A.3.12.1.9 *W—set segment transparent*

```
W—Set—Seg—Transparent      ::= SEQUENCE { transparent—segment—number      [1] INTEGER,
                                          transform—matrix                [2] MAT }
```

—— Set transformation matrix attributes for the identified segment.

```
MAT                        ::= SET { matrix—element—11                    [11] REAL,
                                     matrix—element—12                    [12] REAL,
                                     matrix—element—13                    [13] REAL,
                                     matrix—element—21                    [21] REAL,
                                     matrix—element—22                    [22] REAL,
                                     matrix—element—23                    [23] REAL }
```

—— Transform Matrix Definition.

A.3.12.1.10 *W—set priority*

```
W—Set—Priority             ::= SEQUENCE { priority—segment—number     [1] INTEGER,
                                          priority—value              [2] REAL }
```

—— Set segment priority attribute for the identified segment. This is analogous to display order priority.

A.3.12.2 *Work station independent*

```
Work—Station—Independent   ::= CHOICE { [1] W—Associated,
                                         [2] W—Copy,
                                         [3] W—Insert }
```

A.3.12.2.1 *W—associated*

```
W—Associated               ::= SEQUENCE { associated—w—station—id      [1] INTEGER,
                                          associated—segment—number   [2] INTEGER }
```

—— Associate the identified segment with the identified work station.

A.3.12.2.2 *W—copy*

```
W—Copy                     ::= SEQUENCE { copy—w—station—id                [1] INTEGER,
                                          copy—segment—number             [2] INTEGER }
```

—— Copy the primitives of the identified work station.

A.3.12.2.3 *W—insert*

```
W—Insert                   ::= SEQUENCE { insert—segment—number       [1] INTEGER,
                                          insert—transform—matrix—ref [2] MAT }
```

—— Transform and display segment.

A.3.13 *Colour control string*

All of the source terminal data syntaxes provide the capability to define colour and have available, at least optionally, a colour map capability. However the colour model which is used by each of the source terminal data syntaxes differs significantly. In order to provide a neutral basis for colour, the colour model developed for ISO 8613 Text and Offices Systems — Office Document Architecture is used here.

The basic colour model used in ISO 8613 is a colour cube in terms of the three Red, Green, Blue basic vectors. A colour in the colour model is represented by a three tuple of RGB components. Logically these colours are normalized from 0 (minimum) to 1 (maximum). Therefore `Black` is the three tuple <0,0,0> and `White` is the three tuple <1,1,1>. Since all of the videotex terminal data syntaxes support colour models which are different from this, the mapping of the specific colour models to the basic RGB colour cube must be understood by the conversion process.

Two colour indexing modes are available: Direct and Indexed. In direct colour selection, the colour is defined by providing a three tuple of discrete values for the RGB components. In the indexed colour selection mode, the colour is defined by an index into a single colour table of discrete colour values. The number of colours which may be defined in the colour table is terminal model dependent. The limit assumed in the definition of a particular set of data is specified in the Boundary Value Definition section. If a receiving system cannot image the range of colour values specified by a direct colour value or the colour value indexed by a colour index, then a `closest match` is assumed according to the criteria stated in ISO 8613. A variant of the indexed colour mode called `auxiliary colour mode` is used to define a colour for the background of a text or mosaic character cell.

```
Colour—Control—String        ::= CHOICE { [1] Basic—Colour—Selection,
                                          [2] Direct—Colour—Selection,
                                          [3] Indexed—Colour—Selection,
                                          [4] Auxiliary—Colour—Selection,
                                          [5] Colour—Index—Setup }
```

A.3.13.1 *Basic colour selection*

```
Basic—Colour—Selection       ::= INTEGER { black              (0),
                                           red                (1),
                                           green              (2),
                                           yellow             (3),
                                           blue               (4),
                                           magenta            (5),
                                           cyan               (6),
                                           white              (7),
                                           auxiliary—black    (8),
                                           auxiliary—red      (9),
                                           auxiliary—green    (10),
                                           auxiliary—yellow   (11),
                                           auxiliary—blue     (12),
                                           auxiliary—magenta  (13),
                                           auxiliary—cyan     (14),
                                           auxiliary—white    (15),
                                           auxiliary—foreground (16) }
```

—— Several of the terminal data syntaxes provide a simplified way to access the basic primary colours by the use of C1 set codes. The various C1 control sets differ in a fundamental manner with respect to how the colour command interacts with other attributes. In order to avoid the difficulty in interworking, only the basic colour commands themselves are identified here. The range of a colour specification is terminal model dependent. This tendency has been avoided by defining all the colour selection commands in terms of the colour model specified in ISO 8613. Colour range dependencies (serial row attributes or parallel cell attributes) should be expressed in terms of the abstract colour by the conversion process. That is, all the rules inherent in the serial attribute method of specifying basic colours, or in the parallel attribute method, should be resolved by the conversion process which creates the IDS colour commands.

