EVENT HANDLING IN GNA95GP

GRAPHICS PACKAGE

By

SHAN KUANG

Bachelor of Science
College of Armored Force
Beijing, China
1986

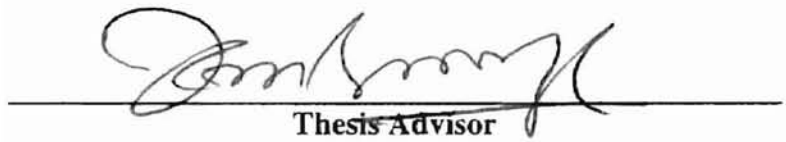
Master of Science
China Mining University
Beijing, China
1989

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1997

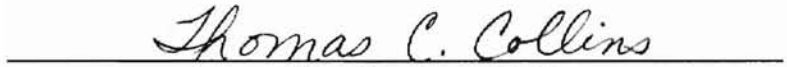
EVENT HANDLING IN GNA95GP

GRAPHICS PACKAGE

Thesis Approved:



Thesis Advisor



Dean of the Graduate College

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my advisor Dr. K. M. George for his guidance, patience, understanding, support, and excellent editorial skills. I also would like to extend my appreciation to my committee members Dr. H. Lu and Dr. J. P. Chandler, who were always there when assistance was needed.

I would like to give special thanks to my wife, Xiaomei Zhu, for her encouragement and love in the process of my pursuing this degree. I also would like to thank my parents for their encouragement and support. I could not have asked for better.

This project is supported by DISA Grant DCA100-96-10007.

TABLE OF CONTENTS

Chapter Page

1. Introduction 1

2. Literature Review 4

2.1 What is event handling? 4

2.2 The theory of the event handling 4

2.2.1 Overview 4

2.2.2 Definitions in event handling 5

2.3 SRGP graphics package 8

2.3.1 A brief description of SRGP 8

2.3.2 Event handling in SRGP 9

(1) Device types 9

(2) Event handling mode 9

(3) Sample mode 9

(4) Event mode 11

2.4 Event handling in Windows 3.1 SDK 13

2.4.1 Message 13

2.4.2 Window procedure 14

2.4.3 Message queue 14

2.4.4 Detailed description of message queue 14

2.5 GKS graphics package 16

2.5.1 A brief descriptin of GKS 16

2.5.2 Event handling in GKS 17

(1) Generating events 17

(2) Input classes 17

(3) Input mode 19

2.6 PHIGS graphics package 20

2.6.1 A brief description of PHIGS 20

2.6.2 Event handling in PHIGS 20

(1) Input device classes 20

(2) Input model and PET 21

(3) Operating modes of input devices 21

2.7 Event handling in X-Window system 23

2.7.1 Events 24

2.7.2 Event queue 24

2.7.3 Event loop 25

2.8 Ada 95 language 25

2.8.1 General description of Ada 95 25

2.8.2 Some important features useful to GNA95GP 26

(1) Packages and private types 26

(2) Type extension 26

(3) Interfacing with other language 26

3. Implementation Issues of Event Handling in GNA95GP 28

3.1 Software architecture 28

3.1.1 Free software for GNA95GP 28

(1) DJGPP 28

(2) GNAT 28

(3) RSXWDK 28

3.1.2 The usage of these free software 29

3.2 Design considerations for event handling package 31

3.2.1 Interface with other package of GNA95GP 31

3.2.2 The desogn style of event handling package 33

3.3 Implementation of event handling 33

3.3.1 WinMain and WndProc function 33

3.3.2 Binding between C and Ada 95 33

3.3.3 Functions supported by this event handling package 35

(1) Functions in event handling of GNA95GP to set parameters 35

(2) Functions in event handling of GNA95GP to get mouse measure in event mode 35

(3) Functions in event handling of GNA95Gp to get keyboard measure in event mode 36

(4) Functions in event handling of GNA95GP to get mouse measure in sample mode 36

(5) Functions in event handling of GNA95Gp to get keyboard measure in sample mode 37

(6) Functions in event handling of GNA95GP to handle I/O 37

3.3.4 Declaration of data types,structure and parameters list 37

(1) Declaration of data types and structures 37

(2) Parameters list 39

3.3.5 Using event handling 39

3.3.5.1 Sample mode 39

3.3.5.2 Event mode 40

3.4 Create a window 42

3.5 Name of event handling package 43

3.6 I/O package for input and output 43

3.6.1 Usage of I/O package 43

3.6.2 Name of I/O package 45

3.7 Examples of event handling usage 45

4. Summary 46

Bibliography 47

Appendix I 50

Appendix II 56

Appendix III 57

LIST OF FIGURES

Figure Page

1. Architecture of event handling 5
2. Sampling versus event-driven 7
3. An example use of sample technique 10
4. An example of keybaord event with edit mode 11
5. Basic usage of locator event mode 12
6. The relationship between message queue and windows procedure in MS-Windows 3.1 16
7. Generating events 17
8. An operator action generates a set of events 20
9. Request mode input 21
10. Sample mode input 22
11. Event mode input 23
12. The server's event queue and client's event queue 24
13. Sofewarre architecture for event handling 29
14. The architecture of GNA95GP 31
15. WinMain and WndProc function 32
16. Binding between C and Ada 95 34
17. Creating a window, setting parameters, and getting events 35

