

AD-A152 954

AFIT GKS--A GKS IMPLEMENTATION IN THE ADA PROGRAMMING
LANGUAGE(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB
OH SCHOOL OF ENGINEERING R S RUEGG DEC 84

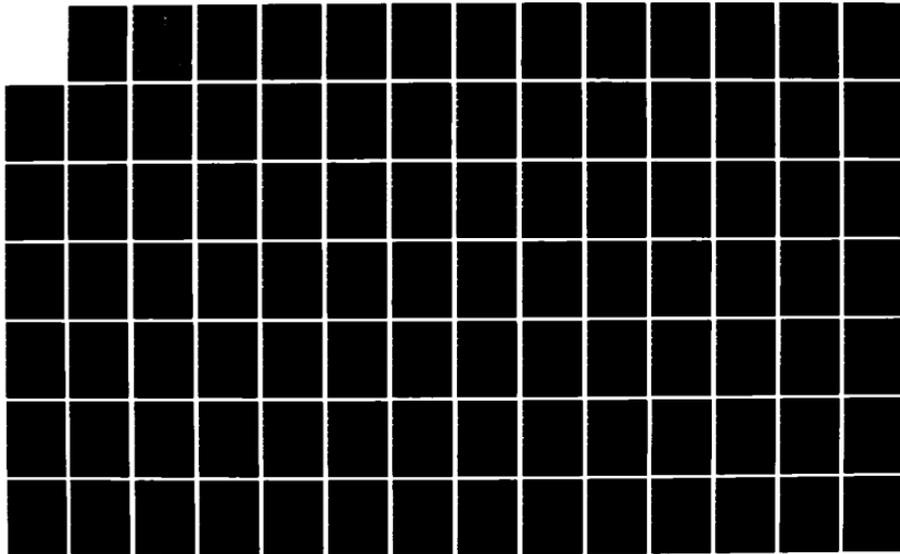
1/2

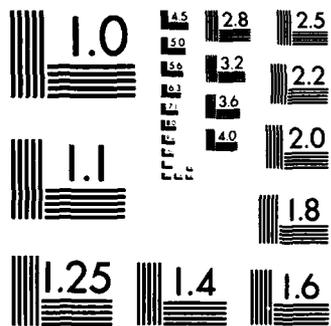
UNCLASSIFIED

AFIT/GCS/MATH/84D-5

F/G 9/2

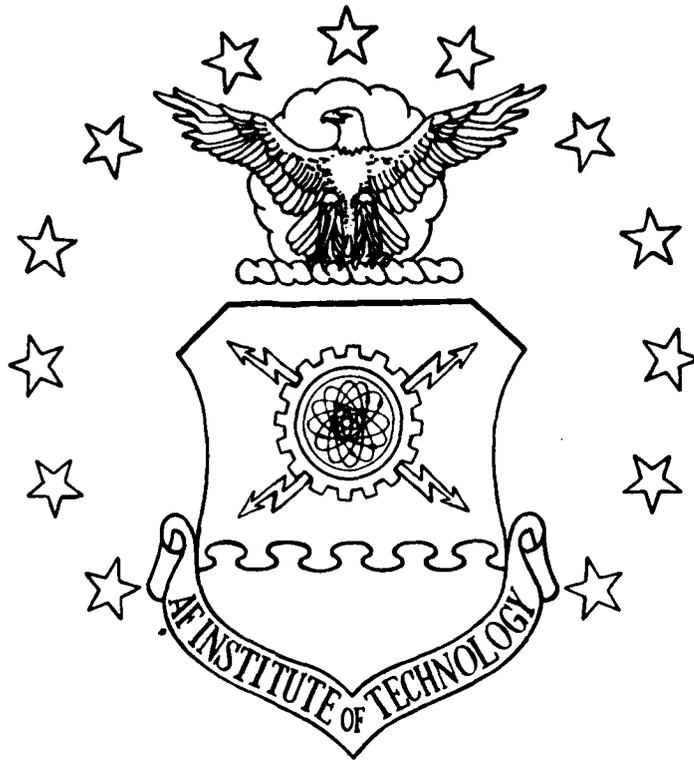
NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A152 954



AFIT_GKS -- A GKS IMPLEMENTATION IN
THE ADA PROGRAMMING LANGUAGE

THESIS

Raymond Scott Ruegg, B.S.
Second Lieutenant, USAF

AFIT/GCS/MATH/84D-5

DTIC FILE COPY

This document has been approved
for public release and sale; its
distribution is unlimited.

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

DTIC
ELECTE
S APR 29 1985 **D**
A

Wright-Patterson Air Force Base, Ohio

85 4 05 023

AFIT/GCS/MATH/84D-5

A-1

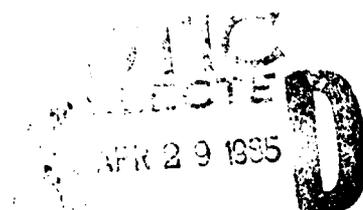


AFIT_GKS -- A GKS IMPLEMENTATION IN
THE ADA PROGRAMMING LANGUAGE

THESIS

Raymond Scott Ruegg, B.S.
Second Lieutenant, USAF

AFIT/GCS/MATH/84D-5



A

Approved for public release; distribution unlimited

AFIT/GCS/MA/84D-5

AFIT_GKS -- A GKS IMPLEMENTATION IN
THE ADA PROGRAMMING LANGUAGE

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Systems

Raymond Scott Ruegg, B.S.

Second Lieutenant, USAF

December 1984

Approved for public release; distribution unlimited

Acknowledgments

I would like to thank all the people that helped make this thesis a success. First, I would like to thank my thesis advisor Professor Charles Richard for all his helpful suggestions that kept this thesis on track. Second, I would like to thank my reader Captain Patricia Lawlis for giving me fresh perspectives on this project. Third, my thanks go out to the Support Systems Branch of the Wright Aeronautical Laboratory, for sponsoring this project. Fourth, I must thank the Harris Corporation who sent me their draft binding of GKS to the Ada programming language. This project would have been extremely difficult if not impossible without this document.

Thanks also goes to Laura and Yoshi Suzuki. I thank Laura for listening to all the problems I had, and offering solutions. While I thank Yoshi for editing and correcting the comment headers which went into the code.

My deepest gratitude goes to Captain Al Deese and the rest of the workers at the Aeronautical Systems Division (ASD) Computer Center. Captain Deese gave many hours throughout this thesis making sure that I had everything I needed to complete my work. To the entire staff of the ASD Computer Center, thanks. I will always remember you.

Raymond Scott Ruegg

Table of Contents

	<u>Page</u>
Acknowledgments	ii
List of Figures	vi
Abstract	viii
I. Introduction	1.1
Background	1.1
History of ANS GKS	1.1
History of Ada	1.2
What is ANS GKS?	1.3
Functional Categories of ANS GKS	1.4
Levels of ANS GKS	1.5
Problem Definition	1.6
Scope of Thesis	1.7
Overview of the Project	1.7
Literature Review of ANS GKS	1.8
What Makes a Good Graphics Package	1.8
Device Independence	1.9
Compactness	1.9
Device Richness	1.10
Portability	1.10
Conclusion	1.11
II. Requirements	2.1
Functional Categories	2.1
Design Considerations	2.1
Documentation	2.2
Life Cycle	2.2
User's Guide	2.2
Software Engineering Tools	2.3
Information Hiding	2.3
Virtual Device Interface	2.3
Harris Binding	2.4
Conclusions	2.4
III. Global Design and Implementation	3.1
User Functions	3.3
Packaging	3.5
Internal Control	3.7
Control	3.7
Primitives	3.7

	<u>Page</u>
Set_Primitives	3.7
Represent	3.8
Transform	3.8
Segments	3.8
Set_Input	3.8
Input	3.9
Inquiry	3.9
Set_Transform	3.9
Emergency	3.9
Error_Handling	3.9
Data Structures and Types	3.10
Internal_Vars	3.10
Operating_State	3.10
GKS_Description_Table	3.12
GKS_State_List	3.14
Workstation_State_Lists	3.14
Workstation_Description_Tables	3.16
Error_State_List	3.17
External_Types	3.17
Internal_Types	3.19
GKS_Coordinate_System	3.20
GKS_List_Uutilities	3.21
GKS_Configuration	3.21
Error_Routines	3.21
Order_of_Checking_Error_Numbers	3.22
Design_Alternatives	3.22
Implementation_of_Error_Functions	3.24
Reporting_an_Error	3.28
Workstations	3.28
The_Original_Design	3.29
Implemented_Design	3.31
OUTIN_Workstations	3.35
Ws_x	3.35
Workstation_Primitive_Functions	3.35
The_Segment_Routines_and_Others	3.38
Input_Routines	3.39
Int_ws_x	3.39
Drive_x	3.40
WISS	3.40
Global_Considerations	3.43
Device_Independent / Dependent_Code	3.44
Transformations	3.45
Deferral_Modes	3.46
Testing	3.47
Conclusions	3.47
IV. Analysis	4.1
Ada	4.1
Positive_Comments	4.1
Ada & Graphics	4.2

	<u>Page</u>
Trouble with Running Ada with the ROLM Compiler . . .	4.3
Specific Errors	4.5
Undesirable Features of Ada on the ROLM Data	
General	4.8
Conclusion	4.9
Harris Binding	4.10
Harris Binding Not Accepted by the ROLM Data	
General	4.11
Style Changes to the Harris Binding	4.12
Conclusions	4.14
ANS GKS	4.14
ANS GKS Proposal	4.14
Conclusion	4.15
V. Conclusions and Recommendations	5.1
Conclusions	5.1
Results	5.2
Recommendations	5.2
Known Bugs in the Program	5.4
Conclusions	5.4
Appendix A: User Guide to AFIT_GKS	A.1
Table of Contents	A.1
Introduction	A.2
Cross Index	A.2
External Types	A.11
AFIT_GKS_Functions	A.38
AFIT_GKS_Errors	A.64
Sample Program	A.68
System Dependent Features of AFIT_GKS	A.71
Appendix B: GKS_COORDINATES, GKS_LIST_UTILITIES, and GKS_	
CONFIGURATION	B.1
GKS_Coordinate_System	B.2
GKS_List_Utilities	B.3
GKS_Configuration	B.4
Appendix C: Harris Functions Not Implemented	C.1
Bibliography	BIB.1
VITA	VIT.1

List of Figures

<u>Figure</u>		<u>Page</u>
1.1	Layer Model of GKS	1.4
3.1	Overall Structure	3.2
3.2	User Functions	3.4
3.3	Data Structures and Types	3.11
3.4	Possible Values of Operating_State	3.13
3.5	GKS_Description_Table	3.12
3.6	The Workstation_State_Lists	3.15
3.7	Original Design of Workstations	3.30
3.8	Modified Data Flow Chart of ANS GKS	3.32
3.9	Workstations	3.34
3.10	Polyline Pipeline	3.36
3.11	Polyline Pipeline Superimposed on the OUTIN Side of the Modified Data Flow Chart of ANS GKS	3.37
3.12	Polyline Attributes Being Bound to Workstation x . .	3.41
3.13	WISS Interfacing with the Polyline Pipeline	3.42
4.1	Procedure DO_SOMETHING	4.4
4.2	DO_SOMETHING Being Debugged	4.4
4.3	Normalization Transformation, and Segment Types . .	4.7
4.4	Illegal Type "X"	4.9
4.5	Variable Strings	4.11
4.6	Input_Class and Choice_Input	4.12
4.7	Present Evaluate_Transformation_Matrix	4.13
4.8	Proposed Evaluate_Transformation_Matrix	4.13
5.1	Inq_Polyline_Representation	5.3

Figure

Page

A.1	Demo Program	A.69
A.2	Output of Demo Program	A.70

Abstract

This project, written in Ada, involved designing and implementing AFIT_GKS which is a subset of the Graphical Kernel System (GKS). This project implemented AFIT_GKS on a ROLM Data General MV/8000-II validated Ada compiler, using a proposed Binding of GKS to Ada developed by the Harris Corporation. After introducing Ada and GKS to the reader, this project considers several alternative ways of designing AFIT_GKS. Selecting what was considered the best design alternative, this project implements AFIT_GKS. It concludes with a discussion of how well Ada, the proposed GKS binding to Ada, and GKS, worked in AFIT_GKS. This thesis found minor problems with the validated ROLM Ada compiler, the proposed GKS binding to Ada, and GKS, but overall they were each excellent products. By using AFIT_GKS as proof, this project concludes that Ada can support large programs, and Ada can support computer graphics.

AFIT_GKS -- A GKS IMPLEMENTATION IN
THE ADA PROGRAMMING LANGUAGE

I. Introduction

This thesis involves the design and implementation of a subset of the American National Standard Graphical Kernel System (ANS GKS) graphics package in the Ada programming language. This project initiates a first step at the Air Force Institute of Technology (AFIT) to provide a graphics package for the Department of Defense (DoD) standard language Ada. Improving portability of software, this subset of the ANS GKS graphics package developed at AFIT (AFIT_GKS) will provide a group of functions in which all future graphics applications can be based. This project's major thrust involves the design and implementation of AFIT_GKS so as to take full advantage of the power of the Ada programming language. The requirements of an ANS GKS graphics package are presently undergoing final approval, therefore AFIT_GKS will use the Technical Committee X3H3/83-25r3 report as its definition of the ANS GKS graphics package(1:i).

Background

This section gives a brief historical accounting of the development of the ANS GKS graphics package and the Ada programming language.

History of ANS GKS. The design of ANS GKS was based on work done by many groups. The early design of GKS and performed by the International Organization for Standardization (ISO) in the Workshop on Graphics Standards Methodology held in May 1976 in Seillac, France(1:i). The ISO

version of GKS (ISO GKS) was first developed by the West German Standardization Institute, DIN, in 1978, and then refined during the period 1980-1982 by Working Group 2 of the Subcommittee on Programming Languages of the Technical Committee on Information Processing of the ISO (ISO TC97/SC5/WG2)(1:i;5:vii).

In parallel with the development of ISO GKS was the work of the Graphic Standards Planning Committee of the Special Interest Group on Computer Graphics of the Association for Computing Machinery (ACM SIGGRAPH GSPC)(1:i). This work, known as the Core System Proposal, was published and widely distributed in 1977 and again (in a revised version) in 1979.

Pooling the ideas of ISO GKS and the Core System Proposal, ANSI developed ANS GKS(1:ii). In February 1984, ANS GKS was published for a four-month period of public review and comment(1). After the initial public review the ANSI Committee voted to accept ANS GKS as a standard.

History of Ada. The other aspect of this project is to explore the Ada programming language. Ada came out of a 1970 DoD study into the software problems facing the DoD. These problems included a diversity of programming languages, many programs using ill suited languages, modern programming practices being ignored, and lack of software support environments(2:12). After developing the requirements for a new language which would support parallel processing, real-time control, exception handling, and unique I/O features, in August 1977 the DoD accepted the design of 4 contractors which were each tasked with coming up with this new language(2:17). In February 1978, the developers came back with 4 different designs two of which were accepted for further development (2:18). After this a final decision was made on the language and in May

III. Design and Implementation

This chapter explores the design and implementation details of AFIT_GKS. The overall design of AFIT_GKS shown in Figure 3.1 consists of four major parts -- the User Functions, the Data Structures and Types, the Error Functions, and the Workstations. As shown in Figure 3.1, the 'Application' program accesses AFIT_GKS by calling the User Functions (See Appendix A). It also uses the package External_Types (See Appendix A) which contains those types found in the parameters of the User Functions. In turn, the User Functions of AFIT_GKS call upon the Error Functions to test any errors in the calling of AFIT_GKS. As shown in Figure 3.1, after testing for errors, the User Functions either call upon the Workstations to output some graphical information, or they access the Data Structures and Types. The Data Structures and Types shown in Figure 3.1 are a collection of types and variables which User Functions, Error Functions and Workstations can use and modify. Although it is not shown in Figure 3.1, the package External_Types is a part of the Data Structures and Types.

This chapter explains how each part of this overall structure was developed and implemented. First, it starts by explaining how the User Functions are separated into different categories (Sectioning the User Functions). Second, the chapter explains the design and implementation considerations that went into the Data Structures and Types. Third, the chapter considers various possible designs of the Error Functions. Then it shows how the accepted design was implemented. Fourth, this chapter covers the design and implementation details of the Workstations. Finally, the chapter concludes with some topics which are involved in the

device functions, but still not overload the devices with too much device dependent code. The workstations in AFIT_GKS are designed in a way that provides a "template" or standard way to handle various device features so that it is easy to add new workstations to AFIT_GKS.

Harris Binding

As specified in the ANS GKS requirements, if a binding of ANS GKS to a particular language has been developed, then any implementation of ANS GKS must use that binding(1:3). The Harris Corporation has developed a proposed binding of ANS GKS to the Ada programming language(10). Therefore, AFIT_GKS will use this binding.

Conclusions

In this section we covered the various requirements of this project, and the reasons for these requirements. In the next chapter, some of the design decisions are given, and the rationale of why they were chosen over other possible decisions.

of what errors may occur when the user calls a function of AFIT_GKS. Also included in this user's guide is a small example program which uses AFIT_GKS.

Software Engineering Tools

Software engineering tools are developed so as to improve the quality of software. There are many software tools designed so as to be effective under different and varying problem environments. AFIT_GKS is a diverse set of functions with different requirements. Therefore, different procedures will best be suited to different software engineering tools. Some techniques, like SADT's, suit those functions that have a flow of control(10:63). Other methods like Jackson's method is better suited to functions which modify data structures like the Set Attribute functions(10:153). Therefore, in this project the software engineering tools will be modeled towards the function of AFIT_GKS being designed.

Information Hiding

Information hiding enables a program to hide its data structures from any outside modifications. By defining data structures inside of procedures and package bodies, the data structure becomes inaccessible to the rest of the program. Therefore, a change to this data structure entails changing only a small core of functions, not the entire program. This concept must be included in the design of AFIT_GKS.

Virtual Device Interface

Another requirement that must be addressed is where to establish the virtual device interface. This is where the device independent AFIT_GKS functions must interact with the device dependent AFIT_GKS functions. This decision should allow each graphical device to implement all its

2. How to group the functions.
3. How to break the functions down so as to maintain a high degree of device independence.
4. How to break the functions down so as to allow advanced hardware to use all its extra capabilities.
5. How to design the GKS package so as to make it expandable.

Documentation

In-line documentation of code enhances the readability, clarity, and maintainability of software, and shall be included in AFIT_GKS. This documentation enables the maintainer of the program to better comprehend what the code is doing, and how it is doing it.

For AFIT_GKS this documentation must be a combination of commented in-line code, and standard headers for the different functions implemented. Since AFIT has a standard header for its software, this header will be used in AFIT_GKS.

Life Cycle

AFIT_GKS attempts to minimize the overall life cycle cost. Clarity and simplicity traditionally minimize the most costly part of the life cycle, the maintenance phase. With this in mind, this implementation of AFIT_GKS shall strive for simple and clear constructs which will improve the maintainability of this project.

User's Guide

Any large program requires a users guide in order to help the user understand how to use the program effectively. The users guide for AFIT_GKS is Appendix A. It shows the user of AFIT_GKS what functions are available, the data types required by the functions, and a detailed list

II. Requirements

This chapter explores the requirements of this project's effort to write an ANS GKS graphical package. Next this chapter justifies the requirements with reasons as to why the different requirements are necessary for this project.

AFIT_GKS will incorporate the following guidelines:

Functional Categories

AFIT_GKS entails designing and implementing a graphical package which is highly dependent on the functional categories specified in ANS GKS. Therefore, AFIT_GKS shall separate the functions into their respective functional categories. These functional categories are Control functions, Output functions, Output Attributes, Transformation functions, Segment functions, Input functions, Metafile functions, Inquiry functions, Set Transformation functions, and Error Handling functions. Together, these categories allow the user of AFIT_GKS to use any set of functional categories that an application program needs (See Appendix A).

Design Considerations

To insure easy expandability, the design of AFIT_GKS will consist of the entire ANS GKS package. This allows a partial implementation of ANS GKS which, by design, can expand to a complete ANS GKS package.

Many ideas must go into a good design of AFIT_GKS. The steps considered in this design are as follows:

1. How to implement the various functions which are defined in ANS GKS.

bundle table is a workstation dependent table associated with a particular output primitive(1:9). Entries in this table specify all the workstation dependent aspects of a primitive(1:9). For example, an entry in a polyline bundle table (a polyline function draws connected line segments) contains a list of possible values for the attributes of the polyline. The attributes of a polyline are line type (e.g. solid, dotted, or dashed), line width, and line color. The polyline bundle table helps portability because each display device has its own polyline table which contains attribute values tailored to the particular device(15:73). Using the polyline bundle tables a programmer can draw a fat red dotted polyline on one display surface, and a thin blue solid polyline on another display surface(14:14-15). This allows ANS GKS to interface concurrently to several different display devices using the different capabilities of the display devices(14:14-15).

Conclusion. Graphical Kernel System (ANS GKS) was accepted as a standard graphical package. Previously adopted by the European graphical community, ANS GKS has been accepted in the US because of its device independence, compactness, device richness, and portability. Having been accepted as a standard, then all future graphics projects should use the ANS GKS graphics system. Therefore, an ANS GKS package should be started now in order to have a working ANS GKS package at AFIT.

full input (level c)(1:64). To give some idea of the range of these levels, level m, minimal level, has 31 functions and 17 Inquiry functions (like what line style is presently being used), where level 2-c (the highest level) has 110 functions and 75 Inquiry functions(12:184). This range of levels enables the implementor of ANS GKS to choose a consistent subset of ANS GKS that will be standard across many different implementations of ANS GKS(11:II-29,30).

Device Richness. ANS GKS's device richness allows the user to take advantage of hardware graphics features of individual machines (8:1). Two examples of this device richness are stroke precision text, and the generalized drawing primitive(1:29;90-91). The stroke precision text asks the device to draw the text in the correct orientation (up, down, or diagonally) but if the machine does not have this capability, then ANS GKS prints the text in the best way possible(7:32-33). A drawback to this method is that sometimes stroke precision text should appear on the viewscreen, but because the graphics device cannot orient the text correctly, the incorrectly oriented text ends up out of the viewing area (4:102). The device richness of the generalized drawing primitive allows the user to input a set of points and a function name which the hardware interprets and executes(9:114). This allows the user to use any of the hardware dependent graphics procedures, such as a circle or arc(1:90-91; 9:114).

Portability. Another ANS GKS goal was portability between different installations(8:1). This portability comes from being a standard graphical package, and also from some of its portable features (14:11). Bundle tables improve the portability of ANS GKS(14:14). The

facilities on a large spectrum of graphical devices(1:2;14:10). "Compactness" insures that the graphics package contains only those functions necessary for the application program(1:1;14:10). "Device Richness" allows the physical graphical devices to implement any special hardware capabilities like filling a polygon(1:1;14:10;12:184-185). "Portability" enables different applications to go from one machine to another(1:1;11:24;14:10;12:187). These criteria will be applied to AFIT_GKS.

Device Independence. One of ANS GKS's main objectives was to design ANS GKS uniformly for a whole range of graphic devices, including vector and raster devices, microfilm recorders, storage tube displays, refresh displays and color displays(1:1). This device independence at the workstation level allows the different machines to use their full capabilities(1:2;12:184-185). This device independence appears in the input and output functions of the ANS GKS package. Here, both the input and the output streams break up into a device independent (not related to a specific device) and a device dependent (related to the hardware capabilities of a specific device) set of code (1:2;14:11). This allows the different devices to use the same device independent code in cooperation with their own device dependent code.

Compactness. ANS GKS was designed with many input/output implementation levels, which define different subsets of the ANS GKS package. This feature of allowing a wide range of subsets of ANS GKS is commonly referred to as compactness(1:64). The output levels (m,0,1,2) range from minimal output (level m) to full workstation independent segment storage (several graphics devices, all using one common storage area)(1:64). The input levels (a,b,c) go from no input (level a) up to

parameters of the different modules were specified so as to take advantage of the encapsulating features of the Ada programming language.

The second step in this project was the detailed design phase. Here, the design was broken down so as to allow the graphics package to use the powerful constructs of the Ada programming language. Moreover, the design allowed each of the different devices to use some of its own machine dependent functions like the locators.

The third step in this project was the implementation of AFIT_GKS. The focus of the implementation was of using those Ada structures which implemented the design quickly, reliably, and clearly.

Literature Review of ANS GKS

In February 1984, the ANSI technical committee X3H3 computer graphics submitted the final draft proposal of ANS GKS to the public for review(1:i). The graphics community had until 1 July 1984 to write in any objectives to the proposed standard(1:i). After final review the proposed ANS GKS was adopted as a standard. This review covers the emerging ANS GKS system in order to justify its implementation on the ROLM computer at the Aeronautical Systems Division (ASD) Computer Center.

The review centers on the use of ANS GKS as a "good" standard graphical package. Because of the enormous size of the ANS GKS package, this literature review does not give the details of ANS GKS's overall design and functions.

What Makes a Good Graphics Package? Over time, different criteria evolved to establish what makes a "good" standard graphics package. These criteria vary but some of the more common ones are as follows. "Device independence" provides any application program with equivalent

The second problem involved the capabilities of Ada using a presently validated Ada compiler. These questions were as follows:

1. Can Ada support a large project?
2. How easy is Ada to use in a large project?
3. Can Ada support computer graphics?

The third problem was reviewing and commenting on the Draft GKS Binding to ANSI Ada (Harris binding)(6). The Harris binding is a proposed list of all the functions of ANS GKS and how they are accessed by any application programmer who uses an Ada implementation of ANS GKS.

Scope of Thesis

In order to review and comment on ANS GKS, and to answer some of the questions about the Ada programming language, this thesis designed, and implemented AFIT_GKS which is a subset of ANS GKS. AFIT_GKS was implemented in Ada using the ROLM Data General compiler at the ASD Computer Center Wright-Patterson AFB, Dayton, Ohio. AFIT_GKS defined its external interface using the Harris binding(6).

AFIT_GKS contains three workstations -- the Tektronix 4014, the Tektronix 4027, and Workstation Independent Segment Storage (WISS). The design of AFIT_GKS includes segmentation control (output level 2), and basic input functions (input level b). The implementation will not cover ANS GKS metafiles (output level 0), event queueing (input level c), and high level input functions such as event sampling (input level c).

Overview of the Project

The first step in this project was the high level system design phase. The different modules of the graphics package were sketched out to get an overall system design. Here the data structures, records and

0: At this level of output, all primitives and attributes are supported.

1: This level of output introduces segmentation.

2: This is the highest output level which does all kinds of output including Workstation Independent Segment Storage (WISS).

On the other hand, the input levels divide the functions according to how powerful the input devices are in the implementation. These levels are as follows:

a: At this low level, no input functions are supported.

b: At this level, input is only accepted in response to a specific REQUEST for input from the application program.

c: This level allows full input including sampling input which will poll an input device continuously.

Overall, ANS GKS is a diverse set of functions which permit application programs to interact with a variety of graphical devices. This diversity allows the program to satisfy the many different needs of various users of computer graphics.

Problem Definition

This thesis addresses three related problems. These problems include the need to review and explore ANS GKS, the Ada programming language, and the Harris binding of Ada to ANS GKS.

The first problem involved a review of ANS GKS. In February 1984, the Association for Computing Machinery (ACM) published a Special GKS issue of Computer Graphics. This issue contained specifications for the draft proposal of ANS GKS with a statement asking for public review and comment(1). Since ANS GKS was proposed as a standard it needed a thorough review.

4. Transformation functions: These functions allow the application programmer to scale, rotate, and translate graphical output that is displayed on the screen.

5. Segment functions: These functions allow the application programmer to organize the graphical output into segments which can then be saved, moved, or deleted.

6. Input functions: These functions allow the user of an application program to give inputs to the program via various input devices (i.e., a locator, the keyboard, or a light pen).

7. Metafile functions: This allows the application program to save graphical pictures.

8. Inquiry functions: They allow the application program to learn what the present attributes of the system are.

9. Utility functions: These functions allow the user to create segment transformation matrices which have a given scale factor, rotation angle, and translation factor relative to a given fixed point.

10. Error Handling: These functions handle any error conditions that may arise.

Levels of ANS GKS. In addition to all these function categories, ANS GKS is also sectioned by both input and output levels. These levels determine how much of ANS GKS is supported by a particular implementation of ANS GKS.

The output levels break the functions up according to how detailed the ANS GKS implementation must display the graphical information. The levels are as follows:

m: The minimal output level which consists of a small set of control and primitive output functions.

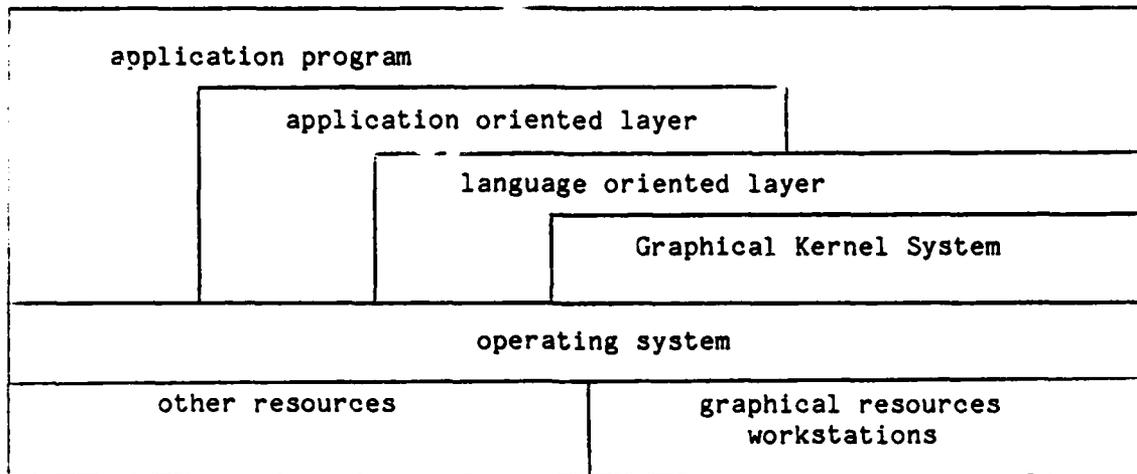


Figure 1.1 Layer Model of GKS (7:7)

workstations layer corresponds to the physical graphical devices and their graphical abilities. The three other layers, Graphical Kernel System, operating system, and other resources are self explanatory.

Functional Categories of ANS GKS. The ANS GKS functions are broken down into categories of related functions. These categories divide the one hundred eighty five functions of ANS GKS into the following ten more manageable functional groupings.

1. Control functions: These functions initialize the GKS system and allow the user to control how and where output is generated onto the graphical devices.

2. Output functions: These functions draw lines and the other graphical outputs.

3. Output Attributes: These functions allow the output functions to take on various characteristics like a thick line width or the color green.

1979, with permission, the language was called Ada(2:18). During this whole process the DoD was writing design and language requirements for Ada so it could keep control of the language. These documents included:

Apr 75 - STRAWMAN - Initial requirements(2:14)

Aug 75 - WOODENMAN - Second draft of requirements based on STRAWMAN
(2:14)

Jan 76 - TINMAN - The complete set of requirements(2:14)

Jan 77 - IRONMAN - A slight revision of TINMAN(2:16)

Jun 78 - STEELMAN - Final language requirements(2:18)

From May 79 through Nov 79, Ada went through a public testing and review process(2:19). At the same time the DoD began work on a computer validation facility which would ensure that the Ada compilers conformed to the specification of the language(2:19). In Aug 80, the Ada Joint Programming Office (AJPO) was created to manage all Ada related activities(2:20). In Jan 81, AJPO applied for, and received, Ada as a trademark of the DoD(2:20). Additionally, the AJPO asked for Ada standardization by ANSI, which was granted in February 1983(2:21).

What is ANS GKS?

ANS GKS is a set of basic functions for computer graphics programmers usable by many graphics producing applications(1:i). The layer model depicted in Figure 1.1 shows the role of ANS GKS as a graphical system(5:7). The application program and the application oriented layer correspond to any program that calls ANS GKS. The language oriented layer is the binding of GKS to the programming language. For AFIT_GKS this is the Draft GKS binding to ANS Ada supplied by the Harris Corporation (Harris Binding)(2). The graphical resources

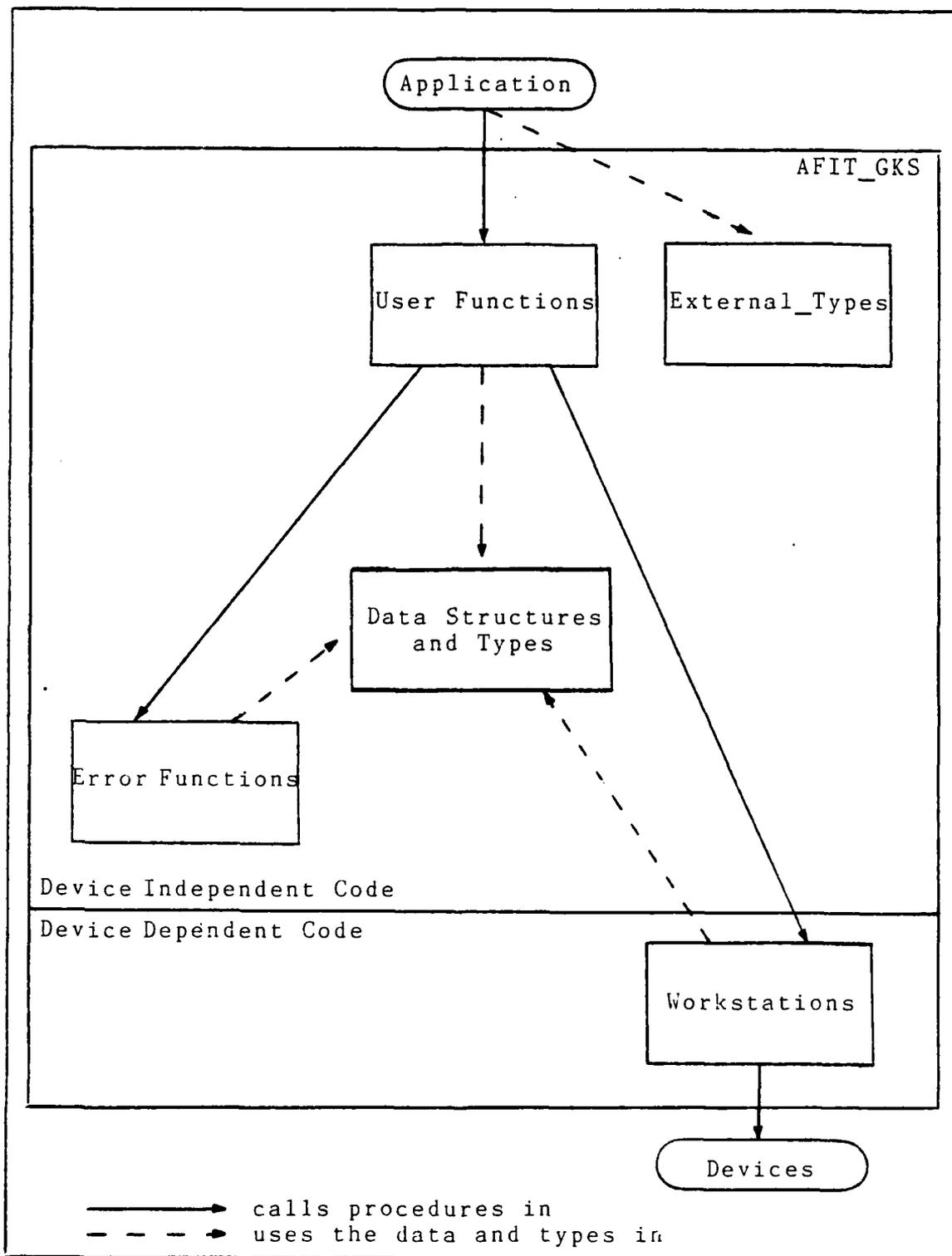


Figure 3.1. Overall Structure

design of AFIT_GKS but do not fall into any one of the different parts of the overall structure depicted in Figure 3.1.