—— The auxiliary colour commands specify the background colour for text and mosaics. The command `auxiliary foreground` specifies that the background colour should be set to the current foreground colour.

A.3.13.2 *Direct colour selection*

```
Direct—Colour—Selection ::= SEQUENCE { REAL, REAL, REAL }
```

—— Direct colour selection permits colours to be specified in terms of the Red Green Blue components of the colour model. The ASN.1 REAL data type is use since this form of number is self—delimiting and of arbitrary length. The real number parameters are relative to the maximum colour value for each component. The parameters are Red, Green and Blue respectively.

A.3.13.3 *Indexed colour selection*

```
Indexed—Colour—Selection ::= INTEGER
```

—— Indexed colour selection permits colours to be specified as an index into an indirect colour map, which contains actual Red, Green and Blue colour specifications for each colour. The length of the colour map and the number of colour maps available is terminal model dependent. The INTEGER parameter is interpreted with respect to the current size of the colour map specified in Boundary Value Definition. In order to accommodate the rules for accommodating differences in the colour value extent, as specified in ISO 8613, the INTEGER parameter is interpreted as a normalized fraction of the specified map lenght. Some terminal data syntaxes provide the capability of multiple colour maps. Multiple maps are logically equivalent to one large map encompassing a number of submaps. In the IDS, the use of several colour maps is handled by arbitrarily partitioning the single IDS colour map.

A.3.13.4 *Auxiliary colour selection*

```
Auxiliary—Colour—Selection ::= INTEGER
```

—— Auxiliary colour selection permits colours to be specified for the background of Text or Mosaics character cells. The operation of this command is similar to the Indexed Colour selection above, except that the current backgroun colour is established.

A.3.13.5 *Colour index setup*

```
Colour—Index—Setup ::= SEQUENCE { INTEGER, REAL, REAL, REAL }
```

—— The Colour Index setup command defines the contents of the colour map. The first parameter takes indexes into the colour map in a similar manner to the Indexed Colour Selection command. The remaining three parameters define the Red, Green and Blue colour values in a manner similar to the Direct Colour Specification command.

A.3.14 *Text colour string*

The manner in which text is presented and the specialized attributes and constraints which pertain to the presentation of texts differs between each of the terminal data syntaxes.

```
Text—Control—String  ::= CHOICE { [1] General—Text—Control,
                                  [2] Word—Wrap—Control }
```

A.3.14.1 *General text control*

```
General—Text—Control          ::= SEQUENCE { [1] General—Text—Control—Code,
                                     [2] G—Text—Par1 OPTIONAL,
                                     [3] G—Text—Par2 OPTIONAL,
                                     [4] Rel—Coord OPTIONAL,
                                     [5] Abs—Coord OPTIONAL }

General—Text—Control—Code ::= GRAPHICSTRING
```

—— General control function from Recommendation T.101 DS III [PDI G Set position 2/2, (10/2)].

—— *Note* — PDI G Set has final character 5/7 within DS III.

```
G—Text—Par1          ::= SET { [1] Char—Rotation OPTIONAL,
                               [2] IMPLICIT Char—Path OPTIONAL,
                               [3] Char—Spacing OPTIONAL,
                               [4] IMPLICIT Text—Precision OPTIONAL,
                               [5] IMPLICIT Char—Expansion—Factor OPTIONAL,
                               [6] Text—Colour—Index OPTIONAL,
                               [7] IMPLICIT Text—Alignment OPTIONAL }

Char—Rotation        ::= CHOICE { predefined [1] IMPLICIT INTEGER {
                                                 char—rotation—0      (0),
                                                 char—rotation—90     (1),
                                                 char—rotation—180    (2),
                                                 char—rotation—270    (3) }

                                  continuous [2] IMPLICIT SEQUENCE {

                                                 height—vector Abs—Coord,
                                                 width—vector Abs—Coord } }

Char—Path            ::= INTEGER { char—path—right        (0),
```

```
                                    char—path—left          (1),
                                    char—path—up            (2),
                                    char—path—down          (3) }

Char—Spacing        ::= CHOICE { predefined [1] IMPLICIT INTEGER {
                                                            char—spacing—1          (0),
                                                            char—spacing—5/4        (1),
                                                            char—spacing—3/2        (2) }
                                    continuous [2] IMPLICIT REAL }

Text—Precision      ::= INTEGER { string                    (0),
                                    char                    (1),
                                    stroke                  (2) }

Char—Expansion—Factor               ::= REAL

Text—Control—Index  ::= CHOICE { [1] IMPLICIT Basic—Colour—Selection,
                                    [2] IMPLICIT Indexed—Colour—Selection }

Text—Alignment      ::= SEQUENCE { Horizontal—Alignment,
                                    Vertical—Alignment }

Horizontal—Alignment ::= INTEGER { normal                   (0),
                                    left                    (1),
                                    centre                  (2),
                                    right                   (3) }
Vertical—Alignment  ::= INTEGER {normal                     (0),
                                    top                     (1),
                                    cap                     (2),
                                    half                    (3),
                                    base                    (4),
                                    bottom                  (5) }

G—Text—Par2         ::= SEQUENCE { INTEGER { cursor—style—underscore        (0),
                                             cursor—style—block             (1),
                                             cursor—style—cross—hair        (2),
                                             cursor—style—custom                (3)
}

                                    INTEGER{cursor—&—geometric—drawing—
                                             position—together              (0),
                                             cursor—leads—geometric—
                                             drawing—position               (1),
                                             geometric—drawing—position
                                             —leads—cursor                  (1),
                                             cursor—&—geometric—drawing—
                                             position—separate              (3) }

                                    INTEGER { char—interrow—spacing—1       (0),
                                             char—interrow—spacing—5/4          (1),
                                             char—interrow—spacing—3/2          (2),
                                             char—interrow—spacing—2        (3) }

                                    Char—Block—Dimension }
```

