A GKS VIRTUAL DEVICE INTERFACE META FILE SYSTEM

by

STEVEN W. TRACHSEL

B. S., The OHIO STATE University, 1979

A MASTER'S REPORT

submitted in parital fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

Approved by:

Major Professor

CONTENTS

List of Figures

1. Introduction

A graphics metafile is a format for storing the representation of graphics data on a physical media. The contents of a metafile is a list of commands which are used to generate a picture. Metafiles are useful in that they allow the results of graphics output to be stored in a format that is device independent, can be transmitted between different sites, and are not concerned with differences in hardware. The metafile also allows complex pictures to be reproduced without having to regenerate the picture from the orginal set of inputs.

The Virtual Device Metafile[1,2] is a paticular metafile first proposed within the the Graphics Kernel System[1]. A complete specification of the Virtual Device Metafile is defined in Annex E of the GKS specification. This specification is not a formal part of GKS, but is included as a suggestion for a metafile. The Virtual Device Metafile has since been proposed as an ANSI standard for metafiles[2].

The Virtual Device Interface (VDI)[3] is the proposed standard for the interface between a graphics package and physical devices, including metafiles. The objective of the VDI is to isolate the physical differences of different hardware devices and metafiles from higher level software

packages, by providing a consistent encoding scheme. This is an excellent idea. The current definition of what the valid VDI codes are, and what types of data should be represented in VDI, is still being developed.

The Graphics Kernel System is a specification for a two dimensional graphics package, that provides a decentralized control of each workstation. GKS provides the ability to delay the effect of an operation on any specific workstation by providing selective control over the current state of that individual workstation. This ability to delay the results on an individual GKS workstations also extends the concept of decentralized control[8,9,10]. The GKS specification does not include a capability for 3 dimensional work, nor does it contain a sophisticated segment definition and reference system.

The Metafile system described in this report functions as a workstation in the Graphics Kernel System (GKS)[1,7]. The scope of this report includes a description of an implementation of a part of the Metafile System within the Graphics Kernel System[1,2], as well as a comparison of the GKS Metafile System with the CORE Metafile Proposal [4], and the ANSI Metafile system. In section 2 the Virtual Device Metafile is defined as it relate to the Graphics Kernel System. Section 3 covers the design and implementation of a VDM subset. Section 4 compares the VDM with the metafile

system defined in the CORE standard[4]. In sections 4 GKS is analyzed, where both the strengths and weaknesses of GKS are discussed in the context of the Virtual Device Metafile. Section 5 contains the conclusions generated from this project. Section 6 is included as list of possible additions which could strengthen the Graphics Kernel System.

2. The Metafile System In GKS

In the GKS specification, the virtual device interface is between the central portion (or kernel) of the graphics package and any workstations or graphical devices. The intent of the metafile system is that any individual workstation could in reality be an online storage medium, which at some later date could be transmitted to another site, or re-used as input to the graphics system. The information that is stored is a device independent representation of a picture or series of pictures at the Virtual Device interface Level. This stored data could then be used to reproduce the graphical pictures. All that is required is that the person has access to a graphics package that understands VDI commands and can read standardized metafile formats. This storage of absolute data values in a VDI format is then called a Virtual Device Metafile(VDM).

There are many reasons for the development of the GKS metafile system. The major function of the metafile is to

provide a facility to transfer information between locations in an orderly way. The user must also be able to use, or preview the transferred data.

There are four major types of elements that are contained in the GKS Virtual Device Metafile:

1. Descriptive Elements- The descriptive elements define and describe the contents of the metafile. This could be compared to a table of contents in a written paper.

2. Formating Elements- The formating elements define which of the various format choices were used in the creation of this metafile. This data is then used to translate the physical metafile representation of the graphical and application element types into the virtual device interface formats.

3. Graphical Elements- The graphical elements are the actual VDI primitive representations.

4. Application Elements- The Application elements are user specific information which is not in a VDI standard format. In many cases this data may be of a format specific to a particular graphics package implementation, and as such would be un-useable to many of the "standard" implementations of the metafile system.

The GKS specification of the metafile allows for the off-line storage of VDI command types, but also includes a slightly different view of the metafile. The basic premise seems changed such that the metafile will be used for the off-line storage of segments and internal graphical states, as well as VDI command types. The unrestricted mixing of the two drastically different basic types of graphical data (internal graphical states and segments vs. bound VDI attributes) present some unsettling issues for both the implementors and users of metafiles. These issue are discussed in detail in later sections.

3. The Design of The Implementation

The portion of the Graphics Kernel System Annex E metafile definition [3] that has been implemented, directly corresponds to the intersection of the GKS Annex E definition and the CORE proposal for Metafiles[4,5] (see section 3.1). The portion of the Definition that deals with the offline storage of segments and the corresponding state information has not been included.

In the implementation that accompanies this report(Appendix 3), the portion of the GKS metafile system that deals with bound VDI commands was implemented. What follows is a high level discussion of the design and philosophy that governed the implementation. A portion of the structure charts that

were used to develop the design are included in Appendix 1. These diagrams represent at a high level the logical structure of the Virtual Device Metafile system in the K.S.U. PASCAL implementation of the Graphics Kernel System. Appendix 2 contains a break down of a sample metafile which was generated using the implemented metafile system. This file contains all of the required definition information, plus a short listing of VDI commands in the VDM format, followed by the trailer item. Appendix 3 contains a listing of the PASCAL source code which makes up the Virtual Device Metafile subsystem.

In order to fulfill the GKS specifications the user must be able to perform operations on metafiles files in the same manner as other workstations. All GKS workstation states would have the same effect on the metafile workstation as on any other user controlled workstation. Delayed output of graphical primitives would also be delayed in being output to the metafile in a manner identical to the methods used to delay output to user controlled workstations. The only restriction on the metafiles verses any other type of workstation is that a metafile can not be open for reading and writing at the same time. However the metafile can be repeatedly written to, closed and opened for reading and then closed and opened for writing again, giving the user a sense of a read-write metafile, with the exception that no

updates or deletions may made, only additions. The sense of updates and deletes can be transferred to the user in that the metafile which was totally rewritten would have the data that was transformed in an update or delete operation.

The specification of internal graphical states, segments, and table indices were excluded as being outside the realm of a VDI level metafile system. The GKS metafile specification also includes the capability for a user to include non-standard VDI data in a standard format. There is the question (created by the CORE system precedent for metafiles) of whether the metafile system is the correct place for the storing of this type of information. This question is addressed in greater detail in section 3.1.

3.1 **Details of the Design**

The metafile subsystem can be divided into two major portions, the input metafile driver(See Appendix 1 page 25), and the output metafile driver(See Appendix 1 page 26). Both drivers have multiple entry points which allow the graphics package to perform operations such as: opening a metafile, closing a metafile, activating a metafile workstation, deactivating a metafile workstation, adding items to a metafile, and reading items from the input metafile(Routines in Appendix 3 on pages 31 - 33, 38&39, 40, 42-46, and 47-51).

