reads FrameMaker MIF files and writes DCL files.

The **code processors** modify and rewrite existing DCL files. For example, a code processor could remap font names and special characters based on user directions. Or it could rasterize any vector drawings. It could check the text for spelling and grammatical errors using only text CTLs, without any interference from formatting data. Since the DCL representation is the same for all source types, the code processor operates without any need to consider the source, although information identifying the source format is present in the DCL code in case it is wanted. An example of a code processor is Omni Systems' **dcx**, which converts DCL's native binary files to and from their ASCII form so that they can be checked and edited by humans.

The **target writers** obtain the information they need to produce the output document by reading the CTLs in the DCL file generated by the source readers. Since the information describing different features in the document is modular and clearly packaged by the CTLs, the target writer can safely skip over any data that it does not understand. This permits extensions to DCL to be added at any time without any risk of breaking the existing target writers. Omni Systems' **dwrtf** reads a DCL file that has been created by one of the source readers, and writes it in RTF form.

Form and Content

DCL files describe a document's layout as a series of pages containing text and graphics frames. The document's content is described as one or more text streams, punctuated by property changes and optional anchored frames. The text streams are segmented and linked to the text frames they flow through, allowing pages to be processed and accessed independently. DCL files are easily used by viewers.

DCL uses named formats to describe repeating text properties and content. Its master frames and pages capture all details of repeating layouts. The model is both simple and comprehensive.

Characteristics of DCL Files

- DCL files are normally written in **binary** form, for maximum speed and minimum size. For convenience, a **human-readable** and editable form is available (typically, but not necessarily, ASCII). Editable DCL files employ **user-editable mnemonics**, in any language desired.
- DCL files are **portable**. For greatest processing speed, DCL binary files are written in the native byte order of the machine on which they are produced, but each DCL file begins with a record that specifies its format in a universally-readable manner. *A DCL file produced on any platform may be read on any other platform.*
- Text types supported include **ISO international character sets** as well as **Unicode**, permitting the unrestricted use of DCL throughout the world.
- Numeric values are given in the native unit set of the source document to avoid unnecessary rounding errors. **Four scalable unit sets** are supported: inches (nominally in ten-thousandths), metric (in microns), points (in hundredths), and Didot (in hundredths). Different unit sets can be used in the same file for different categories of values.

File Format of DCL Files

A DCL file begins with a file identifier which establishes whether the file is binary or ASCII; if binary, the identifier also indicates the internal byte order. Next are one or more control records (CTLs) that specify the file history, beginning with one describing the source application from which the file was written.

The next record is a **group** CTL, which encloses CTLs specifying document-wide defaults, such as the character set in use. This defaults group is followed by the resources group, which includes definitions—such as format definitions—that are used later in the file.

The rest of the file consists of page groups. DCL uses "master pages" to specify page layouts; they also contain headers and footers. The following "body pages" refer to the master pages for repeating page-based text and graphics. Other non-repeating shared text and graphics are defined on "reference pages" as needed.

Each page group consists of CTLs for page attributes (such as page type and hand-edness), followed by groups for each frame (text or graphic) on the page. Frame CTL groups can be nested to any level. Graphics appear inside their frames, or directly on the page if not in any frame. Text appears in groups (text segments) after all the frames on the page; each segment contains the text for one frame,

with CTLs that link it to its frame as well as to any other text segments that logically precede and follow it in the same text stream.

Table 1 diagrams the typical major portions of a DCL file: the file identifier, the file history, the document defaults group, the document resources group, and page groups.

A DCL file typically has many **page groups**—one for each body page, each master page, and each reference page in the document. A page group is frequently quite complex, comprising several page attribute CTLs and several nested **group** CTLs. Each text frame, graphics frame, text stream, and independent graphic on the page has its own group, nested within the page group. Groups can be nested to any level, but cannot overlap. The page group in Table 1 is given in a skeletal form. Table 2 gives more detail about the contents of a typical page group, showing how frames and text are associated with each other on the page.

Table 1: *Typical DCL file organization.*

Functionality	Binary form	ASCII form
DCL identifier byte-order palindrome (SPARC shown)	0xA5C3C3A5 0x000000000	(DCL 100)
file history CTLs	11 1 21 6 timestamp	{dcl source Frame : *time*)
group CTL: begin document-wide defaults, such as the character set in use	15 1 40 1	((dcl set defaults)
CTLs describing the document defaults	– – – –	(. . .)
endgroup CTL: end the document defaults group	14 1 40 1	)
group CTL: begin document resources group	15 1 40 2	((dcl set resources)
CTLs describing document resources	– – – –	(. . .)
endgroup CTL: end the document resources group	14 1 40 2	)
group CTL: begin page group 1	15 3 10 1	((layout pg_def 1)#page_ID is 1
CTLs for page attributes, text and graphics frames, and text streams; a page group typically contains nested groups, as shown in Table 2	– – – –	(. . .)
endgroup CTL: end page group 1	14 3 10 1	)

Notes on Table 1:

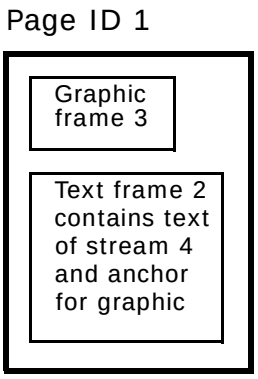
1. The binary DCL identifier is a single eight-byte record, as are all the CTLs.
2. The four numbers for CTLs in the "Binary" column are the **datatype**, **majortype**, **minortype**, and **shortval**, respectively. For **group** CTLs, the **longval**, which is the fifth element of a DCL CTL, is a group ID, and is omitted from this table. See "Record Formats" beginning on page 6 for a full explanation of CTL formats.
3. The binary and ASCII values given in roman type are all actual values, not examples, with the exception of "6" and "Frame" in the file history CTL, and the page_ID in the page group CTL. Type in *italics* is to be replaced by an actual value.
4. In ASCII DCL, the beginning of a group is signalled by an additional open parenthesis at the beginning of the **group** CTL, and the end of a group by a matching closing parenthesis. The pound sign, "#", makes the rest of the line a comment.

Table 2: *Generic page group in a DCL file.*

Functionality	Binary form	ASCII form
group CTL: begin page group 1	15 3 10 1	((layout pg_def 1) #page_ID is 1
group CTL: begin text frame group	15 3 20 2	((layout fr_def 2) #frame_ID is 2
frame text CTL: identify text stream that goes in this frame	2 3 40 4	(layout text 4) #frame contains # text with text_ID 4
end group CTL: end text frame group	14 3 20 2	)
group CTL: begin graphic frame group	15 3 20 3	((layout fr_def 3) #frame_ID is 3
end group CTL: end graphic frame group	14 3 20 3	)
group CTL: begin text segment group	15 2 1 4	((text id 4) #text stream ID is 4
text frame CTL: identify frame containing this text	2 2 2 2	(text frame 2) #text is in frame 2
text anchor CTL: anchor a graphics frame	2 2 5 3	(text anchor 3) #graphics frame 3 is # anchored in this text
end group CTL: end text segment group	14 2 1 4	)
end group CTL: end page group 1	14 3 10 1	)

Notes on Table 2:

1. The page represented by this table has one column of text, and one graphic that is anchored to the text:
2. Each group also contains numerous CTLs that give the properties associated with its page element, such as page size and usage for the page group, size and position for the two frames, and paragraph properties and content for the text segment.
3. The page ID is not the actual page number. The first few page IDs are typically used for reference and master pages.



Record Formats

A DCL file consists of a series of control records (CTLs). Each binary DCL CTL record contains eight bytes, consisting of a **datatype**, a **majortype**, a **minortype**, a **shortval**, and a **longval**. Each ASCII DCL CTL has the equivalent information in a similar form.

The **datatype** is the most significant nibble of the first byte of a binary DCL CTL record. It may mark the record as starting or ending a group of records, or as having an internal or external data area. In the latter case the datatype also specifies the format of the data. The datatype has no explicit representation in the ASCII form of DCL.

The **majortype** values defined in standard DCL are:

1 = DCL internal control
2 = text property
3 = layout property
4 = graphic property
5 = spreadsheet property
6 = audio property
7 = video property

The meaning of the **minortype** depends on the majortype; the minortypes for each majortype are detailed in this document.

The meaning of the shortval and longval fields depends on both the major and minor types.

The **shortval** typically contains enumerated values (providing more subtypes), indices (such as format number or text stream number), or counts of items (table rows and columns).

The **longval** usage depends on the datatype of the record. The longval may contain a numeric value such as type size or line thickness. If the datatype marks the record as having an external data area, the longval is the byte size of the following data. If the record starts or ends a group, the longval contains the group ID (or 0).

Note

All the format descriptions given in this specification are in the SPARC/Motorola/MIPS form, with MSB first.

Binary DCL

The first record in a binary DCL file is the **identifying record**; it is different from all other DCL records. Instead of having the form described above (datatype, majortype, minortype, shortval, longval), the first four bytes identify the file using a "magic" number. The second four bytes specify the byte order used in the file. To make this initial record accessible to all platforms, both the magic number and the byte-order codes are palindromes; they read the same forwards and backwards in binary form.

The binary DCL identifying record has the following format:

`0xA5 0xC3 0xC3 0xA5` *`byteorder byteorder byteorder byteorder`*

where *byteorder* = 0 for MSB first (most common), 0x81 for LSB first (Intel), or 0x42 for PDP-11.

Thus the second 4 bytes of a SPARC record would be 0x00000000, of a PDP-11 record 0x42424242, and of an Intel 80x86 record 0x81818181, all of which would read as the same unsigned long integer on all three architectures.

After reading this record, a program can select the appropriate method to use for reading the rest of the records.

Each of the remaining binary DCL records consists of an eight byte CTL plus any associated external data. Each **binary CTL** is read as two bytes, an unsigned short, and an unsigned long integer, and then byte-swapped as appropriate. The CTL format is:

`(datatype majortype) minortype short1 short0 long3 long2 long1 long0`

The first byte has two parts. The most significant nibble is the **datatype**. Datatypes are discussed at some length beginning on page 10.

The least significant nibble of the first byte is a four-bit **majortype** value. The meaning of majortypes 1 through 7 are given on page 6. The remaining values, 0 and 8 through 15, are unused and reserved.

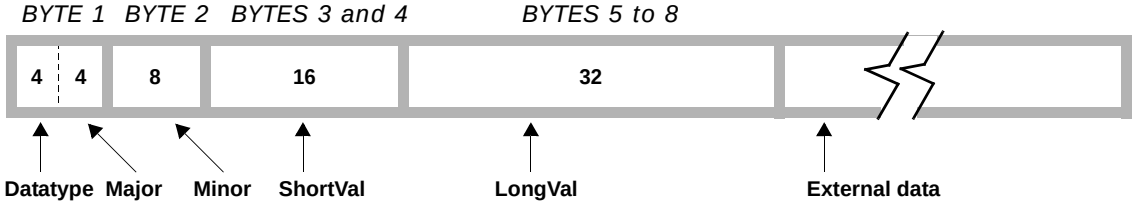
The second byte in a CTL is the **minortype**. It has a range of 1 to 254, with both 0 and 255 reserved for future expansion. Its interpretation is a more detailed expansion of the majortype. Details are given throughout this document.

The structure of the minortype allows users to introduce their own features into DCL. If the top bit of the minortype byte is set, the rest of the record (or group) has a user-defined meaning, and will be ignored if the defined UserID is not one of those understood by the program that is reading the DCL file. (A target writer ignores CTLs by skipping them, but a code processor copies CTLs it does not understand to its output.)

The next two bytes of a CTL are the **shortval**, read as a single short int. The range is 0 to 65,535 when used as an index or unsigned short (the usual case), or else -32,767 to +32,767 when used as a signed short int.

The remaining four bytes of the CTL are the **longval**, read as a single long int. Its usage depends on the datatype. For datatype 0 through 3 (internal), the longval contains the data itself; for example, if the datatype is 3, the longval contains a signed long integer, with a range of -2,147,483,647 to +2,147,483,647. When the datatype indicates external data (4-10), the longval contains the size in bytes of the following data.