—— The relative coordinates define the size of the character field.

```
Char—Block—Dimensions ::= Rel—Coord
```

A.3.14.2 *Word wrap control*

The capability to wrap the presentation of characters on a word boundary rather than on a character boundary is available in one of the terminal data syntaxes. This capability cannot be directly converted to other data syntaxes; however, the effect can be achieved in the converter by issuing appropriate format effector characters.

```
Word—Wrap—Control   ::= INTEGER { Word—Wrap—On             (1),
                                    Word—Wrap—Off           (2) }
```

—— Word—Wrap—On is a function from Recommendation T.101 DS III [C1 set position 5/5, (9/5)].

—— Word—Wrap—Off is a function from Recommendation T.101 DS III [C1 set position 5/6, (9/6)].

A.3.15 *Photographic string synthetic image*

All of the terminal data syntaxes provide a method of handling an array of pixels. Some of the data syntaxes also provide general photographic capabilities which provide more efficient methods of encoding the same type of data. This means that interworking between all of the terminal data syntaxes is possible for photographic data, even though it may be inefficient in some cases.

Two classes of photographic images are identified below. They are the Synthetic and the Natural Image forms of photographic. The synthetic form of photographic corresponds to the photographic capabilities of data Syntax I. Natural Image photographic coding is for further study.

```
Photo—Graphic—String—Synthetic—Image ::= CHOICE { [1] Line—Dot—Pattern,
                                                   [2] Line—Dot—Pattern—Comp,
                                                   [3] Field—Dot—Pattern,
                                                   [4] Colouring—Block,
                                                   [5] Colouring—Block—Comp,
                                                   [6] Field—Colouring—Block,
                                                   [7] Field—Colouring—Block—Comp,
                                                   [8] Free—Format—Colouring—Block }
```

—— Photographic Synthetic Image functions correspond to Recommendation T.101 Data Syntax I. These functions are suitable for displaying synthetic images such as Kanji characters, graphics, etc.

A.3.15.1 *Line dot pattern*

```
Line—Dot—Pattern        ::= SEQUENCE { y—origin—point—coordinate—Idp Abs—Coord,
                                       dot—pattern—data—Idp BITSTRING }
```

—— Line—Dot—Pattern functions indicates a selection of two colours which are defined by a colouring Block, Field Colouring Block, etc. This function gives dot pattern data of one or several lines at a time.

A.3.15.2 *Line dot pattern comp*

```
Line—Dot—Pattern—Comp        ::= SEQUENCE { y—origin—point—coordinate—Idpc Abs—Coord,
                                            mh—run—length coded—data BITSTRING }
```

—— The Line—Dot—Pattern—Comp function is equivalent to the Line—Dot—Pattern function except that the dot patterns are encoded in a compressed manner using the M.H. Run Length Code.

A.3.15.3 *Field dot pattern*

```
Field—Dot—Pattern     ::= SEQUENCE { xy—origin—point—coordinate Abs—Coord,
                                     dx—dy—field—dimensions Rel—Coord,
                                     dot—pattern—data—fdp BITSTRING }
```

—— The Field—Dot—Pattern function is equivalent to the Line—Dot—Pattern function except that this function defines the dot pattern in a rectangular area.