3.1.1 The Output Metafile Driver The output metafile driver is a reentrant driver, in that the metafile that is targeted for the operation is included as one of the parameters to the driver(The mofile parameter of type text in the metaout routine in Appendix 3, page 47). Using this design, the metafile subsystem can support an unlimited number of workstations configured as output metafiles. The data which is entered into the metafile, is transmitted to the output metafile driver in the form of VDI commands(The metaout routine in Appendix 3 on page 47). These VDI commands are then translated into VDM format and written to the output medium.

3.1.2 The Input Metafile Driver The input metafile driver requires that a format definition table be available for each metafile that is open for input(of type metafilehdr, Appendix 3, page 28). The current implementation allows for only one such definition table, thus restricting the graphics system to having only one metafile opened for input. A simple addition to the system would be to create the definition table for each metafile workstation when it is opened, and then pass the definition table as an argument This would be done in the initmetain routine(Appendix 3, page 40). The input metafile driver would then be reentrant, and able to support an unlimited number of workstations configured as input metafiles.

3.2 Net Results of the Implementation Effort

The source files for the implemented metafile system total some 864 lines of PASCAL source code. All of the VDI commands that are supported by the K.S.U. Graphics Kernel System have been included in the metafile system(see figure 1).

Mapping of K.S.U. primitives to GKS standard

K.S.U. primitive name	GSX primitive number	GSM primitive number
vdiclose:	(* gsx #2	, gsm #0 *)
vdiclear:	(* gsx #3	, gsm #1 *)
vdimsg:	(* no gsx	, gsm #5 *)
vdiline:	(* gsx #6	, gsm #11 *)
vdimark:	(* gsx #7	, gsm #12 *)
vditext:	(* gsx #8	, gsm #13 *)
vdiarea:	(* gsx #9	, gsm #14 *)
vdigdp:	(* gsx #11	, gsm #16 *)
vdilinetype:	(* gsx #15	, gsm #22 *)
vdilinescale:	(* gsx #16	, gsm #23 *)
vdilinecolor:	(* gsx #17	, gsm #24 *)
vdimarktype:	(* gsx #18	, gsm #26 *)
vdimarkscale:	(* gsx #19	, gsm #27 *)
vdimarkcolor:	(* gsx #20	, gsm #28 *)
vditextfont:	(* gsx #21	, gsm #30 *)
vditextprec:	(* no gsx	,gsm #30,now 29*)
vditextcolor:	(* gsx #22	, gsm #33 *)
vdichscale:	(* gsx #12	, gsm #31 *)
vdicvector:	(* skip gsx	, gsm #34 *)
vditextspace:	(* no gsx	, gsm #32 *)
vditextpath:	(* no gsx	, gsm #35 *)
vditextalign:	(* no gsx	, gsm #36 *)

Figure 1

This metafile file system has also adopted the definition of the VDI level interface definition that has been developed at K.S.U. This could lead to problems in the future when a

standardized VDI is approved, since the standard will surely be different from previously published VDI specifications. Since standardization problems and format changes will occur, the metafile system has been designed to localize the impact of these change to metafile system itself. This has been done by isolating the areas of the metafile system that deal with the individual elements of the VDM format(The rdmdata and metaout routines in Appendix 3 on pages 42, and 47). This then provides a single location of change when some element of the Virtual Device Metafile format changes. The K.S.U. metafile system can then be used as a translator between sites using a new VDI standard and the local graphics package.

The current implementation will create metafile output as an active workstation. Multiple metafiles may be defined by the graphics package as being opened for output. The input portion of the metafile system may have one metafile open for input at a time. Both portions of the metafile, input and output, have been tested and shown to work with the version of the K.S.U. graphics package that was running at the time of implementation.

4. Current State Of The Metafile standards

In this section an analysis of the GKS metafile standard is presented. The major focus will include pointing out some inconsistencies in the standard, some possible drawbacks owing to the details of the specification, and a short description of how the users view of a graphics system might be impacted by the metafile system.

4.1 A Comparison of The CORE Metafile system to The GKS Metafile system

In the CORE system GSPC metafile proposal [4], the metafile system is defined to have the following functions:

1. Be used to transfer "pictures" between two sites.
2. Act as an audit trail of picture development.
3. Make hard-copies of pictures designed in interactive sessions.
4. Act as an archive.
5. Be a tool for picture verification.
6. Serve as an interface standard.

Also in the picture is a ANSI version of GKS virtual device Metafile definition[2]. This proposal maps very neatly in concept to the GKS Annex E proposal, only differing in minor

syntactical areas. For the purpose of this paper the ANSI version of GKS and the formal GKS proposal will be considered to encompass the same goals and principals. However some of the difference between the CORE metafile proposal and the GKS metafile proposal are due to the different philosophical base on which they founded. In the CORE proposal the metafile itself is the device standard, while in GKS the VDI is the standard, with the Virtual Device Metafile only being a standardized method of storing VDI commands.

In the CORE metafile system the idea of having any segment data is strictly forbidden. The CORE metafile is only to serve as a semi-permanent storage of device and graphics package independent information. Under the GKS specification the metafile system is designed to perform the 6 functions that were listed in the CORE systems GSPC metafile proposal. In addition the concept of segment information being stored in the metafile is introduced.

4.1.1 The Suitability of Storing Bound Images with Segment Storage In GKS the idea of putting the segment information into the metafile system comes from the concept of not having an explicit method of storing any segments, or general status information from workstation session to another. In order to come up with a reasonably portable method with-in GKS type environments the metafile system was

chosen.

The concept of having an implementation independent segment store in GKS is highly desirable. However it seems a bit unwieldy to combine that concept with the virtual device independent data that is stored in the "traditional" metafile system. By combining the two, the graphics package transparency that was present in the CORE GSPC proposal is lost. The user is now limited in the scope of where they can transfer information. For example, no longer can the user send a picture to a system that does not have a graphics package, but which understands VDI, without risking the occurrence of segment defining attributes being in the metafile. The security of the segment storage is also compromised under the GKS proposal. In many instances a finished picture may be the only data that the user wants to transfer. The individual components that comprise the picture many be of separate value, and need to be safeguarded. Since the GKS metafile combines the two elements in one storage medium, the user may have to go to unreasonable lengths to separate the two into separate metafiles.

4.1.2 Metafile Formats In the standard definition of the Virtual Device Metafile there exists a multitude of different formats that may be used to specify the same information. The implementor of a metafile system in a

graphics package has virtually a free hand in how the data is to be represented. The formats vary from different binary permutations, to partial binary and ASCII representations, to all ASCII (or EBCDIC) representations.

While this extreme variability of formats may at first seem to be a boon to the user community, it places an unreasonable expense on the development of a graphics package, when a smaller subset could represent the same information without losing any flexibility or site independence. Multiple formats increase the size and complexity of a graphics package, a price the user eventually has to pay. When a language such as PASCAL is used an additional burden is placed on the graphics system, in that the package itself must be overly concerned with the external representation of the data in order to avoid type checking errors. It would be ideal for the metafile to be configured such that the natural processing facilities of the language being used could handle the necessary conversions. This proliferation of formats is the major short coming of the GKS Virtual Device Metafile System.