The following diagram illustrates the functional parts of a binary DCL CTL.



Editable DCL

The editable form of DCL is used whenever DCL data must be examined or altered by humans, or by programs that do not understand binary DCL. It is useful when debugging programs that read and write binary DCL. Editable DCL need not necessarily use the ASCII character set. Although the description in this specification is in ASCII, the equivalent in any other set, such as Katakana (in Unicode) or EBCDIC, is equally acceptable. For any editable form, the main requirement is that a program exist that converts between that form and binary DCL, allowing the data to make the round trip without semantic alteration. As a reference, Omni Systems furnishes such a bidirectional converter between binary and ASCII DCL, the **dcx** program.

The ASCII form of DCL is readable and writable using any text editor, such as vi or emacs. An ASCII DCL file consists of a set of control records (CTLs) just as the binary DCL files do. Each CTL begins and ends with parentheses, making it easy to move between the record start and end, and to delete or copy records.

The first record is the **DCL identifier**, such as (DCL 100) for DCL of version 1.00. This record should not be preceded by any blank lines or spaces since the UNIX **file** command uses it to identify DCL files.

CTL formats: After the initial identifying record, the remaining records have the following basic format in an editable DCL file:

```
(majortype minortype shortval longval)
```

unless they are group CTLs (datatypes 14 and 15) or have following data (datatypes 4 through 10). The ASCII formats used for each of the datatypes are described in detail starting on page 10.

Group CTLs: The group CTLs, datatypes 14 and 15, generate an additional pair of parentheses that enclose each group. The group CTL is preceded by a start parenthesis, and the endgroup CTL is replaced by a single close parenthesis.

```
((majortype minortype shortval)
. . .
)
```

User-defined CTLs: If the CTL is user-defined (the top minortype bit set in the binary DCL), the first character after the opening parenthesis is an asterisk '*':

```
(*majortype minortype shortval longval)
```

Comments: A comment in an ASCII DCL file starts with a '#' and ends with the next newline; it can occur anywhere outside a CTL, but not within one.

Formatting conventions in editable DCL: The content of the CTLs can be given either in pure numeric form, using decimal numbers, or in mnemonic form, using the mnemonics defined in *stddef.dcl*. This provides complete language independence, and permits unrestricted user customization. For example:

`(1 10 1 0)` is the same as `(dcl page_units inches 0)`

Each record is typically on a line of its own for readability, but white space between records is optional and ignored. Any type and amount of white space within records (except within text strings) is treated as a single space.

The **lines** of a DCL file should never be more than 254 characters long, and 80-character (or shorter) lines are recommended for ease of editing. The only limit on the

byte length of a record is the maximum value of a long integer (2,147,483,647), which is also the UNIX limit on file size.

The case used for **mnemonics** is significant; lower is preferred. The current majortype mnemonics are dcl (for internal controls), text, layout, graph, spread, aud, and vid. The dcl majortype, used for support of ASCII DCL, has among its minortypes the internal controls “define”, “field”, and “scope”, described on page 15. These CTLs permit defining new mnemonics and redefining old ones within normal DCL files as desired.

Numeric data is in decimal, except that datatypes 0 (internal byte, “=X”) and 4 (external byte, “+X”) are hex (commonly used for rasters). Each number contains the data for one data item (typically a long) in the data following a CTL. White space separates the numbers; extra white space is ignored.

For **timestamps**, the CTL should be followed by a comment with readable time:

```
(dcl source omni : 696898721) #Fri Jan 31 14:58:41 1992
```

Text strings are enclosed in double quotes, one string per CTL. In them, literal double quotes are “\"’’, and a literal backslash is “\\”. Only characters in the printable set of **isprint(3)** are acceptable in strings. When a string ends at a line break, it must include a trailing space if one is wanted at that point when the line break shifts. Any line endings and tabs within text strings are ignored.

```
(text obj lit "this line ends, but the paragraph goes on. ")
```

The sequences ’\(’ and ’\)’ may be inserted in text strings as required to “balance” parentheses within them in ASCII DCL; they should be ignored. They are used only to make the editor parenthesis-matching functions work.

The Datatypes

The datatype—the first nibble in a record—indicates how the **longval** is to be interpreted, and whether external data is present. The following diagrams illustrate each of the fourteen defined datatypes. The binary form is the native DCL form. The **dcx** program converts a binary DCL file to and from human-readable ASCII format. Both forms of DCL are displayed below.

Summary of Datatypes

Value	***Meaning***
0, 1, 2, 3	longval of this CTL contains data (byte, short, long, mnemonic)
4, 5, 6, 7	longval of this CTL is the size of the following external numeric data (byte, short, long, double), which is treated logically as part of the CTL
8, 9, 10	longval of this CTL is the size of the following external non-numeric data (name, text, code)
11	longval of this CTL contains timestamp
12, 13	unused, reserved for expansion
14, 15	this CTL is the start or end of group

Note that values 4 through 10 describe the type and size of data *following* the eight-byte portion of the control. Values 15 and 14 indicate that this CTL is respectively the beginning or end of a **group** of CTLs. The other values give information about the type of data contained in the **longval** of the CTL.

Datatype 0: ibyte *(longval contains 4 bytes)*

In an ASCII DCL file, the **longval** portion of datatype 0 is four hexadecimal bytes preceded by **=X**. Datatype 0 is used for user ID.

(*major minor short* =X 4F 4D 4E 49) ← **ASCII form**

← **Binary form**

Datatype 1: ishort *(longval contains 2 shorts)*

In an ASCII DCL file, the longval portion of datatype 1 is two short integers preceded by **=H**.

(*major minor short* =H 12345 12345)

Datatype 2: ilong *(longval contains 1 long)*

In an ASCII DCL file, the longval portion of datatype 2 is one long integer preceded by an **= sign**.

(*major minor short* = 1234567890)

Datatype 3: imnem *(longval contains a mnemonic)*

In binary DCL the longval for datatype 3 is a signed long that has an assigned **mnemonic** value. In an ASCII DCL file, the longval portion is either the mnemonic, or the long in decimal form if no mnemonic is defined for it.

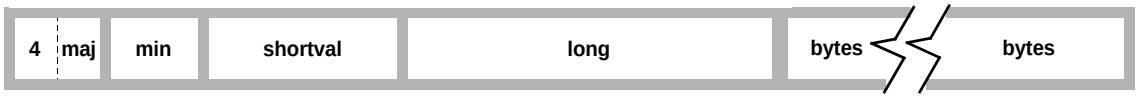
(*major minor short mnemonic*)

Datatype 4: ebyte *(longval contains the size of the following numeric data)*

In a CTL whose datatype is 4, the longval is a long giving the number of bytes in the following **external numeric data**. The external data is read as **bytes**.

In an ASCII DCL file, the external data is represented as hexadecimal bytes, preceded by **+X**. The closing parenthesis of the CTL follows the end of the data. This datatype is typically used for monochrome raster images.

(*major minor short* +X 4F 4D 4E 49 4F 4D 4E 49)



Datatype 5: eshort *(longval contains the size of the following numeric data)*

In a CTL whose datatype is 5, the longval is a long giving the number of bytes in the following **external numeric data**. The external data is read as **shorts**.

In an ASCII DCL file, the external data is represented as decimal numbers, preceded by **+H**. The closing parenthesis of the CTL follows the end of the data.

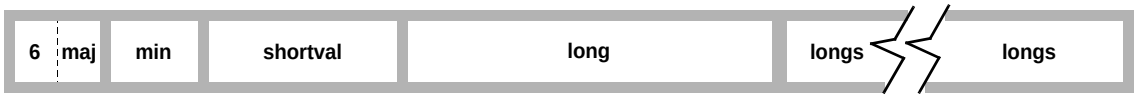
(*major minor short* +H 123 234 345 456)

Datatype 6: elong *(longval contains the size of the following numeric data)*

In a CTL whose datatype is 6, the longval is a long giving the number of bytes in the following **external numeric data**. The external data is read as **longs**.

In an ASCII DCL file, the external data is represented as decimal numbers, preceded by a **+ sign**. The closing parenthesis of the CTL follows the end of the data.

(*major minor short* + 12345 23456 34567)

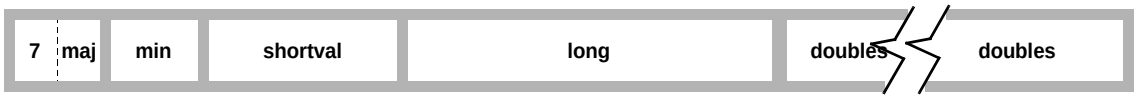


Datatype 7: edoub *(longval contains the size of the following numeric data)*

In a CTL whose datatype is 7, the longval is a long giving the number of bytes in the following **external numeric data**. The external data is read as **doubles**.

In an ASCII DCL file, the external data is represented as floating-point numbers, preceded by **+D**. The closing parenthesis of the CTL follows the end of the data.

(*major minor short* +D 123.234 345.456)

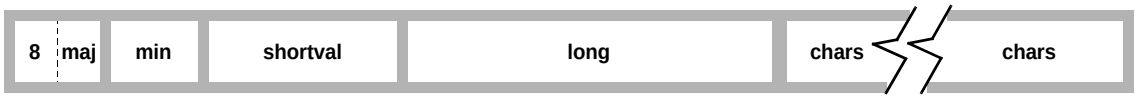


Datatype 8: ename *(longval contains the size of the following text data)*

In a CTL whose datatype is 8, the longval is a long giving the number of bytes in the following **external text data**. The external data is read as **characters** in the default character set of the document. The character data is always null-terminated in a binary DCL file.

In an ASCII DCL file, the external data is represented as a name in single quotes. The closing parenthesis of the CTL follows the end of the data.

```
(major minor short 'name')
```



Datatype 9: etext *(longval contains the size of the following text data)*

In a CTL whose datatype is 9, the longval is a long giving the number of bytes in the following **external text data**. The external data is read as **characters** in the current character set (which may not be the default character set of the document). The character data is always null-terminated in a binary DCL file.

In an ASCII DCL file, the external data is represented as text in double quotes. The closing parenthesis of the CTL follows the end of the data.

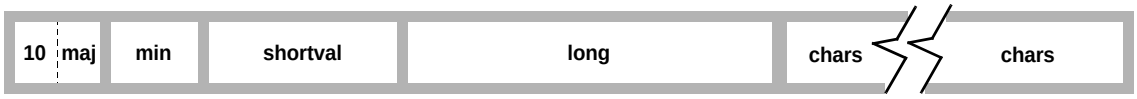
```
(major minor short "text")
```

Datatype 10: ecode *(longval contains the size of the following text data)*

In a CTL whose datatype is 10, the longval is a long giving the number of bytes in the following **external text data**. The external data is read as **characters** in the ANSI set. The character data is always null-terminated in a binary DCL file.

In an ASCII DCL file, the external data is represented as lines of code; the shortval contains the line count. The code begins on a new line, and the closing parenthesis of the CTL is on a new line following the end of the data. This datatype is used for EPS code, among other things.

```
(major minor short %
code
)
```



Datatype 11: stamp *(longval contains a timestamp)*

In datatype 11 CTLs, the longval is an unsigned long that is read as a **timestamp**.

In an ASCII DCL file, the timestamp is represented as a long preceded by a colon. The timestamp should be followed by a comment with readable time. (Text following a pound sign [#] on the same line is an ASCII DCL comment.)

(*major minor short* : 696898721) #Fri Jan 31 14:58:41 1992

? ***Datatypes 12 and 13 are reserved for expansion***.

Datatypes 14 and 15: group *(longval contains group ID or 0)*

Datatypes 15 and 14 are used to signal the beginning and end, respectively, of a **group** of related CTLs. The longval may contain zero, or an ID for matching group start and end.

In an ASCII DCL file, the **group** CTL starts with two open parentheses, and the entire **endgroup** CTL becomes a single close parenthesis. The group ID is not used.

The **group** type is used for CTLs such as graphics definitions. This datatype also allows a target writer or code processor to successfully ignore a set of CTLs that describe a feature that is not understood.