A.3.15.4 *Colouring block*

```
Colouring—Block        ::= SEQUENCE { fg—bg—da—existence—indicator INTEGER,
                                      y—origin—point—coordinate—cb Abs—Coord,
                                SEQUENCE OF { SEQUENCE {
                                      fg—colour BITSTRING,
                                      bg—colour BITSTRING,
                                      display—attributes—cb BITSTRING } } }
```

—— The Colouring—Block function defines a photographic image by specifying the foregroung colour (FG), backgroung colour (BG), and display attributes of certain blocks ahead of which is indicated by the parameter y—origin—point—coordinate.

A.3.15.5 *Colouring block comp*

```
Colouring—Block—Comp        ::= SEQUENCE { colouring—block—comp—function—id INTEGER,
                                    fg—bg—da—existence—indicator—cbc INTEGER,
                                    y—origin—point—coordinate—cbs Abs—Coord,
                                SEQUENCE OF { SEQUENCE {
                                    fg—comp—colour BITSTRING,
                                    fg—runlength BITSTRING,
                                    bg—comp—colour BITSTRING,
                                    bg—runlength BITSTRING,
                                    display—attributes—cbc BITSTRING,
                                    da—runlength BITSTRING } } }
```

—— The Colouring—Block—Comp function is equivalent to that of the Colouring—Block function except that colour and display attributes data are encoded by compressed manner as run—length code.

A.3.15.6 *Field colouring block*

```
Field—Colouring—Block ::= SEQUENCE { field—colouring—block—function—id INTEGER,
                                    fg—bg—da—existence—indicator—fcb INTEGER,
                                    xy—origin—point—coordinate—fcb Abs—Coord,
                                    dx—dy—field—dimensions—fcb Rel—Coord,
                                SEQUENCE OF { SEQUENCE {
                                    fg—colour—fbc BITSTRING,
                                    bg—colour—fbc BITSTRING,
                                    display—attributes—fcb BITSTRING } } }
```

—— The Field—Colouring—Block function defines a photographic image by specifying the foregroung colour (FG), backgroung colour (BG), and display attributes of certain blocks which are contained in the the field allocated by xy—origin—point—coordinate and the dx—dy—field—dimensions.

A.3.15.7 *Field colouring block comp*

```
Field—Colouring—Block—Comp ::= SEQUENCE { field—colouring—block—comp—function—id INTEGER,
                                        fg—bg—da—existence—indicator—fcbc INTEGER,
                                        xy—origin—point—coordinate—fcbc Abs—Coord,
                                        dx—dy—field—dimensions—fcbc Rel—Coord,
                                    SEQUENCE OF { SEQUENCE {
                                        fg—colour—fcbc BITSTRING,
                                        fg—runlength—fcbc BITSTRING,
                                        bg—comp—colour—fcbc BITSTRING,
                                        bg—runlength—fcbc BITSTRING,
                                        display—attributes—fbc BITSTRING,
                                        da—runlength—fcbc BITSTRING } } }
```

—— The Field—Colouring—Block—Comp function is equivalent to that of the Field—Colouring—Block function except that colour and display attributes data are encoded by compressed manner as run—length code.

A.3.15.8 *Free format colouring block*

```
Free—Format—Colouring—Block ::= SEQUENCE { fg—bg—da—existence—indicator—ffcb INTEGER,
                                        fg—bg—da—code—length INTEGER,
                                        run—length—code—length—ffcb INTEGER,
                                        xy—origin—point—coordinate—ffcb Abs—Coord,
                                        dx—dy—field—dimensions—ffcb Rel—Coord,
                                    SEQUENCE OF { SEQUENCE {
                                        fg—colour—ffcb BITSTRING,
                                        runlength—ffcb BITSTRING,
                                        bg—comp—colour—ffcb BITSTRING,
                                        bg—runlength—ffcb BITSTRING,
                                        display—attributes—ffcb BITSTRING,
                                        da—runlength—ffcb BITSTRING } } }
```

—— The Free—Format—Colouring—Block function is equivalent to that of the Field—Colouring—Block—Comp function except that the code length of the Foreground, Background, Display Attributes and Run Lenght can be arbitrarily set.