5. General Analysis of The GKS Virtual Device Metafile

The GKS Virtual Device Metafile System presents a complete package of passive online storage facilities to the user. The ability to add data in a device independent

format in a timely fashion, as well as the ability to display portions of a metafile on command, are powerful tools. These powerful tools present the implementor of a graphics package with some interesting problems that must be solved in a reasonable manner in order for a device independent metafile to be a significant advantage for the user.

The specification of the definition of a metafile contains just about every conceivable combination of formats and data types imaginable. There has been an effort to provide enough general data types to please everyone involved in the specification work. While this generality provides a very versatile tool, it also greatly increases the complexity and expense of the software which implements the metafile system. This expense of variability contradicts the standard ideal, in that no generalized subset exists that provides the necessary functionality. Instead the result has been that all formats have been provided for within a specific framework.

The final factor one must consider when evaluating the GKS virtual device metafile system, is the question of whether the metafile can do what it was intended to. In this respect one can only conclude that the metafile system is a success, and a significant step towards the definition and acceptance of a graphic standard.

5.1 The General Usefulness of The GKS Virtual Device Metafile

The usefulness of having a capability for transferring images between different locations has been proven by graphics systems that have come before GKS[6,11]. The major concern of the user in the GKS virtual device metafile system is for the provision of a tool which can be used as an active tool in the creative process of computer generated graphics, as well as a means for storing final results. In this context the metafile itself seems to have exceeded the capabilities of the Graphics Kernel System itself.

5.2 GKS in an Interactive Environment

Perhaps the major short coming in the GKS metafile proposal, is the non-support of the interactive environment[13,14,15,16]. The metafile makes no distinction between the users responses to solicited input that does not directly map to the normalized data which is eventually displayed on the screen. A good example would be the data representation of an image generated by a light pen, or other similar device. The resulting stream of point definitions is not easily represented in a metafile. Another major problem is the inability to edit the metafile. As a user interacts with a system mistakes are made and corrected. It is usually not the wish of the user for the

mistake to remain as a part of the permanent record of the graphics session.

5.3 The Virtual Device Interface and a Metafile System

The VDI specification determines what level of abstraction is used in the Virtual Device Metafile. That level has been determined to be the point at which hardware dependent levels lie directly below the current level. For the metafile file this creates the opportunity to devise a system that can be independent from the display devices which will be used, but that can also be independent from the graphics system which was used to create the image.

6. Conclusions

The Virtual Device Metafile, when viewed in the context of the Graphics Kernel system, provides a working set of good features. The Virtual Device Metafile follows as a logical extension of the VDI specification. The Virtual Device Metafile needs to make a clear distinction between VDI type commands, and the abstract relations of segments, and other workstation independent information. An entirely different form of metafile, possibly with the same physical structure, should be defined to store segment data. The Virtual Device Metafile specification should be adopted with 2 exceptions. The multiple variations in formats should be

restricted to a reasonable subset that implements the features needed but removes duplication. The other restriction is to insure that the idea of using a metafile for segment storage is not present in a GKS VDM specification.

The VDI specification solves the current problem of the proliferation of different types of graphics hardware. A standardized interface now would greatly reduce the difficulty of implementing a standard metafile. The Virtual Device Interface definition fills the immediate need in this area, and provides a firm base for future development.

The utility of the Virtual Device Metafile system under the Graphics Kernel System is closely tied to the characteristics of workstations in the Graphics Kernel System. Since a metafile is considered a workstation under the functional description of GKS, all of the benefits and restrictions of the workstation concept in GKS apply to the Virtual Device Metafile. To complete a standard in a reasonable time period GKS is designed to function in only 2 dimensions. This restriction reduces the utility of GKS. GKS is written in a way which requires every user to have detailed knowledge of the internals of the system. It is very difficult for a user to maintain only high level knowledge of the system and still be able to utilize most of the functionality of Virtual Device Metafile[12]. This fact

becomes even more apparent when the metafile from GKS is compared to the CORE GSPC metafile proposal. In the GSPC proposal the user is isolated by a layered software approach, which removes the requirement that the user know the details of the lower levels of the software.

7. Possible additions to the Graphics Kernel System Standard

What follows are a few ideas that in my opinion would greatly increase the utility of GKS, and the metafile system, and remove many of the possible obstacles that may slow their acceptance as the graphics standard.

GKS needs to have the concept of 3 dimension returned to it. This is essential because of the growing use and sophistication of computer generated images. It also needs to clean up the interface to the user, providing a higher level of abstraction to the user, and reducing the amount of needless detail that user must concern themselves with. The new user interface should be concerned with both the productivity increases that business is currently demanding, as well as with the concerns of user groups which have to use with systems on a day to day basis. A great improvement would be to include features that are designed to increase the users power over segments. This would enhance the utility of GKS in interactive environments, such as CAD/CAM,

and animation.

REFERENCES

[1] Graphics Kernel System (GKS) Functional Description, Draft International Standard ISO/DIS 7942, Nov. 14, 1982.

[2] Draft Proposed American National Standard for the Virtual Device Metafile, Aug. 13, 1982.

[3] Proposal for an ANSI X3 Standards Project for the Computer Graphics Virtual Device Interface, Aug. 27, 1982.

[4] "The GSPC Metafile Proposal", Status Report of the Graphic Standards Planning Committee, Computer Graphics, Volume 13, Number 3, August 1979.

[5] "A Response to the 1977 GSPC Core Graphics System", Deborah u. Cahn, te. al., Computer Graphics, Vol. 13, No. 2, Aug., 1979, pp. 57-62.

[6] "GPGS A Device-Independent General Purpose Graphic System for Stand-alone and Satellite Graphics", L. C. Caruthers, et. al., pp. 112-119.

[7] "The GKS Impact on Graphics Standardization", Peter R. Bono, Computer Graphics World, Vol. 5, No. 9, Sept., 1982, p. 47.

[8] Principles of Interactive Computer Graphics Robert F. Sproull, 2nd Ed., 1979, McGraw-Hill, Inc., New York. Chap

27: Device-Independent Graphics Systems

[9] Fundamentals of Interactive Computer Graphics Andries Van Dam, 1982, Addison-Wesley Publishing Co., Reading, Mass.

[10] "The Workstation Concept of GKS and the Resulting Conceptional Differences to the GSPC Core System", J. Encarnacao, et. al., Proceedings of SIGGRAPH'80 in Computer Graphics, Vol. 14, No. 2, July 1980, pp. 226-230.

[11] "A Metafile for Efficient Sequential and Random Display of Graphics", Theodore N. Reed, Proceedings of SIGGRAPH'82 in Computer Graphics, Vol. 16, No. 3, July, 1982, pp. 39-43.

[12] "Some Criticisms of the Graphics Kernel System (GKS)", P. Buttuls, Computer Graphics, Vol. 15, No. 4, Dec., 1981, pp. 302-305.