Groups can nest but not overlap.

((*major minor short*)
  other CTLs in the group
)

Internal Controls *(majortype 1)*

For DCL internal controls, the following minortypes are defined:

1 = define
2 = field
3 = scope
4 = attr

10 = page units
11 = leading units
12 = font units
13 = line units

20 = user ID
21 = source ID
22 = processor ID
23 = target ID

30 = alternate

40 = set
41 = duplicate

50 = include

60 = end

70 = ini

80 = debug

Define *(majortype 1, minortype 1)*

The **define** CTL(1) is used to define the mnemonics to be used for numeric values whenever the DCL binary file is converted to editable DCL form. The shortval contains the numeric (short integer) value of the mnemonic. The datatype is **ename**, and the longval gives the length of the naming string that follows. The defined string should be 80 characters or less, should begin with a letter, and should contain only alphanum characters and '_'. All lower case is preferred, but mixed case can be used if desired.

Field *(majortype 1, minortype 2)*

The **field** CTL(2) restricts the usage of defines to a particular field of CTLs. The shortval contains the field identifier:

shortvals
1 = major type
2 = minor type
3 = shortval
4 = longval

The datatype is group; the other CTLs in the group can include defines and scope groups, which may in turn contain more nested field groups. Each of the CTL fields

constitutes a separate name space, so the same defined string may have different meanings when in different fields. In addition, the same string can have different meanings in the same field when it is used within different scopes.

Scope *(majortype 1, minortype 3)*

The **scope** CTL(3) restricts the scope of defines to a particular area of DCL, so that the same names may have different meanings in different contexts. The **scope** CTL is used within a field group; the shortval of the scope CTL then specifies the value for that field within which the subsequent defines are in effect. The value must be given as a number. The datatype is group; the scope group contains a field group, which itself contains defines and possibly more nested scope groups.

To clarify field, scope, and definition usage, here is an excerpt from **stddef.dcl**, which contains the default definitions for ASCII DCL:

```
((dcl field maj)           #field is majortypes
 (dcl def 1 "text")        #majortype 1 is "text"
 ((dcl scope 1)            #scope is within majortype text
  ((dcl field min)         #field is minortypes
   (dcl def 3 "break")     #minortype 3 is "break"
   ((dcl scope 3)          #scope is within minortype break
    ((dcl field short)     #field is shortval mnemonics
     (dcl def 1 "para_start")#data type 1 is a paragraph start
     (dcl def 2 "para_end")#data type 2 is a paragraph ending
     (dcl def 3 "frame")
     (dcl def 4 "page")
     ((dcl scope 4)        #scope is within shortval page
     ((dcl field long)     #field is longval mnemonics
      (dcl def 0 "soft")   #longval 0 means soft page
      (dcl def 1 "hard")
      )                    #end field longval
     )                     #end scope of page
    (dcl def 5 "lpage")
    (dcl def 6 "rpage")
    )                      #end field shortval
   )                       #end scope of break
  )                        #end field minor
 )                         #end scope of text
)                          #end field major
```

Attr *(majortype 1, minortype 4)*

The **attr** CTL(4) specifies the nature of the information in the CTL. It is used only for conversions from binary and editable DCL to “xdcl”, the XML form of DCL, which is experimental.

shortvals
0 = tag
1 = id (string ID)
2 = nid (numeric ID)
3 = ref (to an ID)
4 = type
5 = value

6 = page
7 = rows
8 = col
9 = cols
10 = level
11 = lines
12 = next
13 = rel

Units *(majortype 1, minortypes 10–13)*

The **units** CTLs specify the units sets for position and size information in subsequent CTLs. **Page units**(10) are for layout, graphics, and horizontal text positioning; **leading units**(11) for vertical text positioning, interline and interparagraph; **font units**(12) for type size; and **line units**(13) for graphics line thickness. The shortval identifies the units set:

shortvals
1 = inches in 0.0001" increments
2 = metric in microns
3 = points (72 per inch) in hundredths
4 = Didot points (European standard) in hundredths

The precision of all of these is of a similar order of magnitude, ranging from 6,721 per inch for Didot to 25,400 per inch for microns. This exceeds by a factor of 2 to 10 the best resolution of current typesetting devices.

The longval can be used to scale the units set as desired. A size of 0 or 1000 indicates native scaling, smaller values specify smaller units, and larger values specify larger units. For example, a longval of 1000000 for metric units would make the unit millimeters instead of microns. Units are given near the start of the file, in the "document defaults" set of properties.

User ID *(majortype 1, minortype 20)*

The **user ID** CTL(20) uniquely identifies the user responsible for creation of user-defined controls, so that code processors and target writers can determine whether or not they know how to interpret such codes. The longval contains a four-byte user identifier, such as "OMNI". The shortval gives the minimum revision level (of the user program, not of DCL) required for code comprehension. It is desirable but not essential for user identifiers to be issued by a central source (initially Omni Systems, eventually a user group) to minimize the possibility of accidental duplication. This ID CTL may be reissued as required to distinguish CTLs defined by the source reader from those that may be defined by the code processors. Currently defined IDs are:

longvals
OMNI = new CTL defined by Omni Systems, but not yet part of standard
ASTX = Applix Aster*x feature
FMIF = FrameMaker MIF feature
ILFA = Interleaf ASCII feature
WPER = WordPerfect feature

Source ID *(majortype 1, minortype 21)*

The **source ID** CTL(21) is written at or near the beginning of the DCL file by the source reader. It uses the shortval for the source ID:

shortvals
1 = unknown
2 = Aster*x
3 = Avalon
4 = DCA/RFT
5 = DisplayWrite
6 = FrameMaker
7 = Interleaf
8 = Island Write/Paint/Draw
9 = Lyrix
10 = Microsoft Word (DOS)
11 = Microsoft Word (Mac)
12 = PageMaker
13 = Quark Express
14 = Quintet
15 = Rapport
16 = Signature
17 = troff
18 = Uniplex
19 = Ventura Publisher
20 = Word for Windows
21 = WordMARC
22 = WordPerfect through 4.2
23 = WordPerfect 5.0 and up
24 = GPO Microcomp

Its datatype is 11; the longval is a timestamp as returned by **time(3)**.

Processor ID *(majortype 1, minortype 22)*

A **processor ID** CTL(22) is added after the **source ID** CTL(21) by each code processor that operates on the DCL files. The longval contains either a timestamp as for the **source ID** CTL(21), or, if the datatype is ename, a version identifier string. The shortval identifies the processor type:

shortvals
1 = reader
2 = format constructor
3 = spelling checker
4 = grammar checker
5 = remapper (font or character)

Target ID *(majortype 1, minortype 23)*

The **target ID** CTL(23) specifies information that is intended for use only by a specific target writer. The shortval contains an ID value from the same set of values used for the **source ID** CTL(21). The datatype is group, and the group contains the target-specific CTLs. If the target writer that is reading the file is not the one identified, it should ignore (skip) that data. The datatype is imnem (3) instead when the **target ID** CTL(23) is used within a set of alternates, as defined below, to mark data that is not for that target.

Alternate *(majortype 1, minortype 30)*

The **alternate** CTL(30) groups information which is given in two or more forms to support target writers that have differing capabilities. Each **alternate** CTL(30) in a

set contains the same ID number in its shortval, beginning with 1 and increasing monotonically. This ID distinguishes alternate sets from those adjacent to them or nested within them. The datatype is group, and the group contains the current alternate. A program considering a set of alternates should skip to the end of the current alternate as soon as it sees a CTL describing an object it cannot render. To simplify analysis, the alternate may contain **target ID** CTLs (with datatype imnem, 3) to specify the targets for which the current alternate is not suited. After accepting one alternate, the program should skip all other alternates in the same set.

Set and Duplicate *(majortype 1, minortypes 40 and 41)*

The **set** CTL(40) identifies a group of CTLs for later duplication. The shortval is the set ID, a number starting with 1 and increasing monotonically. The datatype is group. The first two sets are reserved. Set 1 is reserved for document-wide defaults. Set 2 is for packaging all the resources in a DCL file, such as number streams and formats, from the document defaults up to the first page image.

The **duplicate** CTL(41) specifies immediate duplication of the CTL which has the set ID given in the shortval. It avoids needless repetition of CTL data.

Include *(majortype 1, minortype 50)*

The datatype of the **include** CTL(50) is ename (8), and the longval is the length of the following absolute or relative pathname of a file to include. The shortval indicates the type of file:

1 = binary DCL
2 = editable DCL
3 = graphic data (raster or EPS)

If the file referenced is found, its contents are read and interpreted immediately. Such included DCL files generally contain information used in multiple documents, such as format definitions and master frames.

End *(majortype 1, minortype 60)*

The **end** CTL(60) shortval is 1 to mark the end of a DCL file. The longval is the byte size of the file up to the start of the end CTL itself. This provides assurance that the file is complete.

Ini *(majortype 1, minortype 70)*

The datatype of the **ini** CTL(70) is etext (8), and the longval is the length of the following string. The shortval indicates the type of ini-file item described:

1 = application (basename of .ini file)
2 = sect (section heading)
3 = prop (key, on left of equal sign within section)
4 = value (data, on right of equal sign within section)

Debug *(majortype 1, minortype 80)*

The **debug** CTL(80) shortval is 1 to specify a breakpoint, when running a DEBUG version of a DCL processor under a debugger. It is generally inserted and also removed from editable DCL during the debugging process.

Text Properties *(majortype 2)*

Each text stream in a document is associated with a set of layout or graphics text elements, and is considered to flow through them. The text properties CTLs apply to the current text stream, as set by the text stream ID CTL below. Streams all begin with the document default properties, which are those set before the start of any text stream, in effect. Within each stream, footnote text properties are distinct from normal text properties. The minortypes are:

1 = text stream ID
2 = text frame
3 = text prev
4 = text next
5 = text anchor
6 = text stream properties

10 = text objects
11 = text breaks
12 = text keeps
13 = hyphenation
14 = text character set
15 = font attributes
16 = drop box size
17 = text language
18 = revision bar properties

20 = vertical positioning
21 = line spacing
22 = paragraph spacing
23 = vertical justification
24 = column coordination

30 = horizontal positioning
31 = horizontal alignment
32 = indentation
33 = word spacing
34 = character spacing

40 = tab definition
41 = tab position
42 = tab character

50 = table
51 = table properties
52 = table column
53 = table column properties
54 = row
55 = row properties
56 = cell
57 = cell properties
58 = title properties

60 = conditional definition
61 = conditional properties
62 = conditional usage
63 = conditional end
64 = conditional display

70 = footnote type definition
71 = footnotes
72 = footnote properties

75 = text inset
76 = text inset client
77 = text inset flow
78 = text inset text
79 = text inset table

80 = format definition
81 = format properties
82 = format call
83 = format text
84 = format tag

90 = variable definition
91 = variable properties

95 = reference definition
96 = reference properties
97 = reference token

100 = autonumber definition
101 = autonumber properties
102 = autonumber token
103 = autonumber setting

110 = list token
111 = variable value
112 = hypertext token

120 = SGML definition
121 = SGML usage

Text Streams *(majortype 2, minortypes 1–6)*

The **text stream ID** CTL(1) shortval has the ID of the current text stream. The stream ID begins with 1 and increases monotonically. The datatype is **group**, and the group contains the current portion of the stream. Text streams follow the frame definitions within the page descriptions (or the page definitions, for text on master pages).

For the **text frame** CTL(2), the shortval gives the ID of the frame in which the stream is contained; if the text does not fit, it flows on to the next frame as identified by the frame text content CTL in that frame. The longval gives the page number for that frame.

For the **text prev** CTL(3), the shortval gives the ID of the frame in which the previous portion of the stream is contained; 0 indicates the beginning of the stream. The longval gives the page number on which that frame is located.

For the **text next** CTL(4), the shortval gives the ID of the frame in which the next portion of the stream is contained; 0 indicates the end of the stream. The longval gives the page number on which that frame is located.