User Functions

How should the AFIT_GKS User Functions (see Figure 3.1) be sectioned so that the entire AFIT_GKS project is readable, clear, and easily maintained? AFIT_GKS User Functions might be partitioned in several ways.

The first way is by implementation levels(1:64). This means that certain functions go together because they are related by how powerful they are. For example, level ma consists of the minimal related functions that can support GKS where as level mc supports the minimal output but allows for a wide variety of input devices. If the functions were sectioned by levels then several unrelated functions would be grouped together simply because they are considered to be of the same level of difficulty to implement. This would force procedures like emergency close_gks, and set_text_path to be grouped together because they are both required at level 0a. This would create packages of unrelated procedures which defeats much of the intent of Ada packages(2:184).

The second way to separate the functions is by functional categories as shown in Figure 3.2. These categories are Control functions, Output functions, Output attributes, Transformation functions, Segment functions, Input functions, Metafile functions, Inquiry functions, Utility functions (in AFIT_GKS they are called Set Transformation functions), and Error Handling(1:i,v;6:i). By splitting them up in this fashion, each set of functions have a common idea and purpose. This allows the maintainer of the program to work on one specific area of AFIT_GKS without needing to worry about how it might affect other areas. In

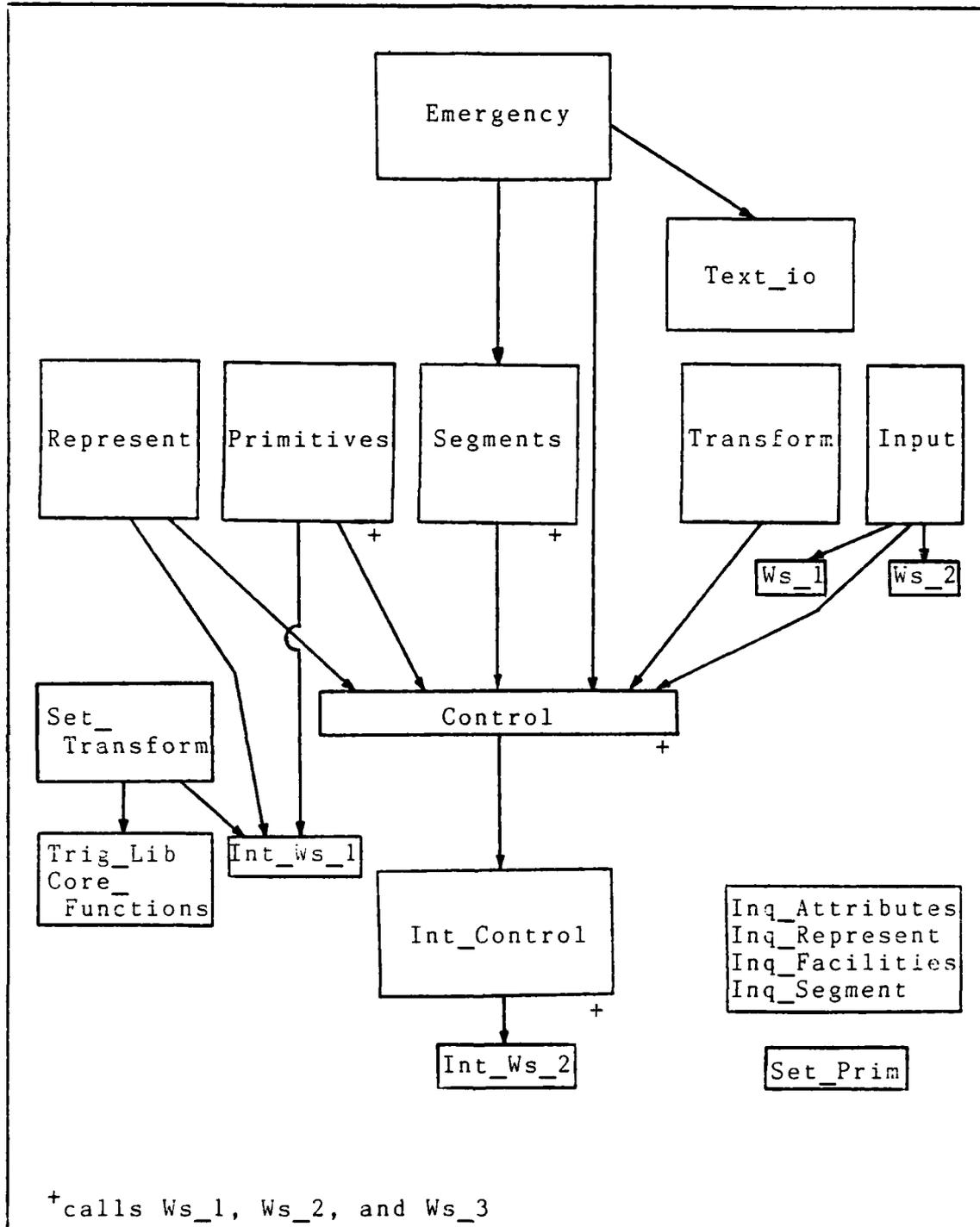


Figure 3.2. User Functions

essence, the various functional areas of AFIT_GKS can be separated and hidden from each other making for many small functional units which are easier to maintain. Also, this allows the user to pick and choose the part of the AFIT_GKS package that is needed for the program. For example, if the user of AFIT_GKS doesn't need any segments then that area of AFIT_GKS can be ignored as a single unit. Overall, the best plan might be a combination of separating the package into functional units in which the lowest level functions are implemented first.

Packaging. If AFIT_GKS is split up into functional units, then how should these units be implemented in Ada? Ada allows for four different types of structures or compilation units -- subprograms, task units, generic units, and packages(3:10-1). First, subprograms are like procedures and as such the user can access any subprogram. This means that no variables or lower level functions can be hidden from the user of AFIT_GKS.

Task units are designed to allow for parallel processing. The problem with parallel processing in AFIT_GKS is it must be carefully controlled or else unpredictable actions may occur. For example, if "activate workstation" and "polyline" procedures were allowed to execute concurrently then the line may or may not be drawn on the newly activated workstation. Task units may be useful in providing AFIT_GKS with parallel processing for multiple active workstations. This idea of task units may be able to speed up the processing time of AFIT_GKS but, due to time constraints it will not be explored in this thesis effort.

Generic units define an algorithm on an unspecified data object which can then be instantiated for whatever data type is needed. For example, a generic program could be written to switch two objects.

```

generic
  type ITEM_TYPE is private;
  procedure SWAP(First, Second : in out ITEM_TYPE);

  procedure SWAP(First, Second : in out ITEM_TYPE) is
    temp_item : ITEM_TYPE;
  begin
    temp_item := First;
    First := Second;
    Second := temp_item;
  end SWAP;

```

The problem with a generic unit is that it must be totally within at most two files (the generic specification and the generic body). This may be helpful for some low level modules but it certainly is not suited for the entire ANS GKS system as required by the Harris binding because then the entire AFIT_GKS source code would have to reside in 1 file (the generic body), which would be impossible to edit.

Finally, Ada allows packaging (which can be used with generics) which involves specifying all the procedures and global types that you want the user to see in a package specification. The rest of the program goes into the package body which is inaccessible to the user of the package. Therefore, by using packages the internal implementation of the AFIT_GKS program will be invisible to the end user of AFIT_GKS. Therefore, this thesis will center on packaging the AFIT_GKS program so that the user does not have access to the internal procedures that implement AFIT_GKS.

Using the packages of Ada and the sectioning of the user callable functions by functional categories, the structure of User Functions is shown in Figure 3.2.

The User Functions consist of all the different functions that the

user of AFIT_GKS can call. The User Functions also contain internal functions to handle the functions of AFIT_GKS that are not involved with a particular workstation (like creating a segment transformation matrix). These functions shown in Figure 3.2 will be briefly described in the order that they were presented in ANS GKS(1:v). The order of presentation is Internal_Control, Control, Primitives, Set_Primitives, Represent, Transform, Segments, Set_Input, Input, Inquiry, Set_Transform, Emergency, and Error Handling.

Internal_Control. The package Internal_Control, shown in Figure 3.2, handles the initialization of the various state and description tables. This package contains various routines which hold the descriptions of the workstations. This includes what line types the workstation can perform, what character heights, and widths. Anything that describes that workstation is found in this package.

Control. The package Control uses the Internal_Control package to initialize the various state and description tables. It also sets the deferral states and does updates on the various workstations.

Primitives. The package Primitives draws the actual primitives (polyline, polymarker, text, and fill_area). It is also involved with the deferral modes. A deferral mode is a variable that can be set on a workstation which tells the workstation when it must have its picture correct. If a segment is open and it is not the highest priority segment then calling any primitive causes the redrawing of the screen (which can be deferred).

Set_Primitives. The package Set_Primitives allows the user of AFIT_GKS to set the various attributes of the primitives. For example, the user can set the line type or line width that is needed.

Represent. The package Represent sets up the various bundle tables which the user can use. Also, this package can set up patterns, and colors on the workstations that support these features.

Transform. The package Transform sets the various windows, and viewports. It also sets the clipping indicator, and the transformation number. The only interesting function in this package is `set_viewport_input_priority`. This function sets one transformation to a priority which is higher or lower than another transformation. To accomplish this the function `set_viewport_input_priority` checks to see if the two transformations are in the correct order with respect to their priorities. If they are not in the correct order then the priorities of the transformations are exchanged.

Segments. The package Segments works on segments. It sets all the segment attributes like segment visibility, highlighting, priority, and detectability. This package also handles the three WISS functions `associate_segment_with_ws`, `copy_segment_to_ws`, and `insert_segment`. The only function that is interesting is the `set_segment_priority` function. This function sets the priority of the given segment and then sorts all the segments from highest priority to lowest priority. This is done so that any redrawing of the screen will draw all the segments in order from lowest priority to highest priority.

Set_Input. The package Set_Input initializes all the input functions. These functions are used to determine which input device will be used when an input function is called. These functions also set the various input modes (request, get, and sample), but at this time only request mode is implemented.

Input. This package performs the six input functions:

request_locator: requests a locator input on the given workstation

request_stroke: requests a stroke input on the given workstation

request_valuator: requests a floating point number from the given workstation

request_choice: requests an integer choice value from the given workstation

request_pick: requests a pick input which returns the highest priority segment that the user picked on the given workstation

request_string: requests text string from the given workstation

Inquiry. The four Inquiry packages shown in Figure 3.2 inquire the values of the various values on all the state and description tables.

Set_Transform. The Set_Transform functions shown in Figure 3.2, allow the user of AFIT_GKS to create segment transformation matrices given the rotation, scaling, and translation parameters.

Emergency. The Emergency package contains the routine 'emergency close_gks' which closes AFIT_GKS no matter what state AFIT_GKS is in at the time of calling this function.

Error_Handling. This package contains the single function error logging which logs any errors that is found in AFIT_GKS.

In conclusion, there are many functions that the user of AFIT_GKS can call. Therefore, AFIT_GKS is split up into many different packages (specified in Appendix A) which the user can choose to include or not include in any application program that uses AFIT_GKS User Functions (see Figure 3.1).

Data Structures and Types

The Data Structures and Types shown in Figure 3.1 are a series of seven related packages shown in Figure 3.3 that contain the types and variables used by AFIT_GKS. The first two packages shown in Figure 3.3, Internal_Vars and Internal_Types, are used exclusively by the internal functions of AFIT_GKS; the user of AFIT_GKS should never access these packages directly. The third package shown in Figure 3.3 is External Types (see Appendix A) which contains the types needed to interface AFIT_GKS with an application (see Figure 3.1). This package along with GKS_List_Uutilities, GKS_Coordinates, and GKS_Configuration (see Appendix B) were supplied as part of the Harris binding of ANS GKS to Ada. The last package shown in Figure 3.3 is Text_io which is a standard package available on all Ada compilers(2:421).

This section will cover Internal_Vars, Internal_Types, External_Types, GKS_List_Uutilities, GKS_Coordinates, and GKS_Configuration as shown in Figure 3.3. This section will not explain the design considerations of Text_io.

Internal_Vars. As shown in Figure 3.3, the six GKS data structures are all contained in one package called Internal_Vars. Internal_Vars is a package of variables which should only be accessed by the internal functions of AFIT_GKS. It contains the Operating_State, GKS_Description_Table, GKS_State_List, Workstation_State_Lists, Workstation_Description_Tables, and the Error_State_List. The three "state" lists hold the current values of their respective types. The "description" tables hold the static values of AFIT_GKS, or of their workstations.

Operating_State. The first structure shown in Internal_Vars (see Figure 3.3) is the Operating_State(1:197). This single variable

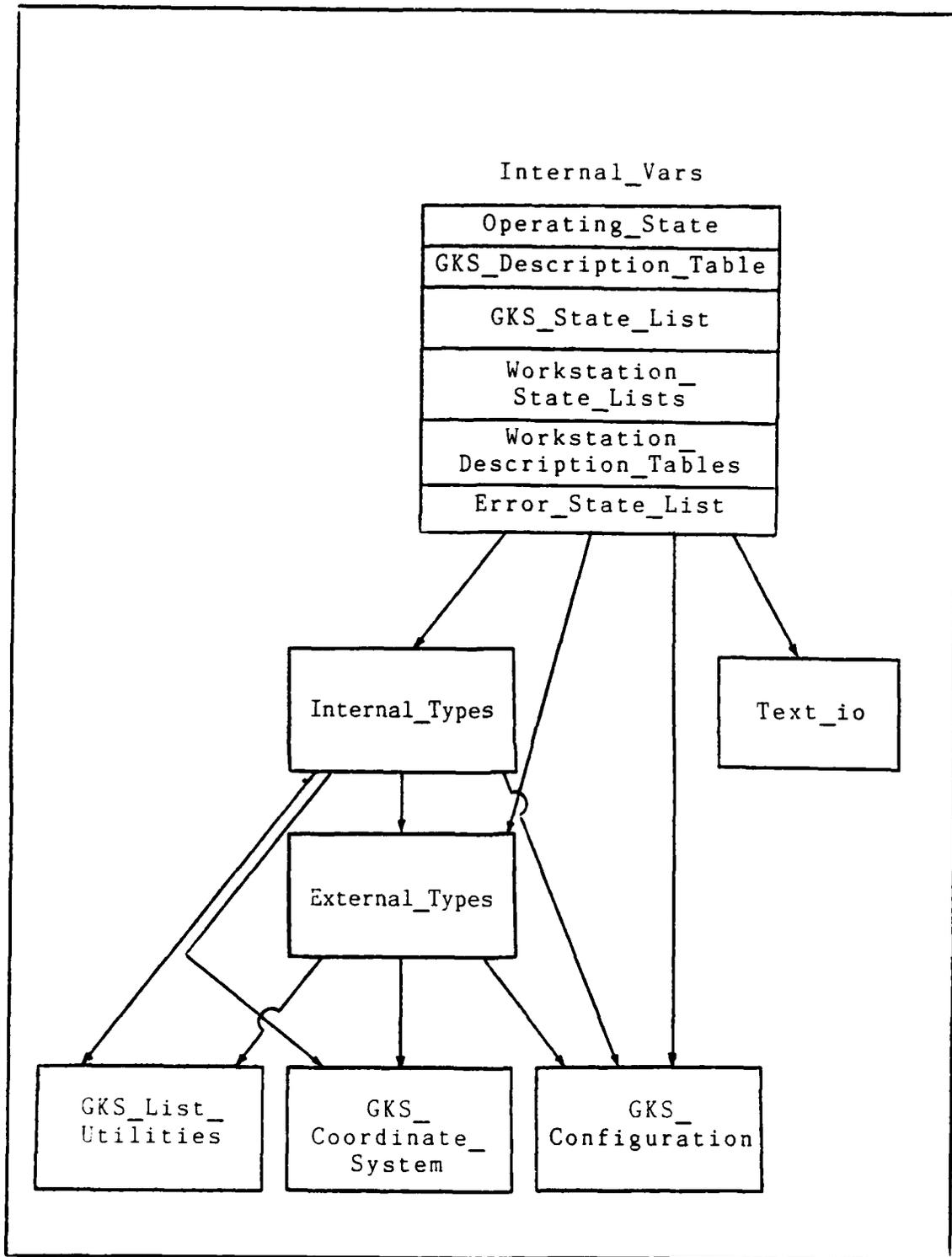


Figure 3.3. Data Structures and Types

holds the operating state of AFIT_GKS. The possible values of the Operating_State as shown in Figure 3.4 are GKCL (GKS is closed), GKOP (GKS is open), WSOP (at least one workstation is open), WSAC (at least one workstation is active), and SGOP (a segment is open). As shown in Figure 3.4, various functions of AFIT_GKS change the Operating_State of AFIT_GKS. By using the Operating_State value, AFIT_GKS can determine if a function can or cannot be called in the present state of AFIT_GKS.

GKS_Description_Table. The second structure of Internal_Vars shown in Figure 3.3 is the GKS_Description_Table(1:198). This data structure holds the constant values associated with AFIT_GKS. It contains the maximum range of abilities of the AFIT_GKS implementation. As such, it can be either a permanent constant in AFIT_GKS or it can be created when necessary by reading the information off of a file. My suggestion is to have it be a permanent part of the AFIT_GKS code as shown in Figure 3.5 since it is needed every time AFIT_GKS is used.

As shown in Figure 3.5 the GKS_Description_Table contains the level of gks implemented, the list of available workstation types, the maximum

```
CURRENT_LEVEL : constant GKS_LEVEL := ma;
CURRENT_LIST_WS_TYPES : WS_TYPES.LIST_OF;
CURRENT_MAX_OPEN_WS : constant POSITIVE := 2;
CURRENT_MAX_ACTIVE_WS : constant POSITIVE := 2;
CURRENT_MAX_SEGMENT_WS : constant POSITIVE := 2;
CURRENT_MAX_TRANSFORMATION_NUM : constant POSITIVE :=
    MAX_TRANSFORMATION_NUMBER;
```

Figure 3.5. GKS_Description_Table

number of open and active workstations, the maximum number of workstations that can be associated with a certain segment, and the maximum

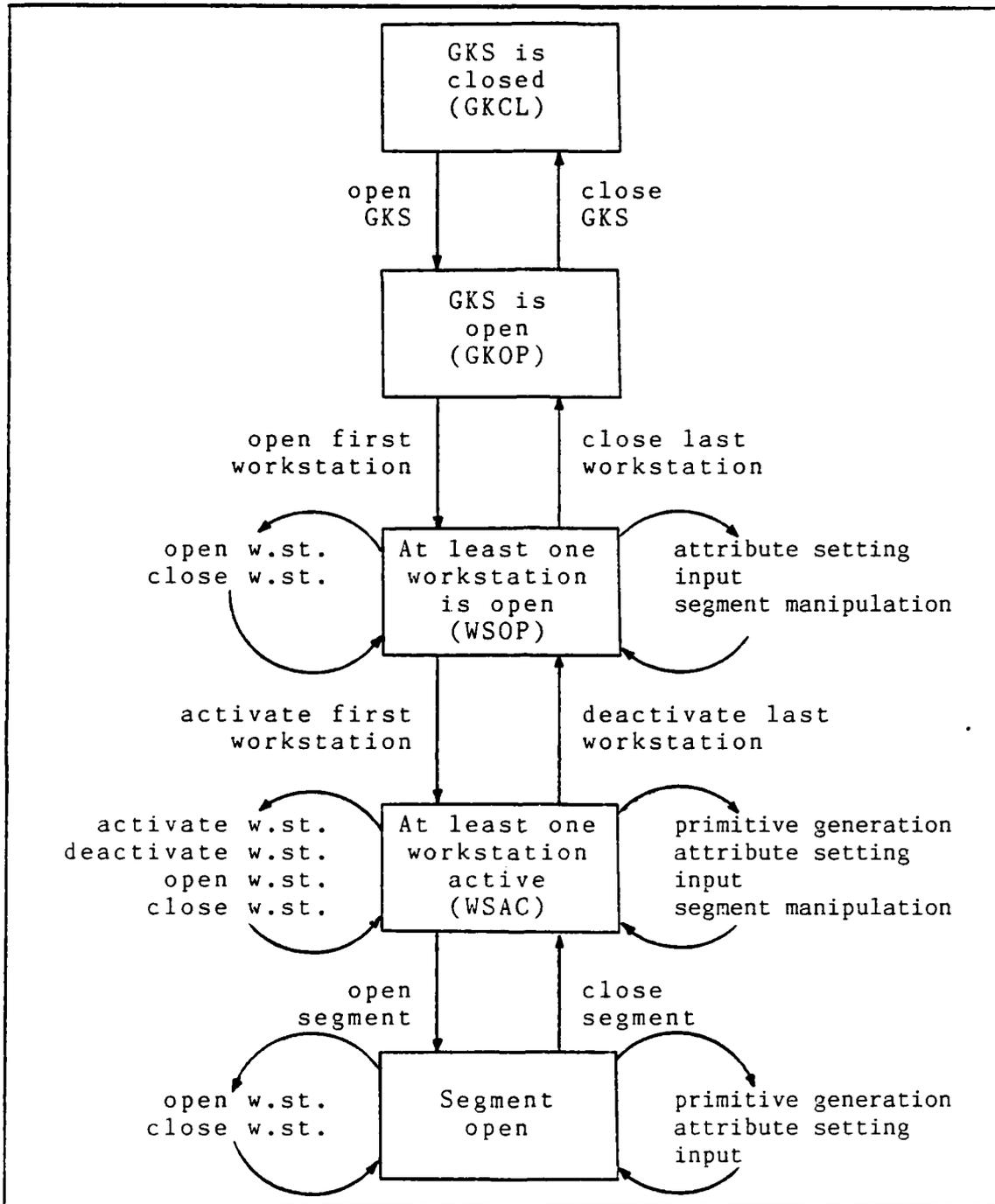


Figure 3.4. Possible Values of Operating_State (1:70)

number of transformations allowed. Overall, the GKS_Description_Table holds the maximum values of various parts of the GKS_State_List.

GKS_State_List. The third structure in Internal_Vars as shown in Figure 3.3 is the GKS_State_List(1:199-200). This structure holds the variable values associated with AFIT_GKS. It is implemented as a list of variables similar to the GKS_Description_Table except that the GKS_State_List variables are declared as variables which in many cases have a maximum value which is specified in the GKS_Description_Table (see Figure 3.5). These variables like the GKS_Description_Table are all prefixed by the word 'current' so that the maintainer of AFIT_GKS can tell where the variables came from.

The GKS_State_List contains the current open and active workstations. It also contains all the current attributes of any polyline, polymarker, text, or fill area that the user invokes. In addition the GKS_State_List contains all the normalization transformation that convert points from world coordinates to normalized device coordinates. Also, the GKS_State_List holds all the information concerning segments, like if they are visible, detectable, or highlighted.

Workstation_State_Lists. The fourth data structure of Internal_Vars shown in Figure 3.6, each workstation has its own Workstation_State_List node. The Workstation_State_Lists are complex structures because they must be accessed given a workstation_id, or a type_of_ws. A workstation_id is the name that the user associates with a workstation when it is opened. A type_of_ws is a permanent name that the implementor of AFIT_GKS associates with a workstation that can be used in AFIT_GKS. As shown in Figure 3.6, the information associated with the Workstation_State_Lists are contained in the Workstation_State_List nodes which are

The Workstation
State Nodes

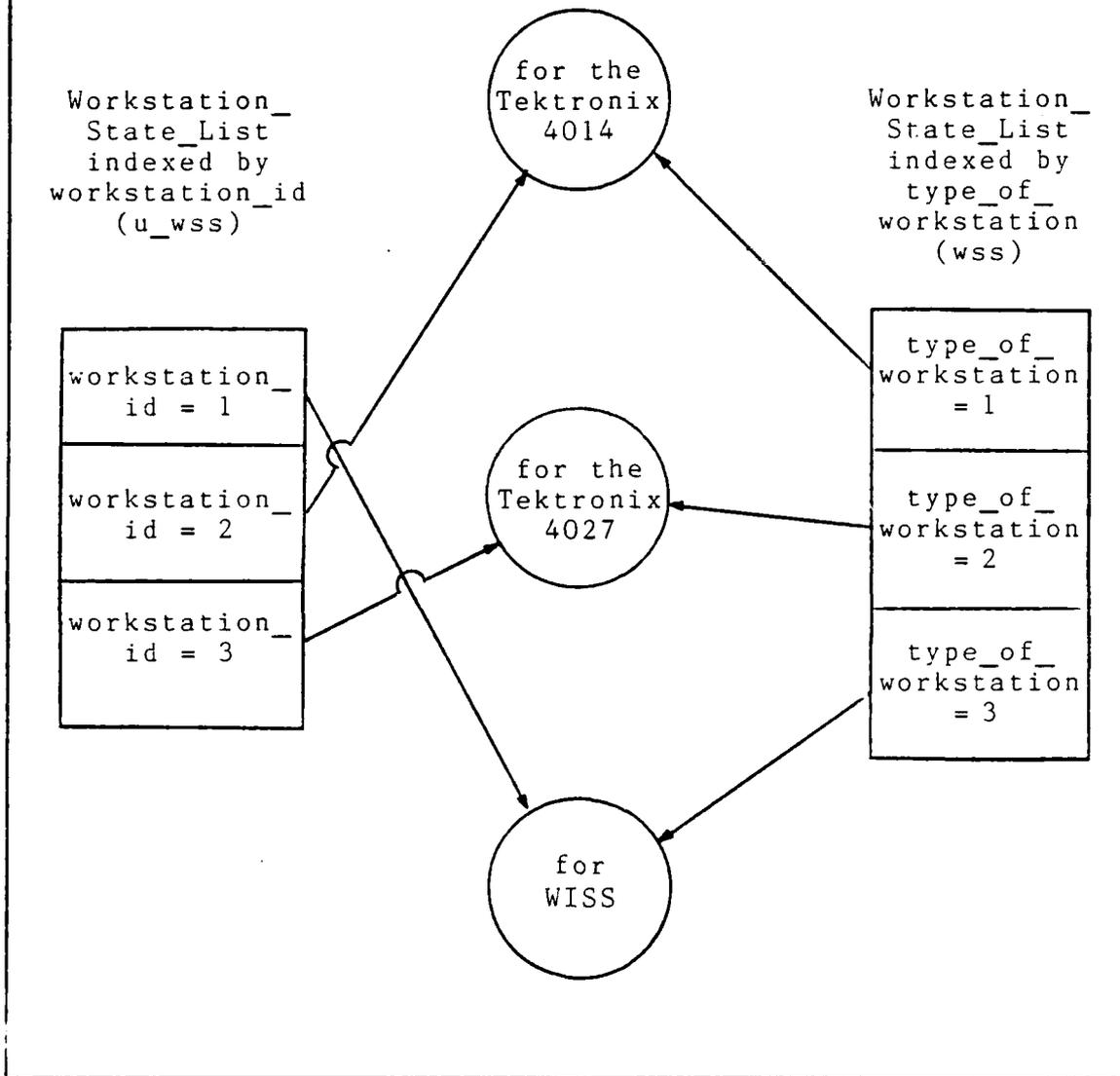


Figure 3.6. The Workstation_State_Lists

workstations. After explaining why the first design was rejected, this section explains in detail the implemented design.

The Original Design. The original design of the Workstations was centered around making the primitive functions as fast as possible. Whenever a primitive call was made to AFIT_GKS the workstation would transform, clip, and output the primitive to the display surface. The primitive function would not need to be concerned with its attributes since the workstation would take care of the attributes whenever an attribute was changed by the user using a Set_Attribute function. This is a nice concept, but it is not as simple as it looks.

First, if the primitive function does not check if its attributes are set properly on the workstation, then whenever a new attribute is set, the attribute function will have to notify each of the active workstations of the change. Second, the workstation must store the current attribute values of the different primitive functions. This is because a polyline might have a color of red while at the same time a fill_area primitive might have a color of blue. Therefore, if a polyline function is called then the workstation must make sure that the line segments are red, and if a fill_area function is called the workstation must make sure that the fill_area is blue. The GKS Data Structures hold the values of the attributes. So, the GKS Data Structures can hold the attribute values of the primitive to be output, and an internal structure will hold the values which tell the workstation which primitives attributes are currently set on the workstation.

The design to this point is shown in Figure 3.7. The workstation

Then the procedure `ERROR_OUTPUT_INDEX_WS` will check error 61 with a polyline index of 2, error 63 with a linetype of 22, and error 87 with a color of 7. It will return in 'error_number' the first error found or a zero if no errors occurred.

Overall, these four types of error routines handle all the errors of `AFIT_GKS`.

Reporting an Error. Finally, how should a procedure in `AFIT_GKS` report the error? As specified in `ANS GKS`, a procedure, other than an inquiry procedure, reports an error by calling the procedure error handling with the error number, and the name of procedure which had the error(1:73). An inquiry procedure simply returns the error in its output parameter 'ei' (error_indicator). In addition, the procedure which called `error_handling` should raise the appropriate user exception and allow the user to handle the error. This permits a procedure to abruptly stop processing and allow the user to handle his/her error.

Now that the Error functions have been explained, Workstations will be discussed next.

Workstations

The Workstations shown in Figure 3.1 are a major part of `AFIT_GKS`. The Workstations are where all the graphical information is output to the graphical devices. In `AFIT_GKS`, there are two types of workstations (`OUTIN`, and `WISS`). The `OUTIN` workstations are the set of device dependent code that performs the `AFIT_GKS` functions on the graphical devices. The `WISS` workstation works exclusively on segments.

First, this section explains the original design of the `AFIT_GKS`

tributes but only a few different types of parameters all of which can be converted to type integer. Therefore, an example error routine would be:

```
procedure ERROR_OUTPUT_INDEX_WS (error_numbers: in error_indicators.  
list_of; range_numbers: in error_indicators.list_of; error_number: out  
error_indicator; id: ws_id);
```

errors checked:

- 61 A representation for the specified polyline index has not been defined on this workstation
- 63 Specified linetype is not supported on this workstation
- 65 A representation for the specified polymarker index has not been defined on this workstation
- 67 Specified marker type is not supported on this workstation
- 69 A representation for the specified text index has not been defined on this workstation
- 76 A representation for the specified fill area index has not been defined on this workstation
- 80 Specified hatch style is not supported on this workstation
- 82 A representation for the specified pattern index has not been defined on this workstation
- 83 Interior style PATTERN is not supported on this workstation
- 87 A representation for the specified colour index has not been defined on this workstation.

An example of how to call ERROR_OUTPUT_INDEX_WS is as follows.

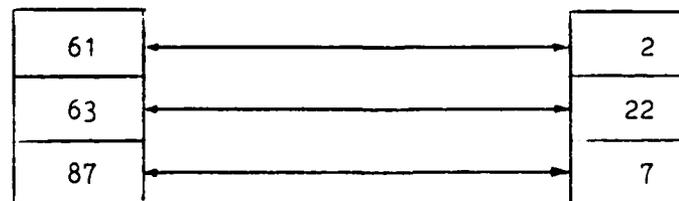
Given that the following errors are to be checked:

- error 61 with a polyline index of 2.
- error 63 with a linetype of 22.
- error 87 with a colour of 7.

The two lists, error_numbers and range numbers, must be defined as:

error numbers:

range numbers:



3. The third type of error routine checks a series of error numbers for some highly related errors. For example,

```
procedure ERROR_WS_IS (error_numbers: in error_indicators.list_of;  
error_number: out error_indicator; id: in ws_id);
```

errors checked:

- 24 Specified workstation is open
- 25 Specified workstation is not open
- 29 Specified workstation is active
- 30 Specified workstation is not active
- 31 Specified workstation is of category MO
- 32 Specified workstation is not of category MO
- 33 Specified workstation is of category MI
- 34 Specified workstation is not of category MI
- 35 Specified workstation is of category INPUT
- 36 Specified workstation is Workstation Independent Segment Storage
- 37 Specified workstation is not of category OUTIN
- 38 Specified workstation is neither of category INPUT nor of category OUTIN
- 39 Specified workstation is neither of category OUTPUT nor of category OUTIN
- 40 Specified workstation has no pixel store readback capability

ERROR_WS_ID accepts a list of error numbers (of length 1 to 14), which are to be checked. It returns the lowest numbered error (in error number) which is true, otherwise, the error_number returned is set to 0. For example, if error_numbers was the list

24	30	35
----	----	----

then ERROR_WS_ID would consecutively check errors 24, 30, and 35.