18. Functions to get mouse measure in event mode 35

19. Functions to get keyboard measure in event mode 36

20. Functions to get mouse measure in sample mode 36

21. Functions to get keyboard measure in sample mode 37

22. I/O functions of GNA95GP under Windows environment 37

23. Declaration of data types and structures 38

LIST OF TABLES

Table **Page**

1. Parameters list for event handling in GNA95GP 39

2. I/O package in GNA95GP 44

Chapter One

Introduction

Computer graphics has been widely used in many different areas of computer science and also in applications of everyday life. A course on computer graphics is taught in almost every university. Currently, many graphics packages are available for graphics applications development and for instructional support. Better user interfaces and "easy to learn" are the common goals of these graphics packages [1]. The publicly available graphics packages either use Pascal language (GKS -- Graphics Kernel System [2] [3]) or use C language (PHIGS -- Programmer's Hierarchical Interactive Graphics System [4] [5], SRGP -- Simple Raster Graphics Package [1] [6], X-Windows [7]) to program graphics applications. Most of the computer graphics courses offered by universities use C, C++, or Pascal. To the author's knowledge, GNA95GP (GNAT Ada 95 Graphics Package) [8] [9] [10] is the first attempt to develop a free graphics library in Ada 95 programming language [8] [9] [10] under Windows environment.

GNA95GP focuses on the development of tools used in undergraduate computer course. It is a free software [31] which is modeled after graphics package SRGP and PHIGS to provide a high-level programming interface for students to write graphics applications using Ada 95 language. It is designed to support dynamic and interactive Ada 95 graphics applications which run under Windows 3.1 in IBM PCs [8] [9] [10].

Currently, GNA95GP is being ported to Windows 95 [11]. It is also used in a computer graphics course at Oklahoma State University.

Ada 95 is an object-oriented programming language and it is the only object-oriented programming language accepted as an ISO standard [12] [13] [14] [15]. It supports abstract data types, data encapsulation and message passing. The development of an Ada 95 graphics package will be helpful for developing graphics applications in Ada 95 programming language. This thesis addresses event handling, one aspect of the 2D graphics package GNA95GP.

This 2D graphics package in Ada 95 is developed under MS Windows 3.1 using free software. It supports 2D drawing, event handling and storage of graphics primitives in the PC. It is interactive and convenient to use with Ada 95 programs. It provides a tool to draw 2D primitives and to handle events with Ada 95 programming language in both Windows 3.1 and Windows 95 environment [10].

Event handling, which handles user inputs, is an essential aspect of computer graphics. Via input devices such as the keyboard and the mouse, users can communicate with programs that respond to events and draw objects on the display device. Event handling allows a user to write interactive programs that communicate between the user and the graphics applications. Like other graphics packages such as GKS, SRGP and PHIGS, event handling is also an important part in GNA95GP. The remainder of this thesis is organized as follows: Chapter 2 describes event handling in several public graphics packages. Chapter 3 describes implementation issues of event handling in GNA95GP. It gives the details of the design of event handling, the functions of event

handling and the usage of event handling. Chapter 4 summarizes the work in this thesis. Two examples illustrating the use of event handling are given in appendix I and II. The source code for event handling is listed in appendix III.

Chapter Two

Literature Review

2.1 What is event handing?

In a computer, the most often used input devices are mouse (locator) and keyboard. An event is a change in a device's state caused by user action. So pressing a keyboard key and clicking a mouse button are all events. Event Handling provides the interface to input devices. It can get the events which are initiated by users using input devices and transfer the actions of the users to the graphics application program [3] [4] [7]. On the other hand, event handling also provides the information about events to the users and lets the users know what they did. Generally, event handling includes the following two logical devices: locator and keyboard. It also uses two basic methods (sample mode and event mode) to get an event. It looks like a "bridge" between users and graphics applications and makes the interactive graphics a reality.

2.2 The Theory of the Event Handing

2.2.1 Overview

According to Foley et. al. [1], the typical application-program schema for interaction handing (event handling) is the event-driven loop. It is easily visualized as a finite-state machine with a central wait state and transitions to other states that are caused by user-input events. Processing a command may include nested event loops of the same format that have their own states and input transitions. An application program may also

sample (or do polling) the input devices such as the locator by asking for their values at any time; the program then uses the returned value as input to processing procedure that also changes the state of the application program, the image, or the database. Figure 1 illustrates the event handling concept.