For the **text anchor** CTL(5), the shortval gives the ID of the frame which is anchored at this point in the stream. The longval gives the page number on which it is described, or 0 if it is not within a page description.

The **text stream properties** CTL(6) shortval has the property type:

Type	***Values***
1 = stream name	text flow tag
2 = auto add frames	0 = stay within frame, 1 = add new frame on overflow
3 = flow content	0 = text, 1 = PostScript code

Text Objects *(majortype 2, minortype 10)*

The **text object** CTL(10) shortval identifies the type of renderable object:

1 = literal text
2 = variable
3 = autonumber
4 = autonumber reference
5 = autonumber page reference
6 = generated text
7 = list token cross-reference

For literal text strings, the datatype is etext (9), and the longval contains the length of the text following. Variables, autonumbers, and cross-references are described later. Generated text includes text from the prefix and suffix content of formats, as well as autonumbers that don't have actual autonumber stream data (such as autonumbers used for bullets).

Text strings are restricted to "printable" characters, in the **isprint(3)** set. In ASCII DCL only, the backslash is used as an escape; a literal backslash is "\\", a literal double quote is "\"", and a balancing parenthesis is "\(" or "\)". In binary DCL, the literal backslash and double quote are themselves, and balancing parentheses are omitted. Binary DCL strings are always null-terminated.

Non-printable characters must be given in text object CTLs of datatype imnem (3). The longval contains the character value. Printable characters may also be given individually in these CTLs, instead of in strings. In ASCII DCL, they are shown as a mnemonic or as a decimal value, rather than as a character or as a string value.

Text Breaks *(majortype 2, minortype 11)*

The **text break** CTL(11) is used to identify points in the text flow at which breaks occur, and specify limits on them. The shortval may be:

Type	***Values***
1 = paragraph start	paragraph ID, or 0 if ID is not needed
2 = paragraph end	paragraph ID, or 0 if ID is not needed
3 = frame or col start	0 = soft, 1 = hard
4 = page start	0 = soft, 1 = hard
5 = left page start	0 = soft, 1 = hard
6 = right page start	0 = soft, 1 = hard

Type	***Values***
7 = return usage	0 = soft returns present, 1 = absent (as in Interleaf pre-5.2)
8 = cancel break	1 = cancel any break in effect from preceding format
9 = break char	group of text object lit CTLs, each containing a character or character sequence after which line breaks are acceptable (such as "-" and "/").
10 = line end	0 = soft, 1 = hard

Soft returns disappear entirely if they are not at line ends; if a space is wanted at such a point instead, one must be put explicitly in the text string before the soft return.

Text Keeps *(majortype 2, minortype 12)*

The **text keeps** CTL(12) describes the limits placed on frame or page breaks before, after, and within paragraphs. The shortval may be:

Type	***Values***
1 = keep prev	0 = allow, 1 = prohibit break before
2 = keep next	0 = allow, 1 = prohibit break after
3 = keep together	0 = allow, 1 = prohibit break within
4 = widows	minimum lines before break, 0 = no widow control
5 = orphans	minimum lines after break, 0 = no orphan control

Hyphenation *(majortype 2, minortype 13)*

The **hyphenation** CTL(13) specifies rules to use for hyphenation. The shortval may be:

Type	***Values***
1 = hyphenation	0 = off, 1 = used
2 = max hyph	maximum consecutive lines ending with hyphens

Type	***Values***
3 = min word	minimum word size to hyphenate
4 = min prefix	minimum characters before hyphen
5 = min suffix	minimum characters after hyphen
6 = respell	1 = hyphenated word at end of line is spelled differently if not hyphenated (as in German), affects only next line.

Fonts and Characters *(majortype 2, minortypes 14–18)*

The **text character set** CTL(14) longval has the size in bits of a character as used in text objects and in names, usually 8 or 16; 0 means variable. The shortval gives the character set used for the text data:

1 = 7-bit ASCII
2 = PC-8
3 = Standard PS (Adobe)
4 = Symbol PS (Adobe)
5 = Interleaf PS
6 = BitStream Standard
7 = Ventura Standard
8 = Ventura Math
9 = Windows
10 = Code Page 437 (United States)
11 = Code Page 850 (Multilingual)
12 = Code Page 860 (Portuguese)
13 = Code Page 863 (Canadian French)
14 = Code Page 865 (Norwegian)
32 = EBCDIC
48 = Macintosh
64 = ISO 8859:1 Latin-1 (also called “ANSI”)
128 = Unicode (default; text strings can contain only U+0020 through U+007E)

For ISO Latin-1 (ANSI), the undefined characters 128 through 159 are used as specified by FrameMaker; they consist primarily of PostScript characters missing from ANSI.

The following values define multibyte characters:

192 = Japanese, Shift-JIS (variable 8 or 16 bit)
193 = Simplified Chinese, GB2312-80EUC (16)
194 = Traditional Chinese, Big5 (16)
195 = Korean, KSC5601-1982 (16)
200 = EUC, ISO 2022 (variable length 8 to 24 bit)
201 = ISO 10646 (32-bit)

The **font attribute** CTL(15) uses the shortval for the attribute type and the longval for the attribute value. The values for each type are mutually exclusive; 0 always means normal or off:

Type	***Values***
1 = name	font number from list below or string name
2 = size	vertical font size, in font units
3 = compression	1 = compressed, 2 = condensed, 3 = narrow, 4 = wide, 5 = expanded
4 = font weight	1 = thin, 2 = light, 3 = demi, 4 = bold, 5 = extra bold, 6 = heavy
5 = font angle	1 = italic, 2 = backslant
6 = baseline	1 = subscript, 2 = superscript, 3 = drop initial
7 = capitalization	1 = lower, 2 = upper, 3 = initial caps, 4 = small caps
8 = font style	1 = shadow, 2 = outline
9 = revision mark	1 = inserted (redline), 2 = deleted (strikethrough) , 3 = changed (revbar)
10 = underscore rule	1 = spaces (default), 2 = tabs, 3 = both
11 = lining	1 = single underscore, 2 = double underscore, 3 = overline, 4 = numeric underline
12 = overstrike	character to use as overstrike char
13 = font color	color index value
14 = visibility	1 = invisible (hidden)
15 = Western name	for combined Western/Asian fonts (FrameMaker)
16 = Combined name	for combined Western/Asian fonts (FrameMaker)
For the following document properties, sizes are in font units:	
21 = superscript size	
22 = superscript offset	
23 = subscript size	
24 = subscript offset	
25 = small cap size	
26 = superscript stretch	
27 = subscript stretch	
28 = small cap stretch	

For the font name, if the source writer cannot find the name in the default font list, the datatype is ename (8) and the longval contains the length of the following font name. The default list is:

1 = unknown
2 = Times Roman
3 = Helvetica
4 = Courier
5 = Avant Garde
6 = Bookman
7 = Garamond
8 = Helvetica Black
9 = Helvetica Light
10 = Helvetica Narrow
11 = Korinna
12 = New Century Schoolbook
13 = Palatino
14 = Zapf Chancery
15 = Zapf Dingbats
16 = Symbol
17 = Webdings

The **drop box size** CTL(16) shortval has the width of the drop box for any following drop initial characters (0 to fit the width to the character). The longval contains the depth of the drop box below the baseline.

The **text language** CTL(17) shortval specifies a subtype, such as ISO (1), Frame (2), Interleaf (3), or Microsoft (4). The longval is either a numeric ID (as for FrameMaker and Microsoft), or the length of the following name.

The **revision bar** CTL(18) shortval contains the property type:

Type	***Values***
1 = auto	0 = no auto revision bars, 1 = auto create revision bars
2 = position	0 = left, 1 = right, 2 = inner margin, 3 = outer margin, 4 = Interleaf autopositioning
3 = color	color number
4 = gap	distance from column edge in lead units
5 = thickness	bar thickness in line units

Vertical Positioning *(majortype 2, minortypes 20–24)*

The **vertical positioning** CTL(20) itself takes immediate effect, specifying one of the following types of move for the distance (relative) or to the position (absolute, from the top) given in the longval. The values are in lead units. The move type is in the shortval:

1 = relative down
2 = relative up
3 = absolute in current frame
4 = absolute on current page

The **line spacing** CTL(21) specifies a mode in the shortval and a value in the longval. The defined modes are:

1 = fixed baseline-to-baseline
2 = fixed, allow for largest size in line
3 = additive to font size
4 = additive, allow for largest size in line
5 = proportional to font size

For the first four modes, the longval contains a height value. For the fifth, it contains a 0 or 1000 for “normal” leading, smaller values to tighten, and larger values to increase; 500 would be half spaced, and 2000 would be double spaced.

The **paragraph spacing** CTL(22) always contains a height value in the longval. It has one of these modes in the shortval:

1 = fixed before
2 = minimum before
3 = fixed after
4 = minimum after

The fixed values are always used, except that the before values are not used at the top of a frame. The minimum values are compared with the corresponding minimum of the adjacent paragraph, and the greater of the two is used (the model employed by FrameMaker, and by Interleaf 4.0 and earlier).

The **vertical justification** CTL(23) shortval contains the property type:

Type	***Values***
1 = feathering	0 = no vert justification, 1 = use vert justification
2 = line max	maximum spread between lines in lead units
3 = para max	maximum spread between paragraphs in lead units
4 = para shrink	maximum shrinkage between paragraphs in lead units
5 = frame max	maximum spread at anchored frame margins in lead units
6 = depth at break	percent of page height at which to attempt vert justification
7 = depth no break	same if no forced break (Interleaf)

The **column coordination** CTL(24) shortval gives the coordination property:

Type	***Values***
1 = sync lines	0 = off, 1 = synchronize baselines in adjacent columns
2 = spacing	nominal line spacing to which to sync, in lead units
3 = first line max	max height of first line to be synchronized, lead units
4 = balance	1 = balance columns on pages before breaks or at end

Horizontal Positioning *(majortype 2, minortypes 30–34)*

The **horizontal positioning** CTL(30) itself takes immediate effect, specifying one of the following types of move for the distance (relative) or to the position (absolute, from the left) given in the longval. The values are in page units. The move type is in the shortval:

1 = relative right
2 = relative left
3 = absolute in current frame
4 = absolute on current page

Horizontal alignment CTL(31) uses the longval (normally 0) as an offset (left is negative, right is positive) to the mode given by the shortval:

1 = left
2 = center
3 = right
4 = justified
5 = force-justified (leadered if in middle of line)

The **indentation** CTL(32) uses the longval as an offset from the left edge of the containing frame (or column) for the first line and body line indents, and as an offset from the right edge of the frame for the right indent. A negative value indicates a position outside the frame boundaries. The indent type is in the shortval:

1 = left for first line of para
2 = left for body of para
3 = right

The **word spacing** CTL(33) uses the shortval for the spacing type:

1 = optimum word space
2 = minimum word space
3 = maximum word space

The longval is 0 or 1000 for word space equal to an en space, smaller values to decrease proportionally, and larger values to increase.

The **character spacing** CTL(34) uses the shortval for the type:

1 = pair kerning
2 = maximum letterspacing to achieve justification
3 = track kerning

For type 1, the longval is 0 for off and 1 for on. For type 2, it is 0 for no letter-spacing, 1000 for en-space width. For type 3, it is in terms of the font size: 0 or 1000 for none, values under 1000 to tighten (500 would be 50% tighter), and values over 1000 to loosen (2000 would be 50% looser).

The **placement** CTL(35) uses the shortval for the type:

1 = span
2 = sidehead
3 = run-in punctuation

For type 1, span, the longval is 1 for placement in column, 2 for col but not sidehead area, 3 for col and sidehead area, and 4 for run-in. For type 2, sidehead, the longval specifies alignment, where 1 is first baseline, 2 is top edge, and 3 is last baseline.

The **sidehead** CTL(36) uses the shortval for the type:

1 = flow
2 = pos
3 = width
4 = gap

For type 1, flow, the longval is 0 for off and 1 for on. For type 2, position, it is 1 for left, 2 for right, 3 for inner, and 4 for outer.

Tabs *(majortype 2, minortypes 40–42)*

The **tab definition** CTL(40) indicates that the following tab values replace the ones previously in effect. The shortval contains a count of tabs. The datatype is group. The tabs are listed by position in ascending sequence.