A.3.16 *Photo graphic string natural image*

```
Photo—Graphic—String—Natural—Image ::= CHOICE { [0] IMPLICIT Header,
                                                 [1] IMPLICIT Transfer,
                                                 [2] IMPLICIT Table—Header,
                                                 [3] IMPLICIT Table—Transfer }

Header ::= SET {     [0] IMPLICIT Components OPTIONAL,

        CHOICE { [1] IMPLICIT Resolution OPTIONAL,
                 [2] IMPLICIT PixelPair OPTIONAL }
                 [3] IMPLICIT BitsPerDisplay OPTIONAL,
                 [4] IMPLICIT SamplingStructure OPTIONAL,
        CHOICE { [5] IMPLICIT Adpcm OPTIONAL,
                 [6] IMPLICIT Adct OPTIONAL } }

Components       ::= INTEGER { colorYU*V*     (0),
                               monochrome     (1) }

Resolution       ::= INTEGER { 4—2—2                (0),
                               2—1—1          (1) }

PixelPair        ::= SEQUENCE { PixHor, PixVer }

PixHor           ::= INTEGER
```

—— Number of horizontal pixels.

```
PixVer           ::= INTEGER
```

—— Number of vertical pixels.

```
BitsPerDisplay   ::= SEQUENCE OF INTEGER { 8 bits/pixel                  (0),
                                           1 bit/pixel                   (1),
                                           2 bits/pixel                  (2),
                                           . . . 9 bits/pixel            (9),. . . }
```

—— One value per component, gives the number or grey or colours a pixel may have.

```
SamplingStructure ::= SEQUENCE {
spatial          ::= SEQ { INTEGER { line and orthogonal                  (0),
                                     line and orthogonal field quincunx   (1),
                                     line quincunx field orthogonal       (2),
                                     line orthogonal single field         (3),
                                     line quincunx single field           (4) }

temporal         ::= SEQ { INTEGER { coincident                           (0),
                                     alternate samples                    (1),
                                     sequential line                      (2) } }

Adpcm            ::= SEQUENCE { INTEGER { Type dpcm                       (1) },
                                INTEGER { Subtype 1 dimension             (0) } }

Adct             ::= SEQUENCE { INTEGER { Type transform                  (2) },
                                INTEGER { Subtype Cosine                  (1) },
                                INTEGER { Subtype 2 dimension             (0) } }

Transfer         ::= SET { Origin, Area, Data }

Origin           ::= CHOICE { [0] IMPLICIT PixelPair OPTIONAL

Area             ::= CHOICE { [1] IMPLICIT PixelPair OPTIONAL

Data             ::= CHOICE { [2] IMPLICIT OCTETSTRING OPTIONAL,
```

—— Any value from 4/0 to 7/F.

[3] IMPLICIT OCTETSTRING OPTIONAL }

—— Transparent mode 8 bit/octet.

```
TableHeader     ::= SET { TableSet, TableSize }

TableSet        ::= [0] IMPLICIT SEQUENCE { type      ::= INTEGER,
                                            number    ::= INTEGER }

TableSize       ::= [1] IMPLICIT SEQUENCE { depth     ::= INTEGER,
                                            heigth    ::= INTEGER,
                                            width     ::= INTEGER OPTIONAL }

TableTransfer   ::= SET { TableSet, Position, Data }

Position        ::= TableSize
```

A.3.17 *Macro*

A Macro capability is available within the syntax of two of the three terminal data syntaxes. This capability permits strings of presentation data to be grouped together, so that it may be executed by the reference to a single command. In essence both terminal data syntax DS I and DS III provide the same Macro capability; however, a Macro in one data syntax cannot in general be converted to a Macro in another data syntax. This is because a Macro may contain any string of presentation data. Since the Terminal Models of the various data syntaxes differ, it is often necessary to sort the commands in the data stream in order to achieve the intended presentation effect. The arbitrary grouping of information into Macros prevents general sorting. Since the purpose of regular Macro functions are to achieve communications efficiency by eliminating the communication of repetitious code, it is possible to expand Macros in the conversion process. The conversion of a Macro is therefore the string of presentation data which it represents.

Two special forms of Macros in data syntax Ds I and DS III are Key Activated Macros and Transit Macros. Key Activated Macros link the execution of the Macro function to a local Key on the terminal. Since this operation depends upon the interaction of the user, the contents of the Macro cannot be expanded in the converter ahead of time. The converter must re—transmit the entire page of information to the terminal with the contents of the Key Activated Macro sorted and factored into the page. This problem must be handled by the Interworking Presentation Architecture. Similarly Transit Macro provides a problem in conversion. The contents of a Transit Macro must be sent back to the source upon a user interaction. In interworking this could mean that data syntax DS I data might be contained within a Transit Macro in a data syntax DS III terminal after a conversion so that it might be sent back to the source unchanged. It is necessary to be able to identify entire coding environments or to identify uniquely each code table in each Data Syntax in order to avoid confusion.

```
MACRO—String    ::= CHOICE { [1] Define—Macro,
                             [2] Define—and—Execute—Macro,
                             [3] Define—Transmit—Macro,
                             [4] Define—End—of—Macro—Definition,
                             [5] Macro—Invocation }
```

—— Key Activated Macros are Macros with reference numbers 0 to 7 in data syntax DS III.