4. The fourth type of error procedure in AFIT_GKS concerns those error routines that need to check an input parameter whose type can be converted to 'integer'. Some of the parameters are polyline indexes while others are polymarker indexes. Here there are many different at-

```
function ERROR_AT_LEAST_1_PT (pt: points.array_of) return boolean;
```

errors checked:

100 Number of points is invalid

ERROR_AT_LEAST_1_PT checks to see if there is at least one point in the input parameter. This function returns a true value if there are no points in 'pt' and a false value otherwise. This particular error routine is called by the output primitive polymarker to check if at least one marker is defined.

2. The second kind of error routine checks the error which corresponds to an input error number. For example,

```
function ERROR_STATES (state : in error_number) return boolean;
```

errors checked:

- 1 GKS not in proper state: GKS shall be in state GKCL
- 2 GKS not in proper state: GKS shall be in state GKOP
- 3 GKS not in proper state: GKS shall be in state WSAC
- 4 GKS not in proper state: GKS shall be in state SGOP
- 5 GKS not in proper state: GKS shall be in either state WSAC or in state SGOP
- 6 GKS not in proper state: GKS shall be in either state WSOP or in state WSAC
- 7 GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP
- 8 GKS not in proper state: GKS shall be in one of the states GKOP, WSOP, WSAC or SGOP

Any call to error_states checks any one of the eight error codes. That is acceptable for this function because no user routine needs to check two of these errors. This function will return a true if the GKS state is in error, otherwise it will return a false. This particular function is called from all of the AFIT_GKS functions except inq_operating_state_value.

to the highest error number, these error functions could easily be programmed without needing to know what procedure called them. This idea would help in error handling but still the implementor of AFIT_GKS would need to make a series of error calls, one for each set of errors that he/she had.

AFIT_GKS uses an error package (shown in Figure 3.1) which contains a combination of the last three different ways of checking errors. These three methods are as follows:

1. For some of the errors, AFIT_GKS writes a single function that tests one particular error.
2. AFIT_GKS has some error functions that test the error which corresponds to an input error number.
3. The last type of AFIT_GKS error function, tests a series, or list of errors and returns the first error found (or a 0 if no error was found).

Overall, this combination of different types of error functions seems like the best solution since it solves the problem of interfacing to the error routines and yet it at least reduces the number of calls to the error routines. Valid input data is checked by calling error routines which specifically check the given input data. These routines are rarely called because the Harris binding restricted all of the input parameters to valid input ranges.

Implementation of Error Functions. Using the three types of error routines described above AFIT_GKS implemented its error routines using one of the following four methods.

1. The first way to check errors is to write a function that checks one specific error. For example,

parameters for various procedures. This would cause a nightmare of overloading of a procedure so that all of the different procedures could be supported by this one error function name. If this solution was taken then every AFIT_GKS function would call say check_errors several times using all different parameter lists so that all its errors could be checked. If this was done then the maintainer of AFIT_GKS would have a difficult time trying to figure out which of the fifty or so check_errors procedures was called in any function of AFIT_GKS.

Another way of handling the errors is to pass a single error procedure the name of the procedure that called it and let it handle the error checking. This still involves the problems of how to pass the input parameters so that they can be checked.

Another method is to have a separate procedure for each individual error. Each procedure would check for one of the specified errors of AFIT_GKS and return a true or false depending on whether or not an error occurred. This would allow for each error procedure to have a proper interface so that it can check the input values. The only problem with this method is that each procedure will have to call several different error procedures before it can determine if it has any errors.

Expanding on the method of using one procedure for each error condition, some of the error procedures could be combined, like those with identical parameters and similar algorithms. This would cut down on the number of error procedures but would not reduce the number of calls to error functions that each procedure must do.

Next suppose that some of these error functions could check a series of error conditions. Since all errors are checked from the lowest error number (for a list of all the error numbers in AFIT_GKS see Appendix A)

shows how the accepted method of implementing errors was realized in AFIT_GKS. Finally the section explores how errors are reported in AFIT_GKS.

Order of Checking Error Numbers. Another design consideration is how to order all the errors that occur in the calling of AFIT_GKS procedures by the user. According to the ANS GKS standard all GKS procedures check on entry (in the following order):

1. That GKS is in the correct state;
2. That the values of input parameters are valid.

At least the first error detected is reported(1:73). To accomplish this goal, AFIT_GKS will check errors from the lowest numbered error (see Appendix A for a list of the error numbers) to the highest numbered error. By the design of ANS GKS this will entail checking the GKS states first, then the workstation states, and finally all the input parameters. This solves the problem of how to order these error functions in a simple concise manner.

Design Alternatives. But the problem still remains as how to design the checking of the different errors.

One way is to have each procedure check all of its errors, directly calling on the inquiry functions whenever it needed information from the data structure in order to determine if there was an error. This would be a bad solution because of the redundant code that each procedure would have to do in order to check its errors.

Another possibility is to have several error procedures all with a single common name (called "overloading" in Ada) which together check the errors of all the different procedures. The problem with this approach is that the error checking must check all the input parameters. Therefore, this error checker would need a different amount and type of

GKS_List_Uutilities. The second generic package shown in Figure 3.3 is GKS_List_Uutilities which allows for arrays, and matrices (both fixed length and variable length) to be defined. GKS_List_Uutilities (whose specification can be found in Appendix B) is used extensively to provide lists of various objects, like the list of active workstations in AFIT_GKS.

This package does have a problem on the ROLM Data General Ada compiler that this project is using. The problem is that the generic parameter must be of a fixed length. Therefore, it would not accept a generic parameter of a variable length string.

This package also has functions which add an item to a list, delete an item from a list, and test if an item is in a list.

GKS_Configuration. GKS_Configuration shown in Figure 3.3 defines all the various maximums of type declarations found in AFIT_GKS. For example, in GKS_Configuration (the specification of GKS_Configuration can be found in Appendix B), the constant max_raster_units defines the highest raster unit found on any workstation in AFIT_GKS.

Error Routines

The Error routines shown in Figure 3.1, contain procedures and functions that test all of the various errors that can occur when an AFIT_GKS User Function is called. A complete listing of all the AFIT_GKS errors can be found in Appendix A.

First, this section will explore the order in which the various errors need to be checked. Second, this section will discuss the many different ways that the error routines could be implemented. Third, it

and therefore should be classified as belonging to the package External_Types. For example, large_ndc types are exactly like the External_Types ndc except that the range of permissible values is larger than that of type ndc. Prefixing the types with "e_" allows the maintainer of AFIT_GKS to quickly recognize whether the type should be found in External_Types or in Internal_Types. Finally, many of these types, like the record which defined a polymarker bundle table entry, must be made into a list (for the polymarker bundle table) by using the generic package GKS_List_Uilities. Their names are always the name of the generic parameter plus an "s". For example:

```
package i_seg_names is new GKS_LIST_UTILITIES (i_seg_name);  
package e_input_q_entrys is new GKS_LIST_UTILITIES (e_input_q_entr  
entry);
```

Note, that it is not necessarily the plural of the generic parameter.

GKS_Coordinate_System. The GKS_Coordinates_System shown in Figure 3.3 allows the program to define points, vectors, sizes and rectangles of any floating type variable. This generic package (whose specifications can be found in Appendix B) is used to create the World Coordinates (wc), Normalized Device Coordinates (ndc), Large Normalized Device Coordinates (large_ndc), and the Device Coordinates (dc). It uses a generic parameter which allows various instantiations of GKS_Coordinate_System using the different coordinate types.

In addition to defining the different types, this package also defines a vector_length operation. This is a function that, given a vector, returns the length of the vector.

these extra functions which could access the private types were not written and the input data structures were not made private. Although the private structures are not presently in AFIT_GKS they may be a nice addition to AFIT_GKS because then the maintainer of AFIT_GKS can change some private data structures implementation without having to worry about what part of AFIT_GKS accessed it.

Internal_Types. As shown in Figure 3.3, Internal_Types is mainly used to define the types needed in Internal_Vars. This package contains the types for binding attributes, segment storage, transformations, input queue entries, Workstation_State_Lists, Workstation_Description_Table, and the error indicators. In addition, AFIT_GKS defined a new type in the package Internal_Types called large_ndc_type. This type allows the ndc points in segment storage to lie outside of the range 0.0..1.0. This is useful when later applying segment transformation, or the WISS function `insert segment(1:44)`.

One problem with Internal_Types is that the Ada compiler used for this project would not allow variant records to be used in pointer lists, or in arrays. Also, a field name in the variant part could not have the same name as in another case of the same variant part. Therefore, this package does not use variant records but simply shows where they should be put if the compiler would accept them.

In the Internal_Types package, type names had to be created. Therefore, this implementation used the prefix "e_" (enumeration of) before any type declared in Internal_Types. The only exceptions to this rule is variable_string, large_ndc types (discussed above), and package dc points, which can be found in Internal_Types. The reason for these exceptions is that they all closely mimic the types found in External_Types

Types given by the Harris binding. There are three differences.

1. The Harris binding specifies that

```
subtype positive_scale_factor is scale_factor range scale
factor'safe_small..scale_facator'safe_large; (2:46)
```

The ROLM Data General will not compile this statement. Therefore, it was changed to "range 0.00001..1.0E50;" which is the value requested.

2. The second change is that the Harris binding requires that AFIT_GKS be a generic package. This is so the user can specify the format of choice_value, input_value, pick_ids, segment_names, world coordinate types, and workstation ids. This is a nice concept but it is unrealistic to have the entire AFIT_GKS project in one generic package, because the ROLM Data General would require that the generic package be contained in one text file which would be impossible to edit. The reason the entire code would have to be in the one generic package is that if External_Types was a generic package then the only piece of code that could use its variables would have to be a part of the actual generic package External_Types (11:10.2.1). This implementation solved the problem by only allowing the user the default types for Harris's generic parameters.

3. The Harris Binding defined the six input data records as private. An input data record holds any values needed to inquire an input device. For example, the valuator input device which gets a floating point number which is inside a given range, has its input data structure contain the range that the floating point number can take on. If the input data records are private, then a series of functions must be written to access those private data types. Due to time limitations

of what the workstation is capable of doing. This includes what kind of workstation is it, how big is the display surface, and is it a raster or vector display. Moreover, the structure stores the line types, line widths, and the colors that the workstation can display. The table holds the marker types, text fonts, character heights, character widths, interior styles, and hatch styles. In addition to handling all the capabilities of the workstation, the GKS_Description_Table nodes also hold the default values for the Workstation_State_Lists. This includes all the default bundle tables and information about the input functions. Finally, the Workstation_Description_Table nodes contain a list of variables which state whether the workstation can perform a certain action with or without a redrawing of the display surface. These actions include changing the various bundle tables, deleting segments, or changing colors on the workstation.

Error_State_List. The sixth data structure in Internal_Vars as shown in Figure 3.3 is the Error_State_List (1:209). This structure holds the information concerned with an error. This structure contains the error state, and the error file. The error file causes the Internal_Vars package to use the package Text_io, this is because the error file is implemented as file_type which is not a type defined in Ada; it is a private type defined in the package Text_io (2:421-42). In order to highlight these variables in AFIT_GKS they are all prefixed with the word "error_." For example, the error file is named "error_GKS."

External_Types. This package shown in Figure 3.3 holds all the types that the user needs to interface with AFIT_GKS. This package shown in Appendix A was specified in the Harris Binding.

AFIT_GKS does not completely adhere to the specification of External_

pointed to by the two different Workstation_State_Lists. Therefore, the Workstation_State_Lists are implemented as two arrays indexed on the range of the workstation ids and workstation types. As shown in Figure 3.6 the array indexed by the workstation id is the variable "u_wss," while the array indexed by the workstation type is the variable "wss."

Note in Figure 3.6 that the "u_wss" can be defined with any of the workstation ids pointing at any of the three different workstation types, but the pointers from "wss" can not be changed. The Tektronix 4014 is always of type '1' (type of workstation = 1), the Tektronix 4027 is always of type '2' (type of workstation = 2), and Workstation Independent Segment Storage (WISS) is always of type '3' (type of workstation = 3).

The Workstation_State_List nodes contain the variables that are needed to deal with a workstation. This structure contains the deferral mode and whether the screen needs to be redrawn. It also contains all the bundle tables for the various output primitives. The Workstation_State_List nodes contain the workstation window and viewport which perform the normalized device coordinates to device coordinates transformations. Finally, this structure contains all the current information needed for the input functions.

Workstation_Description_Tables. The fifth data structure is the Workstation_Description_Tables (1:204-207). This is where the constant values associated with a workstation are stored. The Workstation_Description_Tables are implemented just like the Workstation_State_Lists shown in Figure 3.6. For the Workstation_Description_Tables the two arrays shown in figure 3.6 are "u_wsd," and the "wsd" which point to the Workstation_Description_Table nodes.

The Workstation_Description_Table nodes are simply a structured list

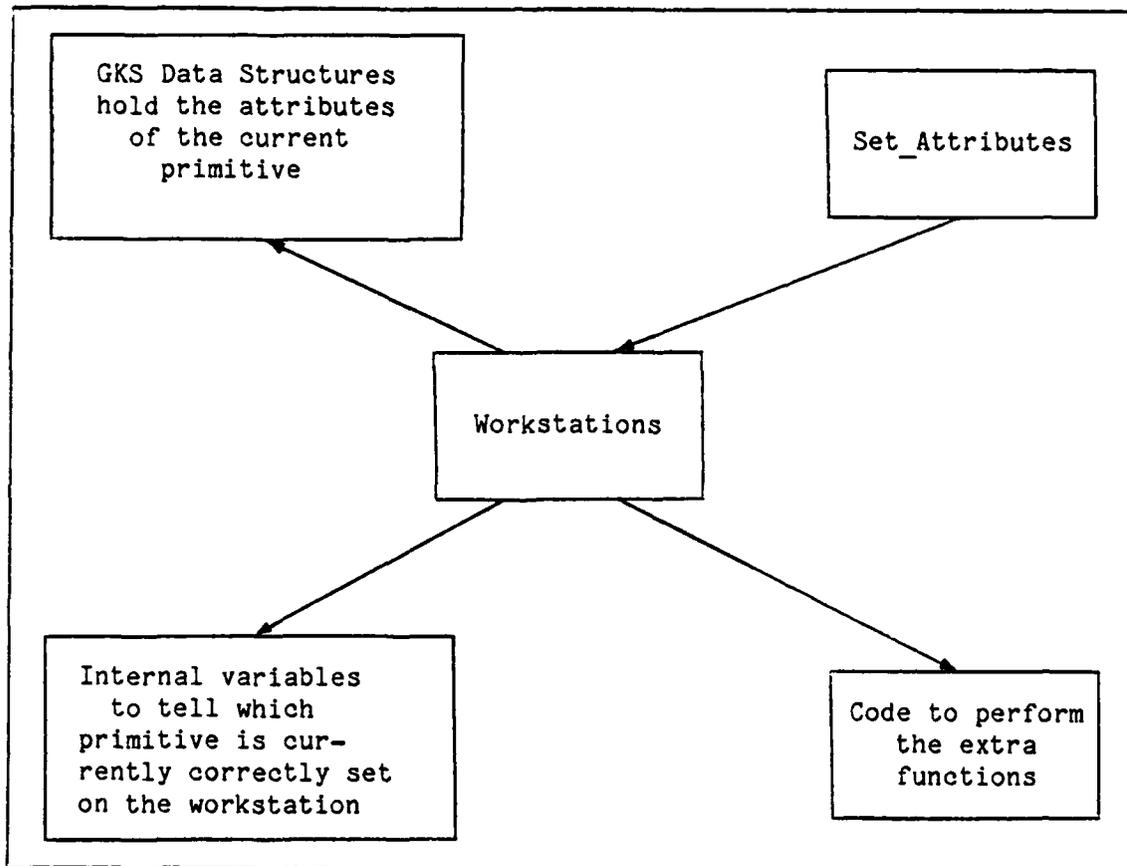


Figure 3.7. Original Design of Workstations

must communicate with the Set_Attribute functions, it must access its internal variables and the GKS Data Structures, and it must have some additional code to perform all the setting of attributes.

Finally, segments were considered using this design. Segments contain a series of primitives and their attributes. Segments are stored on the workstations, and are used to modify displayed images. As such they can be redrawn at any time. This being the case how can this design handle segments? The segment attributes can not be used with the GKS

Data Structures, because that would destroy the current values of the attributes. So another internal structure is needed to handle the attributes of segments. At this stage, this design was determined to be "bad" because of all the extra variables and code needed to implement it. Therefore, the original design was discarded and the design that was eventually implemented is discussed next.

Implemented Design. The implemented design of AFIT_GKS used a slightly modified data flow chart of ANS GKS as shown in Figure 3.8. First, the primitive functions perform the transformation from world coordinates (wc) to normalized device coordinates (ndc). Second, the active workstation is called, where the clipping rectangle is stored with the primitive. The clipping rectangle is the current viewport. When clipping is applied, any graphical information that is not contained in the clipping rectangle will be clipped (discarded). Third, the current attributes of the primitive are bound to the primitive. For example, a polyline primitive is no longer considered as a list of vertices; it is now a dotted, thick, red list of line segments which have a clipping rectangle equal to the current viewport.

Fourth, following Figure 3.8 down the path of the OUTIN workstations, the workstation tests if a segment is open. If a segment is not open then the primitive goes on to the seventh step bind attributes, but assuming a segment is open, the fifth step is to store the primitive in Workstation Dependent Segment Storage. Workstation Dependent Segment Storage is a storage area for segments. Sixth, the primitive is transformed by the segment transformation.

Seventh, the attributes of the primitive are "bound" to the display surface. This means that the attributes of the primitive are set on the

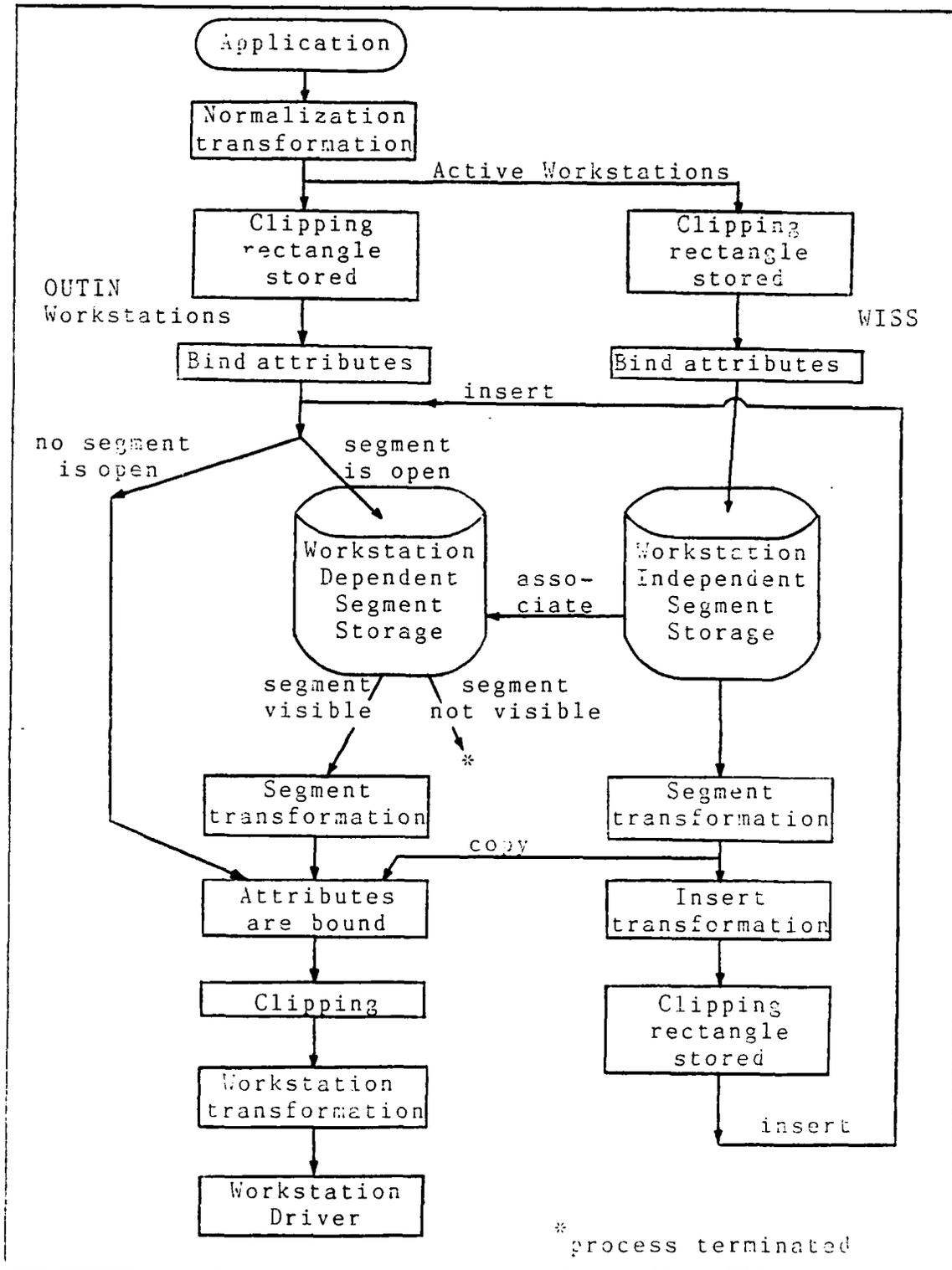


Figure 3.8. Modified Data Flow Chart of ANS GKS (1:47)

display. For example, the polyline primitive described in the third step would set, if necessary, the color on the workstation to red, the line type to dotted, and the line width to "thick". The attributes set are those that were bound to the primitive during `bind_attributes`. Eighth, the primitive is clipped against the appropriate clipping rectangle. The appropriate clipping rectangle is the intersection of the clipping rectangle stored in step two, and the window used in the workstation transformation. Ninth, the primitive is transformed from normalized device coordinates (`ndc`) to device coordinates (`dc`). Tenth, the Workstation Driver is called to output the primitive to the device.

The right side of Figure 3.8 shows WISS. As shown in Figure 3.8, WISS works essentially like the OUTIN workstations except that it doesn't affect a graphical device. Instead, WISS provides three functions, `insert`, `associate`, and `copy`, which can transfer information stored in WISS to any of the OUTIN workstations.

Using the ideas of Figure 3.8 AFIT_GKS designed the workstations shown in Figure 3.9. This design was considered the most critical part of AFIT_GKS. If this works well then the implementation has a good chance of working. If this is done poorly, then AFIT_GKS cannot possibly work.

The OUTIN workstations are split up into three packages called `Ws_x`, `Int_ws_x`, and `Drive_x`, where `x` is the type of workstation (`x = 1` for the Tektronix 4014, `x = 2` for the Tektronix 4027). `Ws_x` handles those functions that the device independent part of AFIT_GKS calls, like `draw polyline`, or `clear the screen`. `Int_ws_x` handles the internal functions of `Ws_x` like doing transformations, binding attributes, doing segment storage, and clipping line segments. `Drive_x` puts out the actual ASCII

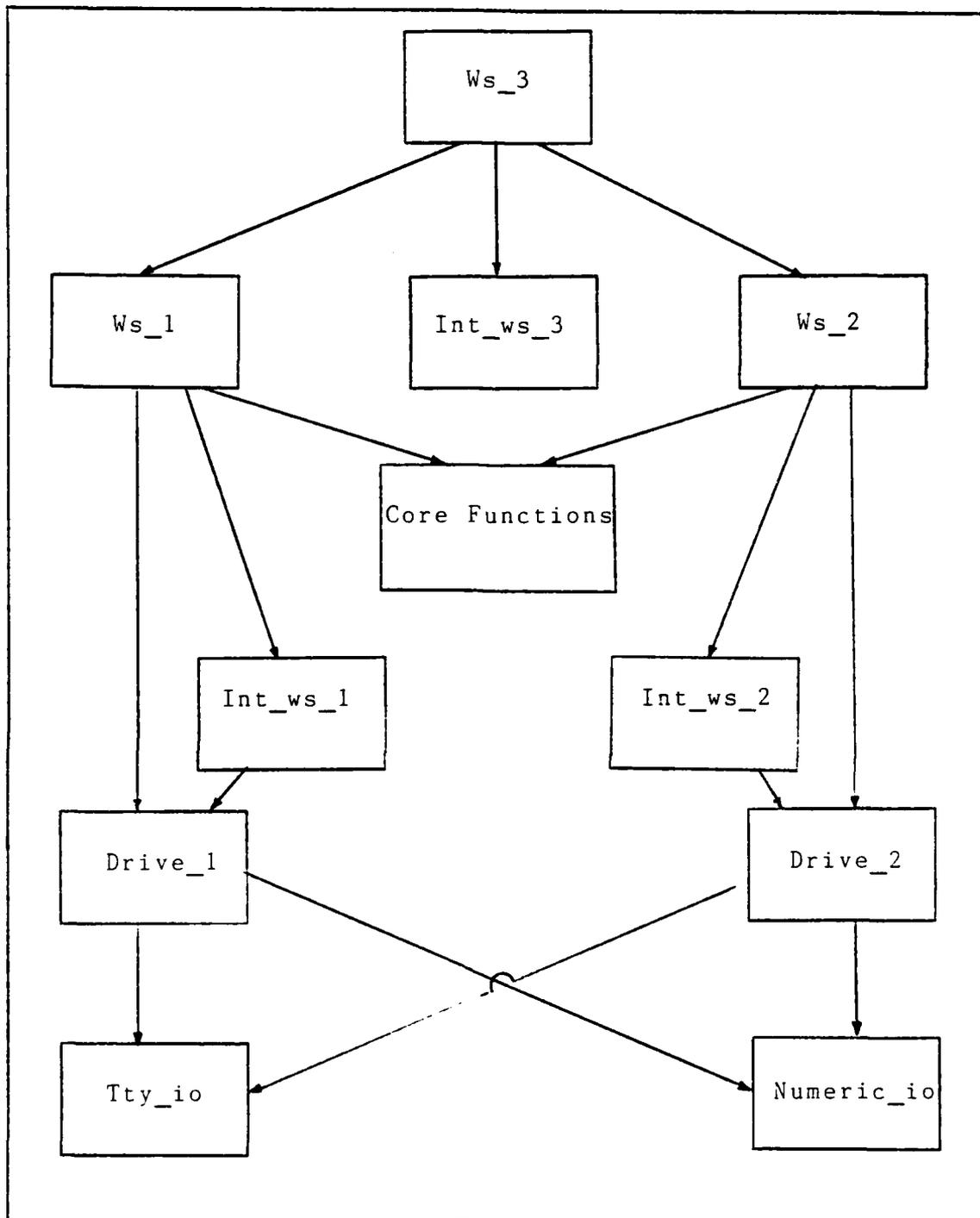


Figure 3.9. Workstations

characters which make the given OUTIN workstation perform. It draws line segments, puts out text and gets input from the graphics device. It is only called from Ws_x and Int_ws_x.

The WISS workstation only works on segments. Whenever it is active it stores any segments being created. In addition, WISS has three functions, associate, copy, and insert. In AFIT_GKS, these functions move segments from WISS to a given OUTIN workstation.

OUTIN Workstations

Ws_x. The package Ws_x shown in Figure 3.9, performs three kinds of functions. It processes the workstation primitive functions, handles the various segment functions (like displaying a segment), and performs all the major input functions. Overall, this package is the primary way for the AFIT_GKS package to access the workstation.

Workstation Primitive Functions. The workstation primitive functions perform the OUTIN side of the data flow chart shown in Figure 3.8. Each primitive function has its own set of code, which performs the operations shown in Figure 3.8. As an example, this section will show how a polyline primitive is implemented in AFIT_GKS. The other primitive functions are implemented the same way.

As shown in Figure 3.10, the polyline function is split into five procedures. These procedures, polyline, ws_polyline_x, ws_polyline_b_x, d_ws_polyline_x, and drive_x, line up in what is known as a pipeline where each procedure listed performs its function and then calls the next procedure in the pipeline. To better understand the relationships of the polyline function (Figure 3.10) and the data flow chart of ANS GKS (Figure 3.8), those two figures are superimposed to create Figure 3.11.

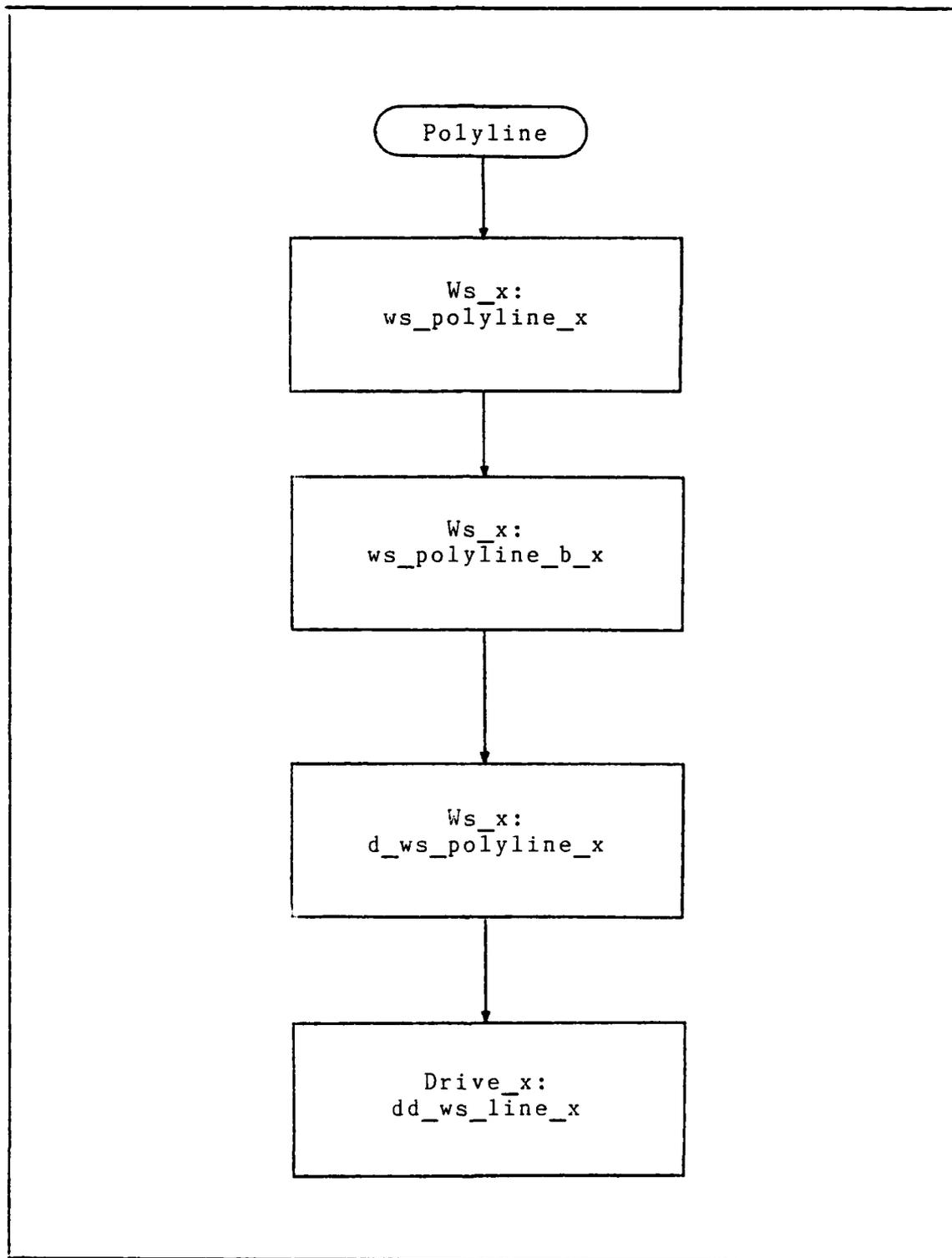


Figure 3.10. The Polyline Pipeline

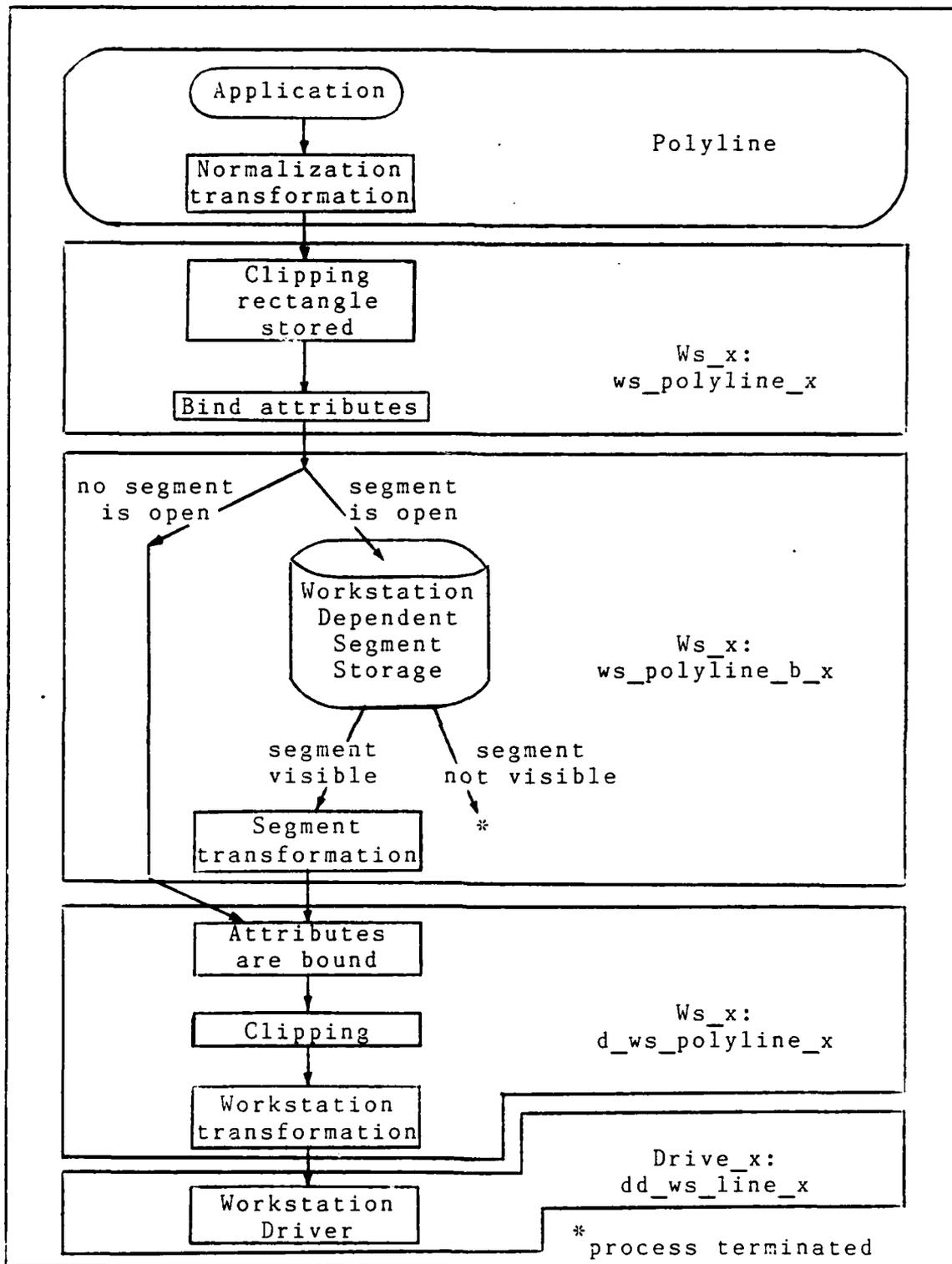


Figure 3.11. Polyline Pipeline Superimposed on the OUTIN side of the Modified Data Flow Chart of ANS GKS

The middle three procedures of the pipeline shown in Figure 3.10 are the workstation primitive functions. As shown in Figure 3.11 `ws_polyline_x`, gets the clipping rectangle for the polyline primitive and then binds the present attributes to the polyline. `Ws_polyline_b_1` handles segments. If a segment is open then this function stores the segment, and then it transforms the segment by the segment transformation and calls the next part of the pipeline `d_ws_polyline_x` if the segment is visible. If a segment is not open then `ws_polyline_b_x` calls `d_ws_polyline_x`.