The **tab position** CTL(41) longval contains the offset of the tab from the left edge of the frame (may be negative). The shortval is the tab type:

1 = left
2 = center
3 = right
4 = align on decimal (period or comma, per country)
5 = align on period
6 = align on comma
7 = align on tab align character
8 = unknown, issued but not set; longval is 0

Within format prefix and suffix groups, and autonumber definition groups, tabs are represented by an “unknown” tab position CTL, with a position of 0. This is necessary because the tab type and position are not determined until the format is called or the autonumber is used in the text; the tabs in effect at define time are not necessarily those in effect later.

The **tab character** CTL(42) shortval contains the character purpose:

1 = fill char, to fill the space up to the next tab, default of space
2 = align char, for align tabs only (type 7 above), default of period

The longval has the character itself, in the set used by the current text stream. For the fill character, if the datatype is etext (9), the longval contains the length of a string to repeat, which follows.

Tables *(majortype 2, minortypes 50–58)*

The **table** CTL(50) has a datatype of group, and the group contains the table data. The shortval contains the row count of the table, or 0 if the row count is unknown.

The **table properties** CTL(51) shortval contains the property type:

Type	***Values***
1 = name	name for master table
2 = placement	group, containing text alignment, indent, break, etc., CTLs
3 = top rule	ruling format number, 0 for no rule
4 = bottom rule	ruling format number
5 = left rule	ruling format number
6 = right rule	ruling format number
7 = autonumbering	0 = by row, 1 = by column
8 = alt shading	0 = by row, 1 = by column
9 = reg shade count	count of normally-shaded rows or columns
10 = alt shade count	count of alternately-shaded rows or columns
11 = alt color	color number
12 = alt fill	fill property number
13 = page end rule	0 = normal, 1 = table bottom rule ends each page
14 = defaults	group, containing title and row defaults CTLs

The **table column** CTL(52) shortval contains the column number starting with 1, or 0 for properties common to all columns. The datatype is group, and the contained CTLs describe the column properties.

The **table column properties** CTL(53) shortval contains the property type:

Type	***Values***
1 = column count	count of columns, if known
2 = column widths	page units, datatype elong (6) for multiple widths, or ilong (3) for the width of the current single column
3 = column color	color number
4 = column fill	fill property number
5 = left rule	ruling format number
6 = right rule	ruling format number

The **row** CTL(54) starts each row of the table; the datatype is group, and the shortval is the row type:

1 = body
2 = head of table
3 = foot of table
4 = running head (used after page break)
5 = running foot (used before page break)

The **row properties** CTL(55) shortval contains the property type:

Type	***Values***
1 = row name	name for master row
2 = auto height	0 or actual height of content (default)
3 = fixed height	height in page units
4 = min height	height in page units
5 = max height	height in page units
6 = row count	count of rows of the current type
7 = row color	color number
8 = row fill	fill property number
9 = top rule	ruling format number
10 = bottom rule	ruling format number
11 = boundary rule	for head and foot rows, rule between them and body
12 = ruling period	count of rows before different rule
13 = periodic rule	ruling format number to use every ruling period

The **cell** CTL(56) starts each cell of the row; the datatype is group, and the shortval is the content:

1 = text or empty
2 = graphic (in graphic frame with cell attribute set)
3 = numeric (represented as text)
4 = formula (application dependent)

Within a cell, the content, alignment, indent, and any other properties CTLs can be used in the usual way. However, such properties are effective only within the cell itself, not for the following text or graphics. When cells straddle, the CTL is given in the row that contains the top left corner of the cell.

The **cell properties** CTL(57) shortval gives the property type:

Type	***Values***
1 = straddle cols	column count
2 = straddle rows	row count
3 = rotation angle	degrees ccw from east, in hundredths
4 = locked	lock state
5 = color	color number
6 = fill type	fill property number
7 = top rule	ruling format number
8 = bottom rule	ruling format number
9 = left rule	ruling format number
10 = right rule	ruling format number
11 = top margin	margin width, in lead units
12 = bottom margin	margin width, in lead units
13 = left margin	margin width, in lead units
14 = right margin	margin width, in lead units
15 = t marg mode	0 = replace, 1 = add to table cell defaults
16 = b marg mode	0 = replace, 1 = add to table cell defaults
17 = l marg mode	0 = replace, 1 = add to table cell defaults
18 = r marg mode	0 = replace, 1 = add to table cell defaults
19 = vertical align	0 = middle, 1 = top, 2 = bottom

The **table title** CTL(58) shortval contains the property type:

Type	***Values***
1 = title placement	1 = at top of table, 2 = at bottom of table
2 = title gap	distance in lead units between title and top or bottom of table
3 = title content	group, including text format CTLs

Conditionals *(majortype 2, minortypes 60–64)*

These CTLs support the Interleaf “effectivity” controls, and the FrameMaker “conditional text” facilities. Note that this CTL is unrelated to the DCL internal alternate CTL, described previously.

The **conditional definition** CTL(60) shortval contains a condition ID number, and the data type is group. Condition IDs start with 1 and increase monotonically.

The **conditional properties** CTL(61) is used inside the conditional definition. The shortval contains the property type:

Type	***Values***
1 = name	datatype is ename (8), name follows
2 = overrides	datatype is group, contains CTLs
3 = visibility	0 = shown, 1 = hidden
4 = color	color number, used when printed

The overrides are applied to the text for which the defined condition is true. The properties given are typically font attribute CTLs, used when the text is not hidden and the conditional display CTL permits.

The **conditional usage** CTL(62) applies the condition ID in the shortval to the current text stream. More than one condition may be applied to the same text. The longval contains either a conditional block ID, or 0. The datatype is not group, because failure to understand the conditional should not result in discarding the data to which it applies. Instead, the **conditional end** CTL(63) with the same shortval and longval content marks the end of the affected data. Conditional usages may nest and overlap.

The **conditional display** CTL(64) properties apply to the entire document:

Type	***Values***
1 = show hidden	0 = no, 1 = show hidden text (for editing purposes)
2 = use overrides	0 = off, 1 = use conditional property overrides on screen

Both of these properties would be off for a file ready to be printed in final form. If “show hidden” is off, text marked as hidden is omitted from the DCL file. This provides the user with a choice between producing a DCL file that can be used as a master for all versions (with “show hidden” on), or one that is for a specific version (or set of versions) only (with “show hidden” off). It eliminates the usual need for producing a native-format file containing only the text for the desired versions before conversion.

Footnotes *(majortype 2, minortypes 70–72)*

Footnotes are considered part of the text stream in which they are referenced, but do not inherit properties from that stream. Instead, they inherit text properties only from one to the next, so that any properties set in the first footnote continue to affect all subsequent footnotes within the same text stream until changed in another footnote.

The **footnote definition** CTL(70) datatype is group, and the group contains the default footnote properties. The shortval contains the footnote type, 1 for text or 2 for table.

The **footnote** CTL(71) shortval contains the footnote type. The datatype is group, and the group contains the footnote data.

Footnote numbers are represented by autonumbers (described below). The autonumber itself is placed in the text at the footnote reference; the footnote number displayed at the beginning of the footnote is a reference to that autonumber. Both numbers are implicit; there are no codes required to describe them at their actual locations. If explicit codes are inserted, they override the implicit code properties.

The **footnote property** CTLs (72) in a footnote definition group specify default footnote properties; when they are inside a footnote, they override the default for that instance. The shortval gives the property type:

Type	***Values***
1 = format	default footnote format number
2 = number	footnote autonumber stream number
3 = max space	space at bottom of page for footnotes in lead units
4 = format tag	when footnote format is undefined, name follows
5 = frame above	ID of a master frame that describes the gap and line (if any) between the body (or table) text and the first footnote
6 = marker	character to use in place of footnote number for next footnote (only); supports Windows Help facilities

Text Insets *(majortype 2, minortypes 75–79)*

The text inset CTLs handle FrameMaker text insets, but may also be used for other cases of content transclusion, such as conrefs. Insets can be nested to any depth.

The **text inset** CTL(75) marks the beginning of a text inset; the shortval specifies the properties for the inset:

Type	***Values***
1 = start	group for starting properties
2 = name	optional ename, usually assigned by FDK client
3 = file	ename, source file path and name for inset
4 = import	ename, import filter hint string
5 = auto update	0 = no, 1 = yes
6 = last update	timestamp in longval
7 = locked	0 = no, 1 = yes
8 = end of inset	placed after inset content

The **text-inset client** CTL(76) is inside the start group, and identifies the API client if any that created the text inset:

Type	***Values***
1 = api	group for API client properties
2 = name	ename of client application
3 = file	ename, source file path and name
4 = type	ename, type of client file
5 = data	etext, added data used by client

The **text-inset flow** CTL(77) is inside the start group, and identifies the flow used within the source file:

Type	***Values***
1 = source	group for flow properties
2 = main	0 = no, 1 = yes
3 = page type	0 = body, 1 = reference
4 = name	ename, flow tag
5 = format source	0 = inset, 1 = container, 2 = plain text
6 = overrides	0 = keep, 1 = remove
7 = page breaks	0 = keep, 1 = remove

The **text-inset text** CTL(78) is inside the start group, and specifies properties for the inset text:

Type	***Values***
1 = prop	group for text properties
2 = EOL is EOP	0 = no, 1 = end of line is end of para
3 = encoding	encoding of inset, as in text charset CTL 2/14

The **text-inset table** CTL(79) is inside the start group, and specifies properties for the inset tables:

Type	***Values***
1 = prop	group for table properties
2 = name	ename, table format name
3 = byrow	0 = para is cell, 1 = para is row
4 = columns	if para is cell, number of columns per row
5 = cellsep	ename: if para is row, cell separator character ilong: if para is row, count of cellsep chars per cell
6 = header	ilong, number of header rows, 0 if none
7 = encoding	encoding of table, as in text charset CTL 2/14

Formats *(majortype 2, minortypes 80–82)*

For the **format definition** CTL(80), the shortval contains the format ID, a number starting with 1 that increases monotonically. The datatype is group. The definition may contain autonumbers and other text as well as property information.

The **format properties** CTL(81) shortval indicates the property type, and the longval contains the value:

Type	***Values***
1 = name	datatype is ename (8), size is length of following name
2 = type	effect on unmentioned properties, described below
3 = prefix	group, content goes at start of item affected by format
4 = suffix	group, content goes at end of item (usually paragraph)

Formats always cause properties specified in their definitions to be applied to the text stream, but they differ in the treatment of the properties not mentioned. The longval of the format properties type CTL identifies the possibilities:

1 = initialize from text stream defaults (default)
2 = inherit from current text stream values
3 = affect only table properties
4 = affect only ruling properties

The table formats (as in the Interleaf “Master Table”, or FrameMaker “Table-Format”) initialize from defaults, and contain other table CTLs so that they provide a template for newly-created tables.

The ruling formats describe a rule used between table columns or rows. These formats include one current line attribute CTL for each line in the rule, used left to right, or top to bottom. For multi-line rules, they specify the gap and line count.

The **format call** CTL(82) shortval contains the format type, and the longval contains the format ID. This CTL changes the current properties to those defined for the named format. If the format inherits current values, only those properties explicitly set in the definition are affected. If the format is initialized from stream defaults, all properties are affected whether mentioned or not. Any local overrides to the resulting properties are given after this CTL.

A call to *inherit* format 0 restores any properties altered by prior inherit formats (such as FrameMaker character formats, or Interleaf in-line components) back to the values in effect from the last init format. It is used for the FrameMaker "Default Para Font", and in editable DCL is "inherit restore".

When overrides affect the default para format, a call to *init* format 0 marks their end, and in editable DCL is "init store".

If the format contains text, such as autonumbers, the **format text** CTL(83) is issued next; its datatype is group, and the generated text, with any CTLs, follows. Its shortval is:

1 = prefix content
2 = suffix content

If a format call is made for an undefined format, the **format tag** CTL(84) is used to specify the requested format name. The shortval is a new format ID that will be associated with the new tag whenever it is used elsewhere. The datatype is ename (8).