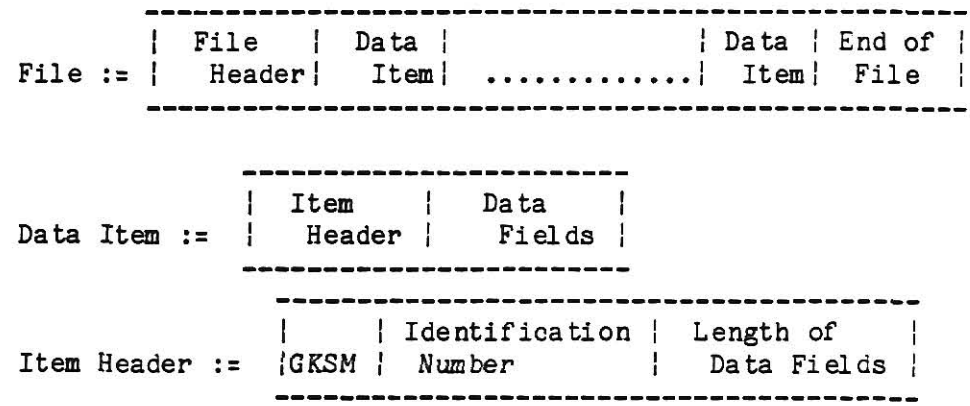
[13] "Device Independent and Decentralized Graphic System", Weiss Guttmann, Computer Graphics, Vol. 13, No. 4, Feb., 1980, pp. 288-302.

[14] "The Graphical Kernel System (GKS). The Standard for Computer Graphics Proposed by the German Institute for Standardization (DIN)".

[15] "Implementing Standard Device-Independent Graphics", James R. Warner and Nikolaus J. Kiefhaber, Mini-Micro Systems, July, 1982, pp. 201-208.

[16] "Implementation of the Core Graphics System GKS in a Distributed Graphics Environment", Peter Wisskirchen, et. al., Proceedings of the International Conference on Interactive Techniques in Computer Aided Design, 1978, pp. 249-254.

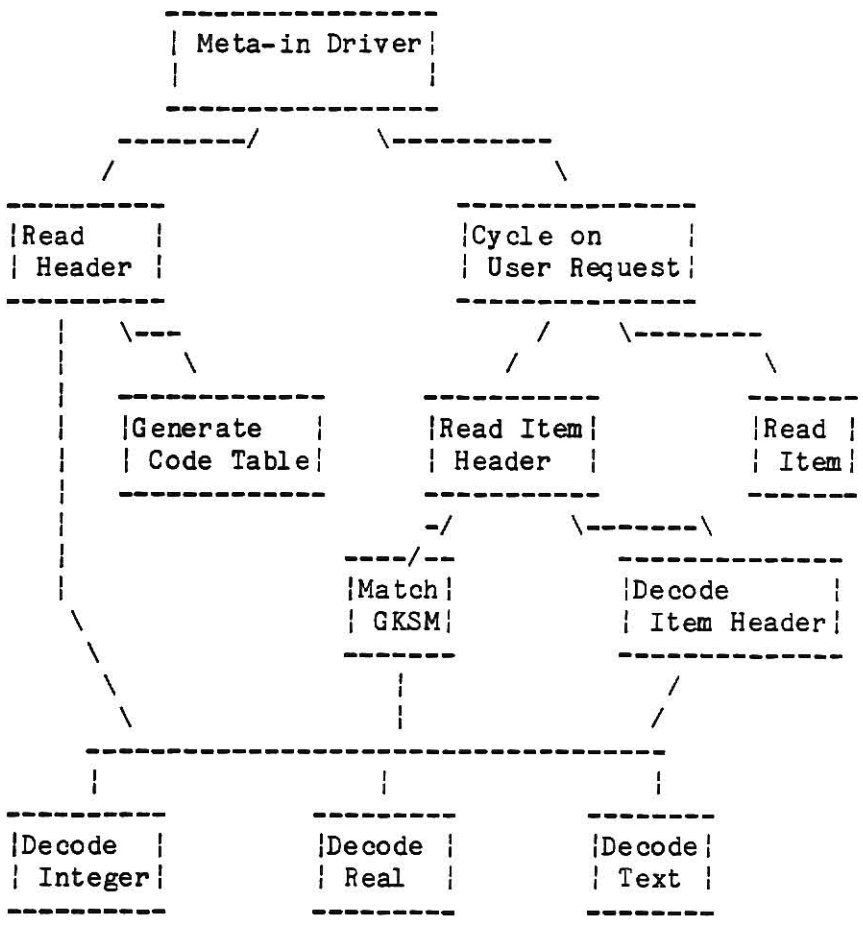
Appendix 1



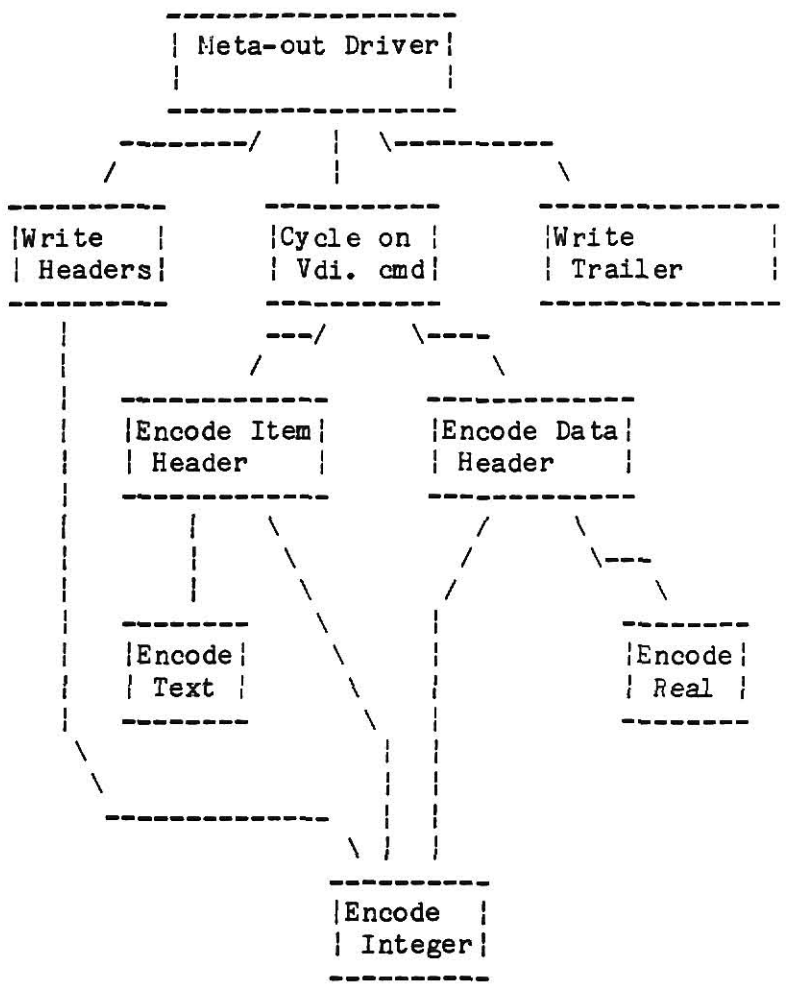
Format of a Metafile

This figure specifies the logical layout of the GKS Virtual Device Metafile. The complete specification of all of the data types can be found on pages 259-268 of the GKS Specification Annex E[1]. The metafile is decomposed into the header, a repeating field which is the data item, and the end of file marker. The header is defined in Appendix 3 on page 29 as the PASCAL type metafilehdr.