Also shown in Figure 3.11, `d_ws_polyline_x` prepares the polyline for output to the screen. This routine does a series of actions to prepare the polyline for the device.

1. It binds the attributes to the output display.
2. It gets the proper clipping rectangle, which is the intersection of the clipping rectangle associated with this primitive and the workstation window.
3. The function then splits the polyline into separate line segments. As shown in Figure 3.11, for each of the separate line segments `d_ws_polyline_x` does the following.
 - A. It clips the line segment to the given clipping rectangle.
 - B. It transforms the line segment to device coordinates.
 - C. It calls the device dependent code in `Drive_x` to output the line segment.

The Segment Routines and Others. This workstation also allows for displaying of segments, and deleting of segments. The displaying of segments works by taking the primitives out of segment storage and modifying them so that they can be put back into the output pipeline. Segments being redrawn enter the pipeline at `d_ws_<primitive name>_x` just

after segment storage.

The other routines simply initialize the workstation and clear the display.

Input Routines. The input routines request various inputs from the terminal. In essence all of the input routines ask the device for the input along with a boolean variable which tells whether the value from the terminal was valid or not.

Int_ws_x. The package Int_ws_x shown in Figure 3.9 performs the internal functions of Ws_x. As such, Int_ws_x does transformations, stores primitives into the open segment, binds attributes to the workstation, and performs line clipping. This package is internal to package Ws_x. Therefore, all of the routines are used only by Ws_x. The only exception is the transform_wc_to_ndc of the package Int_ws_1 which is needed in the packages Ws_1, Primitives, and Set_Transform. Each of these packages needs to transform some points from world coordinates to normalized device coordinates.

The routines are simple in this package. The transform function performs a 2D transform on the given points. The functions that store primitives in segments create a segment node containing the primitive and then they store the node in the segment storage associated with the workstation. The routines that do the line clipping clip line segments.

The binding of attributes is a little more difficult. Here the bind attribute routines check each of the aspect source flags (asf) for any primitive. An asf, in essence, is a boolean variable which tells whether the attribute to be bound should be the one originally bound to the primitive, or the workstation dependent value found in the Workstation Bundle Tables. If the asf = specified, then the value originally bound to the

primitive is used on the workstation. If the `asf = bundled`, then the value given in the Bundle Table associated with the current bundle index (found in the `GKS_State_List`) is used on the workstation. As shown in Figure 3.12 the polyline function `bind_attributes` finds out what values to use on the workstation and then it calls routines which take the input value and set the workstation to that state, if necessary and possible. For example, as shown in Figure 3.12, suppose the polyline function `bind_attributes` calls `set_colour_x` with an input of blue. If the workstation is capable of color, then `set_colour_x` will check if the present color is blue, if it is not blue then it will call upon the device dependent routine `d_set_colour_x` which will set the color on the device to blue.

Drive_x. The package `Drive_x` shown in Figure 3.9 is the driver for the given OUTIN workstation. Here the actual device code for the different primitives are output to the "standard output" device. The actual devices interpret the "standard output" as specific commands and display the various graphical primitive objects. The input functions on `Drive_x` are not supported on all the workstations. Therefore, some of the input functions (like `valuator`) must be simulated by asking input from the keyboard rather than a given valuator device. A valuator device is commonly a potentiometer whose value is read by the graphics device and converted to a floating point number. In `AFIT_GKS`, a valuator is simulated by asking the user to input from the keyboard a floating point number.

WISS. `WISS` works exclusively on segments. It is defined pictorially in Figure 3.8. As shown in Figure 3.8, `WISS` must interface with the OUTIN workstations. Therefore the OUTIN primitives had to be designed so that the `WISS` functions could interface with them. Figure

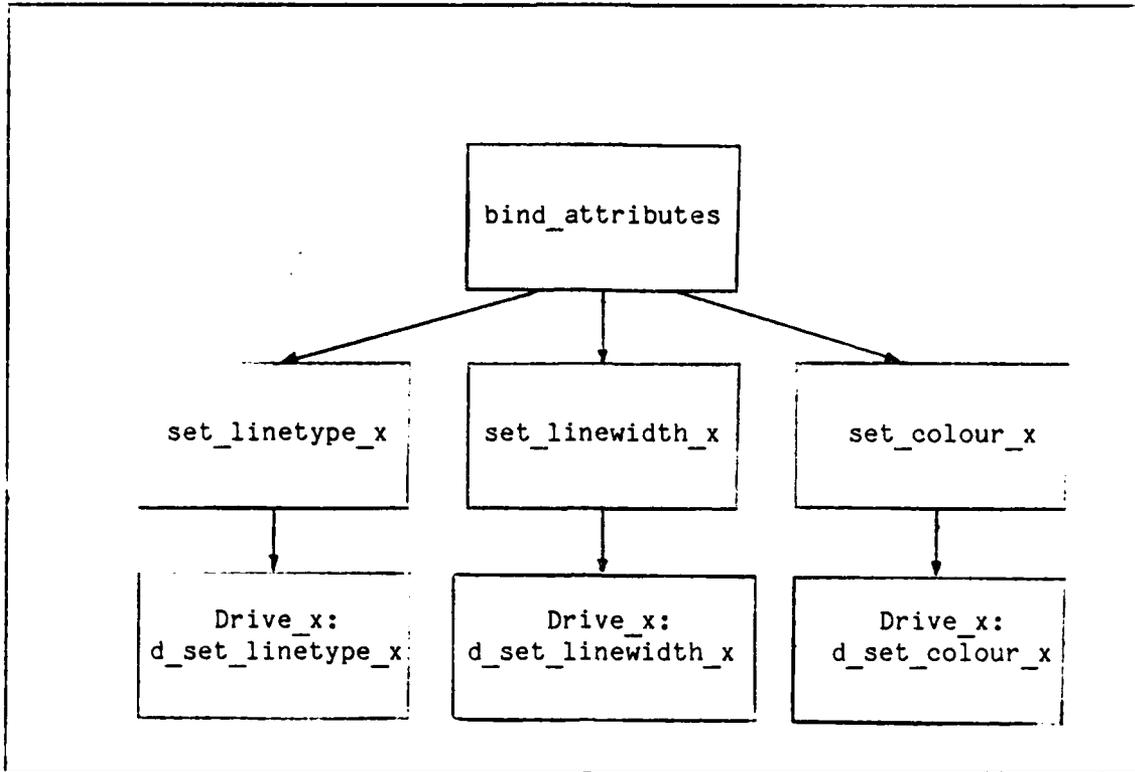


Figure 3.12. Polyline Attributes being Bound to Workstation x

3.13 shows how WISS interfaces with the polyline pipeline of an OUTIN workstation.

This interfacing with the OUTIN workstations is the only important part of WISS. If WISS could not interface with the OUTIN workstations, it would be useless. Therefore, the next three paragraphs will discuss the polyline pipeline shown in Figure 3.13, and how it interacts with WISS.

First, the routine `ws_polyline_x` in Figure 3.13 had to be written so that each of the primitives would get bound to its attributes when it was called by the primitive routines (those found in package Primitives).

The Polyline Pipeline
for Workstation x

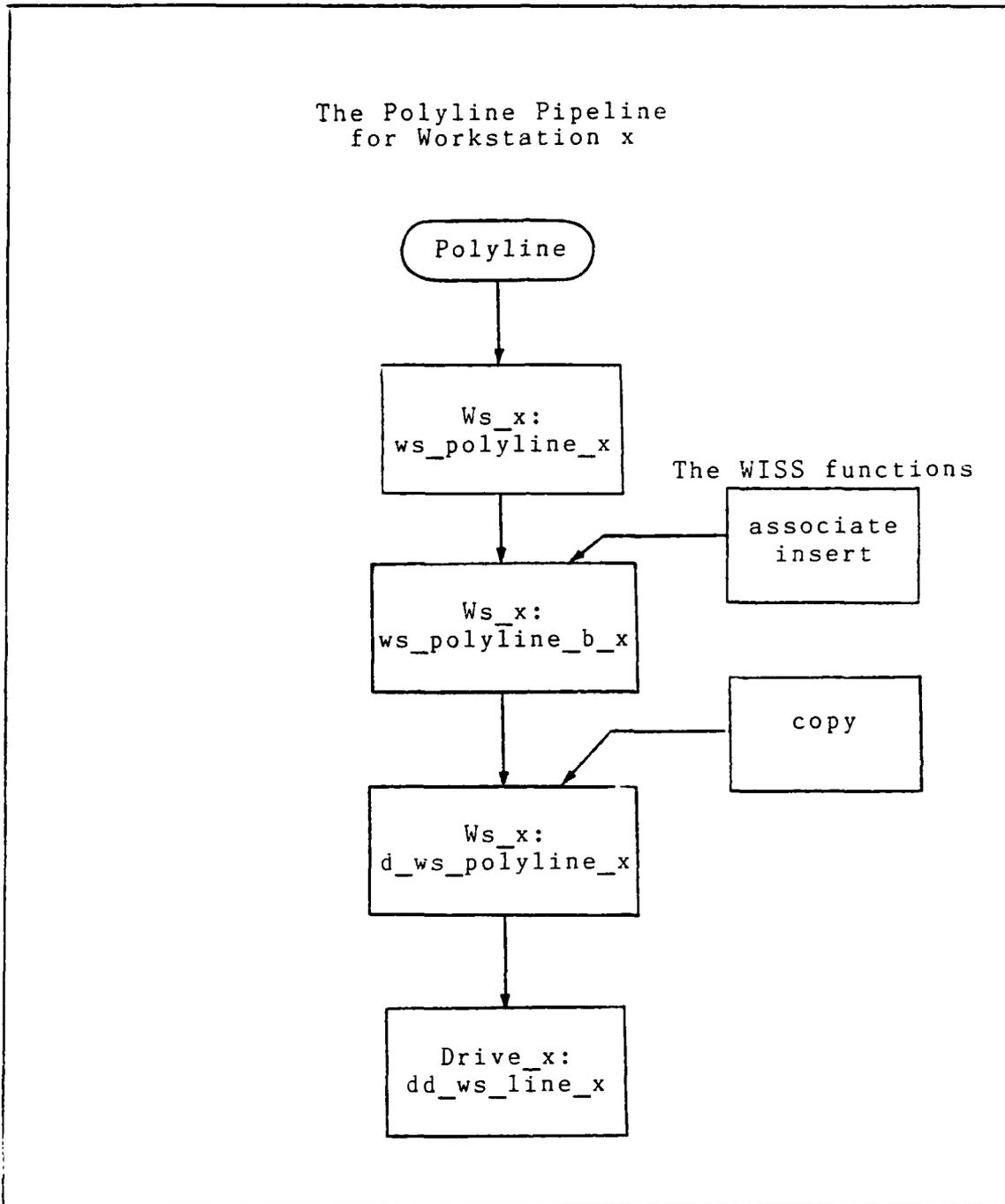


Figure 3.13. WISS interfacing with the Polyline Pipeline

Notice that WISS does not call `ws_polyline_x` because this would rebind the current attributes with the segment. ANS GKS states that once the attributes are bound to a primitive they are never changed(1:20).

Second, in Figure 3.13 `ws_polyline_b_x` handles segments for the primitive, and allows WISS to perform the associate and insert functions. The associate function copies the given segment to the appropriate OUTIN workstation in the same way as if the workstation were active when the segment was created(1:54). As shown in Figure 3.8, the insert function allows previously stored primitives to be transformed and again placed into the stream of output primitives(1:54). The procedure `ws_polyline_b_x` is needed because both the associate and insert routines need to access the segment storage capabilities of the workstation without rebinding the attributes of the workstation to a previously stored and bound segment.

Third, the `d_ws_polyline_x` routine creates the interface for the copy function. As shown in Figure 3.8, the copy function copies segments stored on WISS to the specified workstation(1:54). This is because copy needs to output a segment on a workstation but it does not allow segment storage or rebinding of attributes on the workstation.

Overall, WISS drove the design of the OUTIN workstations. The workstation had to do its operations in the order and with the separate functions described above. Otherwise WISS could not call on the workstation to do its three functions.

Global Considerations

The following four topics, device independent / dependent code, transformations, deferral states, and testing, do not fit into any of the

shown in Figure 4.4, is illegal in Ada. But, if the second "b : integer"

```
type x(id : character) is
  record
    a : float;
    case id is
      when 'a' =>
        b : integer;
        c : integer;
      when 'b' =>
        b : integer;
    end case;
  end record;
```

Figure 4.4. Illegal Type "X"

were "d : integer" then there would be no problem. If the above structure was legal then the `Workstation_State_Lists` and the `Workstation_Description_Tables` could be defined given the `ws_category`. For example, a workstation of type `WISS` needs a `Workstation_State_List` that contains the names of the segments associated with the workstation, but a workstation of type `OUTIN` needs the names of the segments, the deferral information, the bundle tables, the workstation transformations, and the input tables. Now if the variant record shown in Figure 4.4 was allowable then each `Workstation_State_List` would contain only those field names that the type of workstation needed, and the field names for the same information could be the same.

Conclusion. Overall, I want to comment that I like Ada. I think it is a good language which has to be implemented with more consideration given to the programmer debugging programs. Implementations of this language must be explored and tested with users so that the Ada compilers and support environments develop into a friendlier environment for the programmer.

Then I instantiated both types as variable length lists as follows:

```
package e_norm_transformations is new GKS_LIST_UTILITIES (e_norm_
transformation);
package e_seg_names is new GKS_LIST_UTILITIES (e_seg_name);
```

Then I defined the two variables, `current_trans` and `current_seg_state_list`, in the `Internal_vars` package as follows:

```
current_trans: e_norm_transformations.list_of;
current_seg_state_list: e_seg_names.list_of;
```

When I went to compile and run a program that used the `Internal_vars` package, the program bombed. Then I tried deleting the variable `current_trans`, and the program worked fine. Also, if I put the variable `current_trans` back in the package `Internal_vars` and removed the variable `current_seg_state_list`, then the program worked fine.

In the end I changed the maximum length to 50 and both variables worked fine in the `Internal_vars` package. I assume that the problem was that the compiler ran out of room when it tried to compile the two variables.

Undesirable Features of Ada on the ROLM Data General. Some things which were legal but I didn't like about using Ada on the ROLM Data General were as follows.

1. The long compile times of about 10 minutes per package slowed the project down a great deal.
2. Ada allows variant records but the elements in each part of the variant record must be unique. For example, the following structure

A. The first problem was that the compiler would accept variable length variables for the generic parameter, but again any program that used the generic instantiation that had a variable length parameter would cause the program to bomb. As suggested by another student, to solve this problem, I used pointers to the variable length objects, this worked fine.

B. The second problem with GKS_List_Uutilities was one of maximum length. I made the package have a maximum length of 1000. Then I defined the two records shown in Figure 4.3.

```
type e_norm_transformation is
  record
    priority : transformation_number := 0;
    transformation : transformation_number := 1;
    window : wc.rectangle :=
      (x => (min => 0.0, max => 1.0),
       y => (min => 0.0, max => 1.0));
    viewport : ndc.rectangle :=
      (x => (min => 0.0, max => 1.0),
       y => (min => 0.0, max => 1.0));
    a : float := 0.0; -- used in the (wc -> ndc)
    b : float := 0.0; -- transformation
    scalex : float := 1.0;
    scaley : float := 1.0;
    -- ndc_x_point := scalex * (wc_x_point) + a;
    -- ndc_y_point := scaley * (wc_y_point) + b;
  end record;

type e_seg_name is
  record
    segment : segment_name;
    ws : p_ws_ids; -- a pointer to the list of workstations
                -- associated with this segment
    transformation : transformation_matrix :=
      (1 => (1 => 1.0, 2 => 0.0),
       2 => (1 => 0.0, 2 => 1.0),
       3 => (1 => 0.0, 2 => 0.0));
    visibility : segment_visibility := visible;
    highlighting : segment_highlighting := normal;
    priority : segment_priority := 0.0;
    detectability : segment_detectability := undetectable;
  end record;
```

Figure 4.3. Normalization Transformation, and Segment Types

I tried to instantiate the package with the package DC shown below.

```
type DC_TYPE is digits 5;  
package DC is new GKS_COORDINATE_SYSTEM (DC_TYPE);
```

Again the compiler accepted the package instantiation but any program using a part of the instantiation DC would cause the program to raise an exception before executing the first line of code. This is because GKS_Coordinate_System requires a generic parameter which is a range of values (that is what the symbol "<>" means), and DC_Type is not a range of values it is a restriction on the precision of floating point numbers. To solve this problem I changed DC_Type to

```
type DC_Type is new float;
```

This new declaration of DC_Type worked on the ROLM compiler.

This brings up an interesting point, "type DC_Type is digits 5" is simply a floating point number with at least 5 digits of accuracy, and "type DC_Type is new float" is a floating point number without any given degree of accuracy. GKS_Coordinate_System, which does not concern itself with the accuracy of the generic parameter, should treat the two types identically, but it doesn't. Therefore, I believe that there is a bug in the ROLM compiler, because it would not accept "type DC_Type is digits 5" as a generic input parameter.

3. The next problem I had was with a generic package, GKS_List_Utilities (for a complete specification of the package see Appendix B), which creates variable length lists of the given private parameter.

Now, when procedure DO_SOMETHING shown in Figure 4.2 is run it will print 'abcde' followed by 'unhandled exception in...'. This error message allows the programmer to determine what line of code caused the program to bomb. This may seem excessive for this small procedure but when all the programmer knows is that an exception was raised somewhere in 20,000 lines of code, then there is a need for a systematic way to find where the exception was raised.

Specific Errors. Specific errors which the compiler "missed" are as follows:

1. The compiler allowed the following statement:

```
"x: constant := 6;"
```

This is a legal Ada statement, but when I tried to run any program that used this statement I got "unhandled exception..." occurring before the first line of code was executed (note this error was corrected in the newest release of the Ada compiler by the ROLM Corporation).

2. The second compiler glitch occurred when I instantiated a generic package with an illegal type. The generic package GKS_Coordinate_System (see Appendix B for the complete specification) called for a generic parameter of type COORDINATE which was digits <>.

```
generic
  type COORDINATE is digits <>;
package GKS_COORDINATE_SYSTEM is

end GKS_COORDINATE_SYSTEM;
```

determine where this error occurred in the program. To do this I resorted to using the alphabet scheme of debugging shown in Figure 4.1, and Figure 4.2. Figure 4.1 shows the program to be debugged by the alphabet scheme.

```
procedure DO_SOMETHING is
  number : positive;

begin
  number := 2;
  while number >= 0 loop
    number := number - 1;
  end loop;
end DO_SOMETHING;
```

Figure 4.1. Procedure DO_SOMETHING

To use this technique one needs to include a standard output package (text_io will do) in the program that is to be debugged. Then, as shown in Figure 4.2, between each statement of code one outputs a different character of the alphabet.

```
with text_io; use text_io;
procedure DO_SOMETHING is

  number : positive;

begin
  put('a');
  number := 2;
  put('b');
  while number >= 0 loop
    put('c');
    number := number - 1;
    put('d');
  end loop;
  put('e');
end DO_SOMETHING;
```

Figure 4.2. DO_SOMETHING being Debugged

performs these functions. Using these functions and some of the attribute functions defined in Ada, AFIT_GKS was able to handle all the code needed to drive the graphical devices, without resorting to assembly language statements.

Trouble with Running Ada with the ROLM Compiler. The first problem with running Ada with the ROLM compiler is the run time error messages (or lack thereof). When an exception is encountered in the program and it is not handled by an exception handler, then one of the following two messages appears on the screen.

"unhandled exception reaches main program" or

"unhandled exception in library unit, prog"

Nothing else is given. No line number! No error message! No package name! The error encountered could be anything!

I feel this is a major problem with this implementation of Ada. Trying to debug a program can quickly turn into a nightmare. A simple bug takes about 1 hour to find. An average bug takes about 3 to 4 hours. A difficult bug can take up to 12 hours to find. For example, I spent 4 hours trying to determine why a package specification (which contained some variables) would not allow any program to 'with' or 'use' it. I found out that the error was that one of the variables was initialized to 0.0, but it was of a type constrained to be greater than zero. A friendly compiler and/or informative run time message should say something like 'constraint error in initialization of variable x in package 'my_name'', instead of 'unhandled exception...'.

Exceptions are also raised by the compiler whenever a type conflict occurs in the execution of a program. Again, the only error message given is 'unhandled exception...'. Therefore, the programmer must

inquiry function then the routine raises a predefined exception.

This compiler is very good at explaining what syntax errors it found. It gives the line numbers as well as a two to ten line description of the problem encountered. These problems range from type constraints violated to undefined variables. It also allows for initialization of variables when they are defined. This saved initializing many of the variables at the beginning of the procedures.

Another feature of Ada is its portability. This version of AFIT_GKS should run on any validated Ada compiler.

Another observation I found with the language was its strong conformity to the in, out, in out, specification of parameters. If a procedure has a parameter of type 'in' then that parameter variable can not be set in the procedure, if the parameter is of type 'out' then that parameter can not have its value tested in the procedure. This insures that the programmer does not accidentally misuse the parameters.

One last observation is the strong relationship between a module specification header and a module body. If anything is different, even the name of a parameter, then the compiler will not accept the package body of the module. This insures that anyone using the module specification has the correct module specifications for the procedure in the module body.

Ada & Graphics. To effectively do graphics in the Ada programming language one must have input and output (I/O) routines which output characters without putting out a carriage return or a line feed. Also, the I/O package must be able to output control characters since many pieces of graphical equipment need control characters to draw graphical pictures. AFIT_GKS has access to a package called 'tty_io' which

IV. Analysis

As stated in the problem definition (chapter 1), this thesis shall examine three related topics, the Ada programming language, the Harris draft binding of Ada to ANS GKS, and the specification of ANS GKS. This chapter will comment on the good and bad points of each of these topics.

Ada

This project was written in the Ada programming language. The language is new and powerful, but some of the "bugs" haven't been worked out of the validated ROLM Ada compiler that was used in compiling AFIT_GKS. This section will cover the good and bad points of running Ada on the ROLM Data General.

Positive Comments. The Ada programming language allows problems to be broken down into several different compilation units each containing a separate "package" which allows for modularity. The package concept allows for a specified interface between the module specifications and the code in the packages.

Another feature of Ada is typing. Ada allows a wide variety of different types which the compiler checks to see if the variable value stays within its type throughout the program.

Another feature is the generic function. This project used this feature to develop variable length lists.

The error handling features of Ada allow errors to be captured and handled so as to avoid abnormal termination. This feature allows the user of AFIT_GKS to handle any errors in calling AFIT_GKS. Anytime an error is recognized in the calling of AFIT_GKS, the error message is logged in the error file (GKS_Error), and if the routine is not an

Error functions, and the Workstations. After the design consideration of each area was developed, the implementation of each area in AFIT_GKS was explained. Finally, this chapter concluded with a discussion of those overall features of AFIT_GKS that do not fall into any one major area of AFIT_GKS.

The other way that a user of AFIT_GKS can defer the redrawing of all the segments is by setting the implicit regeneration mode. The mode can be set as follows:

1. SUPPRESSED: implicit regeneration of the picture is suppressed, until it is explicitly requested.

2. ALLOWED: implicit regeneration of the picture is allowed(1:41).

By using the deferral mode in conjunction with the regeneration mode the user of AFIT_GKS can specify exactly when a redrawing of the display surface will take place. It is suggested that the user of AFIT_GKS not use the combined deferral and regeneration mode of ASAP and ALLOWED. This is because AFIT_GKS will redraw the screen whenever the display surface may not be perfectly correct (which is very frequently). The user of AFIT_GKS may change the deferral and regeneration mode by invoking the procedure `set_deferral_state` (see Appendix A).

Testing. Whenever a large project like AFIT_GKS is implemented, a thorough test plan must be accomplished. The testing of AFIT_GKS was found to be much more difficult than originally planned because of the poor run time error messages found on the ROLM Data General. AFIT_GKS has been only minimally tested. This minimal testing consisted of executing each AFIT_GKS procedure with one valid set of input. It is recommended that anyone continuing development of AFIT_GKS should develop and run a thorough test plan in accordance with an emerging ANS GKS certification/validation program(5:485).

Conclusions

This chapter has explored the many design considerations that went into AFIT_GKS. Mainly, this chapter explored the design of the four major parts of AFIT_GKS, the User Functions, the Data Structures, the

Deferral Modes. There are four deferral modes available in AFIT_GKS. They are ASAP, BNIG, BNIL, and ASTI, and they are described as follows:

1. ASAP: The visual effect of each function will be achieved As Soon As Possible (ASAP).

2. BNIG: The visual effect of each function will be achieved on the workstation Before the Next Interaction Globally (BNIG), i.e. before the next interaction with any input device happening on any workstation.

3. BNIL: The visual effect of each function will be achieved on the workstation Before the Next Interaction Locally (BNIL), i.e. before the next interaction with an input device happening on this particular workstation.

4. ASTI: The visual effect of each function will be achieved on the workstation At Some Time (ASTI)(1:40-41).

These states permit the user of AFIT_GKS to defer the redrawing of all segments on the display surface. The control function `redraw_all_segments_on_ws` will perform the redrawing of the display surface. `redraw_all_segments_on_ws` is invoked by AFIT_GKS whenever an AFIT_GKS function is called which can only be correctly realized by redrawing all of the segments on the workstation. For example, if the user of AFIT_GKS calls the procedure `delete_segment` for a segment that is being displayed on workstation 1 (the Tektronix 4014), then if the deferral state requires a redraw, the workstation display surface will be cleared and redrawn without the deleted segment. This is because the Tektronix 4014 does not have the ability to erase a segment without clearing the entire screen.

variables are part of the workstation but they are not workstation dependent since they are the same for any type of workstation. This being the case there is no reason to bar the device independent code from inquiring, and setting their values since this will cut down on the device dependent code but not restrict any powerful output functions of the device (see Figure 3.1).

Transformations. There are five transformations that take place in AFIT_GKS. They are as follows:

1. World coordinates to normalized device coordinates (wc \rightarrow ndc).
2. Normalized device coordinates to world coordinates (ndc \rightarrow wc).
3. Normalized device coordinates to device coordinates (ndc \rightarrow dc).
4. Device coordinates to normalized device coordinates (dc \rightarrow ndc).
5. Segment transformation (ndc \rightarrow ndc).

The first two transformations are the device independent transformations. They get the transformation values from the GKS_State_Lists and therefore, they are independent of the workstations. The next two transformations ((ndc \rightarrow dc), and (dc \rightarrow ndc)) transform normalized device coordinates (ndc) to and from device coordinates (dc) of a particular workstation. Therefore, these two functions must be a part of the workstation that they perform the transformation on. The transformation values are found in their respective Workstation_State_Lists. The segment transformation, using an input segment_transformation_matrix, transforms points from (ndc \rightarrow ndc). Both the device independent transformations, and the segment transformations can be workstation independent modules, and it is suggested that these functions may be put into a separate package. Presently, they can be found with the workstation dependent transformations located in Int_ws_1.

boxes shown on Figure 3.1. Instead, these topics are involved with AFIT_GKS as a whole unit.

Device Independent / Dependent Code. One of the major considerations of any graphical package is where to put the interface between the code that doesn't need to know what device it is running on, and the device dependent code that actually invokes the graphical device in a totally machine dependent manner.

The problem is that if the device independent code is too large and does too many things then a device with extra capabilities will not be able to perform all its special capabilities because the package has already broken down that capability into smaller steps. On the other hand, if the device independent code is too small then each new workstation added to AFIT_GKS will cause the maintainer of AFIT_GKS to duplicate a large section of device dependent code.

The solution offered by Simon's article on minimal GKS is very practical(6:185). He suggests that the interface be made at the point where GKS calls the individual workstations. The advantages to this are that each workstation is a different type and most likely will work differently. Therefore, each individual workstation can then execute its own device dependent code that performs the requested function. Next, since each workstation does not perform any function until it is called by its name, there is no sense in having device dependent code being written for sections of AFIT_GKS that haven't called upon the workstation. Therefore, as shown in Figure 3.1 AFIT_GKS defines the device independent / dependent code line directly above the workstations.

The only functions left to decide about are those that access the Workstation_Description_Tables, and the Workstation_State_Lists. These

Harris Binding

The Harris binding is a proposed set of standardized module headers and types which allow all the different Ada implementations of ANS GKS to have the same external interface. This way any application program that uses AFIT_GKS can use the same interface on any other ANS GKS implementation that uses the Harris binding.

The Harris binding used in AFIT_GKS has not been accepted by the ANSI standards committee. It is being reviewed by the public in order to work out its minor problems. I was one of the first to receive the draft.

Overall, I feel that the Harris Corporation did a great job of preparing a binding to ANS GKS. This was a large project that was greatly needed. However, because it was a first draft, it still had some minor errors. Overall, I found 60 syntax errors, 10 logic errors, 8 cases where the ROLM compiler would not accept the standard proposed Ada types, and 4 cases of what I thought was poor Ada style.

The first two kinds of errors, syntax and logic, were simple. They were errors that were obvious to me and Harris and they were changed without discussion. They included spelling errors, mislabeled numbers, and error statements in the wrong place.

The cases where my compiler would not accept the Harris binding were more serious. Harris later tried to implement the types that I had problems with on their ROLM Data General. When the type statement was rejected, they had to change the standard for everyone using the Harris binding. After a telephone conversation with the Harris Corporation, I found out that Harris did not compile their own specification on a validated Ada compiler. Instead, they used the Ada programming manual

(3), and a non-validated IBM-PC Telesoft Compiler, to test out their specifications of the binding.

Harris Binding Not Accepted by the ROLM Data General. The following are statements that the Harris Corporation proposed as part of the ANS GKS binding, but they were not accepted by the ROLM Data General compiler.

1. Harris proposed a package instantiation of the generic package, GKS_List_Uilities (See Appendix B), using a variant length type, Variable_String(6:354). This package instantiation which is shown in Figure 4.5 was rejected by the ROLM Data General Compiler.

```
max_length : constant integer := 50;

subtype VARIABLE_STRING_LENGTH is integer range 0..max_length;

type VARIABLE_STRING (LENGTH : VARIABLE_STRING_LENGTH := 0) is
  record
    CONTENTS : STRING(1..LENGTH);
  end record;

package VARIABLE_STRINGS is new GKS_LIST_UTILITIES
  (VARIABLE_STRING);
```

Figure 4.5. Variable Strings

The ROLM Data General would not accept the package instantiation because VARIABLE_STRING is not of a fixed length. I do not know whether this is a problem with the ROLM Data General or whether this is not a supported feature of Ada.

2. At the beginning of this thesis effort the ROLM Data General was having trouble with the statement:

```
"x : constant := 6;"
```

which was part of the Harris binding. Whenever I tried to use this constant statement in a type declaration, an unhandled exception would be raised before any code executed. I changed the constant to

```
"x : constant integer := 6;"
```

and I had no more problems. (Note: This bug has been fixed on the new version of Ada released by the ROLM Corporation).

3. The two types, `input_class`, and `choice_input`, shown in Figure 4.6, were not allowed together on the ROLM Data General.

```
type INPUT_CLASS is (locator_input,  
                    stroke_input,  
                    valuator_input,  
                    choice_input,  
                    pick_input,  
                    string_input,  
  
type CHOICE_INPUT ...
```

Figure 4.6. `Input_Class` and `Choice_Input`

This is because `choice_input` cannot be defined as an element in an enumeration type, and a declared type. The Harris Corporation agreed that this was a problem with the Harris binding and was changed in the binding.

Style Changes to the Harris Binding. Another input I made to the Harris binding was pointing out that one of their data types causes the access to a variable field to be `"x.style.style"`. The two `"style"`'s involved mean different things. Harris agreed and it has been changed to `"x.style.hatch_style"`.