Variables *(majortype 2, minortypes 90–91)*

The **variable definition** CTL(90) contains a variable ID in the shortval; ID numbers start with 1 and increase monotonically. The datatype is group, and the group contains variable properties CTLs for the variable name and format.

The **variable properties** CTL(91) contains the property type in the shortval:

Type	***Values***
1 = name	name of variable
2 = format	as described for the reference properties format CTL(96/3)
3 = date/time	0 = not used, 1 = current, 2 = file creation, 3 = file modification

When a variable is to be used, the **text object** CTL(10) is given with the variable specifier (2) in the shortval. For simple variable references, the variable format ID is in the longval. For more complex cases, the datatype of the text object CTL is group; the group begins with the reference properties CTL object ID property (which has the variable format ID). Then the group may contain two more reference property CTLs (a new format property, and a current value property), and any other formatting properties CTLs (such as font properties) that are needed.

References *(majortype 2, minortypes 95–97)*

The **reference definition** CTL(95) contains a reference format ID in the shortval. The datatype is group, and the group contains reference properties; it may also contain other properties, such as font attributes, as needed.

The **reference properties** CTL(96) is used in reference definitions, and also with text object CTLs (10) when they refer to a variable, or to a cross-reference list token. The shortval is the property type:

Type	***Values***
1 = reference tag	name of format, used only in reference definition
2 = ref object ID	ID of referenced object, number or string, 0 = unknown
3 = ref format	format ID number, or string (in def or as override), or group containing reference tokens, text objects, and properties
4 = ref value	literal text string, or group if other properties needed
5 = ext file name	name of file in which referenced object is located

The **reference token** CTL(97) contains the token type in the shortval, often with a subtype in the longval. The datatype is usually ilong (2), except for the two paragraph tag-matching (97/31 and 97/32) CTLs, which use ename (8).

The "%c" tokens, where present in the "Meaning" column of the following list, mean that the form of the token is as specified for that token in the POSIX **strftime()** function.

Type	***Sub***	***Meaning***		***Example***
1		Combined date and time		
	1	%c	local date and time	Fri Apr 13 15:25:30 1990
2		Combined date		
	1	%x	short local date	4/13/90
	2		long local date	April 13, 1990
3		Year		
	1	%y	short year	91
	2	%Y	long year	1991
4		Month		
	1	%b	short month name	Jan
	2	%B	long month name	January
	3	%m	month number, 2 digit	03 12
	4		month number, 1-digit	11 2
5		Week		
	1	%U	week of year from Sun	(first Sunday is day 1 of week 1)

Type	*Sub*	*Meaning*		*Example*
	2	%W	week of year from Mon	(first Monday is day 1 of week 1)
6		Day		
	1	%a	short day name	Tue
	2	%A	long day name	Tuesday
	3	%d	date number, 2 digit	02 10
	4		date number, 1 digit	23 5
	5	%j	day of year, 3-digit	031 (for Jan 31)
	6	%w	day of week number	(Sunday = 0, Saturday = 6)
10		Combined time		
	1	%X	short local time	3:25 PM
11		Time zone		
	1	%z	time zone, capital	PST EDT
	2		time zone, lower case	cst gmt
12		AM and PM		
	1	%p	am or pm, capital	AM PM
	2		am or pm, lower case	am pm
13		Hour		
	1	%I	hour, 12-hour, 2 digit	04 11
	2		hour, 12-hour, 1-digit	11 4
	3	%H	hour, 24-hour, 2 digit	02 14
	4		hour, 24-hour, 1 digit	15 3
14		Minute		
	1	%M	minute, 2-digit	05 10
	2		minute, 1-digit	20 3
15		Second		
	1	%S	second, 2-digit	34 09
	2		second, 1-digit	59 7
20		Filesystem		
	1		file name	dcl_spec.txt
	2		path name	/usr/dcl/doc/dcl_spec.txt
21		Page number		

Type	*Sub*	*Meaning*	*Example*
	1	page number	128
	2	last page number in doc	255
22		Table sheet	
	1	sheet number	1
	2	last sheet number in table	5
23		Variable	Running Head
		The longval contains the ID of the variable to be set by the variable value CTL (111).	
24		Document number	
	1	chapter number	
	2	volume number	
30		Paragraph	
	1	paragraph number only	2.5.3
	2	paragraph num string	Section 2.5.3
	3	tag of referenced item	Body
	4	text of referenced item	Using References
	5	pagenum of ref item	35
31		First paragraph on page with matching tag	
32		Last paragraph on page with matching tag	
		For both, datatype is ename (8) and the following data is the tag to match; normally used for running heads.	
35		Xref	
	1	paragraph number only	2.5.3
	2	paragraph num string	Section 2.5.3
	3	tag of referenced item	Body
	4	text of referenced item	Using References
	5	pagenumof ref item	35

Type	*Sub*	*Meaning*	*Example*
40		Index	
	1	no page number	gadget, see widget
	2	single page number	widget, 2
	3	start of range	tools, 2-5 (on page 2)
	4	end of range	tools, 2-5 (on page 5)

The **reference content** CTL(98) datatype is group, and wraps the reference content CTLs so that they can be matched up with their format elements. The shortval is the type:

Type	*Values*
1 = number	paragraph number only
2 = num string	paragraph numbering string
3 = tag	tag of referenced item
4 = text	text of referenced item
5 = pagenum	page number of referenced item
6 = chapnum	chapter number string
7 = volnum	volume number string

Autonumbers *(majortype 2, minortypes 100–103)*

The **autonumber definition** CTL(100) contains an autonumber stream ID in the shortval. Stream ID numbers start with 1 and increase monotonically. The datatype is group.

The **autonumber properties** CTL(101) shortval contains the property type:

Type	*Values*
1 = name	datatype is ename (8), name follows
2 = type	type of autonumber, described below
3 = stream ID	ID of definition, used in references
4 = format	group, described below
5 = reference form	same as 4, but applies to references to the autonumber
6 = sequence	sequence number in stream, omitted for master
7 = level number	level to be incremented or set, starting with 1

Type	***Values***
8 = value	group, content of current instance with properties
9 = symbol list	group, symbols to use for symbol token in format
10 = auto reset	0 = none, 1 = per page, 2 = per stream, 3 = per table
11 = increment	

The symbol list and auto reset properties are used mainly for footnotes. The reference form is used for the number in the footnote itself, which is a reference to the number in the text.

For the **type**(2) property, the longval contains the autonumber type:

1 = page number
2 = footnote number
3 = paragraph number
4 = figure number
5 = table number
6 = equation number
7 = list (default)
8 = chapter number
9 = volume number
10 = table footnote number

For the **format**(4) and **reference format**(5) properties, the datatype is group; the group contains text object CTLs for fixed text and autonumber token CTLs for variables, both interspersed with any required properties (such as font property or format use CTLs). Such properties are in effect only within the group.

The **symbol list**(9) is a group of text obj lit CTLs containing symbols to be used in the order given, typically for footnotes where numbering is per page. Symbols may be multiple characters. If the count of symbols needed in the text (before an auto reset) exceeds the count of symbol CTLs in the list, the last symbol is reused.

The **autonumber token** CTL(102) longval contains the level (counter) number described by the token, and the shortval specifies the counter type:

1 = no display (not used in definition format)
2 = arabic number
3 = lower-case roman
4 = upper-case roman
5 = lower-case alpha
6 = upper-case alpha
7 = symbol (from symbol list)
8 = text (fixed)
9 = chapter number
10 = volume number

The **autonumber set** CTL(103) longval contains the level number to be set; the shortval contains the value to which that level (counter) should be set.

When an autonumber is used in the text, the **text object** CTL(10) is given with one of the following specifiers in the shortval:

3 = autonumber
4 = autonumber reference (content of referenced autonumber itself)
5 = autonumber page reference (number of page on which autonumber falls)

Page numbers are automatically incremented, so they are always used as autonumber references, rather than as autonumbers per se. For them, the longval may contain just the **stream ID**(3). For more complex situations, the text object CTL datatype is group. The group begins with the **stream ID** property(3), may contain a new **format** property(4), and may include a current **value** property(8). It always includes a **sequence number**(6), except for autonumbers in master shared content (such as format definitions).

The autonumber **level** property(7) specifies the level to be incremented after the current instance. At define time all levels are set to 1 unless otherwise specified by one or more **autonumber set** CTLs(103).

List Tokens *(majortype 2, minortypes 110–111)*

The **list token** CTL(110) shortval contains the token type:

1 = index
2 = TOC
3 = list
4 = cross-reference
5 = hypertext
6 = subject (ALink)

The token content is usually a group of text obj lit CTLs and property CTLs.

In the index entry content, multiple entries are separated by semicolons, multiple levels by colons, and sort strings are in square brackets (as in FrameMaker format). The index entry group often contains property information, such as format calls, or includes reference tokens.

In the TOC token, the group content consists of the level number first as an ilong, then the title as etext, then the reference as an ename.

For other lists, the list name is first, then a colon, then the list entry text.

For cross-reference tokens, the content may be either a number or a string. When all references are within a single document, the numeric form is preferred. When the cross-references may be to or from external files, as with FrameMaker, the original marker string must be preserved in the token.

For hypertext, the literal command contained in the original document is used only if the command does not fit any of the types given for the hypertext token CTL(112).

The **variable value** CTL(111) shortval contains a variable ID which matches the ID used with the variable reference token (97/22). The datatype is ename (8), and the variable text follows. The text is used as the replacement for the variable token. This CTL is employed to set running heads that change based on page content. The reference is placed on a master page, and the variable value tokens are placed in the text stream whenever the content of the running head should be changed.

The **hypertext token** CTL(112) shortval contains a command type:

Type	***Values***
1 = option	1 = display in new window
2 = location	Frame newlink
3 = filename	Can include full path
4 = jump	Frame gotolink
5 = popup	Frame openlink
6 = alert	Frame alert, alerttitle
7 = URL	Frame message URL
8 = macro	
9 = macro definintion	
10 = alternate location	Type 11 markers
11 = jump to page	
12 = page location	Target for jump to page
13 = alert title	
14 = popup menu	
15 = target window	Window for hypertext jumps, often "_blank"

SGML *(majortype 2, minortypes 120–121)*

Global **SGML definition** CTLs(120) are given in the resources set, so that they precede the first real page. The shortval identifies:

1 = elements
2 = entities
3 = attribute lists

Within the text, **SGML usage** CTLs(121) use the shortval to specify:

1 = tags, which are references to the elements named
2 = ents, which are references to the entities named
3 = endtag for the named element
4 = attribute name
5 = attribute value
6 = element path (from current elem to root)
7 = PI (Processing Instruction) content

For all of them, the datatype is ename (8); the string name, stripped of its usual SGML delimiters (such as “<>” and “&;”) follows.

The SGML CTLs are also used for info from (or for) XML and HTML files.

Layout Properties *(majortype 3)*

Page definitions are given for each page in the document. Frame definitions are given in the page on which the frame appears. The frame is normally anchored to that page unless the frame anchor CTL is used to specify that it is anchored to the text. Frames may contain other frames anchored within them. Text streams follow all frames on the page. The following minortypes are defined:

10 = page definition
11 = page attributes
12 = page size

20 = frame definition
21 = frame attributes
22 = frame size
23 = frame anchor
24 = frame line
25 = frame pen
26 = frame fill

30 = frame preview

35 = frame properties
36 = frame property

40 = text content
50 = graphic content
60 = ps content

Page Properties *(majortype 3, minortypes 10–12)*

The **page definition** CTL(10) contains the page ID number in the shortval. The ID starts with 1 and increases monotonically. The datatype is group.