The data item consists of the item header and then variable fields which are determined by the item header. The item header is read in the rdmtype routine(Appendix 3, page 41). The rdmdata routine reads in the data items(Appendix 3 page 42). In the vdimsg case the length of the message is read in. Using this value the contents of the message is read into the vdiparm record to be returned to the user.



Structure Chart for the Meta-in Driver



Structure Chart for Meta-out Driver

Appendix 2
Character dump of a Metafile

```
byte     |          Character values
address  |
-------------------------------------------------------------
                                  /* header information     */
0000000          GKSMKSU_GKS_GRAPHICS_PACKAGE_META_FILE_____

0000054          yy/mm/dd          /* date information       */
0000064          010202000306      /* file representation    */
0000100          010200            /* definitions            */
0000106          0000000000
0000120          0000032000

0000132          GK05041          /* message item, len=41   */
0000141          hello~there~meta~file~~~~~~~~3~~~~~~~~~4~

0000212          GK05041          /* message item, len=41   */
0000221          hello~there~meta~file~~~~~~~~3~~~~~~~~~4~

0000275          GK11             /* polyline primitive     */
0000301          002001002003005  /* point definitions      */

0000320          GK00             /* end of file item       */
```

Appendix 3

```
(*
 *
 *
 *
 *        METAFILE SUBSYSTEM constant definitions
 *
 *        Written by Steve Trachsel,
 *                   Summer 1983.
 *
 *
 *
 *)
;const  cversion = 1
        ;gksmdef = 'GKSM'
        ;mheadercont = 'KSU GKS GRAPHICS PACKAGE META FILE       '
        ;mdatacont = 'yy/mm/dd'
        ;mversion = '01'
        ;mgklen = '02'
        ;vmgklen = 2
        ;mitlen = '02'
        ;vitlen = 2
        ;mdatlen = '00'
        ;vdatlen =  0
        ;mintlen = '03'
        ;vintlen = 3
        ;mrealen = '06'
        ;vrealen = 6
        ;mintype = '01'
        ;vintype = 1
        ;mrtype = '02'
        ;vrtype = 2
        ;mzero = '00000000000'
        ;vzero = 0
        ;mone = '00000032000'
        ;vone = 32000
        ;headcont
        ='GKSMKSU GKS GRAPHICS PACKAGE META FILE      yy/mm/dd'
        ;headcont1 = '0102020003060102000000000000000000032000'

;type
        metafilehdr =
        record
                gksstr : array [1..4] of char
                ; strcomment : array [ 1..40] of char
                ; date : array [1..8] of char
                ; version : integer
                ; lengksm : integer
                ; lopcode : integer
                ; lenbytesfld : integer
                ; lenint : integer
```

```
        ; lenreal : integer
        ; decrbin : integer
        ; roiforreal : integer
        ; zero : integer
        ; one : integer
end (* record metafilehdr *)

;wsname = 1..30
```

```
(*
 *
 *
 *
 *        METAFILE SUBSYSTEM Variable definitions
 *
 *        Written by Steve Trachsel,
 *                   Summer 1983.
 *
 *
 *
 *)
        ; metahdr : metafilehdr
        ; opcode : vdicmdtype
        ; linbytes : integer
        ; in_init : boolean
```

```
(*****************************************************************
 *                                                               *
 *                                                               *
 *                    METAFILE SOURCE CODE                       *
 *                 WRITTEN BY STEVEN W. TRACHSEL                 *
 *              FOR MASTERS PROJECT IMPLEMENTATION               *
 *                    SUMMER SESSION 1983                        *
 *                                                               *
 *                                                               *
 *                                                               *
 *                                                               *
 *****************************************************************)
(*
 *
 *
 *
 *        METAFILE SUBSYSTEM procedure : initmetaout
 *
 *        Written by Steve Trachsel,
 *                   Summer 1983.
 *
 *
 *
 *)
; procedure initmetaout (var ifile : text )
;var hdr :array [1..52] of char
        ;hdr1 :array [1..38] of char
        ;x : integer
;begin
        hdr := headcont
        ; for x:=1 to 52
          do
                writechar( ifile,hdr[x])
        ; hdr1 := headcont1
        ; for x:=1 to 38
          do
                writechar( ifile,hdr1[x])
end
```

```
(*
 *
 *
 *
 *        METAFILE SUBSYSTEM procedure : closemetaout
 *
 *        Written by Steve Trachsel,
 *                   Summer 1983.
 *
 *
 *
 *)
;procedure closemetaout(var ifile :text)
;var        x:integer
       ;chr :array [1..4] of char
;begin
       chr := 'GK00'
       ; for x:=1 to 4 do
              writechar(ifile, chr[x])
end
```

```
(*
 *
 *
 *
 *        METAFILE SUBSYSTEM procedure : closemetain
 *
 *        Written by Steve Trachsel,
 *                   Summer 1983.
 *
 *
 *
 *)
;procedure closemetain(var ifile :text)
;begin
       in_init := FALSE
end
```

```
(*
 *
 *
 *
 *        METAFILE SUBSYSTEM procedure : fwrite
 *
 *        Written by Steve Trachsel,
 *                   Summer 1983.
 *
 *
 *
 *)
;procedure fwrite(var ifile: text; value : integer;
                  format : integer)
;var str: array[1..11] of char
        ;count,temp : integer
        ;temvalue        : integer
;begin
        temvalue := value
        ;for count := 11 downto (12-format)
        do begin
                str[count] := chr((temvalue mod 10)
                                  + ord('0'))
                ; temvalue := temvalue div 10
        end
        ;temp := 12 - format
        ;for count := temp to 11
        do begin
                writechar(ifile,str[count])
        end
end
```

```
(*
 *
 *
 *
 *        METAFILE SUBSYSTEM procedure : int_real
 *
 *        Written by Steve Trachsel,
 *                   Summer 1983.
 *
 *
 *
 *)
;procedure int_real( intvalue:integer; var result :real;
                         zero:integer; one :integer)
;var inr : real
        ; zr: real
        ;onr :real
;begin
        inr := conv(intvalue)
        ;zr := conv(zero)
        ;onr := conv(one)
        ;result := (inr - zr) / onr
end
```

```
(*
 *
 *
 *
 *        METAFILE SUBSYSTEM procedure : real_int
 *
 *        Written by Steve Trachsel,
 *                   Summer 1983.
 *
 *
 *
 *)
;procedure real_int(var int:integer; result :real;
                        zero:integer; one :integer)
;var tem : real
        ; zr: real
        ;onr :real
;begin
         zr := conv(zero)
        ;onr := conv(one)
        ;tem := (result * onr) + zr
        ;int := trunc(tem)
end
```

```
(*
 *
 *
 *
 *        METAFILE SUBSYSTEM procedure : fread
 *
 *        Written by Steve Trachsel,
 *                   Summer 1983.
 *
 *
 *
 *)
;procedure fread( var ifile : text; var value: integer;
                  format : integer; var b_errcode : boolean)
;var
        count : integer
        ;power : integer
        ;string : array [1..11] of char
;begin
        value:= 0
        ;power:= 1
        ;b_errcode:= FALSE
        ;if (format <> 0) and ( in_init = TRUE ) then
        begin
                ;for count:= 1 to format
                do
                        if not eof(ifile) then
                                begin
                                        readchar(ifile,
                                          string[count])
                                end
                ;for count := format downto 1
                do begin
                        value := value +
                                (ord(string[count])-