A.3.17.1 *Define macro*

```
Define—Macro    ::= SEQUENCE { SID, INTEGER }
```

—— General control character (DEF MACRO) from Recommendation T.101 DS III [C1 set position 4/0, (8/0)] and (P—DEF MACRO) from DS I [C1 set position 5/5, (9/5) followed by parameter 4/0].

—— Integer number from 0 to 95 correspondig to the Macro reference number of the Macro being defined.

A.3.17.2 *Define and execute macro*

```
Define—and—Execute—Macro    ::= SEQUENCE { SID, INTEGER }
```

—— General control character (DEFP MACRO) from Recommendation T.101 DS III [C1 set position 4/1, (8/1)] and (P—DEFP MACRO) from DS I [C1 set position 5/5, (9/5) followed by parameter 4/1].

—— Integer number from 0 to 95 correspondig to the Macro reference number of the Macro being defined.

A.3.17.3 *Define transit macro*

```
Define—Transmit—Macro    ::= SEQUENCE { SID, INTEGER }
```

—— General control character (DEFT MACRO) from Recommendation T.101 DS III [C1 set position 4/2, (8/2)] and (P—DEFT MACRO) from DS I [C1 set position 5/5, (9/5) followed by parameter 4/2].

—— Integer number from 0 to 95 correspondig to the Macro reference number of the Macro being defined.

A.3.17.4 *Define end—of—macro definition*

Define—End—of—Macro—Definition ::= SID

—— General control character [END (Macro)] from Recommendation T.101 DS III [C1 set position 4/5, (8/5)] and (END MACRO) from DS I [C1 set position 5/5, (9/5) followed by parameter 4/15].

A.3.17.5 *Macro invocation*

Macro—Invocation ::= INTEGER

—— Integer number from 0 to 95 correspondig to the Macro reference number of the Macro being invoked.

—— *Note* — Macros may invoke other Macros at any time and to any depth.

A.3.18 DRCS string

The Dynamically Redefinable Character Set (DRCS) capability allows additional text or mosaic characters to be defined and used as regular alphanumeric text or mosaics. All three of the terminal data syntaxes include a form of DRCS capability; however, the operation of DRCS is quite different in the various Display Environments. In general it is not possible to convert exactly from one type of DRCS to another because of the boundary conditions imposed by each of the Terminal Data Syntaxes. Different limits exist on the number of DRCS characters which may be defined or the amount of memory which may be used to store DRCS characters. The definition of DRCS characters is a particular difficulty. One of the source terminal data syntaxes takes the approach of allowing any presentation information to be used in the definition of a DRCS character, including geometric drawing commands, bit (photographic) and text and even other DRCS characters. The other two source data syntaxes define DRCS characters using a bit oriented (photographic) approach. Even the two photographic approaches to the definition of DRCS are not equivalent since they have different pixel densities and serious quantization errors may result from mapping an array of pixels to another array of a different size. Three forms of DRCS definition are included in the Interworking Data Syntax to accommodate the requirements of the three source data syntaxes. The conversion process would therefore have sufficient information to make the best conversion possible.

```
DRCS—String          ::= CHOICE { [1] Define—DRCS—Type—I—1byte,
                                  [2] Define—DRCS—Type—I—2byte,
                                  [3] Define—DRCS—Type—II,
                                  [4] Define—DRCS—Type—III,
                                  [5] End—of—DRCS—Definition—Type—III,
                                  [6] DRCS—Invocation,
                                  [7] DRCS—Invocation—2byte }
```

A.3.18.1 *Define DRCS Type—I 1 byte*

```
Define—DRCS—Type—I—1byte ::= SEQUENCE { DRCS—I—Char—Size,
                                         DRCS—I—Code,
                                         DRCS—I—Data }

DRCS—I—Char—Size         ::= INTEGER { normal—size    (1),
                                       medium—size    (2),
                                       small—size     (3) }

DRCS—I—Code              ::= INTEGER
```

—— Integer number from 0 to 95 correspondig to the DRCS reference number of the 1 byte DRCS being invoked.

```
DRCS—I—Data              ::= BITSTRING
```

A.3.18.2 *Define DRCS Type—I 2 byte*

```
Define—DRCS—Type—I—2byte ::= SEQUENCE { DRCS—I—Char—Size,
                                         DRCS—I—Code,
                                         DRCS—I—Data }
```

—— This structure is the same as "Define—DRCS—Type—I—1byte" except that "DRCS—I—Code" is an integer number from 0 to 8835 correspondig to the DRCS reference number of the 2—byte DRCS being invoked.