I also proposed a change in the style of the Utility Functions (called `Set_Transform` in `AFIT_GKS`). As described previously in the User Functions section of chapter 3, these functions create transformation matrices for use in segment transformations. Presently, the Utility Functions listed in Appendix A (under the package `Set_Transform`) do not have any default values. If these functions had default values, then the user of `AFIT_GKS` could call these functions with only those values which were not the standard default values. For example, the procedure `Evaluate_Transformation_Matrix` shown in Figure 4.7, could be specified using the default values shown in Figure 4.8.

```
procedure EVALUATE_TRANSFORMATION_MATRIX
(FIXED_POINT : in wc.point;
SHIFT_VECTOR : in wc.vector;
ROTATION_ANGLE : in radians;
SCALE_FACTORS : in transformation_factor;
TRANSFORMATION : out transformation_matrix);
```

Figure 4.7. Present `Evaluate_Transformation_Matrix`

```
procedure EVALUATE_TRANSFORMATION_MATRIX
(FIXED_POINT : in wc.point := (x => 0.0, y => 0.0);
SHIFT_VECTOR : in wc.vector := (x => 0.0, y => 0.0);
ROTATION_ANGLE : in radians := 0.0;
SCALE_FACTORS : in transformation_factor :=
(x => 1.0, y => 1.0);
TRANSFORMATION : transformation_matrix);
```

Figure 4.8. Proposed `Evaluate_Transformation_Matrix`

By using the proposed `Evaluate_Transformation_Matrix` the user of `AFIT_GKS` would only have to specify the `rotation_angle` if the user wanted a transformation matrix which would rotate a picture 90 degrees around the origin. The user would not have to specify the `fixed_point`, `shift_`

vector, or the scale_factor.

Conclusions. The Harris Corporation created a draft binding of GKS to ANSI Ada. This section covered the minor problems that occurred when AFIT_GKS used the Harris binding. It concludes with some suggestions as to how certain small parts of the binding might be improved. Overall, I found that the Harris binding was a great help to this project, because it eliminated designing the entire external interface of AFIT_GKS.

ANS GKS

ANS GKS is the graphics system which was implemented. Overall, I found this graphical package was well designed. It allows for various devices and different levels of graphical devices. I especially like how it allows the program to use all of the facilities of the graphics terminals with relative ease.

ANS GKS Proposal. I did however find two errors in the ANS GKS proposal as given in the Special GKS issue of Computer Graphics February 1984 (1).

The first error found in the ANS GKS proposal was on page 121. On this page error 144, is duplicated(1:121). The second error 144 should be:

"error 145 echo area is outside display space"

Second, I believe that a variable "readback" should be added to the Workstation_Description_Table(1:204). The reason is that

Error #40 Specified workstation has no pixel store readback capability"

should be able to check the Workstation_Description_Table to inquire as to whether or not the workstation has this capability.

Conclusion

In this chapter I have discussed the three major areas which this thesis explored. Overall, the Ada programming language, the Harris binding, and the ANS GKS specifications were well designed. They each had some errors which were discovered as AFIT_GKS was developed.

V. Conclusions and Recommendations

Conclusions

In conclusion, as stated in the problem definition, there are several questions that this project wanted to explore.

1. Can Ada support a large project?

Except for some problems with the maximum size of variables, the ROLM Ada compiler permits large projects to be developed.

2. How easy is Ada to use in a large project?

The ROLM compiler used was excellent at catching and explaining syntax errors. The run time error messages were terrible. They need to include at a minimum:

A. The exception that was raised but not handled.

B. The line and package name where the exception was raised.

Without this information the programmer has no idea what went wrong with the program and on the average it took about 1 hour to find a bug. The minimum time was about 15 minutes, and the maximum time was about 24 hours.

3. Can Ada handle computer graphics?

Yes, it has the two capabilities which are essential for computer graphics. First, Ada can send out control characters. Second, Ada can convert ascii characters to their numeric equivalents and vice versa.

For anyone using Ada to do computer graphics, an I/O package must be developed which outputs ascii characters without putting out a <cr> or a <lf> control character. On the ROLM Data General this package was already present. Another I/O package that might be necessary is one that outputs 8-bit binary bytes since some graphical devices (like the Raster

Technologies Model One) require this kind of information in order to display graphical information.

Results

AFIT_GKS implemented 169 functions of ANS GKS using the Harris binding (see Appendix A for a complete list of these functions). However, AFIT_GKS did not implement the other 45 functions of ANS GKS as defined by the Harris binding. In addition, AFIT_GKS did not implement an additional 38 functions which were defined to work on the private data structures. A complete listing of all the functions that are specified by the Harris binding and are not implemented by AFIT_GKS can be found in Appendix C.

AFIT_GKS implemented and did a cursory testing of all the User functions defined in AFIT_GKS (See Appendix A). All of the AFIT_GKS functions were tested given one set of valid input data. Complete rigorous testing was not done.

Recommendations

This thesis was a first attempt at creating ANS GKS in Ada. The following are suggestions to further and improve AFIT_GKS.

1. The number and type of workstations could be expanded.
2. The transformation and segment priority list could be redesigned to use a generic sorting function.
3. The number of input devices may be expanded. For example, another locator device could be added to the Tektronix 4027 that would draw lines as the cross hairs moved (using the ink command)(15:8-11).
4. The rest of the ANS GKS functions (listed in Appendix C) should be coded.

5. The inquiry functions that have an input parameter of 'returned_valued' should be recoded so that when 'returned_values' is "realized", the function returns values as they are actually realized on the given workstation.

For example, the function "Inq_Polyline_Representation" shown in Figure 5.1 should return the "line" type, the line "width" scale factor, the polyline "colour" index for the bundle specified by the polyline "index" on workstation "ws".

```
procedure INQ_POLYLINE_REPRESENTATION
  (ws : in ws_id;
   index : in polyline_index;
   returned_values : in return_value_type;
   ei : out error_indicator;
   line : out line_type;
   width : out line_width;
   colour : out colour_index);
```

Figure 5.1. Inq_Polyline_Representation

The "returned_values" parameter indicates whether the returned values should be as set by the program, or as they were actually realized (10:244). Presently, the "returned_values" parameter is ignored.

6. AFIT_GKS should be thoroughly tested using a certification/validation program(14:485).

7. Stroke precision text should be added to AFIT_GKS. By definition, stroke precision text is displayed in the requested text font, at the text position by applying all text attributes(1:29).

8. The world coordinate to normalized device coordinate transformation for the primitive functions shown in Figure 3.8, should be moved down to the workstation. This will separate the Primitive package, from

the `Int_ws_1` package which contains the transformation function.

9. The binding of attributes shown in Figure 3.8, should be moved after clipping so that totally clipped primitives do not bind their attributes to the workstation.

Known Bugs in AFIT_GKS

AFIT_GKS is a first attempt at writing an ANS GKS graphics package. Therefore, there are some known errors in this program.

1. If the user sets up a `fill_area_bundle_table` entry that has an interior style of hollow, or solid, and the `pattern_index` or `hatch_index` is inquired from this `bundle_table` entry, then program will bomb. ANS GKS states that the `bundle_tables` should always contain a 'style' index and therefore, this should never cause an error.

2. The number of segments supported on AFIT_GKS is less than 32,000 because the ROLM Data General compiler will not allow a segment list to be this long. So far AFIT_GKS can only support 50 segments before the compiler refuses to allow the variables to be created.

3. If the program terminates before closing the error file then any information put into the file is lost.

Conclusions

AFIT_GKS is a subset of ANS GKS written in the Ada programming language. First, this thesis introduced the reader to ANS GKS and the Ada programming language. Second, the thesis covered the requirements of AFIT_GKS. Third, the thesis covered the design and implementation of AFIT_GKS. Included in the chapter on design was a section on how AFIT_GKS was tested. Fourth, the thesis analyzed the strengths and weaknesses

of the three major areas explored by the thesis, the Ada programming language, the Harris binding, and ANS GKS. Finally, the thesis concluded with a discussion about what was learned from this project and how AFIT_GKS might be extended and improved.

Appendix A

Users Guide to AFIT_GKS

This appendix contains the information necessary to write an application program that uses AFIT_GKS. AFIT_GKS took its external interface from the Harris Corporation's draft binding of Ada to ANS GKS. Therefore, this entire appendix is practically a direct quote of the Harris binding(6).

Table of Contents

	<u>Page</u>
Introduction	A.2
Cross Index	A.2
External_Types	A.11
AFIT_GKS Functions	A.38
AFIT_GKS Errors	A.64
Sample Program	A.68
System Dependent Features of AFIT_GKS	A.71

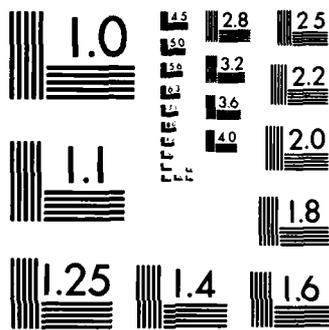
Introduction

This appendix is intended to be a users guide to AFIT_GKS. As such it includes a cross index, a list of all the types needed to access AFIT_GKS (external_types), and a list of all the functional specifications of the AFIT_GKS functions (AFIT_GKS Functions). Next, this appendix contains a list of all the errors that can occur in AFIT_GKS along with their error numbers (AFIT_GKS Errors). It concludes with a sample program running on AFIT_GKS, and a section describing some of the implementation dependent features of AFIT_GKS.

Cross Index

The cross index lists in alphabetical order all the functions of ANS GKS as specified in the proposed Harris binding of GKS to ANSI Ada. The AFIT_GKS functions are those listed in the cross index that have a "Y" under the column "Implemented". Also included in the cross index is the name of the package that contains the function (or should contain the function if the function is not implemented), and the possible errors that can occur in calling any ANS GKS function. Finally, the cross index lists the level of ANS GKS that the function is a part of and the page that the procedure's specification can be found in this appendix.

<u>Function Name</u>	<u>Implemented</u>	<u>Package</u>	<u>Errors Checked</u>	<u>Level</u>	<u>Page</u>
Accumulate_transformation_matrix	Y	Set_transform	8	1a	A.62
Activate_ws	Y	Control	6,25,29,33,35	ma	A.38
Associate_segment_with_ws	Y	Segments	6,25,27,33,35,124	2a	A.44
Await_event	N	Input	7,147,505	mc	
Cell_array	N	Primitives	5	0a	
Clear_ws	Y	Control	6,25,33,35	ma	A.38
Close_gks	Y	Control	2	ma	A.38
Close_segment	Y	Segments	4	1a	A.44
Close_ws	Y	Control	7,25,29,147	ma	A.38
Copy_segment_to_ws	Y	Segments	6,25,27,33,35,36,124	2a	A.45
Create_segment	Y	Segments	8,121	1a	A.44
Deactivate_ws	Y	Control	3,30,33,35	ma	A.38
Delete_segment	Y	Segments	7,122,125	1a	A.44
Delete_segment_from_ws	Y	Segments	7,25,33,35,123,125	1a	A.44
Emergency_close_gks	Y	Emergency	None	0a	A.62
Error_logging	Y	Error Handling	None	0a	A.63
Escape	N	Control	8,501	ma	
Evaluate_transformation_matrix	Y	Set_transform	8	1a	A.61
Fill_area	Y	Primitives	5,100	ma	A.39
Flush_device_events	N	Input	7,25,38,140,147	mc	
Gdp	N	Primitives	5,104	0a	
Get_choice	N	Input	7,150	mc	
Get_item_type_from_gksm	N	Metafile	7,25,34,162	0a	
Get_locator	N	Input	7,150	mc	
Get_pick	N	Input	7,150	1c	
Get_string	N	Input	7,150	mc	
Get_stroke	N	Input	7,150	mc	
Get_valuator	N	Input	7,150	mc	
Initialise_choice	Y	Set_input	7,25,38,51,123,140,141,144,145	mb	A.46
Initialise_locator	Y	Set_input	7,25,38,51,140,141,144,145	mb	A.46
Initialise_pick	Y	Set_input	7,25,37,51,140,141,144,145	1b	A.47



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

<u>Function Name</u>	<u>Implemented</u>	<u>Package</u>	<u>Errors Checked</u>	<u>Level</u>	<u>Page</u>
Initialise_string	Y	Set_input	7,25,38,51,140, 141,144,145	mb	A.47
Initialise_stroke	Y	Set_input	7,25,38,51,65, 140,141,144,145	mb	A.46
Initialise_valuator	Y	Set_input	7,25,38,51,140, 141,144,145	mb	A.46
Inq_char_expansion_factor	Y	Inq_attributes	8	ma	A.52
Inq_char_height	Y	Inq_attributes	8	ma	A.50
Inq_char_spacing	Y	Inq_attributes	8	ma	A.52
Inq_char_up_vector	Y	Inq_attributes	8	ma	A.50
Inq_choice_device_state	N	Inq_input	7,25,38,140	mb	
Inq_clipping	Y	Inq_attributes	8	ma	A.53
Inq_colour_facilities	Y	Inq_facilities	8,39	ma	A.60
Inq_colour_representation	Y	Inq_represent	7,25,33,35,36, 87	ma	A.56
Inq_current_normalization_transformation_number	Y	Inq_attributes	8	ma	A.52
Inq_default_choice_device_data	N	Inq_input	8,38,140	mb	
Inq_default_deferral_state_values	Y	Inq_represent	3,39	1a	A.57
Inq_default_locator_device_data	N	Inq_input	8,38,140	mb	
Inq_default_pick_device_data	N	Inq_input	8,37,140	1b	
Inq_default_string_device_data	N	Inq_input	8,38,140	mb	
Inq_default_stroke_device_data	N	Inq_input	8,38,140	mb	
Inq_default_valuator_device_data	N	Inq_input	8,38,140	mb	
Inq_display_space_size	Y	Inq_represent	8,31,33,36	0a	A.57
Inq_dynamic_modification_of_segment_attributes	Y	Inq_facilities	8,39	1a	A.61
Inq_dynamic_modification_of_ws_attributes	Y	Inq_represent	8,39	1a	A.57
Inq_fill_area_colour_index	Y	Inq_attributes	8	ma	A.52
Inq_fill_area_facilities	Y	Inq_facilities	8,39	ma	A.59
Inq_fill_area_index	Y	Inq_attributes	8	ma	A.51
Inq_fill_area_interior_style	Y	Inq_attributes	8	ma	A.52

<u>Function Name</u>	<u>Implemented</u>	<u>Package</u>	<u>Errors Checked</u>	<u>Level</u>	<u>Page</u>
Inq_fill_area_representation	Y	Inq_represent	7,25,33,35,36,76	1a	A.55
Inq_fill_area_style_index	Y	Inq_attributes	8	ma	A.52
Inq_gdp	Y	Inq_facilities	8,39,41	0a	A.60
Inq_input_queue_overflow	N	Inq_input	7,148,149	mc	
Inq_level_of_gks	Y	Inq_attributes	8	ma	A.49
Inq_line_type	Y	Inq_attributes	8	ma	A.51
Inq_linewidth_scale_factor	Y	Inq_attributes	8	ma	A.51
Inq_list_of_asf	Y	Inq_attributes	8	ma	A.52
Inq_list_of_available_gdp	Y	Inq_facilities	8,39	0a	A.60
Inq_list_of_available_ws_types	Y	Inq_attributes	8	0a	A.50
Inq_list_of_colour_indices	Y	Inq_represent	7,25,33,35,36	ma	A.56
Inq_list_of_fill_area_indices	Y	Inq_represent	7,25,33,35,36	1a	A.55
Inq_list_of_normalization_transformation_numbers	Y	Inq_attributes	8	0a	A.53
Inq_list_of_pattern_indices	Y	Inq_represent	7,25,33,35,36	1a	A.55
Inq_list_of_polyline_indices	Y	Inq_represent	7,25,33,35,36	1a	A.54
Inq_list_of_polymarker_indices	Y	Inq_represent	7,25,33,35,36	1a	A.54
Inq_list_of_text_indices	Y	Inq_represent	7,25,33,35,36	1a	A.55
Inq_locator_device_state	N	Inq_input	7,25,38,140	mb	
Inq_max_length_of_ws_state_tables	Y	Inq_segment	8,39	ma	A.60
Inq_max_normalization_transformation_number	Y	Inq_attributes	8	0a	A.50
Inq_more_simultaneous_events	Y	Inq_attributes	7	mc	A.53
Inq_name_of_open_segment	Y	Inq_attributes	4	1a	A.53
Inq_normalization_transformation	Y	Inq_attributes	8,50	ma	A.53
Inq_number_of_available_logical_input_devices	N	Inq_input	8,38	mb	

<u>Function Name</u>	<u>Implemented</u>	<u>Package</u>	<u>Errors Checked</u>	<u>Level</u>	<u>Page</u>
Inq_number_of_segment_priorities_supported	Y	Inq_segment	8,39	1a	A.60
Inq_operating_state_value	Y	Inq_attributes	None	0a	A.49
Inq_pattern_facilities	Y	Inq_facilities	8,39	0a	A.59
Inq_pattern_height_vector	N	Inq_attributes	8	ma	
Inq_pattern_size	Y	Inq_attributes	8	ma	A.51
Inq_pattern_reference_point	Y	Inq_attributes	8	ma	A.51
Inq_pattern_representation	Y	Inq_represent	7,25,33,35,36,83	1a	A.56
Inq_pattern_width_vector	N	Inq_attributes	8	ma	
Inq_pick_device_state	N	Inq_input	7,25,140	mb	
Inq_pick_id	Y	Inq_attributes	8	1b	A.51
Inq_pixel	N	Inq_pixels	7,25,39,40	0a	
Inq_pixel_array	N	Inq_pixels	7,25,39,40	0a	
Inq_pixel_array_dimensions	N	Inq_pixels	7,25,39	0a	
Inq_polyline_colour_index	Y	Inq_attributes	8	ma	A.51
Inq_polyline_facilities	Y	Inq_facilities	8,39	ma	A.57
Inq_polyline_index	Y	Inq_attributes	8	ma	A.50
Inq_polyline_representation	Y	Inq_represent	7,25,33,35,36,61	1a	A.54
Inq_polymarker_representation	Y	Inq_represent	7,25,33,35,36,65	1a	A.54
Inq_polymarker_colour_index	Y	Inq_attributes	8	ma	A.52
Inq_polymarker_index	Y	Inq_attributes	8	ma	A.50
Inq_polymarker_facilities	Y	Inq_facilities	8,39	ma	A.58
Inq_polymarker_size_scale_factor	Y	Inq_attributes	8	ma	A.52
Inq_polymarker_type	Y	Inq_attributes	8	ma	A.51
Inq_predefined_colour_representation	Y	Inq_facilities	8,39,86	0a	A.60
Inq_predefined_fill_area_representation	Y	Inq_facilities	8,39,75	0a	A.59
Inq_predefined_pattern_representation	Y	Inq_facilities	8,39,79,83	0a	A.59
Inq_predefined_polyline_representation	Y	Inq_facilities	8,39,60	0a	A.58

<u>Function Name</u>	<u>Implemented</u>	<u>Package</u>	<u>Errors Checked</u>	<u>Level</u>	<u>Page</u>
Inq_predefined_ polymarker_ representation	Y	Inq_facilities	8,39,64	0a	A.58
Inq_predefined_ text_representation	Y	Inq_facilities	8,39,68	0a	A.59
Inq_segment_ attributes	Y	Inq_segment	7,122	1a	A.61
Inq_set_of_active_ws	Y	Inq_attributes	8	1a	A.50
Inq_set_of_ associated_ws	Y	Inq_segment	7,122	1a	A.61
Inq_set_of_open_ws	Y	Inq_attributes	8	0a	A.50
Inq_set_of_segment_ names_in_use	Y	Inq_attributes	7	1a	A.53
Inq_set_of_segment_ on_ws	Y	Inq_represent	7,25,33,35	1a	A.56
Inq_string_device_ state	N	Inq_input	7,25,38,140	mb	
Inq_stroke_device_ state	N	Inq_input	7,25,38,140	mb	
Inq_text_alignment	Y	Inq_attributes	8	ma	A.51
Inq_text_colour_index	Y	Inq_attributes	8	ma	A.52
Inq_text_extent	N	Inq_represent	7,25,33,35,36,69	ma	
Inq_text_facilities	Y	Inq_facilities	8,39	ma	A.58
Inq_text_font_and_ precision	Y	Inq_attributes	8	ma	A.52
Inq_text_index	Y	Inq_attributes	8	ma	A.50
Inq_text_path	Y	Inq_attributes	8	ma	A.51
Inq_text_ representation	Y	Inq_represent	7,25,33,35,36,69	1a	A.55
Inq_valuator_device_ state	N	Inq_input	7,25,38,140	mb	
Inq_ws_category	Y	Inq_represent	8	0a	A.56
Inq_ws_class	Y	Inq_represent	8,39	0a	A.57
Inq_ws_connection_ and_type	Y	Inq_attributes	7,25	ma	A.53
Inq_wd_deferral_and_ update_states	Y	Inq_attributes	7,25,33,35,36	0a	A.54
Inq_ws_max_numbers	Y	Inq_attributes	8	1a	A.50
Inq_ws_state	Y	Inq_attributes	7,25	0a	A.53
Inq_ws_transformation	Y	Inq_represent	7,25,33,36	ma	A.56
Insert_segment	Y	Segments	5,27,124,125	2a	A.45
Interpret_item	N	Metafile	7	0a	
Message	N	Control	7,25,36	1a	
Open_gks	Y	Control	1,500	ma	A.38
Open_ws	Y	Control	8,21,24,26,28	ma	A.38

<u>Function Name</u>	<u>Implemented</u>	<u>Package</u>	<u>Errors Checked</u>	<u>Level</u>	<u>Page</u>
Polyline	Y	Primitives	5,100	ma	A.39
Polymarker	Y	Primitives	5,100	ma	A.39
Read_item_from_gksm	N	Metafile	7,25,34,162	0a	
Redraw_all_Segments_ on_ws	Y	Control	7,25,33,35,36	1a	A.38
Rename_segment	Y	Segments	7,121,122	1a	A.44
Request_choice	Y	Input	7,25,38,140, 141,504	mb	A.49
Request_locator	Y	Input	7,25,38,140, 141,504	mb	A.48
Request_pick	Y	Input	7,25,37,140, 141,504	1b	A.49
Request_string	Y	Input	7,25,38,140, 141,504	mb	A.49
Request_stroke	Y	Input	7,25,38,140, 141,504	mb	A.48
Request_valuator	Y	Input	7,25,38,140, 141,504	mb	A.49
Sample_choice	N	Input	7,25,38,140,142	mc	
Sample_locator	N	Input	7,25,38,140,142	mc	
Sample_pick	N	Input	7,25,37,140,142	1c	
Sample_string	N	Input	7,25,38,140,142	mc	
Sample_stroke	N	Input	7,25,38,140,142	mc	
Sample_valuator	N	Input	7,25,38,140,142	mc	
Select_normalization_ transformation	Y	Transform	8	ma	A.44
Set_asf	Y	Set_primitives	8	0a	A.41
Set_detectability	Y	Segments	7,122	1b	A.45
Set_deferral_state	Y	Control	7,25,33,35,36	1a	A.39
Set_char_expansion_ factor	Y	Set_primitives	8	0a	A.40
Set_char_height	Y	Set_primitives	8	ma	A.40
Set_char_spacing	Y	Set_primitives	8	0a	A.40
Set_char_up_vector	Y	Set_primitives	8,74	ma	A.40
Set_choice_mode	Y	Set_input	7,25,38,140,143	mb	A.48
Set_clipping_indicator	Y	Transform	8	ma	A.44
Set_colour_ representation	Y	Represent	7,25,33,35,36	ma	A.43
Set_fill_area_colour_ index	Y	Set_primitives	8	ma	A.41
Set_fill_area_index	Y	Set_primitives	8	0a	A.41
Set_fill_area_ interior_style	N	Set_primitives	8	ma	
Set_fill_area_ representation	Y	Represent	7,25,33,35,36, 77,80	1a	A.42

<u>Function Name</u>	<u>Implemented</u>	<u>Package</u>	<u>Errors Checked</u>	<u>Level</u>	<u>Page</u>
Set_fill_area_style_index	Y	Set_primitives	8	0a	A.41
Set_highlighting	Y	Segments	7,122	1a	A.45
Set_line_type	Y	Set_primitives	8	ma	A.39
Set_line_width_scale_factor	Y	Set_primitives	8	0a	A.40
Set_locator_mode	Y	Set_input	7,25,38,140,143	mb	A.47
Set_marker_size_scale_factor	Y	Set_primitives	8	0a	A.40
Set_marker_type	Y	Set_primitives	8	ma	A.40
Set_pattern_reference_point	Y	Set_primitives	8	0a	A.41
Set_pattern_representation	Y	Represent	7,25,33,35,36,83	1a	A.43
Set_pattern_size	Y	Set_primitives	8	0a	A.41
Set_pick_id	Y	Set_primitives	8	1b	A.41
Set_pick_mode	Y	Set_input	7,25,37,140,143	1b	A.48
Set_polyline_colour_index	Y	Set_primitives	8	ma	A.40
Set_polyline_index	Y	Set_primitives	8	0a	A.39
Set_polyline_representation	Y	Represent	7,25,33,35,36,63	1a	A.41
Set_polymarker_colour_index	Y	Set_primitives	8	ma	A.40
Set_polymarker_index	Y	Set_primitives	8	0a	A.40
Set_polymarker_representation	Y	Represent	7,25,33,35,36,67	1a	A.42
Set_segment_priority	Y	Segments	7,122	1a	A.45
Set_segment_transformation	Y	Segments	7,122	1a	A.45
Set_string_mode	Y	Set_input	7,25,38,140,143	mb	A.48
Set_stroke_mode	Y	Set_input	7,25,38,140,143	mb	A.47
Set_text_alignment	Y	Set_primitives	8	ma	A.41
Set_text_colour_index	Y	Set_primitives	8	ma	A.40
Set_text_font_and_precision	Y	Set_primitives	8	0a	A.40
Set_text_index	Y	Set_primitives	8	0a	A.40
Set_text_path	Y	Set_primitives	8	0a	A.41
Set_text_representation	Y	Represent	7,25,33,35	1a	A.42
Set_valuator_mode	Y	Set_input	7,25,38,140,143	mb	A.48
Set_viewport	Y	Transform	8,51	ma	A.43
Set_viewport_input_priority	Y	Transform	8	0b	A.43
Set_visibility	Y	Segments	7,122	1a	A.45
Set_window	Y	Transform	8,51	ma	A.43
Set_ws_viewport	Y	Transform	7,25,33,36,51,54	ma	A.44
Set_ws_window	Y	Transform	7,25,33,36,51	ma	A.44

<u>Function Name</u>	<u>Implemented</u>	<u>Package</u>	<u>Errors Checked</u>	<u>Level</u>	<u>Page</u>
Text	Y	Primitives	5	ma	A.39
Update_ws	Y	Control	7,25,33,35,36	ma	A.38
Write_item_to_gksm	N	Metafile	5,30,32	0a	

External_Types (6:25-58)

This section contains an alphabetical listing of all of the data type definitions used to define the Ada binding to GKS.

```
package EXTERNAL_TYPES is
```

```
-----  
type ASF is (BUNDLED, INDIVIDUAL);
```

```
-- Level 0a
```

```
-- This type defines an aspect source flag whose value indicates whether  
-- individual attributes are to be used, or attributes as specified in a  
-- bundle table.  
-----
```

```
type ASF_LIST is
```

```
record  
  LINE_TYPE           : ASF := INDIVIDUAL;  
  LINE_WIDTH          : ASF := INDIVIDUAL;  
  LINE_COLOUR         : ASF := INDIVIDUAL;  
  MARKER_TYPE         : ASF := INDIVIDUAL;  
  MARKER_SIZE         : ASF := INDIVIDUAL;  
  MARKER_COLOUR       : ASF := INDIVIDUAL;  
  TEXT_FONT_PRECISION : ASF := INDIVIDUAL;  
  CHAR_EXPANSION      : ASF := INDIVIDUAL;  
  CHAR_SPACING        : ASF := INDIVIDUAL;  
  TEXT_COLOUR         : ASF := INDIVIDUAL;  
  INTERIOR_STYLE      : ASF := INDIVIDUAL;  
  STYLE_INDEX         : ASF := INDIVIDUAL;  
  FILL_AREA_COLOUR   : ASF := INDIVIDUAL;  
end record;
```

```
-- level 0a
```

```
-- A list containing all of the aspect source flags, with components in-  
-- dicating the specific flag. The flags are all initialized as individ-  
-- ual.  
-----
```

```
type ATTRIBUTES_FLAG is (CURRENT, SPECIFIED);
```

```
-- Level 0a
```

```
-- Indicates whether output attributes used are to be as currently set,  
-- or as explicitly specified.  
-----
```

```
package ATTRIBUTES_USED is new GKS_LIST_UTILITIES(ATTRIBUTES_USED_TYPE);
```

```
-- Level 0a
```

```
-- Provides for a list of the attributes used.
```

```
-----  
type ATTRIBUTES_USED_TYPE is (POLYLINE_ATTRIBUTES,  
                               POLYMARKER_ATTRIBUTES,  
                               TEXT_ATTRIBUTES,  
                               FILL_AREA_ATTRIBUTES);
```

```
-- Level 0a
```

```
-- The types of attributes which may be used in generating output.
```

```
-----  
subtype CHAR_EXPANSION is POSITIVE_SCALE_FACTOR;
```

```
-- Level ma
```

```
-- Defines a character expansion factor. Factors are unitless, and must  
-- be greater than zero.
```

```
-----  
type CHAR_HEIGHT is new WC.MAGNITUDE;
```

```
-- Level ma
```

```
-- Defines character height, which must be a positive World Coordinates  
-- value.
```

```
-----  
subtype CHAR_SPACING is SCALE_FACTOR;
```

```
-- Level ma
```

```
-- Defines a character spacing factor. The factors are unitless. A pos-  
-- itive value indicates the amount of space between characters in a text  
-- string, and a negative value indicates the amount of overlap between  
-- characters in a text string.
```

```
-----  
-- This type was changed so as to avoid implementing the extra functions  
-- which are needed to work on the private types.  
-- type CHOICE_DATA_RECORD is private;
```

```
type CHOICE_DATA_RECORD is
  record
    prompt_echo : choice_prompt_echo_type;
    choices : integer;
    the_prompts : prompts.list_of;
    strings : prompt_strings.list_of;
    segment : segment_name;
    pick : pick_ids.list_of;
  end record;
```

-- Level mb

-- Defines a choice data input record.

```
type CHOICE_INPUT_TYPE (STATUS : CHOICE_STATUS := NOCHOICE) is
  record
    case STATUS is
      when OK => VALUE : CHOICE_VALUE;
      when NOCHOICE => null;
    end case;
  end record;
```

-- Level mb

-- Defines a choice input type. If STATUS is OK, then the choice input
-- value is indicated.

```
type CHOICE_PROMPT_ECHO_TYPE is range
    1..MAX_CHOICE_PROMPT_ECHO_TYPE;
```

-- Level mb

-- Defines the choice prompt and echo types supported by the implementa-
-- tion.

```
package CHOICE_PROMPT_ECHO_TYPES is new GKS_LIST_UTILITIES
    (CHOICE_PROMPT_ECHO_TYPE);
```

-- Level mb

-- Provides for lists of choice prompt and echo types.

```
type CHOICE_STATUS is (OK, NOCHOICE);
```

-- Level mb

-- indicates if a choice was made by the operator.

-- This is not done as a generic type because then all of the GKS program
-- would have to be in 1 file!
-- type CHOICE_VALUE is (<>);

type CHOICE_VALUE is range 0..20;

-- Level mb

-- This generic formal parameter specifies the range of values for a
-- choice input. The full range of values indicated may be only partial-
-- ly supported.

type CLIPPING_INDICATOR is (CLIP, NOCLIP);

-- Level ma

-- Indicates whether or not clipping is to be performed.

type COLOUR_AVAILABLE is (COLOUR, MONOCHROME);

-- Level ma

-- Indicates whether color output is available on a workstation.

subtype COLOUR_INDEX is PIXEL_COLOUR range 0..Max_colour_index;

-- Level ma

-- Indices into color tables are of this type.

package COLOUR_INDICES is new GKS_LIST_UTILITIES(COLOUR_INDEX);

-- Level ma

-- Provides for a list of color indices which are available on a partic-
-- ular workstation, and matrices containing color indices corresponding
-- to a cell array or pattern array.

```

type COLOUR_REPRESENTATION is
  record
    RED : INTENSITY;
    GREEN : INTENSITY;
    BLUE : INTENSITY;
  end record;

-- Level ma

-- Defines the representation of a color as a combination of intensities
-- in a RGB color system.

-----

subtype CONNECTION_ID is STRING;

-- Level ma

-- Defines the type for a connection identifier. The string must corre-
-- spond to an external device or file as defined by the GKS implementa-
-- tion.

-----

type CONTROL_FLAG is (CONDITIONALLY, ALWAYS);

-- Level ma

-- The control flag is used to indicate the conditions under which the
-- display surface should be cleared.

-----

package DC is new GKS_COORDINATE_SYSTEM (DC_TYPE);

-- Level ma

-- Defines the Device Coordinate System.

-----

-- ROLM Data General would not allow the package DC to use the following
-- statement as a formal generic parameter.
-- type DC_TYPE is digits MAX_DEVICE_PRECISION;

type DC_TYPE is new float;

-- Level ma

-- The type of a coordinate in the Device Coordinate System.

-----

```

type RELATIVE_PRIORITY is (HIGHER, LOWER);

-- Level ma

-- Indicates the relative priority between two normalization transformations.

type RETURN_VALUE_TYPE is (SET, REALIZED);

-- Level ma

-- Indicates whether the returned values should be as they were set by the program or as they were actually realized.

type SCALE_FACTOR is digits PRECISION;

-- Level ma

-- The type used for unitless scaling factors.

type SECONDS is digits PRECISION;

-- Level mb

-- This type is used for referencing timeout times for input events.

type SEGMENT_DETECTABILITY is (UNDETECTABLE, DETECTABLE);

-- Level 1b

-- Indicates whether a segment is detectable or not.

type SEGMENT_HIGHLIGHTING is (NORMAL, HIGHLIGHTED);

-- Level 1a

-- Indicates whether a segment is highlighted or not.

subtype PROMPT_STRING is STRING(1..MAX_PROMPT_STRING_LENGTH);

-- Level mb

-- A string which may be displayed as a choice prompt type.

package PROMPT_STRINGS is new GKS_LIST_UTILITIES (PROMPT_STRING);

-- Level mb

-- Defines a list of strings displayed as a choice device prompt type.

type RADIANS is digits PRECISION;

-- Level 1a

-- Values used in performing segment transformations (rotation angle).
-- Positive indicates an anticlockwise direction.