                                ord('0')) * power
                         ;power := power *10
                end
        end
end
```

```
(*
 *
 *
 *
 *        METAFILE SUBSYSTEM procedure : vdiread
 *
 *        Written by Steve Trachsel,
 *                   Summer 1983.
 *
 *
 *
 *)
;procedure vdiread( var ifile : text; var value: vdicmdtype;
                    format : integer; var b_errcode : boolean)
;var        temp :integer
        ;power : integer
        ;string : array [1..11] of char
;begin
        temp:= 0
        ;power:= 1
        ;b_errcode:= FALSE
        ;fread ( ifile, temp, format, b_errcode )
        ;case temp of

                0: begin
                             value := vdiclose
                             end(* gsx #2 , gsm #0 *)
                ;5: begin
                             value := vdimsg
                             end(* no gsx, gsm #5 *)
                ;11: begin
                             value := vdiline
                             end(* gsx #6, gsm #11 *)
                ;12: begin
                             value := vdimark
                             end(* gsx #7, gsm #12 *)
                ;13: begin
                             value := vditext
                             end(* gsx #8, gsm #13 *)
                ;16 : begin
                             value :=vdigdp
                             end(* gsx #11, gsm #16 *)
                ;22: begin
                             value :=vdilinetype
                             end(* gsx #15, gsm #22 *)
                ;23: begin
                             value :=vdilinescale
                             end(* gsx #16, gsm #23 *)
                ;24: begin
                             value :=vdilinecolor
                             end(* gsx #17, gsm #24 *)
```

```
                ;26: begin
                                value :=vdimarktype
                                end(* gsx #18, gsm #26 *)
                ;27: begin
                                value :=vdimarkscale
                                end(* gsx #19, gsm #27 *)
                ;28: begin
                                value :=vdimarkcolor
                                end(* gsx #20, gsm #28 *)
                ;29: begin
                                value :=vditextfont
                                end(*gsx #21,gsm for now 29*)
                ;30: begin
                                value :=vditextprec
                                end(*no gsx , gsm #30 *)
                ;33: begin
                                value :=vditextcolor
                                end(* gsx #22, gsm #33 *)
                ;31: begin
                                value :=vdichscale
                                end(* gsx #12, gsm #31 *)
                ;34: begin
                                value :=vdicvector
                                end(* skip gsx, gsm #34 *)
                ;32: begin
                                value :=vditextspace
                                end(* no gsx , gsm #32 *)
                ;35: begin
                                value :=vditextpath
                                end(* no gsx , gsm #35 *)
                ;36: begin
                                value :=vditextalign
                                end(*no gsx, gsm #36 *)
                ;else: begin
                      end
        end
end
```

```
(*
 *
 *
 *
 *        METAFILE SUBSYSTEM procedure : initmetain
 *
 *        Written by Steve Trachsel,
 *                   Summer 1983.
 *
 *
 *
 *)
; procedure initmetain
        ; var
                count : integer
                ; b_errcode : boolean
; begin (* initmetain *)
        in_init := true
        ;with metahdr do
        begin (* read commands and issue gks calls *)
                gksstr := '    '
                ; for count := 1 to 4
                        do
                            readchar(mifile,gksstr[count])
                ; for count := 1 to 40
                        do
                          readchar(mifile,strcomment[count])
                ;for count := 1 to 8
                        do
                            readchar( mifile, date[count])
                ; count:=2
                ; fread (mifile,version,count,b_errcode)
                ; fread (mifile,lengksm,count,b_errcode)
                ; fread (mifile,lopcode,count,b_errcode)
                ; fread (mifile,lenbytesfld,count,b_errcode)
                ; fread (mifile,lenint,count,b_errcode)
                ; fread (mifile,lenreal,count,b_errcode)
                ; fread (mifile,decrbin,count,b_errcode)
                ; fread (mifile,roiforreal,count,b_errcode)
                ; count:=11
                ; fread (mifile,zero,count,b_errcode)
                ; fread (mifile,one,count,b_errcode)
        end (* with *)
end
```

```
(*
 *
 *
 *
 *        METAFILE SUBSYSTEM procedure : rdmtype
 *
 *        Written by Steve Trachsel,
 *                   Summer 1983.
 *
 *
 *
 *)
; procedure rdmtype ( wid : wsname; var kind : vdicmdtype)
;var count : integer
        ; b_errcode :boolean
;begin
        b_errcode := false
        ;with metahdr do
        begin (* read vdicmd type commands *)
                gksstr := '    '
                ;if lengksm <> 0 then
                  for count := 1 to lengksm
                        do if not eof(mifile) then
                        begin
                           readchar(mifile,gksstr[count])
                        end
                ; vdiread (mifile,kind,lopcode,b_errcode)
                (* assign it to global var now *)
                ; opcode := kind
        end
end
```

```
(*
 *
 *
 *
 *        METAFILE SUBSYSTEM procedure : rdmdata
 *
 *        Written by Steve Trachsel,
 *                   Summer 1983.
 *
 *
 *
 *)
; procedure rdmdata ( wid : wsname; var rec : vdiparm)
;var ncoord : integer
    ;b_errcode : boolean
    ;count : integer
    ;temp : integer
;begin
    b_errcode := false
    ;with metahdr do
    begin
           if lenbytesfld <> 0 then
                  fread (mifile,linbytes,
                         lenbytesfld,b_errcode)
           ;case opcode of
             vdiclose:begin
                         rec.cmdcode := vdiclose
                         end(* gsx #2 , gsm #0 *)
             ;vdiclear:     begin
                         rec.cmdcode := vdiclear
                         end(* gsx #3, gsm #1 *)
             ;vdimsg:     begin
                         rec.cmdcode := vdimsg
                         ;fread(mifile,
                         ncoord,lenint,b_errcode)
                         ;for count :=1 to ncoord
                         do
                            readchar(mifile,
                                  rec.msg[count])
                         end(* no gsx, gsm #5 *)
             ;vdiline:     begin
                          rec.cmdcode := vdiline
                         ; fread (mifile,ncoord,
                              lenint,b_errcode)
                         ;rec.npts := ncoord
                         ; for count := 1 to ncoord
                         do
                            begin (* read coords *)
                            ; fread (mifile,temp,
                               lenint,b_errcode)
                            ; rec.pts[count].ix :=
```

```
                        temp
                    ; fread (mifile,temp,
                        lenint,b_errcode)
                    ;rec.pts[count].iy:=temp
                    end (* read coords *)
                end(* gsx #6, gsm #11 *)
;vdimark:           begin
                rec.cmdcode := vdimark