A.3.18.3 *Define DRCS Type—II*

```
Define—DRCS—Type—II                ::= SEQUENCE { [1] IMPLICIT DRCS—Header OPTIONAL,
```

—— Description of general properties of the DRCS to be loaded. It is applied for all subsequent DRCS—pattern transfer units.

```
                                       [2] IMPLICIT DRCS—Pattern OPTIONAL }
```

—— Actual pattern data.

```
DRCS—Header                ::= SEQUENCE { Identification—of—Char—Set,
                                  Select—Dot—Composition }

Identification—of—Char—Set ::= SEQUENCE { repertory—info SET {
                                  repertory—# INTEGER { first repertory   (1),
                                                        second repertory  (2) },

                                  delete—existing—drcs BOOLEAN

Identification—of—Char—Set ::= SEQUENCE { registration—info CHOICE {
                                  iso—registration    [1] IMPLICIT GRAPHICSTRING,
                                  private—drcs—#      [2] IMPLICIT INTEGER } }

Select—Dot—Composition     ::= SEQUENCE { Character—Cell—Structure,
                                  Blocking—Factor,
                                  Pixel—Characteristics }

Character—Cell—Structure   ::= CHOICE { matrix—dimensions      [1] IMPLICIT SEQUENCE {
                                  horizontal                INTEGER,
                                  vertical                  INTEGER },
```

—— According to SDC Type 1.

```
                                  predefined—matrices [2] IMPLICIT INTEGER {
                                        n16*24      (0),
                                        n16*20      (1),
                                        n16*12      (2),
                                        n16*10      (3),
                                        n12*24      (4),
                                        n12*20      (5),
                                        n12*12      (6),
                                        n12*10      (7),
                                        n8*12       (8),
                                        n8*10       (9),
                                        n6*12       (10),
                                        n6*10       (11),
                                        n6*5        (12),
                                        n4*10       (13),
                                        n4*5        (14),
                                        n6*6        (15) } }
```

—— According to SDC Type—2.

```
Blocking—Factor    ::= SEQUENCE { horizontal INTEGER,
                                  vertical INTEGER }
```

—— Grouping of character cells, which are considered as a single character cell during character description.

```
Pixel—Characteristics ::= CHOICE { number of bits      [1] IMPLICIT INTEGER,
                                  predefined—numbers   [2] IMPLICIT INTEGER }

                                                          basic—DRCS      (1),
```

—— 1 bit/dot.

four—colour—DRCS (4),

—— Black, red, green, yellow from `Basic—Colour—Selection`.

eight—colour—DRCS (8),

—— First 8 colours from `Basic—Colour—Selection`.

sixteen—colour—DRCS (16) }

—— 16 redefinable colours.

—— This data type describes the pattern for the characters of the down—loaded DRCS, according the last transmitted header unit. It contains no compression for the pattern data. Data Syntax I and III have no similar encodings and this method can be used for an adequate mapping. All encoding different from the direct method and the codes for improvement of the efficiency (S—bytes) have to be transformed to the following description.

DRCS—Pattern ::= SEQUENCE { first character GRAPHICSTRING,

—— Code of the first character or character block

pattern—units SEQUENCE OF {
pattern—block—# SEQUENCE OF INTEGER,
pattern—block BIT STRING } }

—— Each pattern block contains one bit of each of the dots, starting from the top left hand corner, running row by row from left to right. The pattern block numbers are ordered from the least significant bit on. If the pattern block is preceded by two or more block numbers, the pattern block is applied to all of them. The block numbers are in the range of 0 to `pixel—characteristics`—1. The length of the pattern block equals to the number of pixels in the block.

A.3.18.4 *Define DRCS Type—III*

Define—DRCS—Type—III ::= INTEGER

—— A function from Recommendation T.101 DS III [C1 set position 4/3, (8/3)].

—— Integer number from 0 to 95 correspondig to the DRCS reference number of the DRCS character being defined to be followed by data string.

A.3.18.5 *End—of—DRCS definition Type—III*

End—of—DRCS—Definition—Type—III ::= GRAPHICSTRING

—— General control character [END (DRCS)] from Recommendation T.101 DS III [C1 set position 4/5, (8/5)].

A.3.18.6 *DRCS invocation*

DRCS—Invocation ::= INTEGER

—— Integer number from 0 to 95 correspondig to the DRCS reference number of the DRCS being invoked.

A.3.18.7 *DRCS invocation 2 byte*

DRCS—Invocation—2byte ::= INTEGER

—— Integer number from 0 to 8835 correspondig to the DRCS reference number of the 2—byte DRCS being invoked.