) type RASTER_UNITS is range 1..MAX_RASTER_UNITS;

-- Level ma

-- Defines the legal range of raster unit references.

type RASTER_UNIT_SIZE is
 record
 X : RASTER_UNITS;
 Y : RASTER_UNITS;
 end record;

-- Level ma

-- Defines the size of an object in raster units on a raster device.

type REGENERATION_MODE is (SUPPRESSED, ALLOWED);

-- Level 0a

-- Indicates whether implicit regeneration of the display is suppressed
-- or allowed.

```
package POLYLINE_INDICES is new GKS_LIST_UTILITIES (POLYLINE_INDEX);
```

```
-- Level 0a
```

```
-- Provides for the declaration of a list of polyline indices.
```

```
-----  
type POLYMARKER_INDEX is range 1..MAX_POLYMARKER_INDEX;
```

```
-- Level 0a
```

```
-- Defines the maximum range for polymarker bundle table indices.
```

```
-----  
package POLYMARKER_INDICES is new GKS_LIST_UTILITIES (POLYMARKER_INDEX);
```

```
-- Level 0a
```

```
-- Provides for the declaration of a list of polymarker indices.
```

```
-----  
-- The ROLM Data General compiler cannot implement "safe_small", or  
-- "safe_large".
```

```
-- subtype POSITIVE_SCALE_FACTOR is SCALE_FACTOR range  
-- SCALE_FACTOR'SAFE_SMALL..SCALE_FACTOR'SAFE_LARGE;
```

```
subtype POSITIVE_SCALE_FACTOR is SCALE_FACTOR range 0.00001..1.0E50;
```

```
-- Level ma
```

```
-- Define the positive range of scale factors.
```

```
-----  
type PROMPT is (OFF, ON);
```

```
-- Level mb
```

```
-- Indicates for a choice prompt and echo type whether a specified prompt  
-- is to be displayed or not.
```

```
-----  
package PROMPTS is new GKS_LIST_UTILITIES (PROMPT);
```

```
-- Level mb
```

```
-- Provides for a list of prompts.
```

-- plementation.

package PICK_PROMPT_ECHO_TYPES is new GKS_LIST_UTILITIES
(PICK_PROMPT_ECHO_TYPE);

-- Level mb

-- Provides for lists of pick prompt and echo types.

type PICK_STATUS is (OK, NOPICK);

-- Level 1b

-- Defines the status of a pick input operation.

type PIXEL_COLOUR is range -1..MAX_COLOUR_INDEX;

-- level 0a

-- This special data type is created for inquiries of pixel colors, since
-- an invalid color index is a legal value.

package PIXEL_COLOURS is new GKS_LIST_UTILITIES (PIXEL_COLOUR);

-- Level 0a

-- Provides for variable sized matrices of pixel colors.

package POINTS is new GKS_LIST_UTILITIES (WC.POINT);

-- Level ma

-- Provides for the declaration of arrays and lists of points.

type POLYLINE_INDEX is range 1..MAX_POLYLINE_INDEX;

-- Level 0a

-- Defines the maximum range of polyline indices.

-- Provides for the declaration of a list of pattern table indices.

-- This type was changed so as to avoid implementing extra functions
-- which are needed to work on the private types.
-- type PICK_DATA_RECORD is private;

```
type PICK_DATA_RECORD is
  record
    prompt_echo : pick_prompt_echo_type;
  end record;
```

-- Level 1b

-- Defines a pick input data record.

-- This is not done as a generic type because then all of the GKS program
-- would have to be in 1 file!
-- type PICK_ID is (<>);

```
type PICK_ID is new default_pick_id;
```

-- Level 1b

-- Defines a pick identifier. Pick identifiers are specified as a ge-
-- neric formal parameter.

type PICK_INPUT_TYPE (STATUS : PICK_STATUS := NOPICK) is
 record
 case STATUS is
 when OK => SEGMENT : SEGMENT_NAME;
 PICK : PICK_ID;
 when NOPICK => null;
 end case;
 end record;

-- Level 1b

-- Defines the type for pick. If STATUS is OK, then the segment name and
-- pick identifier are indicated.

type PICK_PROMPT_ECHO_TYPE is range 1..MAX_PICK_PROMPT_ECHO_TYPE;

-- Level mb

-- Defines the string prompt and echo types that are supported by the im-

-- Defines the Normalized Device Coordinate System.

type NDC_TYPE is digits PRECISION range 0.0..1.0;

-- Level ma

-- Defines the type of a coordinate in the Normalized Device Coordinate System.

type NEW_FRAME_NECESSARY is (NO, YES);

-- Level 0a

-- Indicates whether a new frame action is necessary at update.

type OPERATING_MODE is (REQUEST_MODE, SAMPLE_MODE, EVENT_MODE);

-- Level mb

-- Defines the operating modes of an input device.

type OPERATING_STATE is (GKCL, GKOP, WSOP, WSAC, SGOP);

-- Level 0a

-- At run time, GKS may be in one of the five predefined operating states. GKCL indicates GKS is closed, GKOP indicates GKS is open, WSOP indicates at least one workstation is open, WSAC indicates at least one workstation is active, and SGOP indicates a segment is currently open.

type PATTERN_INDEX is range 1..MAX_PATTERN_INDEX;

-- Level 0a

-- Defines the maximum range of pattern table indices.

package PATTERN_INDICES is new GKS_LIST_UTILITIES(PATTERN_INDEX);

-- Level 0a

-- A record containing information needed to specify the appearance of a
-- marker.

subtype MARKER_SIZE is POSITIVE_SCALE_FACTOR;

-- Level ma

-- The size of a marker is indicated by a scale factor larger than zero.

type MARKER_TYPE is range 1..MAX_MARKER_TYPE;

-- Level ma

-- Defines the types of markers provided by the implementation. Marker
-- types one through five are predefined as DOT_MARKER, PLUS_MARKER, AS-
-- TERISK_MARKER, CIRCLE_MARKER, and DIAGONAL_CROSS_MARKER.

package MARKER_TYPES is new GKS_LIST_UTILITIES (MARKER_TYPE);

-- Level ma

-- Provides for lists of marker types.

type MEMORY_UNITS is range 0..MAX_MEMORY_UNITS;

-- Level ma

-- Defines the type of the units of memory that may be allocated for GKS.
-- An implementation must indicate the type of the memory units being
-- used, such as bytes or fixed size blocks. Some standardization is
-- needed here.

type MORE_EVENTS is (NOMORE, MORE);

-- Level mc

-- Indicates whether more events are contained in the input event queue.

package NDC is new GKS_COORDINATE_SYSTEM (NDC_TYPE);

-- Level ma

-- which are needed to work on the private types.
-- type LOCATOR_DATA_RECORD is private;

```
type LOCATOR_DATA_RECORD is
  record
    prompt_echo : locator_prompt_echo_type;
    control : attributes_used_type;
    line : line_data;
    fill : fill_area_data;
  end record;
```

-- Level mb

-- Defines a locator data input record.

type LOCATOR_PROMPT_ECHO_TYPE is range 1.. MAX_LOCATOR_PROMPT_ECHO_TYPE;

-- Level mb

-- Defines the locator prompt and echo types supported by the implemen-
-- tation.

package LOCATOR_PROMPT_ECHO_TYPES is new GKS_LIST_UTILITIES
 (LOCATOR_PROMPT_ECHO_TYPE);

-- Level mb

-- Provides for lists of locator prompt and echo types.

type MARKER_DATA (ATTRIBUTES : ATTRIBUTES_FLAG := CURRENT) is
 record

```
    case ATTRIBUTES is
      when SPECIFIED =>
        MARKER_ASF : ASF;
        SIZE_ASF : ASF;
        COLOUR_ASF : ASF;
        INDEX : POLYMARKER_INDEX;
        MARKER : MARKER_TYPE;
        SIZE : MARKER_SIZE;
        COLOUR : COLOUR_INDEX;
      when CURRENT =>
        null;
    end case;
  end record;
```

-- Level mb

```
type LINE_DATA (ATTRIBUTES : ATTRIBUTES_FLAG := CURRENT) is
record
  case ATTRIBUTES is
  when SPECIFIED =>
    LINE_ASF : ASF;
    WIDTH_ASF : ASF;
    COLOUR_ASF : ASF;
    INDEX : POLYLINE_INDEX;
    LINE : LINE_TYPE;
    WIDTH : POSITIVE_SCALE_FACTOR;
    COLOUR : COLOUR_INDEX;
  when CURRENT =>
    null;
  end case;
end record;
```

-- Level mb

-- A record containing information needed to specify the appearance of a
-- line.

```
type LINE_TYPE is range 1..MAX_LINE_TYPE;
```

-- level ma

-- Defines the types of line styles provided by the implementation. Line
-- types one through four are predefined as SOLID_LINE, DASHED_LINE, DOT-
-- TED_LINE and DASHED_DOTTED_LINE. Additional line types are implemen-
-- tation defined.

```
package LINE_TYPES is new GKS_LIST_UTILITIES (LINE_TYPE);
```

-- Level ma

-- Provides for list of line types.

```
subtype LINE_WIDTH is POSITIVE_SCALE_FACTOR;
```

-- Level ma

-- The width of a line is indicated by a scale factor greater than zero.

-- This type was changed so as to avoid implementing extra functions

-- Defines the maximum length of a variable length string supported by
-- the implementation. This affects both string input values as well as
-- strings used as prompts for choice input.

-- This is not done as a generic type because then all of the GKS program
-- would have to be in 1 file!
-- type INPUT_VALUE is digits <>;

type INPUT_VALUE is new float;

-- level mb

-- This is not done as a generic because then all of the GKS program
-- would have to be in 1 file! An input value from a valuator device is
-- specified as a generic formal parameter. Not all input devices may
-- be able to support the full range of input values indicated.

type INTENSITY is digits PRECISION range 0.0..1.0;

-- Level ma

-- Defines the range of possible intensities of a color.

type INTERIOR_STYLE is (HOLLOW, SOLID, PATTERN, HATCH);

-- Level ma

-- Defines the predefined types of interior styles for fill areas

package INTERIOR_STYLES is new GKS_LIST_UTILITIES (INTERIOR_STYLE);

-- Level ma

-- Defines the predefined types of interior styles for fill area.

type INVALID_VALUES_INDICATOR is (ABSENT, PRESENT);

-- Level 0a

-- Indicates whether invalid values are contained in a pixel array or ma-
-- trix.

type HATCH_STYLE_TYPE is range 1..MAX_HATCH_STYLE;

-- Level 0a

-- Defines the hatch styles supported by the implementation. There must
-- be at least three hatch styles supported by the implementation.

package HATCH_STYLES is new GKS_LIST_UTILITIES (HATCH_STYLE_TYPE);

-- Level 0a

-- Provides for a list of hatch styles.

type HORIZONTAL_ALIGNMENT is (NORMAL, LEFT, CENTER, RIGHT);

-- Level ma

-- The alignment of the text extent rectangle with respect to the ver-
-- tical positioning of the text.

type INPUT_CLASS is (LOCATOR_INPUT, STROKE_INPUT, VALUATOR_INPUT,
CHOICE_INPUT, PICK_INPUT, STRING_INPUT);

-- Level mb

-- Defines the input device classifications for workstations of category
-- INPUT or OUTIN.

type INPUT_STRING (LENGTH : INPUT_STRING_LENGTH := 0) is
record
CONTENTS : STRING(1..LENGTH);
end record;

-- Level mb

-- Provides a variable length string. Objects of this type should be de-
-- clared unconstrained to allow for dynamic modification of the length.

subtype INPUT_STRING_LENGTH is INTEGER range 0..MAX_INPUT_STRING_LENGTH;

-- Level mb

-- Defines the maximum range of fill area bundle table indices.

package FILL_AREA_INDICES is new GKS_LIST_UTILITIES (FILL_AREA_INDEX);

-- Level 0a

-- Provides for the declaration of a list of fill area bundle table indices.

type FONT_TYPE is range 1..MAX_FONT_TYPE;

-- Level ma

-- Defines the types of fonts provided by the implementation. The implementation must provide at least one font capable of generating the standard ASCII character set. This font is font number one.

type GDP_ID is range 1..MAX_GDP_ID;

-- Level 0a

-- Defines a range of values for selecting a Generalized Drawing Primitive.

package GDP_IDS is new GKS_LIST_UTILITIES (GDP_ID);

-- Level 0a

-- Provides for lists of GDP ID's

type GKS_LEVEL is (Lma, Lmb, Lmc,
 L0a, L0b, L0c,
 L1a, L1b, L1c,
 L2a, L2b, L2c);

-- Level ma

-- The valid Levels of GKS. M, 0, 1, 2 indicate the level of output supported by the implementation, and a, b, and c indicate the level of input supported by the implementation. Certain other capabilities and capacities are also indicated by the level.

```
type ECHO_SWITCH is (ECHO, NOECHO);
```

```
-- Level mb
```

```
-- Indicates whether or not echoing of the prompt is performed.
```

```
subtype ERROR_FILE_TYPE is STRING;
```

```
-- Level ma
```

```
-- Defines the type for error file specification. The name used must  
-- conform to an external file name as defined for the host system imple-  
-- mentation.
```

```
type ERROR_INDICATOR is range 0..999;
```

```
-- Level ma
```

```
-- Defines the range of error indicator values.
```

```
type FILL_AREA_DATA (ATTRIBUTES : ATTRIBUTES_FLAG := CURRENT) is  
record
```

```
  case ATTRIBUTES is  
    when SPECIFIED =>  
      STYLE_ASF : ASF;  
      STYLE_INDEX_ASF : ASF;  
      COLOUR_ASF : ASF;  
      INDEX : FILL_AREA_INDEX;  
      STYLE : STYLE_INDEX;  
      COLOUR : COLOUR_INDEX;  
    when CURRENT =>  
      null;  
  end case;  
end record;
```

```
-- Level mb
```

```
-- A record containing information needed to specify the appearance of a  
-- filled area.
```

```
type FILL_AREA_INDEX is range 1.. MAX_FILL_AREA_INDEX;
```

```
-- Level 0a
```

type DC_UNITS is (METRES, OTHER);

-- Level ma

-- Device coordinate units for a particular workstation may be in meters,
-- or some other units (such as inches).

type DEFERRAL_MODE is (ASAP, BNIG, BNIL, ASTI);

-- Level 0a

-- Indicates how long output to a workstation is delayed. ASAP indicates
-- as soon as possible, BNIG indicates before the next interaction glo-
-- bally, BNIL indicates before the next interaction locally, and ASTI
-- indicates at some time.

type DEVICE_NUMBER is range 1..MAX_DEVICE_NUMBER;

-- Level mb

-- Logical devices are referenced as device numbers. The maximum number
-- of input devices may not be supported on all workstations.

type DISPLAY_CLASS is (VECTOR_DISPLAY, RASTER_DISPLAY,
 OTHER_DISPLAY);

-- Level 0a

-- The classification of a workstation of category OUTPUT or OUTIN.

type DISPLAY_SURFACE_EMPTY is (EMPTY, NOTEMPTY);

-- Level 0a

-- Indicates whether the display surface is empty.

type DYNAMIC_MODIFICATION is (IRG, IMM);

-- Level 1a

-- Indicates whether an update to the state list is performed immediately
-- (IMM) or is implicitly regenerated (IRG).

```
-- This is not done as a generic type because then all of the GKS program
-- would have to be in 1 file!
-- type SEGMENT_NAME is (<>);
```

```
type SEGMENT_NAME is new default_segment_name;
```

```
-- Level 1a
```

```
-- Segment names are specified as a generic formal parameter.
```

```
-----
package SEGMENT_NAMES is new GKS_LIST_UTILITIES (SEGMENT_NAME);
```

```
-- Level 1a
```

```
-- Provides for a list of segment names.
```

```
-----
type SEGMENT_PRIORITY is digits PRECISION range 0.0..1.0;
```

```
-- Level 1a
```

```
-- Defines the priority of a segment.
```

```
-----
type SEGMENT_VISIBILITY is (VISIBLE, INVISIBLE);
```

```
-- Level 1a
```

```
-- Indicates whether a segment is visible or not.
```

```
-----
-- This type was changed so as to avoid implementing extra functions
-- which are needed to work on the private types.
-- type STRING_DATA_RECORD is private;
```

```
type STRING_DATA_RECORD is
  record
    prompt_echo : string_prompt_echo_type;
    buffer : integer;
    cursor_position : integer;
  end record;
```

```
-- Level mb
```

```
-- Defines a string data input record.
```

```
type STRING_PROMPT_ECHO_TYPE is range 1..MAX_STRING_PROMPT_ECHO_TYPE;
```

```
-- Level mb
```

```
-- Defines the string prompt and echo types supported by the implementa-  
-- tion.
```

```
-----  
package STRING_PROMPT_ECHO_TYPES is new GKS_LIST_UTILITIES  
  (STRING_PROMPT_ECHO_TYPE);
```

```
-- Level mb
```

```
-- Provides for lists of string prompt and echo types.
```

```
-----  
-- This type was changed so as to avoid implementing extra functions  
-- which are needed to work on the private types.  
-- type STROKE_DATA_RECORD is private;
```

```
type STROKE_DATA_RECORD is  
  record  
    buffer : integer;  
    edit_position : integer;  
    interval : wc.point;  
    time : seconds;  
    prompt_echo : stroke_prompt_echo_type;  
    marker : marker_data;  
    line : line_data;  
  end record;
```

```
-- Level mb
```

```
-- Defines a string data input record.
```

```
-----  
type STROKE_PROMPT_ECHO_TYPE is range 1..MAX_STROKE_PROMPT_ECHO_TYPE;
```

```
-- Level mb
```

```
-- Defines the stroke prompt and echo types supported by the implementa-  
-- tion.
```

```
-----  
package STROKE_PROMPT_ECHO_TYPES is new  
  GKS_LIST_UTILITIES (STROKE_PROMPT_ECHO_TYPE);
```

```
-- Level mb
```

-- Provides for lists of stroke prompt and echo types.

```
type STYLE_INDEX (INTERIOR: INTERIOR_STYLE := HOLLOW) is
  record
    case INTERIOR is
      when HOLLOW | SOLID => null;
      when PATTERN => INDEX : PATTERN_INDEX;
      when HATCH => HATCH_STYLE : HATCH_STYLE_TYPE;
    end case;
  end record;
```

-- Level 0a

-- Defines a fill area style index. Such an index is null for interior styles hollow and solid. For interior style pattern, the style index is an index into the pattern tables. For interior style hatch, the style index indicates which hatch style is to be used. An attempt to index the pattern tables using a hatch style will result in an exception, as will attempting to reference a hatch style using a pattern table index.

```
subtype SUBPROGRAM_NAME is STRING;
```

-- Level ma

-- Defines the name of a GKS function detecting an error.

```
type TEXT_ALIGNMENT is
  record
    HORIZONTAL : HORIZONTAL_ALIGNMENT;
    VERTICAL : VERTICAL_ALIGNMENT;
  end record;
```

-- Level ma

-- The type of the attribute controlling the positioning of the text extent rectangle in relation to the text position, having horizontal and vertical components as defined above.

```
type TEXT_EXTENT_RECTANGLE is
  record
    LOWER_LEFT : WC.POINT;
    LOWER_RIGHT : WC.POINT;
    UPPER_LEFT : WC.POINT;
    UPPER_RIGHT : WC.POINT;
```

```

end record;

-- Level ma

-- Defines the corner points of the text extent rectangle with respect
-- to the vertical positioning of the text.

-----

type TEXT_FONT_PRECISION is
  record
    FONT : FONT_TYPE;
    PRECISION : TEXT_PRECISION;
  end record;

-- Level ma

-- This type defines a record describing the text font and precision as-
-- pect.

-----

package TEXT_FONT_PRECISIONS is new
  GKS_LIST_UTILITIES (TEXT_FONT_PRECISION);

-- Level ma

-- Provides for lists of text font and precision pairs.

-----

type TEXT_INDEX is range 1..MAX_TEXT_INDEX;

-- Level 0a

-- Defines the maximum range of text bundle table indices.

-----

package TEXT_INDICES is new GKS_LIST_UTILITIES (TEXT_INDEX);

-- Level 0a

-- Provides for a list of text indices.

-----

type TEXT_PATH is (RIGHT, LEFT, UP, DOWN);

-- Level ma

-- The direction taken by a text string.

```

```
type TEXT_PRECISION is (STRING_PRECISION, CHAR_PRECISION,  
                        STROKE_PRECISION);
```

```
-- Level ma
```

```
-- The precision with which individual text units may be regarded.
```

```
type TRANSFORMATION_FACTOR is  
  record  
    X : SCALE_FACTOR;  
    Y : SCALE_FACTOR;  
  end record;
```

```
-- Level 1a
```

```
-- Scale factors used in building transformation matrices for performing  
-- segment transformations.
```

```
type TRANSFORMATION_MATRIX is array (1..3,1..2) of SCALE_FACTOR;
```

```
-- Level 1a
```

```
-- A segment transformation matrix for mapping NDC to NDC. The elements  
-- for the array are all scale factors for convenience. Elements M11,  
-- M12, M21, M22 comprise the scaling and rotation portion of the matrix,  
-- and are unitless. The remaining elements (M13 and M23) comprise the  
-- translation portion. The latter are of type NDC.
```

```
type TRANSFORMATION_NUMBER is range 0..MAX_TRANSFORMATION_NUMBER;
```

```
-- Level ma
```

```
-- A normalization transformation number.
```

```
package TRANSFORMATION_NUMBERS is new  
  GKS_LIST_UTILITIES (TRANSFORMATION_NUMBER);
```

```
-- Level 0a
```

```
-- Provides for a list of normalization transformation number. All im-  
-- plementations must supply the predefined UNITY_TRANSFORMATION from  
-- World Coordinate space [0,1] x [0,1] to Normalized Device Coordinate  
-- space [0,1] x [0,1].
```

```
type UPDATE_REGENERATION_FLAG is (PERFORM, POSTPONE);
```

```
-- Level 0a
```

```
-- Flag indicating regeneration action on display.
```

```
type UPDATE_STATE is (NOTPENDING, PENDING);
```

```
-- Level ma
```

```
-- Indicates whether or not a workstation transformation change has been  
-- requested and not yet provided.
```

```
-- This type was changed so as to avoid implementing extra functions  
-- which are needed to work on the private types.  
-- type VALUATOR_DATA_RECORD is private;
```

```
type VALUATOR_DATA_RECORD is  
  record  
    prompt_echo : valuator_prompt_echo_type;  
    low_value : input_value;  
    high_value : input_value;  
  end record;
```

```
-- Level mb
```

```
-- Defines a valuator data input record.
```

```
type VALUATOR_PROMPT_ECHO_TYPE is range  
  1..MAX_VALUATOR_PROMPT_ECHO_TYPE;
```

```
-- Level mb
```

```
-- Defines the possible range of valuator prompt and echo types.
```

```
package VALUATOR_PROMPT_ECHO_TYPES is new  
  GKS_LIST_UTILITIES (VALUATOR_PROMPT_ECHO_TYPE);
```

```
-- Level mb
```

```
-- Provides for lists of valuator prompt and echo types.
```

```
-----  
type VERTICAL_ALIGNMENT is (NORMAL, TOP, CAP, HALF, BASE, BOTTOM);  
  
-- Level ma  
  
-- The alignment of the text extent rectangle with respect to the verti-  
-- cal positioning of the text.  
-----
```

```
package WC is new GKS_COORDINATE_SYSTEM (WC_TYPE);
```

```
-- Level ma  
  
-- Defines the World Coordinate System.  
-----
```

```
-- This is not done as a generic type because then all of the GKS program  
-- would have to be in 1 file!  
-- type WC_TYPE is digits <>;
```

```
type WC_TYPE is new float;
```

```
-- Level ma  
  
-- Defines the type of a coordinate in the World Coordinate System.  
-- World coordinate type is specified as a generic formal parameter.  
-- The type given as an actual parameter must at least include the range  
-- 0.0 to 1.0.  
-----
```

```
type WS_CATEGORY is (OUTPUT, INPUT, OUTIN, WISS, MO, MI);
```

```
-- Level 0a  
  
-- All workstation types fall into one of these six predefined worksta-  
-- tion categories.  
-----
```

```
-- This is not done as a generic type because then all of the GKS program  
-- would have to be in 1 file!  
-- type WS_ID is (<>);
```

```
type WS_ID is new default_ws_id;
```

```
-- Level ma  
  
-- Workstation identifiers are of this type. Workstation identifier  
-- type is specified as a generic formal parameter.  
-----
```

```
package WS_IDS is new GKS_LIST_UTILITIES (WS_ID);
```

```
-- Level ma
```

```
-- Provides for lists of workstation identifiers.
```

```
type WS_STATE is (ACTIVE, INACTIVE);
```

```
-- Level 0a
```

```
-- The state of a workstation.
```

```
type WS_TYPE is range 1..MAX_WS_TYPE;
```

```
-- Level ma
```

```
-- Range of values corresponding to valid workstation types. Constants  
-- specifying names for the various types of workstations should be pro-  
-- vided by an implementation, such as GKSM_OUTPUT, FLAT_BED_PLOTTER.
```

```
package WS_TYPES is new GKS_LIST_UTILITIES (WS_TYPE);
```

```
-- Level 0a
```

```
-- provides for variable length lists of workstation types.
```

```
end EXTERNAL_TYPES;
```

AFIT_GKS Functions (6:67-345)

The following is all the functions implemented in AFIT_GKS. They are divided into their respective packages.

package CONTROL is

procedure OPEN_GKS

(error_file : in error_file_type := "";
amount_of_memory : in memory_units := max_memory_units);
This function initializes GKS. It must be invoked before any other GKS function.

procedure CLOSE_GKS;

This function closes GKS.

procedure OPEN_WS

(ws : in ws_id;
connection : in connection_id;
type_of_ws : in ws_type);
This function adds "ws" to the list of open workstations.

procedure CLOSE_WS

(ws : in ws_id);
This function releases the connection between workstation "ws" and GKS. No further references to workstation "ws" are allowed.

procedure ACTIVATE_WS

(ws : in ws_id);
This function activates the specified workstation "ws".

procedure DEACTIVATE_WS

(ws : in ws_id);
No further output primitives and/or segments will be sent to workstation "ws".

procedure CLEAR_WS

(ws : in ws_id;
flag : in control_flag);
CLEAR_WS causes the display surface on "ws" to be cleared. All segments stored on the workstation are deleted.

procedure REDRAW_ALL_SEGMENTS_ON_WS

(ws : in ws_id);
This function causes all of the segments stored for workstation "ws" to be redrawn on that workstation. The display surface is cleared first.

procedure UPDATE_WS

(ws : in ws_id;
regeneration : in update_regeneration_flag);
UPDATE_WS causes all of the deferral actions for "ws" to be performed

without any intermediate clearing of the display. If the "regeneration" flag is set to "perform", then the display surface is cleared, if necessary. The workstation transformation is updated if it is pending. The segments stored on the workstations are redrawn. None of the additional functions are executed if the "regeneration" flag is set to "postpone".

```
procedure SET_DEFERRAL_STATE
```

```
  (ws : in ws_id;  
   deferral : in deferral_mode;  
   regeneration : in regeneration_mode);
```

This function sets the "deferral" and implicit "regeneration" modes on workstation "ws".

```
end CONTROL;
```

```
package PRIMITIVES is
```

```
procedure POLYLINE
```

```
  (line_points : in points.array_of);
```

POLYLINE draws a line connecting the specified "line_points". There must be at least two points in "line_points".

```
procedure POLYMARKER
```

```
  (marker_points : in points.array_of);
```

POLYMARKER draws a sequence of markers at the specified "marker_points". There must be at least one point in "marker_points".

```
procedure TEXT
```

```
  (position : in wc.point;  
   text_string : in string);
```

The character string "text_string" is drawn starting at the "position" which is given in world coordinates.

```
procedure FILL_AREA
```

```
  (fill_area_points : in points.array_of);
```

This function fills a polygon defined by "fill_area_points". There must be at least three points in "fill_area_points".

```
end PRIMITIVES;
```

```
package SET_PRIMITIVES is
```

```
procedure SET_POLYLINE_INDEX
```

```
  (index : in polyline_index);
```

The current polyline index is set to "index". This value will be used for all subsequent polyline primitives.

```
procedure SET_LINE_TYPE
```

```
  (line : in line_type);
```

The current linetype is set to "line".

```
procedure SET_LINE_WIDTH_SCALE_FACTOR  
  (width : in line_width);
```

The current line width scale factor becomes "width".

```
procedure SET_POLYLINE_COLOUR_INDEX  
  (colour : in colour_index);
```

The current polyline colour index becomes "colour".

```
procedure SET_POLYMARKER_INDEX  
  (index : in polymarker_index);
```

The current polymarker index is set to the value specified by "index".

```
procedure SET_MARKER_TYPE  
  (marker : in marker_type);
```

The current marker type becomes "marker".

```
procedure SET_MARKER_SIZE_SCALE_FACTOR  
  (size : in marker_size);
```

The current marker size scale factor is set to the value of "size".

```
procedure SET_POLYMARKER_COLOUR_INDEX  
  (colour : in colour_index);
```

The current polymarker colour index becomes "colour".

```
procedure SET_TEXT_INDEX  
  (index : in text_index);
```

The current text index is set to the value specified by "index".

```
procedure SET_TEXT_FONT_PRECISION  
  (font_precision : in text_font_precision);
```

The current text font and precision is set to the value "font_precision".

```
procedure SET_CHAR_EXPANSION_FACTOR  
  (expansion : in char_expansion);
```

The current character expansion factor is set to the value "expansion".

```
procedure SET_CHAR_SPACING  
  (spacing : in char_spacing);
```

The current character spacing is set to "spacing".

```
procedure SET_TEXT_COLOUR_INDEX  
  (colour : in colour_index);
```

The current text colour index is set to the value "colour".

```
procedure SET_CHAR_HEIGHT  
  (height : in char_height);
```

The current character height is set to the value "height".

```
procedure SET_CHAR_UP_VECTOR  
  (char_up_vector : in wc.vector);
```

The current character up vector is set to the value "char_up_vector".

The character up vector sets the rotation of the text path from the origin.

```
procedure SET_TEXT_PATH  
  (path : in text_path);
```

The current text path is set to "path". The text path determines the direction that the characters are displayed.

```
procedure SET_TEXT_ALIGNMENT  
  (alignment : in text_alignment);
```

The current text alignment is set to "alignment".

```
procedure SET_FILL_AREA_INDEX  
  (index : in fill_area_index);
```

The current fill area index is set to "index".

```
procedure SET_FILL_AREA_STYLE_INDEX  
  (index : in style_index);
```

The current fill area style index becomes "index".

```
procedure SET_FILL_AREA_COLOUR_INDEX  
  (colour : in colour_index);
```

The current fill area colour index becomes "colour".

```
procedure SET_PATTERN_SIZE  
  (size : in wc.size);
```

The current pattern size is set to the value "size".

```
procedure SET_PATTERN_REFERENCE_POINT  
  (point : in wc.point);
```

The current pattern reference point becomes "point". When the currently selected fill area interior style is PATTERN, this value is used, where possible, in conjunction with the current pattern size for displaying the fill area output primitives.

```
procedure SET_ASF  
  (asf : in asf_list);
```

The aspect source flags are assigned the values contained in ASF.

```
procedure SET_PICK_ID  
  (pick : in pick_id);
```

The current pick identifier is set to the value specified by the parameter "pick".

```
end SET_PRIMITIVES;
```

package REPRESENT is

```
procedure SET_POLYLINE_REPRESENTATION  
  (ws : in ws_id;  
   index : in polyline_index;
```

```
line : in line_type;  
width : in line_width;  
colour : in colour_index);
```

This function is used to define (or redefine) the contents of the poly-line bundle table for workstation WS according to the contents of the parameters "index", "line", "width", and "colour" where:

"index" specifies the entry in the bundle table to be defined.

"line" is the line type value.

"width" is the line width scale factor.

"colour" is the line colour.

```
procedure SET_POLYMARKER_REPRESENTATION
```

```
(ws : in ws_id;  
index : in polymarker_index;  
marker : in marker_type;  
size : in marker_size;  
colour : in colour_index);
```

This function is used to define (or redefine) the contents of the poly-marker bundle for workstation "ws" according to the parameters "index", "marker", "size", "colour" where

"index" specifies the entry in the bundle table to be defined.

"marker" specifies the marker type value.

"size" is the scale factor to be applied to the nominal marker size.

"colour" is the value for the marker colour.

```
procedure SET_TEXT_REPRESENTATION
```

```
(ws : in ws_id;  
index : in text_index;  
font_precision : in text_font_precision;  
expansion : in char_expansion;  
spacing : in char_spacing;  
colour : in colour_index);
```

This function is used to define (or redefine) the contents of a text bundle for workstation "ws" according to the parameters "index", "font_precision", "expansion", "spacing" and "colour", where:

"index" specifies the entry in the bundle table to be defined.

"font_precision" is used to select a particular font on this workstation.

"expansion" specifies the deviation of the width to height ratio indicated by the font designer.

"spacing" specifies how much additional space is to be inserted between two adjacent character bodies.

"colour" is the text colour.

```
procedure SET_FILL_AREA_REPRESENTATION
```

```
(ws : in ws_id;  
index : in fill_area_index;  
style : in style_index;  
colour : in colour_index);
```

This function is used to define (or redefine) the contents of a fill area bundle for workstation "ws" according to the parameters "index", "style", and "colour" where:

"index" specifies the entry in the bundle table to be defined.

"style" determines which PATTERN or HATCH style is selected. It is ignored for interior styles HOLLOW and SOLID.

"colour" is the fill area colour.

procedure SET_PATTERN_REPRESENTATION

(ws : in ws_id;
index : in pattern_index;
pattern : in colour_indices.matrix_of);

This function is used to define (or redefine) the contents of a pattern bundle for workstation "ws" according to the parameters "index" and "pattern", where:

"index" specifies the entry in the bundle table to be defined.