                ; fread (mifile,ncoord,
                    lenint,b_errcode)
                ;rec.npts := ncoord
                ; for count := 1 to ncoord
                do
                        begin(*read coords*)
                        ; fread (mifile,temp,
                            lenint,b_errcode)
                        ;rec.pts[count].ix:=temp
                        ; fread (mifile,temp,
                            lenint,b_errcode)
                        ;rec.pts[count].iy:=temp
                        end (* read coords *)
                end(* gsx #7, gsm #12 *)
;vditext:           begin
                rec.cmdcode := vditext
                ; fread (mifile,temp,
                        lenint,b_errcode)
                ; rec.textpos.ix := temp
                ; fread (mifile,temp,
                        lenint,b_errcode)
                ; rec.textpos.iy := temp
                ; fread (mifile,temp,
                        lenint,b_errcode)
                ; rec.numchar := temp
                ;for count := 1 to temp
                do
                        readchar(mifile,
                            rec.string[count])
                end(* gsx #8, gsm #13 *)
;vdiarea:           begin
                rec.cmdcode := vdiarea
                ; fread (mifile,ncoord,
                    lenint,b_errcode)
                ;rec.npts := ncoord
                ; for count := 1 to ncoord
                do
                        begin (* read coords *)
                        ; fread (mifile,
                            temp,lenint,b_errcode)
                        ; rec.pts[count].ix := temp
                        ; fread (mifile,temp,
                              lenint,b_errcode)
```

```
                        ; rec.pts[count].iy := temp
                        end (* read coords *)
                    end(* gsx #9, gsm #14 *)
;vdigdp:              begin
                    rec.cmdcode := vdigdp
                    ; fread (mifile,ncoord,
                           lenint,b_errcode)
                    ;rec.gdpcmd := ncoord
                    ; fread (mifile,temp,
                           lenint,b_errcode)
                    ; rec.numgdppts := temp
                    ; for count := 1 to temp
                    do
                        begin (* read coords *)
                        ; fread (mifile,temp,lenint,
                              b_errcode)
                        ;rec.gdppts[count].ix:=temp
                        ; fread (mifile,temp,lenint,
                              b_errcode)
                        ;rec.gdppts[count].iy:=temp
                        end (* read coords *)
                    end(* gsx #11, gsm #16 *)
;vdilinetype:        begin
                    rec.cmdcode := vdilinetype
                    ; fread (mifile,temp,lenint
                        ,b_errcode)
                    ;rec.kind := temp
                    end(* gsx #15, gsm #22 *)
;vdilinescale:       begin
                    rec.cmdcode := vdilinescale
                    ; fread (mifile,temp,lenint
                        ,b_errcode)
                    ;rec.scale := temp
                    end(* gsx #16, gsm #23 *)
;vdilinecolor:       begin
                    rec.cmdcode := vdilinecolor
                    ; fread (mifile,temp,lenint
                        ,b_errcode)
                    ;rec.color := temp
                    end(* gsx #17, gsm #24 *)
;vdimarktype:        begin
                    rec.cmdcode := vdimarktype
                    ; fread (mifile,temp,lenint
                        ,b_errcode)
                    ;rec.kind := temp
                    end(* gsx #18, gsm #26 *)
;vdimarkscale:       begin
                    rec.cmdcode := vdimarkscale
                    ; fread (mifile,temp,lenint
                        ,b_errcode)
                    ;rec.scale := temp
```

```
                end(* gsx #19, gsm #27 *)
;vdimarkcolor:        begin
                rec.cmdcode := vdimarkcolor
                ; fread (mifile,temp,lenint
                        ,b_errcode)
                ;rec.color := temp
                end(* gsx #20, gsm #28 *)
;vditextfont:         begin
                rec.cmdcode := vditextfont
                ; fread (mifile,temp,lenint
                        ,b_errcode)
                ;rec.kind := temp
                end(* gsx #21, gsm #30 *)
;vditextprec:         begin
                rec.cmdcode := vditextprec
                ; fread (mifile,temp,lenint
                        ,b_errcode)
                ;rec.prec := temp
                end(*no gsx,gsm #30,now 29*)
;vditextcolor:        begin
                rec.cmdcode := vditextcolor
                ; fread (mifile,temp,lenint
                        ,b_errcode)
                ;rec.color := temp
                end(* gsx #22, gsm #33 *)
;vdichscale:         begin
                rec.cmdcode := vdichscale
                ; fread (mifile,temp,lenint
                        ,b_errcode)
                ;rec.scale := temp
                end(* gsx #12, gsm #31 *)
;vdicvector:         begin
                rec.cmdcode := vdicvector
                ; fread (mifile,temp,lenint
                        ,b_errcode)
                ; rec.htvec.ix := temp
                ; fread (mifile,temp,lenint
                        ,b_errcode)
                ; rec.htvec.iy := temp
                ; fread (mifile,temp,lenint
                        ,b_errcode)
                ; rec.wdvec.ix := temp
                ; fread (mifile,temp,lenint
                        ,b_errcode)
                ; rec.wdvec.iy := temp
                end(* skip gsx, gsm #34 *)
;vditextspace:        begin
                rec.cmdcode := vditextspace
                ; fread (mifile,temp,lenint
                        ,b_errcode)
                ; rec.space := temp
```

```
                        end(* no gsx , gsm #32 *)
            ;vditextpath:       begin
                        rec.cmdcode := vditextpath
                        ; fread (mifile,temp,lenint
                                ,b_errcode)
                        ; rec.path := temp
                        end(* no gsx , gsm #35 *)
            ;vditextalign:      begin
                        rec.cmdcode := vditextalign
                        ; fread (mifile,temp,lenint
                                ,b_errcode)
                        ; rec.h := temp
                        ; fread (mifile,temp,lenint
                                ,b_errcode)
                        ; rec.v := temp
                        end(* no gsx , gsm #36 *)
                ;else: begin end
        end (* case *)
    end
end
```

```
(*
 *
 *
 *
 *        METAFILE SUBSYSTEM procedure : metaout
 *
 *        Written by Steve Trachsel,
 *                   Summer 1983.
 *
 *
 *
 *)
;procedure metaout(var mofile: text;
    vdicommand: vdicmdtype;rec:vdiparm)
;var count : integer
    ;temp : integer
    ;ncoord :integer
    ;buf: array[1..4] of char
;begin
    buf := gksmdef
    ;temp := 2
    ; for count :=1 to temp
    do
            writechar( mofile, buf[count])
    ;case vdicommand of
             vdiclose:begin
                          temp := 0
                          ;fwrite( mofile,temp,vitlen)
                          end(* gsx #2 , gsm #0 *)
             ;vdiclear:       begin
                          temp :=1
                          ; fwrite(mofile, temp, vitlen)
                          end(* gsx #3, gsm #1 *)
             ;vdimsg:       begin
                          temp := 5
                          ; fwrite(mofile, temp, vitlen)
                          ;temp := maxstring
                          ;fwrite(mofile,temp,vintlen)