A.3.19 *Fill pattern control string*

The capability to fill a geometrically defined area with an arbitrary Fill Pattern, interior style, hatch or texture, is provided in two of the source videotex data syntaxes. Since one of the terminal Videotex data syntaxes, data syntax DS I, does not provide this capability it must be accommodated in the conversion process by assigning distinguishing colours or other means to indicate the difference between patterned areas. The method by which this capability is supported in the other two source data syntaxes is quite different. Data syntax DS III provides four predefined texture patterns, including solid fill, and four redefinable texture masks. These masks are rectilinear and are referenced to the origin of the normalized display area. This means that abutting areas filled with the same pattern will align perfectly. In Interior style patterns defined in data syntax DS II, the pattern may be defined on a parallelogram shaped area and is referenced to the origin of the area. Data syntax DS II also provides eight predefined fill patterns (hatch patterns). In general, any texture of interior style pattern may be simulated in the conversion process; however, secondary effects such as exact alignment of patterns cannot be guaranteed. Texture patterns in data syntax DS III are defined by including any string of presentation data in the definition of the pattern whereas interior styles defined in data syntax DS II are defined in terms of a cell array. The conversion process must resolve the pattern before the conversion. Limits to global variables, such as the available amount of texture memory, is defined by the boundary value condition indicators in the state vector.

Fill—Pattern—Control—String ::= CHOICE { [1] Define—Texture,
[2] End—of—Texture—Definition,
[3] Texture—Mask—Size,
[4] Set—Pattern—Representation,
[5] Pattern—Selection }

A.3.19.1 *Define texture*

Define—Texture ::= INTEGER

—— A function from Recommendation T.101 DS III [C1 set position 4/4, (8/4)].

—— Integer number from 4 to 7 correspondig to the redefinable Texture Mask to be defined. Note texture masks 0 to 3 are predefined and cannot be redefined to be followed by data string.

A.3.19.2 *End—of—texture definition*

End—of—Texture—Definition ::= GRAPHICSTRING

—— General control character [END (TEXTURE)] from Recommendation T.101 DS III [C1 set position 4/5, (8/5)].

A.3.19.3 *Texture mask size*

Texture—Mask—Size ::= Rel—Coord

—— Establish the texture mask size up to the limit defined by the boundary conditions.

—— A function from Recommendation T.101 DS III [C1 set position 2/3, (10/3)].

A.3.19.4 *Set pattern representation*

Set—Pattern—Representation ::= SEQUENCE { pattern—index INTEGER,

—— This number corresponds to the current pattern definition. It can be referenced by subsequent fill area style indices.

delta—x INTEGER,
delta—y INTEGER,

—— A grid of delta—x* delta—y* cells is specified. The colour of each cell is individually given by the

pattern—cell—data Pixel—Array—Data }

—— The colour array is associated with the cells as follows: the element (1, delta—y) is associated with the cell having the pattern reference point at one corner. Elements with increasing first dimension are associated with succesive cells in the direction of the pattern with vector; elements with decreasing second dimension are associated with succesive cells in the direction of the pattern height vector.

—— These definitions of patterns are from DS II and are applicable in conjugation with the fill area attributes defined by the data type "Display—Element—Attributes".

A.3.19.5 *Pattern selection*

Pattern—Selection ::= INTEGER

—— Integer number from 4 to 7 correspondig to the Texture Mask being selected.

A.3.20 *Music string*

The music capability is an option unique to only one of the terminal data syntaxes. It requires special capabilities for presentation and cannot be converted in any reasonable manner. Music information is included in the Interworking Data Syntax for future compatibility so that interworking may be accomplished between information from data syntax DS I and any future versions of data syntax DS II or III which might include a music capability.

Music—String ::= CHOICE { [1] Music—Code—Sequence,
[2] Music—Control—Sequence }

A.3.20.1 *Music code sequence*

Music—Code—Sequence ::= GRAPHICSTRING

—— Characters from Recommendation T.101 DS I [Musical Tone Set (pitch/duration)]. Note that the Musical Tone set is a two byte set which can be described as the combination of two one byte sets, one for duration and one for pitch. Reference is made to Recommendation T.101 since this code table has not yet been registered.

A.3.20.2 *Music code sequence*

Music—Control—Sequence ::= GRAPHICSTRING

—— Control characters from Recommendation T.101 DS I (Musical Control C1 Set). The Musical Control set contains the functions: Start Music Sequence, End Music Sequence, Start Melody Part, Start Rythm Part, End Part, Music Label, Jump to Part, Music Repeat, Music Branch, Sound Level, Change of Timbre, Long Duration Rest or Tone. Reference is made to Recommendation T.101 since this code table has not yet been registered.

A.3.21 *Telesoftware string*

Telesoftware—String ::= Further Study

A.3.22 *Audio data string*

Audio—Data—String ::= Further Study