The **page attribute** CTL(11) uses the shortval for the attribute type and the longval for the attribute value:

Type	***Values***
1 = name	datatype is ename (8), the name follows
2 = usage	0 = body, 1 = first (of section), 2 = special (exception)
3 = master type	0 = not master, 1= page description master, 2 = special page master, 3 = reference master
4 = master ID	page ID of master page for this page
5 = orientation	0 = portrait, 1 = landscape, 2 = turned
6 = handedness	0 = none, 1 = right, 2 = left
7 = background	ID of frame to be used as page background
8 = number	string value of page num autonumber for current page

Master pages are templates for use as needed for a class of pages (such as left or right pages). Headers and footers are represented as frames on such pages. Interleaf Master Frames are stored on a single reference page. Reference pages (as in FrameMaker) are considered master pages.

Turned pages are portrait pages on which the content (but not the page headers and footers) has been rotated 90 degrees counterclockwise.

The **page size** CTL(12) shortval specifies the type of size information it has:

1 = full page size in page layout units
2 = count of columns and rows (for label pages)

The datatype is 6, and the longval contains the size of the following data, the width then height (or columns then rows, for label pages).

Frame Properties *(majortype 3, minortypes 20–36)*

The **frame definition** CTL(20) contains the frame ID number in the shortval. The ID starts with 1 and increases monotonically. The datatype is group.

The **frame attribute** CTL(21) uses the shortval for the attribute type and the longval for the attribute value. The value 0 always means normal or off:

Type	***Values***
1 = name	datatype is ename(8), name follows
2 = special use	1 = body, 2 = header, 3 = footer, 4 = border, 5 = label
3 = master type	1 = content, 2 = frame only
4 = shared	ID of master frame containing real content
5 = table use	1 = full table, 2 = cell content
6 = horizontal	1 = left, 2 = center, 3 = right, 4 = full, 5 = inner margin, 6 = outer margin
7 = vertical	1 = top, 2 = center, 3 = bottom, 4 = full
8 = ground	1 = foreground (overlay), 2 = background (underlay)
9 = position	1 = inline, 2 = next (following anchor), 3 = float, 4 = run-in
10 = autosizing	1 = auto width, 2 = auto height, 3 = auto both
11 = cropping	1 = cropped, 2 = overlap
12 = constraints	1 = horizontal, 2 = vertical, 3 = both (stay in col of anchor)

Type	***Values***
13 = rotation	degrees ccw from east, in hundredths
14 = gap	spacing around frame, in page units, for run-in
15 = columns	count of snaking columns
16 = column gap	gap between snaking columns, in page units

The **frame size** CTL(22) shortval specifies the reference point for frame positioning:

1 = text anchor
2 = text paragraph start
3 = page origin (top left)
4 = frame origin (top left of enclosing frame or column)
5 = column edge at vertical position of text anchor, as specified by the horizontal frame attribute CTL (21/6), which must be 1, 3, 5 or 6

A frame may have two size CTLs with different references, such as anchor and page, when both positions are known; the first is controlling. The datatype is elong (6), and the data follows. The x and y position (**xypos**) values are all relative to the reference point, and are given as pairs of signed longs. The interpretation of the data depends on the count of longs, based on four bytes of data for each:

Count of longs	***Following data***
0 = at reference	None
1 = anchor offset	Offset from anchor in text to bottom (y, for type 1)
2 = column offset	Offset from anchor at col edge (x and y, for type 5)
3 = size unknown	Top left corner xypos, width (0 if unknown)
4 = rectangle	Top left corner xypos, width and height
5 = ellipse	Center xypos, x and y diameter, rotation angle
6 and up = polygon	Vertices of polygon, xypos (2 longs) for each

For column offset frames, the x value is the distance from the specified column edge to the nearest side of the frame, with the frame placed outside the column. Positive values move the frame away from the column, negative values move the frame into the column. The y value is positive to move up, negative to move down; 0 aligns the base of the frame to the reference. This usage is in accord with that of FrameMaker and RTF.

For elliptical frames, the diameters are for a normalized ellipse (major axis horizontal), then the rotation (in hundredths of a degree) is applied counterclockwise.

The **frame anchor** CTL(23) specifies a frame anchored to a text stream. The shortval is the text stream ID, and the longval is the page ID number of the page containing the part of that stream that has the anchor.

The frame **line**, **pen** and **fill** CTLs(24, 25, 26) have the same syntax as the related CTLs for graphics attributes, described below, but apply only to frames, not to the objects inside them.

The **frame preview** CTL(30) specifies a graphic image that may be used in place of the entire frame. It is ename (8), and the filepath follows. The shortval is the preview type:

1 = GIF
2 = DIB
3 = TIFF
4 = WMF
5 = JPEG
6 = PNG

The **frame properties** CTL(35) is a group with shortval = 1 (for FrameMaker), used to identify arbitrary attributes assigned to graphic frames. It contains two **frame property** CTLs(36) of type ename, one each with the following shortvals:

Type	***Values***
1 = name	property name (FrameMaker ObjectAttribute tag)
2 = value	value string

Frame Content *(majortype 3, minortypes 40–60)*

Graphic content includes vector (draw) objects, raster (paint) objects, and EPSI (PostScript) objects as well as graphics text. Master frames may have content that is shared by later copies. Graphic frames contain graphic groups (4/1) and objects (4/10).

Text frames contain some portion of a text flow. The **text content** CTL(40) shortval contains the text stream ID of the flow (0 for empty master frames), and the longval contains the ID of the next frame to use. When the next frame is a master text frame, a new copy of that frame (and new page if needed) should be generated in the target application.

The **graphic content** CTL(50) is reserved for future use.

For PostScript page image support, the **ps content** CTL(60) shortval indicates defs (1) or page (2); the datatype is ecode (10). The defs CTL contains prologue information, and must be in the resources set that precedes the first page. The page CTL appears within the layout page group of its page. Concatenating all of these CTLs produces a PS file of the whole document; concatenating the defs CTL and any one page CTL creates a PS file that images that page.

Graphics Properties *(majortype 4)*

Graphics are contained in graphics frames. They consist of vector (draw) objects, raster (paint) objects, and EPS (PostScript) objects. The frame itself is described under “Layout Properties (majortype 3)”. The content of the frame is described in back-to-front sequence, without explicit z coordinates.

Graphic position and size data is given as two long integers, an x value then a y value, called **xypos** and **xysize**. Graphic positions are always relative to the top left corner of the smallest enclosing frame. Angles are measured in hundredths of a degree counterclockwise from east, in a range of 0 to 36000 inclusive.

The following minortypes are defined for the graphic majortype:

1 = graphic group

10 = graphic object

20 = line attribute
21 = pen attribute
22 = fill attribute
23 = marker attribute
24 = arrow attribute
25 = name attribute

30 = graphic shape

40 = graphic text attributes

50 = equation type
51 = equation properties
52 = equation content

60 = raster content
61 = raster properties
62 = raster colormap

70 = EPS content
71 = EPS properties
72 = EPS header

80 = color definition
81 = line pattern definition
82 = fill pattern definition
83 = marker definition
84 = arrow definition

Graphic Grouping *(majortype 4, minortype 1)*

The **graphic group** CTL(1) identifies the start of a set of grouped graphic objects, such as all the objects in one image plane, or all the lines and arcs that make up one visual item. This CTL may be repeated and nested to any degree desired. The shortval contains the group type, 1 for a normal group, 2 for a fillable group, in which the endpoints of the individually-unfillable objects in the group must connect, or 3 for a complex group, in which the first object (which may itself be a group) contains “holes” defined by the following objects (which may also be groups). The datatype is group.

Graphic Objects *(majortype 4, minortype 10)*

The **graphic object** CTL(10) has a datatype of group, to permit skipping of unrecognized graphic objects. The shortval has the type of object:

1 = vector
2 = graphics text
3 = equation
4 = raster
5 = PostScript
6 = unknown external

The CTL for unknown external objects contains an include CTL with the name of the external object's file. It may also have a frame size CTL, a frame attribute angle CTL, and line, pen, and fill CTLs. It may not have a frame definition CTL.

Graphic Attributes *(majortype 4, minortypes 20–25)*

Graphic attribute CTLs given outside a graphic object are persistent, and apply to all objects following them in the same frame until reissued. The starting defaults are given at the start of the frame before the first group; any not present receive the standard DCL defaults, described below for each, as modified by graphic attributes given in the starting document defaults set. All attributes given within a graphic object definition apply only to that object.

The **line attribute** CTL(20) shortval has the attribute type, and the longval contains the attribute value:

Type	***Value***
1 = line pattern	Line pattern number, 0 = invisible, 1 = solid
2 = line color	Color number, 0 = invisible, 1 = black
3 = line thickness	Thickness in line units
4 = line spacing	Distance between multiple lines in rule in line units
5 = line count	Number of parallel lines, usually 1 or 2

The default line is black, solid, and 1 pt, 0.5 mm, or 0.02" thick (depending on the line thickness units in effect).

The **pen attribute** CTL(21) shortval has the attribute type, and the longval contains the attribute value:

Type	***Value***
1 = fill pattern	Fill pattern number, 0 = invisible, 1 = solid, 2-15 = hatch
2 = foreground	Color number, 0 = use line color
3 = background	Color number, 0 = transparent
4 = shading	0-255 of foreground color, 0 = solid, 255 = whited out

T he fill pattern is applied only to the solid parts of the current line attribute. The foreground color number is 0 to use the current line color. The background color number is 0 for transparent. The default pen pattern is solid in the current line color. For FrameMaker, the line attribute pattern is always solid, and the pen attribute alone is used to vary line appearance.

The **fill attribute** CTL(22) shortval has the same syntax as the pen attribute shortval. The background color number is 0 for transparent. Fill pattern 0 is background (or invisible). The default fill pattern is opaque solid black. Fill applies to the area that lies inside the fillable object according to the odd-parity rule.

The **current marker style** CTL(23) shortval has the attribute type, and the longval contains the attribute value:

Type	***Value***
1 = marker style	Marker style number
2 = marker color	Color number
3 = marker size	Height

The marker size is measured from top to bottom at a line drawn vertically through the horizontal centerpoint. The default marker is a black dot, 1 pt, 0.5 mm, or 0.02" diameter (depending on thickness units in effect).

The **arrow attribute** CTL(24) shortval has the attribute type, and the longval contains the attribute value:

Type	***Value***
1 = arrow head	Arrow style number
2 = arrow tail	Arrow style number
3 = arrow length	Length
4 = arrow width	Width

The default is head 0, tail 0 (no arrow). The length and width apply to both the head and the tail. The head is the starting point of the polyline or arc with the arrow.

The **name attribute** CTL(25) shortval contains the ID of the mutually-exclusive name space to which the name belongs. The datatype is ename (8), and the name follows. The name replaces any name in effect in the same name space, but applies in addition to any names in effect in other name spaces.

Vector Graphics *(majortype 4, minortypes 30 and 40)*

For vector graphics, the **graphic shape** CTL(30) specifies the object form. Its data-type is typically elong (6), and the longval contains the size of the following data.

For **polys**, the shortval contains:

1 = polylines
2 = polygons
3 = polymarkers

The data following has one xypos for each mark or vertex; the last vertex of a polygon need not be the same as the first, but may be if convenient. Polygons are always filled; polylines are not filled even if they are closed.

For **rectangles**, the shortval contains:

10 = square-cornered
11 = rounded

Both types of rectangles are specified by the xypos of the upper left corner, then an xysize giving the width and height. In addition, the rounded rectangle uses another xysize to give the corner radius (for circular corners, the x and y sizes are equal).

For **ellipses**, the shortval contains:

20 = normal ellipses, with the major and minor axes at right angles
21 = Interleaf ellipses, where the axes may be at any angles

The normal ellipse CTL is followed by the xypos of the center, an xysize with the major radius as x and the minor radius as y, and the rotation angle from the x axis (east) counterclockwise to the right end of the major axis. The Interleaf ellipse data contains the xypos of the center, the xypos of the left end of the major axis, and the xypos of the top end of the minor axis; if the rotation is exactly 90 degrees, the top end of the major axis and left end of the minor axis are used.

For **elliptic arcs**, the shortval is:

30 = normal unfilled
31 = pie-slice filled (with the center connected to each endpoint)
32 = chord filled (endpoints connected to each other)

The data contains the same xypos and xysize used for normal ellipses, the rotation angle, then the start and end angles. The start and end angles are both measured counterclockwise from the east end of the normalized major axis (before the rotation angle is applied), and the arc is drawn from start to end counterclockwise. If the start and end angles are equal, the arc is a normal ellipse.