                          ;for count :=1 to temp
                          do
                                  writechar(mofile,
                                          rec.msg[count])
                          end(* no gsx, gsm #5 *)
             ;vdiline:      begin
                          temp := 11
                          ; fwrite(mofile, temp, vitlen)
                          ;ncoord := rec.npts
                          ; fwrite (mofile,ncoord,vintlen)
                          ; for count := 1 to ncoord
                          do
                                  begin (* read coords *)
```

```
                        ;temp:=rec.pts[count].ix
                        ; fwrite (mofile,temp
                                ,vintlen)
                        ;temp:=rec.pts[count].iy
                        ; fwrite (mofile,temp
                                ,vintlen)
                        end (* read coords *)
                end(* gsx #6, gsm #11 *)
;vdimark:        begin
                temp :=12
                ; fwrite(mofile, temp, vitlen)
                ;ncoord := rec.npts
                ; fwrite (mofile,ncoord,vintlen)
                ; for count := 1 to ncoord
                do
                        begin (* read coords *)
                        ;temp := rec.pts[count].ix
                        ;fwrite(mofile,temp,vintlen)
                        ;temp := rec.pts[count].iy
                        ;fwrite(mofile,temp,vintlen)
                        end
                end(* gsx #7, gsm #12 *)
;vditext:        begin
                temp :=13
                ; fwrite(mofile, temp, vitlen)
                ;temp := rec.textpos.ix
                ; fwrite (mofile,temp,vintlen)
                ;temp := rec.textpos.iy
                ; fwrite (mofile,temp,vintlen)
                ; temp := rec.numchar
                ;for count:= 1 to temp
                do
                        writechar( mofile,
                             rec.string[count])
                end(* gsx #8, gsm #13 *)
;vdiarea:        begin
                temp :=14
                ; fwrite(mofile, temp, vitlen)
                ;ncoord := rec.npts
                ; fwrite (mofile,ncoord,vintlen)
                ; for count := 1 to ncoord
                do
                        begin (* read coords *)
                        ;temp := rec.pts[count].ix
                        ;fwrite(mofile,temp,vintlen)
                        ;temp := rec.pts[count].iy
                        ;fwrite(mofile,temp,vintlen)
                        end
                end(* gsx #9, gsm #14 *)
;vdigdp:         begin
                temp :=16
```

```
                ;fwrite(mofile, temp, vitlen)
                ;ncoord := rec.gdpcmd
                ;fwrite(mofile,ncoord,vintlen)
                ;temp := rec.numgdppts
                ;fwrite (mofile,temp,vintlen)
                ;for count := 1 to temp
                do
                        begin (* read coords *)
                        ;temp:=rec.gdppts[count].ix
                        ;fwrite (mofile,temp,vintlen)
                        ;temp:=rec.gdppts[count].iy
                        ;fwrite (mofile,temp,vintlen)
                        end (* read coords *)
                end(* gsx #11, gsm #16 *)
;vdilinetype:       begin
                temp := 22
                ; fwrite(mofile, temp, vitlen)
                ; temp := rec.kind
                ;fwrite (mofile, temp, vintlen)
                end(* gsx #15, gsm #22 *)
;vdilinescale:       begin
                temp := 23
                ; fwrite(mofile, temp, vitlen)
                ; temp := rec.scale
                ;fwrite (mofile, temp, vintlen)
                end(* gsx #16, gsm #23 *)
;vdilinecolor:       begin
                temp := 24
                ; fwrite(mofile, temp, vitlen)
                ; temp := rec.color
                ;fwrite (mofile, temp, vintlen)
                end(* gsx #17, gsm #24 *)
;vdimarktype:       begin
                temp := 26
                ; fwrite(mofile, temp, vitlen)
                ; temp := rec.kind
                ;fwrite (mofile, temp, vintlen)
                end(* gsx #18, gsm #26 *)
;vdimarkscale:       begin
                temp := 27
                ; fwrite(mofile, temp, vitlen)
                ; temp := rec.scale
                ;fwrite (mofile, temp, vintlen)
                end(* gsx #19, gsm #27 *)
;vdimarkcolor:       begin
                temp := 28
                ; fwrite(mofile, temp, vitlen)
                ; temp := rec.color
                ;fwrite (mofile, temp, vintlen)
                end(* gsx #20, gsm #28 *)
;vditextfont:       begin
```

```
                temp := 29
                ; fwrite(mofile, temp, vitlen)
                ; temp := rec.kind
                ;fwrite (mofile, temp, vintlen)
                end(*gsx #21,gsm #30,fornow 29*)
;vditextprec:       begin
                temp :=30
                ; fwrite(mofile, temp, vitlen)
                ; temp := rec.prec
                ;fwrite (mofile, temp, vintlen)
                end(* no gsx , gsm #30 *)
;vditextcolor:       begin
                temp := 33
                ; fwrite(mofile, temp, vitlen)
                ; temp := rec.color
                ;fwrite (mofile, temp, vintlen)
                end(* gsx #22, gsm #33 *)
;vdichscale:       begin
                temp := 31
                ; fwrite(mofile, temp, vitlen)
                ; temp := rec.scale
                ;fwrite(mofile, temp, vintlen)
                end(* gsx #12, gsm #31 *)
;vdicvector:       begin
                temp := 34
                ; fwrite(mofile, temp, vitlen)
                ; temp := rec.htvec.ix
                ; fwrite (mofile,temp,vintlen)
                ; temp := rec.htvec.iy
                ; fwrite (mofile,temp,vintlen)
                ; temp := rec.wdvec.ix
                ; fwrite (mofile,temp,vintlen)
                ; temp := rec.wdvec.iy
                ; fwrite (mofile,temp,vintlen)
                end(* skip gsx, gsm #34 *)
;vditextspace:       begin
                temp := 32
                ; fwrite(mofile, temp, vitlen)
                ;temp := rec.space
                ;fwrite(mofile, temp, vintlen)
                end(* no gsx , gsm #32 *)
;vditextpath:       begin
                temp := 35
                ; fwrite(mofile, temp, vitlen)
                ;temp := rec.path
                ;fwrite(mofile, temp, vintlen)
                end(* no gsx , gsm #35 *)
;vditextalign:       begin
                temp := 36
                ;fwrite(mofile, temp, vitlen)
                ;temp := rec.h
```

```
                                ;fwrite(mofile, temp, vintlen)
                                ;temp := rec.v
                                ;fwrite(mofile, temp, vintlen)
                                end(* no gsx , gsm #36 *)
                        ;else: begin end
    end
end
```

A GKS VIRTUAL DEVICE INTERFACE META FILE SYSTEM

by

STEVEN W. TRACHSEL

B. S., The Ohio State University, 1979

AN ABSTRACT OF A MASTER'S REPORT

submitted in parital fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1984

ABSTRACT

This report presents a PASCAL implementation of a subset of the meta-in and the meta-out portions of the Metafile System described in the Graphics Kernal Standard Annex E. During the course of implementing the Metafile System a number of unresolved issues became evident: how the concept of the meta file is related to GKS as a whole, how it relates to the CORE graphics standard meta file system, and how it fulfills the needs of users working in an interactive graphics environment.