"pattern" specifies the interior style for fill areas.

procedure SET_COLOUR_REPRESENTATION

(ws : in ws_id;
index : in colour_index;
colour : in colour_representation);

This function is used to define (or redefine) the contents of a colour bundle for workstation "ws" according to the parameters "index" and "colour" where:

"index" specifies the entry in the bundle table to be defined.

"colour" index refers to an entry in the colour table when output primitives are displayed.

end REPRESENT;

package TRANSFORM is

procedure SET_WINDOW

(transformation : in transformation_number;
window_limits : in wc.rectangle);

The window limits for the specified normalization "transformation" is set to the value specified in world coordinate points by "window_limits".

procedure SET_VIEWPORT

(transformation : in transformation_number;
viewport_limits : in ndc.rectangle);

The viewport limits for the specified normalization "transformation" is set to the value specified by the normalized device coordinates in "viewport_limits".

procedure SET_VIEWPORT_INPUT_PRIORITY

(transformation : in transformation_number;
reference_transformation : in transformation_number;
priority : in relative_priority);

The ordering of the normalization transformation with regard to input priority is changed such that "transformation" will be ordered according to "priority" as either having a "higher" or "lower" priority than the "reference_transformation".

category : out ws_category);
This function returns the "category" of the "type_of_ws".

procedure INQ_WS_CLASS
(type_of_ws : in ws_type;
ei : out error_indicator;
class : out display_class);
This function returns the display "class" of the "type_of_ws".

procedure INQ_DISPLAY_SPACE_SIZE
(type_of_ws : in ws_type;
ei : out error_indicator;
units : out dc_units;
max_dc_size : out dc.size;
max_raster_unit_size : out raster_unit_size);
This function returns the Device Coordinate "units", the maximum display surface size in Device Coordinate units "max_dc_size", and the maximum display surface size in raster units "max_raster_unit_size" for the "type_of_ws".

procedure INQ_DYNAMIC_MODIFICATION_OF_WS_ATTRIBUTES
(type_of_ws : in ws_type;
ei : out error_indicator;
polyline_representation : out dynamic_modification;
polymarker_representation : out dynamic_modification;
text_representation : out dynamic_modification;
fill_area_representation : out dynamic_modification;
pattern_representation : out dynamic_modification;
colour_representation : out dynamic_modification;
transformation : out dynamic_modification);
This function returns the polyline, polymarker, text, fill_area, pattern, and colour representation changeable. These tell whether a redrawing of the screen is needed if the representation is changed. "transformation" does the same thing for setting the transformations.

procedure INQ_DEFAULT_DEFERRAL_STATE_VALUES
(type_of_ws : in ws_type;
ei : out error_indicator;
deferral : out deferral_mode;
regeneration : out regeneration_mode);
This function returns the default value for "deferral" mode and the default value for "regeneration" mode for workstation type "type_of_ws".

end INQ_REPRESENT;

package INQ_FACILITIES is

procedure INQ_POLYLINE_FACILITIES
(type_of_ws : in ws_type;
ei : out error_indicator;
list_of_types : out line_types.list_of;

```
index : in pattern_index;
returned_values : in return_value_type;
ei : out error_indicator;
indices : out pattern_indices.list_of);
This function returns the "pattern" array of colour indices.
```

```
procedure INQ_PATTERN_REPRESENTATION
(ws : in ws_id;
index : in pattern_index;
returned_values : in return_value_type;
ei : out error_indicator;
pattern : out colour_indices.matrix_of);
This function returns "pattern" array of colour indices.
```

```
procedure INQ_LIST_OF_COLOUR_INDICES
(ws : in ws_id;
ei : out error_indicator;
indices : in colour_indices.list_of);
This function returns a list of color "indices" for workstation "ws".
```

```
procedure INQ_COLOUR_REPRESENTATION
(ws : in ws_id;
index : in colour_index;
returned_values : in return_value_type;
ei : out error_indicator;
colour : out colour_representation);
This function returns the "colour" for the specified color "index" being
inquired of on workstation "ws". The "returned_values" parameter indi-
cates whether the returned values should be as they were set by the pro-
gram, or as they were actually realized.
```

```
procedure INQ_WS_TRANSFORMATION
(ws : in ws_id;
ei : out error_indicator;
update : out update_state;
requested_window : out ndc.rectangle;
current_window : out ndc.rectangle;
requested_viewport : out dc.rectangle;
current_viewport : out dc.rectangle);
This procedure returns the workstation transformation "update" state, the
"requested_window", "current_window", "requested_viewport", and "current_
viewport".
```

```
procedure INQ_SET_OF_SEGMENT_NAMES_ON_WS
(ws : in ws_id;
ei : out error_indicator;
segments : out segment_names.list_of);
This function returns a set of stored segment names "segments" for work-
station "ws".
```

```
procedure INQ_WS_CATEGORY
(type_of_ws : in ws_type;
ei : out error_indicator;
```

```
returned_values : in return_value_type;
ei : out error_indicator;
marker : out marker_type;
size : out marker_size;
colour : out colour_index);
```

This function returns the 'marker' type, the marker "size" scale factor, and the polymarker "colour" index. The "returned_values" parameter indicates whether the returned values should be as set by the program, or as they were actually realized.

```
procedure INQ_LIST_OF_TEXT_INDICES
(ws : in ws_id;
ei : out error_indicator;
indices : out text_indices.list_of);
```

This function returns the list of text "indices" for workstation "ws".

```
procedure INQ_TEXT_REPRESENTATION
(ws : in ws_id;
index : in text_index;
returned_values : in return_value_type;
ei : out error_indicator;
font_precision : out text_font_precision;
expansion : out char_expansion;
spacing : out char_spacing;
colour : out colour_index);
```

This function returns the text "font_precision", the character "expansion" factor, the character "spacing", and the text "colour" index. The "returned_values" parameter indicates whether the returned values should be as set by the program, or as they were actually realized.

```
procedure INQ_LIST_OF_FILL_AREA_INDICES
(ws : in ws_id;
ei : out error_indicator;
indices : out fill_area_indices.list_of);
```

This function returns a list of defined fill area "indices" for workstation "ws".

```
procedure INQ_FILL_AREA_REPRESENTATION
(ws : in ws_id;
index : in fill_area_index;
returned_values : in return_value_type;
ei : out error_indicator;
style : out style_index;
colour : out colour_index);
```

This function returns the fill area "style" index and the fill area "colour" index. The fill area interior style is a discriminant component of the style index. The "return_values" parameter indicates whether the returned values should be as they were set by the program, or as they were actually realized. The "index" is the fill area index being inquired of on workstation "ws".

```
procedure INQ_LIST_OF_PATTERN_INDICES
(ws : in ws_id;
```

```
ei : out error_indicator;  
state : out ws_state);  
The "state" workstation "ws" is returned. The "state" is either "active"  
or "inactive".
```

```
procedure INQ_WS_DEFERRAL_AND_UPDATE_STATES
```

```
(ws : in ws_id;  
ei : out error_indicator;  
deferral : out deferral_mode;  
regeneration : out regeneration_mode;  
display : out display_surface_empty;  
frame_action : out new_frame_necessary);
```

This function returns the "deferral" mode, the "regeneration" mode, the "display" mode, and the new "frame_action" necessary for update for the workstation "ws".

```
end INQ_ATTRIBUTES;
```

```
package INQ_REPRESENT is
```

```
procedure INQ_LIST_OF_POLYLINE_INDICES
```

```
(ws : in ws_id;  
ei : out error_indicator;  
indices : out polyline_indices.list_of);
```

This function returns the list of defined polyline "indices" for the workstation "ws".

```
procedure INQ_POLYLINE_REPRESENTATION
```

```
(ws : in ws_id;  
index : in polyline_index;  
returned_values : in return_value_type;  
ei : out error_indicator;  
line : out line_type;  
width : out line_width;  
colour : out colour_index);
```

The function returns the "line" type, the line "width" scale factor, and the polyline "colour" index for the bundle specified by the polyline "index" on workstation "ws". The "returned_values" parameter indicates whether the returned "returned_values" parameter indicates whether the returned whether the returned values should be as set by the program, or as they were actually realized.

```
procedure INQ_LIST_OF_POLYMARKER_INDICES
```

```
(ws : in ws_id;  
ei : out error_indicator;  
indices : out polymarker_indices.list_of);
```

This function retruns the list of defined polymarker "indices" for "ws".

```
procedure INQ_POLYMARKER_REPRESENTATION
```

```
(ws : in ws_id;  
index : in polymarker_index;
```

(ei : out error_indicator;
transformation : out transformation_number);
The current normalization "transformation" number is returned.

procedure INQ_LIST_OF_NORMALIZATION_TRANSFORMATION_NUMBERS
(ei : out error_indicators;
list : out transformation_numbers.list_of);
This function returns a "list" of transformation numbers.

procedure INQ_NORMALIZATION_TRANSFORMATION
(transformation : in transformation_number;
ei : out error_indicator;
window_limits : out wc.rectangle;
viewport_limits : out ndc.rectangle);
This function returns the "window_limits" in world coordinates and the
"viewport_limits" in normalized device coordinates for the specified nor-
malization "transformation".

procedure INQ_CLIPPING
(ei : out error_indicator;
clipping : out clipping_indicator;
clipping_rectangle : out ndc.rectangle);
This function returns the "clipping" indicator and the "clipping_rectan-
gle."

procedure INQ_NAME_OF_OPEN_SEGMENT
(ei : out error_indicator;
segment : out segment_name);
This function returns the name of the current open "segment".

procedure INQ_SET_OF_SEGMENT_NAMES_IN_USE
(ei : out error_indicator;
segments : out segments_names.list_of);
The function returns the set "segments" of segment names in use. It in-
cludes the number of segment names in use.

procedure INQ_MORE_SIMULTANEOUS_EVENTS
(ei : out error_indicator;
events : out more_events);
A value of "more" or "nomore" will be returned for "events" to indicate
whether there are other input reports in the same group of simultaneous
events as the last removed report.

procedure INQ_WS_CONNECTION_AND_TYPE
(ws : in ws_id;
ei : out error_indicator;
connection : out connection_id;
type_of_ws : out ws_type);
This function returns the connection "identifier" and the "type_of_ws"
for the workstation "ws".

procedure INQ_WS_STATE
(ws : in ws_id;

procedure INQ_POLYMARKER_SIZE_SCALE_FACTOR

(ei : out error_indicator;
size : out scale_factor);

This function returns the current marker "size".

procedure INQ_POLYMARKER_COLOUR_INDEX

(ei : out error_indicator;
colour : out colour_index);

This function returns the current polymarker "colour" index.

procedure INQ_TEXT_FONT_AND_PRECISION

(ei : out error_indicator;
font_precision : out text_font_precision);

This function returns the current text font and precision (font_precision).

procedure INQ_CHAR_EXPANSION_FACTOR

(ei : out error_indicator;
expansion : out char_expansion);

This function returns the current character "expansion" factor.

procedure INQ_CHAR_SPACING

(ei : out error_indicator;
spacing : out char_spacing);

This function returns the current character "spacing".

procedure INQ_TEXT_COLOUR_INDEX

(ei : out error_indicator;
colour : out colour_index);

This function returns the current text "colour".

procedure INQ_FILL_AREA_INTERIOR_STYLE

(ei : out error_indicator;
style : out interior_style);

This function returns the current fill area interior "style"

procedure INQ_FILL_AREA_STYLE_INDEX

(ei : out error_indicator;
index : out style_index);

This function returns the current fill area "style" index.

procedure INQ_FILL_AREA_COLOUR_INDEX

(ei : out error_indicator;
colour : out colour_index);

This function returns the current fill area "colour" index.

procedure INQ_LIST_OF_ASF

(ei : out error_indicator;
list : out asf_list);

This function returns the list of aspect source flags.

procedure INQ_CURRENT_NORMALIZATION_TRANSFORMATION_NUMBER

```

    (ei : out error_indicator;
     vector : out wc.vector);
This function returns the current character up "vector".

procedure INQ_TEXT_PATH
    (ei : out error_indicator;
     path : out text_path);
This function returns the current text "path".

procedure INQ_TEXT_ALIGNMENT
    (ei : out error_indicator;
     alignment : out text_alignment);
This function returns the current text "alignment".

procedure INQ_FILL_AREA_INDEX
    (ei : out error_indicator;
     index : out fill_area_index);
This function returns the current fill area "index".

procedure INQ_PATTERN_SIZE
    (ei : out error_indicator;
     size : out wc.size);
This function returns the current pattern "size".

procedure INQ_PATTERN_REFERENCE_POINT
    (ei : out error_indicator;
     reference_point : out wc.point);
This function returns the current pattern "reference_point".

procedure INQ_PICK_ID
    (ei : out error_indicator;
     pick : out pick_id);
This function returns the current "pick" identifier.

procedure INQ_LINE_TYPE
    (ei : out error_indicator;
     line : out line_type);
This function returns the current "line" type.

procedure INQ_LINEWIDTH_SCALE_FACTOR
    (ei : out error_indicator;
     width : out line_width);
This function returns the current line "width" scale factor.

procedure INQ_POLYLINE_COLOUR_INDEX
    (ei : out error_indicator;
     colour : out colour_index);
This function returns the current polyline "colour" index.

procedure INQ_POLYMARKER_TYPE
    (ei : out error_indicator;
     marker : out marker_type);
This function returns the current "marker" type

```

```
procedure INQ_LIST_OF_AVAILABLE_WS_TYPES
  (ei : out error_indicator;
   types : out ws_types.list_of);
This function returns a list of workstation types.
```

```
procedure INQ_WS_MAX_NUMBERS
  (ei : out error_indicator;
   max_open_ws : out positive;
   max_active_ws : out positive;
   max_segment_ws : out positive);
This function returns the maximum number of simultaneously open workstations (max_open_ws), the maximum number of active workstations (max_active_ws), and the maximum number of workstations associated with segment (max_segment_ws).
```

```
procedure INQ_MAX_NORMALIZATION_TRANSFORMATION_NUMBER
  (ei : out error_indicator;
   transformation : out transformation_number);
This function returns the maximum normalization "transformation" number allowed by this implementation of GKS.
```

```
procedure INQ_SET_OF_OPEN_WS
  (ei : out error_indicator;
   ws : out ws_ids.list_of);
A list of open workstations is returned.
```

```
procedure INQ_SET_OF_ACTIVE_WS
  (ei : out error_indicator;
   ws : out ws_ids.list_of);
A list of active workstations is returned.
```

```
procedure INQ_POLYLINE_INDEX
  (ei : out error_indicator;
   index : out polyline_index);
This function returns the current polyline "index".
```

```
procedure INQ_POLYMARKER_INDEX
  (ei : out error_indicator;
   index : out polymarker_index);
This function returns the current polymarker "index".
```

```
procedure INQ_TEXT_INDEX
  (ei : out error_indicator;
   index : out text_index);
This function returns the current text "index".
```

```
procedure INQ_CHAR_HEIGHT
  (ei : out error_indicator;
   height : out char_height);
This function returns the current character "height".
```

```
procedure INQ_CHAR_UP_VECTOR
```

GKS performs a request on the specified stroke "device" on the specified "ws". The current measure of the stroke device consists of a sequence of "stroke_points" in world coordinates and the normalization "transformation" number which was used in the conversion to World Coordinates.

```
procedure REQUEST_VALUATOR
  (ws : in ws_id;
   device : in device_number;
   value : out input_value);
```

GKS performs a request on the specified valuator "device" number on workstation "ws". The "value" returned is the current measure of the valuator device.

```
procedure REQUEST_CHOICE
  (ws : in ws_id;
   device : in device_number;
   choice : out choice_input_type);
```

GKS performs a request on the specified choice "device" on workstation "ws". The "choice" returned is the current measure of the "choice" device.

```
procedure REQUEST_PICK
  (ws : in ws_id;
   device : in device_number;
   pick : out pick_input_type);
```

GKS performs a request on the specified pick "device" on workstation "ws". If the measure of the pick device indicates no pick, "status" is returned "nopick"; otherwise, "ok" is returned together with a "segment" name and "pick" identifier which are set according to the current measure of the pick device.

```
procedure REQUEST_STRING
  (ws : in ws_id;
   device : in device_number;
   char_string : out input_string);
```

GKS performs a request on the specified string "device" on workstation "ws". The "char_string" returned is the current measure of the string device.

end INPUT;

package INQ_ATTRIBUTES is

```
procedure INQ_OPERATING_STATE_VALUE
  (value : out operating_state);
```

This function returns the "value" of the GKS operating state.

```
procedure INQ_LEVEL_OF_GKS
  (ei : out error_indicator;
   level : out gks_level);
```

The level of this GKS implementation is returned.

```
procedure SET_VALUATOR_MODE
  (ws : in ws_id;
   device : in device_number;
   mode : in operating_mode;
   switch : in echo_switch);
Input device number device on workstation "ws" is set to the specified
"mode", and the echoing is set to "switch".
```

```
procedure SET_CHOICE_MODE
  (ws : in ws_id;
   device : in device_number;
   mode : in operating_mode;
   switch : in echo_switch);
Input device number device on workstation "ws" is set to the specified
"mode", and the echoing is set to "switch".
```

```
procedure SET_PICK_MODE
  (ws : in ws_id;
   device : in device_number;
   mode : in operating_mode;
   switch : in echo_switch);
Input device number device on workstation "ws" is set to the specified
"mode", and the echoing is set to "switch".
```

```
procedure SET_STRING_MODE
  (ws : in ws_id;
   device : in device_number;
   mode : in operating_mode;
   switch : in echo_switch);
Input device number device on workstation "ws" is set to the specified
"mode", and the echoing is set to "switch".
```

```
end SET_INPUT;
```

```
package INPUT is
```

```
procedure REQUEST_LOCATOR
  (ws : in ws_id;
   device : in device_number;
   transformation : out transformation_number;
   position : out wc.point);
GKS performs a request on the specified locator device number "device"
on the specified "ws". The locator "position" in World Coordinates and
the normalization "transformation" number, which was used in the conver-
sion to World Coordinates, are the current measure of the locator device.
```

```
procedure REQUEST_STROKE
  (ws : in ws_id;
   device : in device_number;
   transformation : out transformation_number;
   stroke_points : out points.list_of);
```

```
    echo_area : in dc.rectangle;
    data_record : in choice_data_record);
```

The input device with device number "device for the workstation "ws" is initialized. This function provides the following information to the device:

- The "initial_choice" number.
- The "echo_area" rectangle in device coordinates.
- the choice "data_record".

```
procedure INITIALISE_PICK
  (ws : in ws_id;
   device : in device_number;
   initial_pick : in pick_input_type;
   echo_area : in dc.rectangle;
   data_record : in pick_data_record);
```

The input device with device number "device" for the workstation "ws" is initialized. This function provides the following information to the device:

- The "initial_status" of "pick" or "no_pick".
- The name of the "initial_segment".
- The "initial_pick" identifier.
- The "echo_area" rectangle in device coordinates.
- The pick "data_record".

```
procedure INITIALISE_STRING
  (ws : in ws_id;
   device : in device_number;
   initial_string : in input_string;
   echo_area : in dc.rectangle;
   data_record : in string_data_record);
```

The input device with device number "device" for the workstation "ws" is initialized. This function provides the following information to the device:

- The "initial_string" which contains the initial input string.
- The "echo_area" rectangle in device coordinates.
- The string "data_record".

```
procedure SET_LOCATOR_MODE
  (ws : in ws_id;
   device : in device_number;
   mode : in operating_mode;
   switch : in echo_switch);
```

Input device number device on workstation "ws" is set to the specified "mode", and the echoing is set to "switch".

```
procedure SET_STROKE_MODE
  (ws : in ws_id;
   device : in device_number;
   mode : in operating_mode;
   switch : in echo_switch);
```

Input device number device on workstation "ws" is set to the specified "mode", and the echoing is set to "switch".

package SET_INPUT is

procedure INITIALISE_LOCATOR

(ws : in ws_id;
device : in device_number;
initial_transformation : in transformation_number;
initial_position : in wc.point;
echo_area : in dc.rectangle;
data_record : in locator_data_record);

The input device with device number "device" for the workstation "ws" is initialized. The function provides the following information to the device:

The "initial_position" of the locator in world coordinates.
The "initial_transformation" which provides the initial normalization transformation number.
The "echo_area" rectangle in device coordinates.
The locator "data_record".

procedure INITIALISE_STROKE

(ws : in ws_id;
device : in device_number;
initial_transformation : in transformation_number;
initial_stroke : in points.array_of;
echo_area : in dc.rectangle;
data_record : in stroke_data_record);

The input device with device number "device" for the workstation "ws" is initialized. The function provides the following information to the device:

"initial_stroke" which contains the number of points in the initial stroke and the points in the stroke.
"initial_transformation" which provides the initial normalization transformation number.
The "echo_area" rectangle in device coordinates
"data_record" which provides the stroke data record.

procedure INITIALISE_VALUATOR

(ws : in ws_id;
device : in device_number;
initial_value : in input_value;
echo_area : in dc.rectangle;
data_record : in valuator_data_record);

The input device with device number "device" for the workstation "ws" is initialized. The function provides the following information to the device:

"initial_value" which contains the initial value.
The "echo_area" rectangle in device coordinates.
The valuator "data_record".

procedure INITIALISE_CHOICE

(ws : in ws_id;
device : in device_number;
initial_choice : in choice_input_type;

segment : in segment_name);
The "segment" is sent to the workstation "ws" in the same way as if the workstation were active when the segment was created. Clipping rectangles are copied unchanged.

```
procedure COPY_SEGMENT_TO_WS  
  (ws : in ws_id;  
   segment : in segment_name);
```

The primitives in the segment pointed to by "segment" are sent to the specified "ws".

```
procedure INSERT_SEGMENT  
  (segment : in segment_name;  
   transformation : in transformation_matrix);
```

INSERT_SEGMENT allows previously stored primitives to be transformed, according to the transformed coordinates specified by "transformation", and again placed into the stream of output primitives.

```
procedure SET_SEGMENT_TRANSFORMATION  
  (segment : in segment_name;  
   transformation : in transformation_matrix);
```

This function transforms the "segment" as specified by the "transformation" matrix.

```
procedure SET_VISIBILITY  
  (segment : in segment_name;  
   visibility : in segment_visibility);
```

The visibility of the specified "segment" is set to the value of "visibility".

```
procedure SET_HIGHLIGHTING  
  (segment : in segment_name;  
   highlighting : in segment_highlighting);
```

The highlighting of the specified "segment" is set to the value of "highlighting".

```
procedure SET_SEGMENT_PRIORITY  
  (segment : in segment_name;  
   priority : in segment_priority);
```

The segment priority of the named "segment" is set to the value specified by "priority". Segment priority affects the display of segments and pick input if segments overlap, in which case GKS gives precedence to segments of higher priority.

```
procedure SET_DETECTABILITY  
  (segment : in segment_name;  
   detectability : in segment_detectability);
```

The detectability of the specified "segment" is set to the value of "detectability".

```
end SEGMENTS;
```

procedure SELECT_NORMALIZATION_TRANSFORMATION
(transformation : in transformation_number);
The specified normalization TRANSFORMATION becomes the current normalization transform to be used for subsequent output primitives.

procedure SET_CLIPPING_INDICATOR
(clipping : in clipping_indicator);
The clipping indicator is set to specify whether there is clipping or not by the parameter "clipping".

procedure SET_WS_WINDOW
(ws : in ws_id;
ws_window_limits : in ndc.rectangle);
The requested workstation window on workstation "ws" is set to the rectangle specified by the normalized device coordinates "ws_window_limits".

procedure SET_WS_VIEWPORT
(ws : in ws_id;
ws_viewport_limits : in dc.rectangle);
The requested workstation viewport on workstation "ws" is set to the rectangle specified by the device coordinates "ws_viewport_limits".

end TRANSFORM;

package SEGMENTS is

procedure CREATE_SEGMENT is
(segment : in segment_name);
This function creates a new segment. The segment is stored on all workstations active at the time the segment is created.

procedure CLOSE_SEGMENT;
The current open segment is closed.

procedure RENAME_SEGMENT
(old_name : in segment_name;
new_name : in segment_name);
The name attribute of the segment is changed from "old_name" to "new_name".

procedure DELETE_SEGMENT
(segment : in segment_name);
The segment specified by "segment" is deleted on all workstations.

procedure DELETE_SEGMENT_FROM_WS
(ws : in ws_id;
segment : in segment_name);
The specified "segment" is deleted from the workstation "ws".

procedure ASSOCIATE_SEGMENT_WITH_WS
(ws : in ws_id;

```
number_of_widths : out line_types.list_of;  
nominal_width : out dc.magnitude;  
range_of_widths : out dc.limits;  
number_of_indices : out natural);
```

This function returns a list of available linetypes (list_of_types), the number of available linewidths (number_of_widths), the "nominal_width", the range of linewidths (range_of_widths), and the number of predefined polyline indices (number_of_indices) for workstation type (type_of_ws).

procedure INQ_PREDEFINED_POLYLINE_REPRESENTATION

```
(type_of_ws : in ws_type;  
index : in polyline_index;  
ei : out error_indicator;  
line : out line_type;  
width : out line_width;  
colour : out colour_index);
```

This function returns the linetype (line), the linewidth scale factor (width), and the polyline "colour" index. The "index" is the polyline index being inquired on the "type_of_ws".

procedure INQ_POLYMARKER_FACILITIES

```
(type_of_ws : in ws_type;  
ei : out error_indicator;  
list_of_types : out marker_types.list_of;  
number_of_sizes : out natural;  
nominal_size : out dc.magnitude;  
range_of_sizes : out dc.limits;  
number_of_indices : out natural);
```

This function returns the list of available marker types (list_of_types), the number of available marker sizes (number_of_sizes), the "nominal_size" of the marker, the range of marker sizes (range_of_sizes), and the number of predefined polyarker indices (number_of_indices) for the "type_of_ws".

procedure INQ_PREDEFINED_POLYMARKER_REPRESENTATION

```
(type_of_ws : in ws_type;  
index : in polymarker_index;  
ei : out error_indicator;  
marker : out marker_type;  
size : out marker_size;  
colour : out colour_index);
```

This function returns the "marker", the marker "size" scale factor, and the polymarker "colour" index for the polymarker "index" being inquired on this "type_of_ws".

procedure INQ_TEXT_FACILITIES

```
(type_of_ws : in ws_type;  
ei : out error_indicator;  
list_of_font_precision_pairs :  
    out text_font_precisions.list_of;  
number_of_heights : out natural;  
range_of_heights : out dc.limits;  
number_of_expansions : out natural);
```

```
range_of_expansions : out dc.limits;  
number_of_indices : out natural);  
This function returns a "list_of_font_precision_pairs" available, the  
number of available character heights (number_of_heights), the minimum  
and maximum character heights (range_of_heights), the number of available  
character expansion factors (number_of_expansions), the minimum and maximum  
character expansion factors (range_of_expansions), and the number  
of predefined text indices (number_of_indices) for the "type_of_ws".
```

```
procedure INQ_PREDEFINED_TEXT_REPRESENTATION
```

```
(type_of_ws : in ws_type;  
index : in text_index;  
ei : out error_indicator;  
font_precision : out text_font_precision;  
expansion : out char_expansion;  
spacing : out char_spacing;  
colour : out colour_index);
```

This function returns the text font and precision (font_precision), the
character "expansion" factor, the character "spacing", and the text "col-
out" index for the predefined text "index" on this "type_of_ws".

```
procedure INQ_FILL_AREA_FACILITIES
```

```
(type_of_ws : in ws_type;  
ei : out error_indicator;  
list_of_interior_styles : out interior_styles.list_of;  
list_of_hatch_styles : out hatch_styles.list_of;  
number_of_indices : out natural);
```

This function returns the "list_of_interior_style" available, the "list
of_hatch_styles" available, and the number of predefined fill area in-
dices (number_of_indices) for the "type_of_ws".

```
procedure INQ_PREDEFINED_FILL_AREA_REPRESENTATION
```

```
(type_of_ws : ws_type;  
index : in fill_area_index;  
ei : out error_indicator;  
style : out style_index;  
colour : out colour_index);
```

This function returns the fill area "style" index and the fill area "co-
lour" index for the fill area "index" being inquired on this "type_of_
ws".

```
procedure INQ_PATTERN_FACILITIES
```

```
(type_of_ws : in ws_type;  
ei : out error_indicator;  
number_of_indices : out natural);
```

This function returns the number of predefined pattern indices (number_
of_indices) for this "type_of_ws".

```
procedure INQ_PREDEFINED_PATTERN_REPRESENTATION
```

```
(type_of_ws : in ws_type;  
index : in pattern_index;  
ei : out error_indicator;  
pattern : out colour_indices.variable_matrix_of);
```

This function returns the "pattern" array of colour indices for the pattern "index" being inquired on this "type_of_ws".

```
procedure INQ_COLOUR_FACILITIES
```

```
(type_of_ws : in ws_type;  
ei : out error_indicator;  
number_of_colours : out positive);
```

This function returns the "number_of_colours" supported on this "type_of_ws".

```
procedure INQ_PREDEFINED_COLOUR_REPRESENTATION
```

```
(type_of_ws : in ws_type;  
index : in colour_index;  
ei : out error_indicator;  
colour : out colour_representation);
```

This function returns the "colour" for the color "index" being inquired on this "type_of_ws".

```
procedure INQ_LIST_OF_AVAILABLE_GDP
```

```
(type_of_ws : in ws_type;  
ei : out error_indicator;  
list_of_gdp : out gdp_ids.list_of);
```

This function returns the "list_of_gdp" identifiers for the "type_of_ws".

```
procedure INQ_GDP
```

```
(type_of_ws : in ws_type;  
gdp : in gdp_id;  
ei : out error_indicator;  
list_of_attributes_used : out attributes_used.list_of);
```

This function returns the list of sets of attributes used (list_of_attributes_used) for the "gdp" identifier on the "type_of_ws".

```
end INQ_FACILITIES;
```

```
-----  
package INQ_SEGMENT is
```

```
procedure INQ_MAX_LENGTH_OF_WS_STATE_TABLES
```

```
(type_of_ws : in ws_type;  
ei : out error_indicator;  
max_polyline_entries : out natural;  
max_polymarker_entries : out natural;  
max_text_entries : out natural;  
max_fill_area_entries : out natural;  
max_pattern_indices : out natural;  
max_colour_indices : out natural);
```

This function returns the maximum number of polyline, polymarker, text, and fill area table entries. It also returns the maximum number of pattern and colour indices on this "type_of_ws".

```
procedure INQ_NUMBER_OF_SEGMENT_PRIORITIES_SUPPORTED
```

```
(type_of_ws : in ws_type;
```

```
    ei : out error_indicator;  
    number_of_priorities : out natural);  
This function returns the number of segment priorities (number_of_priorities) supported on this "type_of_ws".
```

```
procedure INQ_DYNAMIC_MODIFICATION_OF_SEGMENT_ATTRIBUTES
```

```
  (type_of_ws : in ws_type;  
   ei : out error_indicator;  
   transformation : out dynamic_modification;  
   visible_to_invisible : out dynamic_modification;  
   invisible_to_visible : out dynamic_modification;  
   highlighting : out dynamic_modification;  
   priority : out dynamic_modification;  
   adding_primitives : out dynamic_modification;  
   deletion_visible : out dynamic_modification);
```

This function returns the segment "transformation" changeable, the visibility changeable from "visible_to_invisible", the visibility changeable from "invisible_to_visible", the segment "priority" changeable, "adding_primitives" to open segment, and the segment "deletion_visible" for this "deletion_visible" for this "type_of_ws".

```
procedure INQ_SET_OF_ASSOCIATED_WS
```

```
  (segment : in segment_name;  
   ei : out error_indicator;  
   list_of_ws : out ws_ids.list_of);
```

This function returns the set of workstations, (list_of_ws), associated with "segment".

```
procedure INQ_SEGMENT_ATTRIBUTES
```

```
  (segment : in segment_name;  
   ei : out error_indicator;  
   transformation : out transformation_matrix;  
   visibility : out segment_visibility;  
   highlighting : out segment_highlighting;  
   priority : out segment_priority;  
   detectability : out segment_detectability);
```

This function returns the segment "transformation" matrix, the segment "visibility", the segment "highlighting", the segment "priority", and the segment "detectability" for the given "segment".

```
end INQ_SEGMENT;
```

```
-----  
package SET_TRANSFORM is
```

```
procedure EVALUATE_TRANSFORMATION_MATRIX
```

```
  (fixed_point : in wc.point;  
   shift_vector : in wc.vector;  
   rotation_angle : in radians;  
   scale_factors : in transformation_factor;  
   transformation : out transformation_matrix);
```

The transformation specified by "fixed_point", "shift_vector", "rotation_

angle", and "scale_factors", is evaluated and the result is put in "transformation".

```
procedure EVALUATE_TRANSFORMATION_MATRIX
```

```
(fixed_point : in ndc.point;  
shift_vector : in ndc.vector;  
rotation_angle : in radians;  
scale_factors : in transformation_factor;  
transformation : out transformation_matrix);
```

The transformation specified by "fixed_point", "shift_vector", "rotation_angle", and "scale_factors", is evaluated and the result is put in "transformation".

```
procedure ACCUMULATE_TRANSFORMATION_MATRIX
```

```
(source_transformation : in transformation_matrix;  
fixed_point : in wc.point;  
shift_vector : in wc.vector;  
rotation_angle : in radians;  
scale_factors : in transformation_factor;  
result_transformation : in transformation_matrix);
```

The transformation defined by "fixed_point", "shift_vector", "rotation_angle", and "scale_factors", is premultiplied by the "source_transformation" and the result is returned in "result_transformation".

```
procedure ACCUMULATE_TRANSFORMATION_MATRIX
```

```
(source_transformation : in transformation_matrix;  
fixed_point : in ndc.point;  
shift_vector : in ndc.vector;  
rotation_angle : in radians;  
scale_factors : in transformation_factor;  
result_transformation : in transformation_matrix);
```

The transformation defined by "fixed_point", "shift_vector", "rotation_angle", and "scale_factors", is premultiplied by the "source_transformation" and the result is returned in "result_transformation".

```
end SET_TRANSFORM;
```

```
package EMERGENCY is
```

```
procedure EMERGENCY_CLOSE_GKS;
```

This function is used to close GKS in case of a nonrecoverable error. Any open segment is closed. All workstations are updated. All active workstations are deactivated. All open workstations are closed. GKS is closed.

```
end EMERGENCY;
```

```
package ERROR_HANDLING is
```

```
state_error, ws_error, transformation_error : exception;  
output_attribute_error, output_primitive_error : exception;  
segment_error, input_error, language_binding_error : exception;
```

```
procedure ERROR_LOGGING
```

```
  (ei : in error_indicator;  
   name : in subprogram_name);
```

```
This function writes the error "number" and the GKS function "name" de-  
tecting the error to the error file specified in "open_gks".
```

```
end ERROR_HANDLING;
```

AFIT_GKS Errors (6:59-64)

This binding requires the use of Ada exceptions to notify the application program of error conditions detected by AFIT_GKS functions, except the inquiry functions. The exceptions correspond to the classes of errors described in Appendix B of the ANS GKS specification. The Ada concept of allowing the application program to provide exception handlers replaces the ANS GKS requirement of a global user-supplied Error Handling procedure. When an exception is raised, the application program may read the error file to get the error number, the name of the subprogram detecting the error, and any implementation-defined messages.