For **Hermite arcs**, used by Interleaf only, the shortval is:

33 = non-extended
34 = extended

The non-extended form uses five points, each described by an xypos, in accordance with Interleaf rules. The extended form adds a sixth point. Refer to Interleaf docs for details. This form is never filled, although it can be grouped with polylines (and other open arcs and curves) so as to construct a fillable object.

For **spline** curves, the shortval contains:

40 = open spline
41 = closed spline

The data uses the same syntax as polys. The closed form is fillable.

For **Bezier** curves, the shortval contains:

50 = open Bezier
51 = closed Bezier

The descriptive data is the same as the spline curve, except that the data contains six long integers per vertex instead of two. Each vertex xypos is preceded by the xypos of the preceding control point, and followed by the xypos of the following control point. The closed form is fillable. The open form begins and ends with a vertex, not a control point, so the ends have one control point each instead of two.

Graphic Text *(majortype 4, minortype 40)*

Graphic text objects (10/2) contain the **graphic text attributes** CTL, any needed text property CTLs, and text object CTLs for the text itself.

The **graphic text attributes** CTL(40) shortval specifies the alignment of the text to its anchor point:

1 = left (default)
2 = center
3 = right

The datatype is elong (6), and the longval has the length of the following data, the xypos of the anchor point and rotation of the text around it.

Equations *(majortype 4, minortype 50)*

The **equation type** CTL(50) shortval gives the format type:

1 = standard troff eqn
2 = Interleaf math object
3 = WordPerfect near-eqn
4 = Frame MathFullForm

The **equation properties** CTL(51) shortval gives the property type:

1 = frame size, elong (6) with xypos and xysize of math object
2 = alignment, 0 = left, 1 = center, 2 = right, 3 = justified
3 = text size, 0 = small, 1 = medium, 2 = large

The **equation content** CTL(52) shortval gives the content type:

1 = Frame MathFullForm as an ename (8)
2 = graphic file, ename (8) containing file path

Rasters *(majortype 4, minortypes 60–62)*

The **raster properties** CTL(61) contains the pixel depth in the shortval. The datatype is elong (6), and the data follows: an xypos for the top left corner, an xysize for the display size, another xysize in pixels, and the rotation angle.

A negative display size means the object is flipped around the corresponding axis; if the width is negative, the object is flipped left-to-right, and if the height is negative, the object is flipped top-to-bottom. The rotation angle is applied last, after any flipping.

The **raster content** CTL(60) contains the compression method in the shortval:

1 = uncompressed, 8-bit row roundup
2 = Portable Bit Map (PBM), no colormap
3 = PCX run-length-encoded
4 = SunRaster RT_BYTE_ENCODED
5 = TIFF PackBits
6 = MS DIB uncompressed (32-bit roundup)
7 = MS DIB compressed for 4-bit pixels
8 = MS DIB compressed for 8-bit pixels
9 = WMF (not necessarily raster)

If the datatype is group, an include CTL follows with the pathname of a file containing the raster data in its standard format (TIFF, SunRaster, PBM, PCX, or DIB, with normal header). Otherwise the datatype is ebyte (4), and the longval contains the size in bytes of the following raster data.

The **raster colormap** CTL(62) shortval contains the colormap type, 1 for SunRaster or 2 for DIB. The datatype is ebyte (4), and the longval is the length of the uncompressed color map following. If the pixel depth is 1, the raster colormap has two entries. If the depth is 24 or greater, the raster colormap is omitted. If the map is not large enough to reset all 256 color values, the unset values retain their previous states. If the raster is in a referenced external file, identified in an include CTL, the colormap is also contained in that file, not in the DCL file.

EPS Images *(majortype 4, minortypes 70–72)*

The **EPS properties** CTL(71) shortval contains the type:

1 = EPSI with ASCII preview
2 = EPSI without preview
3 = EPS with binary preview
4 = EPS without preview
5 = PostScript without BoundingBox (non-EPS compliant)

The datatype is elong (6), and the data follows: an xypos and xysize for the position and size in the frame, then the lower-left xypos and upper-right xypos from the BoundingBox (in PostScript units, both 0 for type 5), then the rotation angle. As described for rasters, negative sizes mean the object is flipped; the rotation angle is applied last, after any flipping.

If the **EPS content** CTL(70) datatype is group, then an include CTL with the pathname of a standard EPS file containing the data follows. Otherwise, the datatype is ecode (10) and the longval contains the length of the PostScript data that follows. The last byte of the PS data is always 0 in binary DCL, to permit string operations on it (such as **strstr()** for searching). The shortval contains the count of PostScript lines, excluding the binary header if any.

If the EPS file has a binary header, it is in an **EPS header** CTL(72) with datatype ebyte (4). The header format is specified in the same way as for the raster format CTL (60), with the compression method in the shortval.

Attribute Definitions *(majortype 4, minortypes 80–84)*

The attributes identified by numeric indexes (color, line pattern, fill pattern, marker style, and arrow style) can be redefined at any time.

The **color definition** CTL(80) shortval has the color number defined; color 0 is reserved to mean “invisible”. The longval is treated as four unsigned chars, datatype ibyte (0). The MSB, long3, contains the definition type: 1 for RGB (also used for values originally in CMY or CMYK, since conversion is trivial), 2 for HSV (hue, saturation, value), and 3 for CIELUV. The three LSBs contain the three color values in their usual order. If more precision is needed than is provided by 8 bits, the CTL datatype is elong (6); then the data that follows has a 32-bit long int for the type and for each value.

The DCL default colors include the FrameMaker defaults; the “1” values below represent the maximum value, 0xFF for 8-bit color:

Color	***R***	***G***	***B***
1 = black	0	0	0
2 = white	1	1	1
3 = red	1	0	0
4 = green	0	1	0
5 = blue	0	0	1
6 = cyan	0	1	1
7 = magenta	1	0	1
8 = yellow	1	1	0

The **line pattern definition** CTL(81) has the line pattern number in the shortval; pattern 0 is invisible. The datatype is elong (6), and the longval has the length of the following data. The first long is the real length of one segment of the line pattern; the second is the bit length of that segment. It is followed by a bitmap of the pattern, where set bits are present and zero bits are absent, MSB to LSB. The default pattern definitions depend on the source application; 1 is always solid.

The **fill pattern definition** CTL(82) has the fill pattern number in the shortval; pattern 0 is invisible. The datatype is elong (6), and the longval has the length of the following data. The first long is the real width of a tile of the pattern. The second long contains the bit depth of the pattern (1 for bitmap, usually 8 for pixmap). The third long contains the map width, and the fourth the map height. For FrameMaker, the map is 16 bits wide and 8 bits high; for Interleaf, the map has 16 rows of 16 pixels each. For a bitmap, the rest of the long ints contain a bitmap of the pattern; set bits are foreground and zero bits are background. For a pixmap, each pixel is specified by an 8-bit (or more) value, packed MSB to LSB in long ints; the values are each color numbers (not raster colormap numbers), with zero used for transparent background. The default pattern definitions depend on the source application; for FrameMaker, styles 0 through 14 map directly to DCL styles 1 through 15.

The **marker definition** CTL(83) has the marker style number in the shortval. The datatype is group, and the group contains CTLs that define the vector or raster object to be used as a marker. All positions in those CTLs are relative to the mark point, which need not be the center of the marker (or even within the marker bounds). The default marker definitions depend on the source application. Style 0 is invisible; style 1 is dot.

The **arrow definition** CTL(84) has the arrow style number in the shortval. The datatype is elong (6), and the longval has the length of the following data. The first long contains the arrow definition type:

1 = Aster*x
2 = Avalon
3 = FrameMaker

The remaining content depends on the definition type. Dimensions are in line units. It always begins with an xysize containing the arrow length (as x) and the arrow head type (as y, 1 = filled, 2 = hollow, 3 = open). Next, for type 1 (Aster*x), it has an xysize with the width (as x) and the base indent (as y), in the same units as the length. For type 2 (Avalon), it has an xysize with the width (as x) and the base indent (as y), both in percent of length. For type 3 (FrameMaker), it has an xysize containing the tip angle (as x) and the base angle (as y). The default arrow definitions depend on the source application; style 0 is reserved to mean no arrow.

In addition, type 3 (FrameMaker) treats line caps as arrow head types: 0 = square (default), 4 = butt, and 5 = round.

The default styles are 0 = square, 1 = butt, 2 = round, and 3 through 10 are the standard FrameMaker arrows.

Objects *(majortype 4, minortypes 90–92)*

The **objects** CTLs can be used for graphics in place of the graphics CTLs whenever they have sufficient content to satisfy the output modules. In general, that is true for all HTML and most XML outputs. They can also handle audio and video.

The **object definition** CTL(90) has a datatype of group, and wraps the object properties and param CTLs. The shortval can be used for ID.

The **object properties** CTL(91) contains the information that would be used for attributes of the object element in HTML 4 and in DITA. The shortvals are:

Type	***Value***
1 = data	etext, URL of data file
2 = classid	etext, identifier for object type
3 = type	etext, MIME type such as image/jpg
4 = codebase	etext, URL of implementation code if needed
5 = codetype	etext, MIME type for classid item
6 = ID	ename
7 = alignment	0 = none, 1 = left, 2 = right, 3 = center, 4 = texttop, 5 = textmiddle, 6 = textbottom, 7 = middle, 8 = baseline
8 = height	ilong
9 = width	ilong
10 = hspace	ilong
11 = vspace	ilong
12 = border	thickness, 0 = none
13 = declare	0 = normal, 1 = postpone load until called
14 = standby	etext, message to display while loading

The **object param** CTL(92) describes one param element used in the object element. Its shortvals are:

Type	***Value***
1 = param	group containing the other param CTLs
2 = name	etext, name attribute
3 = value	etext, value attribute
4 = value type	0 = data (default), 1 = ref, 2 = object

Spreadsheet Properties *(majortype 5)*

DCL may be used to describe included spreadsheets, such as those used in Aster*x and Rapport. The DCL minortypes are not yet defined.

Audio Properties *(majortype 6)*

Audio annotations may appear at any point in text or graphics. The DCL majortype is 6; the minortypes are not yet defined.

Video Properties *(majortype 7)*

Video annotations may appear at any point in text or graphics. The DCL majortype is 7; the minortypes are not yet defined.

User-defined Properties

DCL permits definition of properties specific to the source application in a manner that can be disregarded by target writers that are not aware of those properties.

A user-defined CTL is identified by setting the top bit of the minortype. If it is the start of a group, the datatype is group, so that related data can be skipped by a target writer that does not understand it. The other datatypes also have their usual DCL meanings. Otherwise, the CTL can be defined in any way desired, although it is wise to keep the basic design concepts of DCL in mind when planning its use.

The specific user by which the CTL is defined is identified by the user ID CTL. The user identification is persistent and may be changed at any time.

For example, suppose that Frame wanted to include the <Pgf Next Tag> data in the DCL. There is no CTL defined for that, so Frame could make one up with a datatype of ename (8), a majortype of 2 (text property), a minortype of 201 (128 + 83, in the format area), a shortval containing the format ID for the Next Tag name, and a longval giving the length of the tag name, followed by the name itself. Sometime before the first use of this CTL, perhaps near the start of the DCL file, Frame would issue a user ID CTL with a shortval of 200 (for version 2.0) and a longval of “FMIF”. Then any program that recognized CTLs written by that user ID would understand the Pgf Next Tag item. Frame could define many such CTLs to “enrich” its DCL files.

In general, the DCL version of a document should be a *complete* version, so that if it is converted back to its original format *nothing* is lost. Then it is possible for third-party products to operate on the DCL form without having to understand dozens of unique formats in order to do their job. A single language translator, for example, can work with DCL documents from many sources without affecting, or even understanding, their native formatting. The ability to define new DCL properties at will, without breaking existing translators, is critical to the success of such a process.

Email: <info@omsys.com> *Web:* <http://www.omsys.com>