The AFIT_GKS inquiry functions do not raise exceptions. Instead, they return an error indicator parameter containing the number of the "error" which was detected. This is consistent with the ANS GKS philosophy that no errors occur during inquiries. The error numbers correspond to the error numbers from Appendix B of the ANS GKS specification, plus additional errors defined in this binding. Note that certain known error conditions may be detected outside the control of AFIT_GKS due to the nature of the Ada language, and may result in an exception being raised on an inquiry.

Error Code Definition

This section provides the mapping of the ANS GKS error numbers to Ada exceptions. The names of the exceptions correspond to the classes of errors defined in Appendix B of the ANS GKS specification. For each of these error "classes", the number of the errors covered by this exception are specified. These numbers are also the same numbers used as error indicator return values on inquiries. Certain of the known ANS GKS errors will never be detected by AFIT_GKS due to features of the Ada language, such as strong data typing. These errors are not included in this section.

STATE_ERROR

The State_Error exception is raised when a AFIT_GKS function is called from an incorrect state. The following ANS GKS error conditions correspond to this exception:

- 1 GKS not in proper state: GKS shall be in state GKCL
- 2 GKS not in proper state: GKS shall be in state GKOP
- 3 GKS not in proper state: GKS shall be in state WSAC
- 4 GKS not in proper state: GKS shall be in state SGOP
- 5 GKS not in proper state: GKS shall be in either state WSAC or in state SGOP
- 6 GKS not in proper state: GKS shall be in either state WSOP or in state SGOP
- 7 GKS not in proper state: GKS shall be in one of the states WSOP, WSAC or SGOP
- 8 GKS not in proper state: GKS shall be in one of the states GKOP, WSOP, WSAC or SGOP

WS_ERROR

The exception `WS_ERROR` is raised when an error occurs during manipulation of a workstation. The following error numbers correspond to this exception:

- 21 Specified connection identifier is invalid
- 22 Specified workstation type is invalid
- 24 Specified workstation is open
- 25 Specified workstation is not open
- 26 Specified workstation cannot be opened
- 27 Workstation Independent Segment Storage is not open
- 28 Workstation Independent Segment Storage is already open
- 29 Specified workstation is active
- 30 Specified workstation is not active
- 31 Specified workstation is of category MO
- 32 Specified workstation is not of category MO
- 33 Specified workstation is of category MI
- 34 Specified workstation is not of category MI
- 35 Specified workstation is of category INPUT
- 36 Specified workstation is Workstation Independent Segment Storage
- 37 Specified workstation is not of category OUTIN
- 38 Specified workstation is neither of category INPUT nor of category OUTIN
- 39 Specified workstation is neither of category OUTPUT nor of category OUTIN
- 40 Specified workstation has no pixel store readback capability
- 41 Specified workstation type is not able to generate the specified generalized drawing primitive

TRANSFORMATION_ERROR

The `TRANSFORMATION_ERROR` exception is raised when an error occurs during a transformation manipulation. The following error numbers correspond to this exception:

- 51 Rectangle definition is invalid
- 54 Workstation viewport is not within the display space

OUTPUT_ATTRIBUTE_ERROR

The `OUTPUT_ATTRIBUTE_ERROR` exception is raised when an error occurs during manipulation of an output attribute. The following ANS GKS error numbers correspond to this exception:

- 61 A representation for the specified polyline index has not been defined on this workstation
- 63 Specified linetype is not supported on this workstation
- 65 A representation for the specified polymarker index has not been defined on this workstation
- 67 Specified marker type is not supported on this workstation
- 69 A representation for the specified text index has not been defined on this workstation

- 71 Requested text font is not supported for the specified precision on this workstation
- 74 Length of character up vector is zero
- 76 A representation for the specified fill area index has not been defined on this workstation
- 77 Specified fill area interior style is not supported on this workstation.
- 80 Specified hatch style is not supported on this workstation
- 82 A representation for the specified pattern index has not been defined on this workstation
- 83 Interior style PATTERN is not supported on this workstation
- 87 A representation for the specified colour index has not been defined on this workstation

OUTPUT_PRIMITIVE_ERROR

The exception OUTPUT_PRIMITIVE_ERROR is raised when an error occurs during manipulation of an output primitive. The following ANS GKS error numbers correspond to this exception:

- 100 Number of points is invalid
- 103 Content of generalized drawing primitive data record is invalid
- 104 At least one active workstation is not able to generate the specified generalized drawing primitive

SEGMENT_ERROR

The exception SEGMENT_ERROR is raised if an error is detected during manipulation of a segment. The following ANS GKS error numbers correspond to this exception:

- 121 Specified segment name is already in use
- 122 Specified segment does not exist
- 123 Specified segment does not exist on specified workstation
- 124 Specified segment does not exist on Workstation Independent Segment Storage
- 125 Specified segment is open

INPUT_ERROR

The exception INPUT_ERROR is raised when an error is detected during an AFIT_GKS input operation. The following ANS GKS error numbers correspond to this exception:

- 140 Specified input device is not present on workstation
- 141 Input device is not in REQUEST mode
- 142 Input device is not in SAMPLE mode
- 143 EVENT and SAMPLE input mode are not available at this level of GKS
- 144 Specified prompt and echo type is not supported on this workstation
- 145 Echo area is outside display space
- 147 Input queue has overflowed
- 150 No input value of the correct class is in the current event report

LANGUAGE_BINDING_ERROR

LANGUAGE_BINDING_ERROR is raised when an error is detected that is specific to this binding of ANS GKS to Ada. Error numbers 500 to 600 are reserved for language binding dependent errors. The following error numbers are defined by this binding for the specific identification of language binding errors:

- 500 Error file identification is invalid
- 503 Invalid use of input data record
- 504 Operator break on input
- 505 Timeout occurred before input received

Error Codes Precluded by Function but included in the Inquiry Procedures.

- 50 Transformation number is invalid
- 60 Polyline index is invalid
- 64 Polymarker index is invalid
- 68 Text index is invalid
- 75 Fill area index is invalid
- 79 Specified pattern index is invalid
- 86 Colour index is invalid

Sample Program

This section of Appendix A shows a small sample program that uses AFIT_GKS. This sample program, shown in Figure A.1, draws the house shown in Figure A.2. Note that the procedure demo in Figure A.1 only included those AFIT_GKS packages that it needed. If the user of AFIT_GKS needed a function from some other package of AFIT_GKS then he/she would have to include those additional packages.

```

with external_types, control, primitives, transform;
procedure DEMO is
use external_types, control, primitives, transform;
wind : wc.rectangle :=
    (x => (min => 0.0, max => 50.0),
     y => (min => 0.0, max => 50.0));

procedure HOUSE is
roof : points.array_of(1..3) :=
    (1 => (x => 10.0, y => 30.0),
     2 => (x => 20.0, y => 40.0),
     3 => (x => 30.0, y => 30.0));
wall : points.array_of(1..4) :=
    (1 => (x => 10.0, y => 10.0),
     2 => (x => 10.0, y => 30.0),
     3 => (x => 30.0, y => 30.0),
     4 => (x => 30.0, y => 10.0));
door : points.array_of(1..4) :=
    (1 => (x => 15.0, y => 10.0),
     2 => (x => 15.0, y => 25.0),
     3 => (x => 20.0, y => 25.0),
     4 => (x => 20.0, y => 10.0));
handle : points.array_of(1..1) :=
    (1 => (x => 19.0, y => 17.5));
text_pt : wc.point :=
    (x => 35.0, y => 10.0);
begin
polyline(roof); -- draw the roof
fill_area(wall); -- draw the wall
polyline(door); -- draw the door
polymarker(handle); -- draw the door handle
text(text_pt, "house"); -- label the house
end HOUSE;

begin -- DEMO
open_gks;
open_ws(ws => 2,
        connection => "tek_4014",
        type_of_ws => 1);
activate_ws(2);
set_window(transformation => 1,
           window_limits => wind);
select_normalization_transformation(1);
HOUSE;
deactivate_ws(2);
close_ws(2);
close_gks;
end DEMO;

```

Figure A.1. DEMO Program

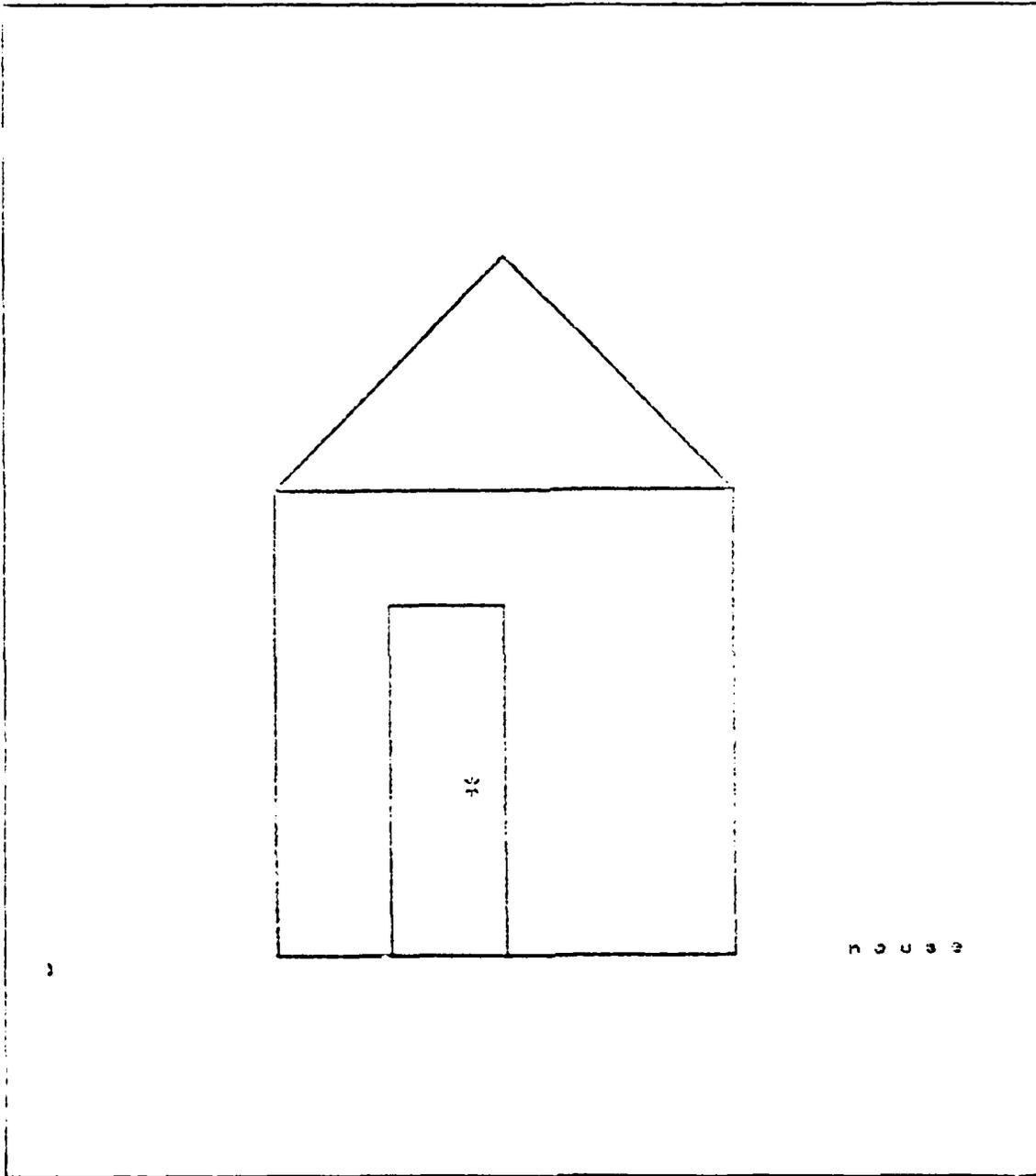


Figure A.2. Output of Demo Program

System Dependent Features of AFIT_GKS

The user of AFIT_GKS needs to know a few things before trying to use AFIT_GKS.

First, the packages of AFIT_GKS are located in the site libraries of the ROLM Data General computer located at the ASD Computer Center. Therefore, if the users account has access to the site libraries then to use AFIT_GKS the user simply "with"s the AFIT_GKS packages into his/her program just like the library package "text_io".

Second, in order to use the "open_ws" command of gks correctly the user must call:

```
open_ws(ws => <any workstation id>,  
        connection => "tek-4014",  
        type_of_ws => 1);
```

to open the Tektronix 4014 workstation. The user must call:

```
open_ws(ws => <any workstation id>,  
        connection => "tek-4027",  
        type_of_ws => 2);
```

to open the Tektronix 4027 workstation. Finally, the user must call:

```
open_ws(ws => <any workstation id>,  
        connection => "wiss",  
        type_of_ws => 3);
```

to open the Workstation Independent Segment Storage (WISS) workstation.

Centrum Amsterdam Netherlands, Department Computer Science, April 1981 (AD-8059606).

15. -----. "Graphics Standards -- Where are We?," Eurographics '81, 71-73 (September 1981).

Bibliography

1. ACM SIGGRAPH. "Special GKS Issue," Computer Graphics, (February 1984).
2. Booch, Grady. Software Engineering with Ada. Menlo Park, California: Benjamin/Cummings, 1983.
3. Department of Defense. Military Standard Ada Programming Language. ANSI/MIL-STD-1815A. Washington: Ada Joint Programming Office, (February 1983).
4. Ducrot and others. "A GKS Implementation for Meteorological Applications," Eurographics '81, 101-102 (September 1981).
5. Enderle, G. and others. Computer Graphics Programming GKS -- The Graphics Standard. Berlin: Springer-Verlag, 1984.
6. Harris Corporation. Draft GKS Binding to ANSI Ada. GKS/Ada Binding. Government Information Systems Software Operation, 407 John Rodes Blvd., Melbourne, Florida 32901, 20 July 1984 (Contract F49642-83-C0083).
7. Hopgood, F.R.A. and others. Introduction to the Graphical Kernel System GKS. London: Academic Press, 1983.
8. Ica, R. EZDRAW -- An Interactive Computer Graphics Program to Design Bar, Line, or Pie Graphs. MS thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1982 (AD-A124694).
9. Lindener, R. and J. Rix. "A GKS Interface to a Real Time Oriented Raster Workstation for CAD Applications," Eurographics '81, 114 (September 1981).
10. Peters, Lawrence J. Software Design: Methods & Techniques. New York: Yourdon Press, 1981.
11. Rose, K.W. Development of an Interactive Computer Graphics System Library and Graphics Tool. MS thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1982 (AD-A124694).
12. Simons, Randall W. "Minimal GKS," Computer Graphics, 17 number 3: 183-189 (July 1983).
13. Tektronix. 4027A Color Graphics Terminal. Programmer's Reference Manual. Tektronix, USA, 1981 (Part No. 070-4173-00).
14. ten Hagen, Paul J.W. The GKS Reviewing Process. Mathematisch

```
set_pick_data_record;  
function prompt_echo_type;  
set_string_data_record;  
function prompt_echo_type;  
function input_buffer_size;  
function initial_cursor_position;  
end EXTERNAL_TYPES;
```

AFIT_GKS adhered to the Harris binding whenever possible. However, AFIT_GKS differs from the Harris binding in that it does not define the following variables as private.

```
locator_data_record;  
stroke_data_record;  
valuator_data_record;  
choice_data_record;  
pick_data_record;  
string_data_record;
```

This is because AFIT_GKS did not implement the functions manipulating these data records.

Overall, the functions not implemented in AFIT_GKS mostly deal with the high level input functions and the metafile functions. Mainly, these functions were not implemented because of time constraints encountered in this thesis.

```

type ERROR_INFORMATION is
  record
    number : error_indicator;
    name   : subprogram_name;
  end record;

function get_error;
function get_next_error;
function end_of_file;

```

The next set of functions not implemented cover the interfaces that would need to be designed if the input data records were private. Therefore, if the following functions are added to AFIT_GKS then they should be placed where the private data structures are placed, namely package 'external_types'.

```

package EXTERNAL_TYPES is
  set_locator_data_record; (3 overloaded procedures with
this name)
  function attribute_flag;
  function attributes_used;
  function line_attributes;
  function fill_area_attributes;
  function prompt_echo_type;
  set_stroke_data_record; (3 overloaded procedures with
this name)
  function buffer_size;
  function position;
  function interval;
  function time;
  function line_attributes;
  function marker_attributes;
  function prompt_echo_types;
  set_valuator_data_record;
  function high_value;
  function low_value;
  function prompt_echo_type;
  set_choice_data_record; (4 overloaded procedures with
this name)
  function prompt_echo_type;
  function array_of_prompts;
  function array_of_strings;
  function segment;

```

```

package METAFILE is
  write_item_to_gksm;
  get_item_type_from_gksm;
  read_item_from_gksm;
  interpret_item;
end METAFILE;

package INQ_ATTRIBUTES is
  inq_pattern_height_vector;
  inq_pattern_width_vector;
  inq_char_nominal_width;
  inq_char_base_vector;
  inq_text_extent;
end INQ_ATTRIBUTES;

package INQ_INPUT is
  inq_locator_device_state;
  inq_stroke_device_state;
  inq_valuator_device_state;
  inq_choice_device_state;
  inq_pick_device_state;
  inq_string_device_state;
  inq_number_of_available_logical_input_devices;
  inq_default_locator_device_data;
  inq_default_stroke_device_data;
  inq_default_valuator_device_data;
  inq_default_choice_device_data;
  inq_default_pick_device_data;
  inq_default_string_device_data;
  inq_input_queue_overflow;
end INQ_INPUT;

procedure INQ_PIXELS is
  inq_pixel_array_dimensions;
  inq_pixel_array;
  inq_pixel;
end INQ_PIXELS;

```

In addition, the Harris binding defines additional functions, and an additional type not specified in ANS GKS. None of these functions were implemented in AFIT_GKS. The first set of additional functions are used to handle the error file.

Appendix C

Harris Functions Not Implemented

AFIT_GKS is a subset of an ANS GKS graphical package. Below is a list of those functions that are part of the Harris Binding of ANS GKS to Ada, but are not implemented in AFIT_GKS. They are listed by the package that the functions should be in if they were implemented. This appendix is designed for a maintainer of AFIT_GKS, and as such this appendix is not intended for the average user of AFIT_GKS, or the average reader of this thesis.

```
package CONTROL is
  message;
  escape;
end CONTROL;
```

```
package PRIMITIVES is
  cell_array;
  gdp;
end PRIMITIVES;
```

```
package SET_PRIM is
  set_fill_area_interior_style;
end SET_PRIM;
```

```
package INPUT is
  sample_locator;      get_locator;
  sample_stroke;      get_stroke;
  sample_valuator;    get_valuator;
  sample_choice;      get_choice;
  sample_pick;        get_pick;
  sample_string;      get_string;
  await_event;
  flush_device_events;
end INPUT;
```

```
type DEFAULT_WS_ID is range 1..3;
-- AFIT_GKS has three workstations, the Tektronix 4014, the Tektronix
-- 4027, and Workstation Independent Segment Storage (WISS).
-- Level ma

type DEFAULT_PICK_ID is range 1..1000;
-- This is an arbitrary number of pick_ids allowed in AFIT_GKS.
-- Level 1b;

type DEFAULT_SEGMENT_NAME is range 1..32000;
-- This is an arbitrary number of segments allowed in AFIT_GKS, but
-- ANS GKS states at least 32,000 segments must be supported. Due to
-- space limitations AFIT_GKS only supports 50 segments.
-- Level 1a

end GKS_CONFIGURATION;
```

```

MAX_STRING_PROMPT_ECHO_TYPE : constant integer := 1;
-- At this time, AFIT_GKS allows only 1 input device per device type.
-- Level mb

MAX_WS_TYPE : constant integer := 3;
-- AFIT_GKS has 3 workstations Tektronix 4014, Tektronix 4027, and WISS.
-- Level ma

MAX_LINE_TYPE : constant integer := 24;
-- The Tektronix 4027 has the 4 ANS GKS defined linetypes 1..4, and the
-- Tektronix 4027 has 4 additional linetypes 21..24.
-- Level ma

MAX_MARKER_TYPE : constant integer := 5;
-- AFIT_GKS only uses the 5 markers defined by ANS_GKS.
-- Level ma

MAX_FONT_TYPE : constant integer := 1;
-- None of the devices support more than 1 font
-- Level ma

MAX_HATCH_STYLE : constant integer := 2;
-- The Tektronix 4027 has been programmed to reserve pat 118..pat 119
-- for the hatch styles. This gives 2 hatches.
-- Level 0a

PRECISION : constant integer := 5;
-- This is the standard precision of AFIT_GKS.
-- Level ma

MAX_GKSM_STRING_LENGTH : constant integer := 72;
-- AFIT_GKS allows 72 characters per line.
-- Level mb

MAX_INPUT_STRING_LENGTH : constant integer := 72;
-- AFIT_GKS allows 72 characters per line.
-- Level mb

MAX_PROMPT_STRING_LENGTH : constant integer := 72;
-- AFIT_GKS allows 72 characters per line.
-- Level mb

MAX_ERROR_FILE_STRING_LENGTH : constant integer := 72;
-- AFIT_GKS allows 72 characters per line.
-- Level mb

MAX_DEVICE_PRECISION : constant integer := 5;
-- This is the standard precision of AFIT_GKS.
-- Level ma

type DEFAULT_WC_TYPE is digits 5;
-- This is the standard precision of AFIT_GKS.
-- Level ma

```

```

MAX_GDP_ID : constant integer := 0;
-- No GDP's implemented yet.
-- Level 0a

MAX_PATTERN_INDEX : constant integer := 119;
-- Tektronix 4027 supports 120 patterns from pattern 0 .. 119.
-- Level 0a

MAX_POLYLINE_INDEX : constant integer := 20;
-- It is an arbitrary maximum number of entries in the bundle table.
-- Level 0a

MAX_POLYMARKER_INDEX : constant integer := 20;
-- It is an arbitrary maximum number of entries in the bundle table.
-- Level 0a

MAX_TEXT_INDEX : constant integer := 20;
-- It is an arbitrary maximum number of entries in the bundle table.
-- Level 0a

MAX_FILL_AREA_INDEX : constant integer := 20;
-- It is an arbitrary maximum number of entries in the bundle table.
-- Level 0a

MAX_TRANSFORMATION_NUMBER : constant integer := 20;
-- It is an arbitrary maximum number of transformations allowed from
-- World Coordinates to Normalized Device Coordinates.
-- Level ma

MAX_DEVICE_NUMBER : constant integer := 1;
-- At this time, AFIT_GKS allows only 1 input device per device type.
-- Level mb

MAX_LOCATOR_PROMPT_ECHO_TYPE : constant integer := 1;
-- At this time, AFIT_GKS allows only 1 input device per device type.
-- Level mb

MAX_STROKE_PROMPT_ECHO_TYPE : constant integer := 1;
-- At this time, AFIT_GKS allows only 1 input device per device type.
-- Level mb

MAX_VALUATOR_PROMPT_ECHO_TYPE : constant integer := 1;
-- At this time, AFIT_GKS allows only 1 input device per device type.
-- Level mb

MAX_CHOICE_PROMPT_ECHO_TYPE : constant integer := 1;
-- At this time, AFIT_GKS allows only 1 input device per device type.
-- Level mb

MAX_PICK_PROMPT_ECHO_TYPE : constant integer := 1;
-- At this time, AFIT_GKS allows only 1 input device per device type.
-- Level mb

```

```

function IS_IN (ITEM : ITEM_TYPE; THE_LIST : LIST_OF)
  return BOOLEAN;
  -- This function returns the value TRUE if the ITEM is found in THE_
  -- LIST

procedure ADD_ITEM (ITEM : ITEM_TYPE;
                   TO_LIST : in out LIST_OF);
  -- This procedure adds the ITEM to the specified list

procedure DELETE_ITEM (ITEM : ITEM_TYPE;
                      FROM_LIST : in out LIST_OF);
  -- This procedure deletes the specified item from the list

type MATRIX_OF is array (INDEX range <>, INDEX range <>) of
  ITEM_TYPE;
  -- Defines a two-dimensional array of the item.

type VARIABLE_MATRIX_OF (DX : INDEX := 1, DY : INDEX := 1) is
  record
    MATRIX : MATRIX_OF (1 .. DX, 1 .. DY);
  end record;
  -- Defines a matrix whose dimensions may vary dynamically as both an
  -- input and output parameter. Warning : when declaring objects of
  -- this type, be sure to let the discriminant components default (to
  -- 1), or else the size of the matrix will always be constrained.

```

```

end GKS_LIST_UTILITIES;

```

GKS Configuration (6:356-358)

Package GKS_CONFIGURATION is a package of AFIT_GKS which contains implementation-defined constants.

package GKS_CONFIGURATION is

```

MAX_RASTER_UNITS : constant integer := 1023;
  -- It is the largest raster units on any possible workstation. In AFIT_
  -- GKS that is the Tektronix 4014 with a maximum x raster unit size of
  -- 1023.
  -- Level ma

MAX_MEMORY_UNITS : constant integer := 2;
  -- It is not used at all to determine size of gks.
  -- Level ma

MAX_COLOUR_INDEX : constant integer := 7;
  -- Tektronix 4027 allows for 8 colours ranging from colour 0 to 7.
  -- Level ma

```

```
MIN : COORDINATE;
MAX : COORDINATE;
end record;
-- Defines a range of values along an axis in the coordinate system.
-- MIN should always be less than MAX.
```

```
type RECTANGLE is
record
  X : LIMITS;
  Y : LIMITS;
end record;
-- Defines the extent of a rectangle in the coordinate system parallel
-- to the X and Y axes.
```

```
end GKS_COORDINATE_SYSTEM;
```

GKS List Utility (6:354-355)

This section contains the specification of the generic package GKS_LIST_UTILITIES, instantiated by the AFIT_GKS binding for the declaration of arrays, lists (variable-sized arrays), matrices, and variable-sized matrices of many of the AFIT_GKS data types. This package also contains a few examples of optional utilities for manipulating some of the data types. This generic package is required at level ma.

```
generic
  type ITEM_TYPE is private;
```

```
package GKS_LIST_UTILITIES is
```

```
MAX_INDEX : constant := 50;
-- This value defines the maximum dimensions of any of the following
-- data types.
```

```
type INDEX is range 0.. MAX_INDEX;
-- Defines the valid range of indices of the following data types.
```

```
type ARRAY_OF is array (INDEX range <>) of ITEM_TYPE;
-- Defines an unconstrained array of the item.
```

```
type LIST_OF (LENGTH : INDEX := 0) is
record
  LIST : ARRAY_OF (1 .. LENGTH);
end record;
-- Defines a list whose length may vary dynamically. Warning : when
-- declaring objects of this type, be sure to let the discriminant
-- component (LENGTH) default (to 0) to initialize a null list, or
-- else the length of the object will be constrained always.
```

GKS Coordinate System (6:352-353)

This section contains the specification for the coordinate systems template, and Ada generic package defining a Cartesian coordinate system for use by AFIT_GKS.

```
generic
  type COORDINATE is digits <>;
  -- Coordinates in the system are floating point values. Values on
  -- both axes are of the same type.

package GKS_COORDINATE_SYSTEM is

  -- Due to compiler problems on the ROLM Data General the type MAGNI-
  -- TUDE could not be implemented as:
  -- MAGNITUDE_PRECISION : CONSTANT := 6;
  --
  -- type MAGNITUDE_BASE_TYPE is digits MAGNITUDE_PRECISION;
  --
  -- subtype MAGNITUDE is MAGNITUDE_PRECISION range
  -- MAGNITUDE_BASE_TYPE (COORDINATE'safe_small)..
  -- MAGNITUDE_BASE_TYPE (ABS (COORDINATE'last - COORDINATE'first));
  -- Instead it was implemented as:

  subtype MAGNITUDE is COORDINATE range
    COORDINATE'small .. abs (COORDINATE'last - COORDINATE'first);
  -- Defines the length of an object in the coordinate system. In GKS,
  -- all such values must be greater than zero.

  type POINT is
    record
      X : COORDINATE;
      Y : COORDINATE;
    end record;
  -- Defines a point in the coordinate system.

  type VECTOR is new POINT;
  -- Defines a vector in the coordinate system.

  function VECTOR_LENGTH (V : VECTOR) return MAGNITUDE;
  -- This is an optional function which returns the length of the speci-
  -- fied vector.

  type SIZE is
    record
      X : MAGNITUDE;
      Y : MAGNITUDE;
    end record;
  -- Defines the size of an object in the coordinate system as length
  -- along the X and Y axes.

  type LIMITS is
    record
```

Appendix B

GKS Coordinate System,

GKS List Utilities, and GKS Configuration

This appendix contains the specifications of the three packages, GKS_Coordinate_System, GKS_List_Utilities, and GKS_Configuration, used in the Harris draft binding of ANS GKS to Ada.

Table of Contents

	<u>Page</u>
GKS_Coordinate_System	B.2
GKS_List_Utilities	B.3
GKS_Configuration	B.4

Third, the Error file of AFIT_GKS is "Error_GKS", this is the name of the error file no matter what input is given (if any) for the open_gks parameter "error_file".

Fourth, the type_of_ws of AFIT_GKS are defined as follows:

The Tektronix 4014 is type_of_ws equal to 1.
The Tektronix 4027 is type_of_ws equal to 2.
WISS is type_of_ws equal to 3.

Overall, these are the only system dependent features of AFIT_GKS that should concern the user. If any other problems are found in running AFIT_GKS, they should be reported to

Professor Charles Richard
Math Department
Air Force Institute of Technology
School of Engineering
Wright-Patterson AFB
Dayton, Ohio

VITA

Raymond Scott Ruegg was born on 4 February 1961 in Syracuse, New York. He graduated fourth from Holliston High School in Holliston, Massachusetts in 1979, and attended the University of Massachusetts (Amherst campus), from which he graduated cum laude, receiving two Bachelor of Science degrees, one in Mathematics, the other in Computer Science. Upon graduation, he received a commission in the United States Air Force through the ROTC program, and immediately entered into the School of Engineering, Air Force Institute of Technology in June 1983. He is a member of Phi Beta Kappa.

Permanent address: 19 High Rock Road
Holliston, Massachusetts 01746

REPORT DOCUMENTATION PAGE				
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b. RESTRICTIVE MARKINGS		
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT		
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE		Approved for public release; distribution unlimited		
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/MATH/84D-5		5. MONITORING ORGANIZATION REPORT NUMBER(S)		
6a. NAME OF PERFORMING ORGANIZATION School of Engineering	8b. OFFICE SYMBOL (If applicable) AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION		
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB, Ohio 45433		7b. ADDRESS (City, State and ZIP Code)		
8a. NAME OF FUNDING/SPONSORING ORGANIZATION System Avionics Division	8b. OFFICE SYMBOL (If applicable) AFWAL/AAAF	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER		
8c. ADDRESS (City, State and ZIP Code) Avionics Laboratory Wright-Patterson AFB, Ohio 45433		10. SOURCE OF FUNDING NOS.		
11. TITLE (Include Security Classification) See Box 19		PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.
12. PERSONAL AUTHOR(S) Raymond Scott Ruegg, B.S., 2d Lt, USAF		WORK UNIT NO.		
13a. TYPE OF REPORT MS Thesis	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 1984 December	15. PAGE COUNT 180	
16. SUPPLEMENTARY NOTATION				
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)		
FIELD	GROUP	SUB. GR.		
12	01		Computer Graphics, Graphics, GKS,	
09	02		Programming Languages, High Level Languages, Ada	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)				
Title: AFIT_GKS -- A GKS IMPLEMENTATION IN THE ADA PROGRAMMING LANGUAGE				
Thesis Advisor: Charles W. Richard Jr. Associate Professor of Mathematics				
<div style="text-align: right;"> Approved for public release: IAW AFB 190-17. 1984 E. V. </div>				
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT <input type="checkbox"/> DTIC USERS <input type="checkbox"/>		21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED		
22a. NAME OF RESPONSIBLE INDIVIDUAL Charles W. Richard Jr.	22b. TELEPHONE NUMBER (Include Area Code) 513-255-3098	22c. OFFICE SYMBOL AFIT/ENG		

This project, written in Ada, involved designing and implementing AFIT_GKS which is a subset of the Graphical Kernel System (GKS). This project implemented AFIT_GKS on a ROLM Data General MV/8000-II validated Ada compiler, using a proposed Binding of GKS to Ada developed by the Harris Corporation. After introducing Ada and GKS to the reader, this project considers several alternative ways of designing AFIT_GKS. Selecting what was considered the best design alternative, this project implements AFIT_GKS. It concludes with a discussion of how well Ada, the proposed GKS binding to Ada, and GKS, worked in AFIT_GKS. This thesis found minor problems with the validated ROLM Ada Compiler, the proposed GKS binding to Ada, the GKS, but overall they were each excellent products. By using AFIT_GKS as proof, this project concludes that Ada can support large programs, and Ada can support computer graphics.

END

FILMED

5-85

DTIC