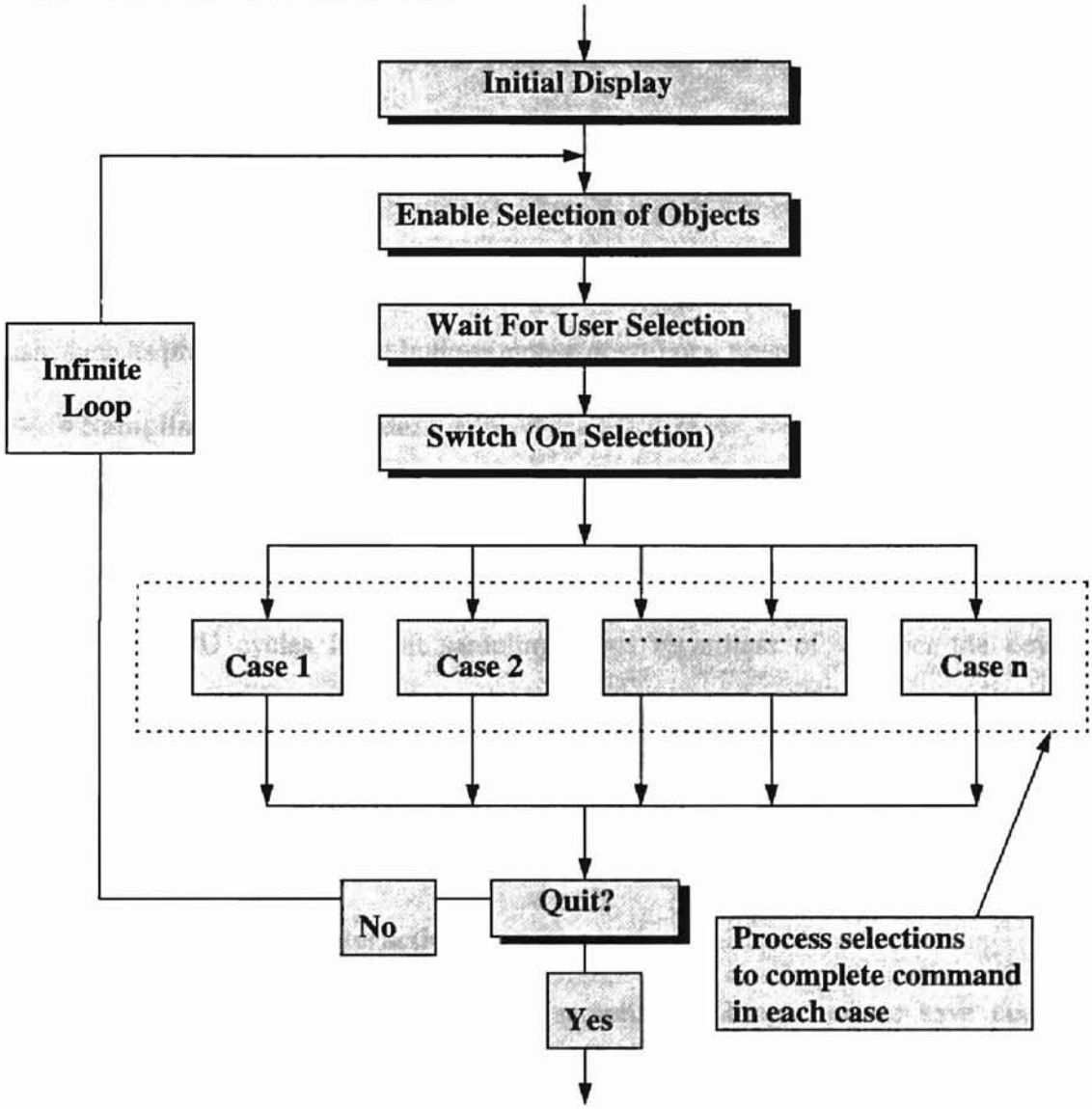


Figure 1 Architecture of event handling.

2.2.2 Definitions in event handling

- **Measure:**

It is the value of a logical input device. It includes position of points, button status (UP, DOWN) and keyboard values. Usually it is defined as a data structure.

- **Event:**

It is a change in a device's state caused by user action. All the changes of input devices (mouse and keyboard) such as press and release any key of keyboard and mouse buttons are events.

- **Event trigger:**

It is the user action that causes an event to occur. Typically, the trigger is a button push, such as press of the mouse button (mouse down) or a press of a keyboard key.

- **Sampling (Polling) mode:**

The application program queries the current measure of a logical input device in a time interval and continues execution (repeating this query all the time). Sampling spends most of the CPU cycles in tight sampling loops regardless of whether the device's measure has changed. If there is no new event occurring, the measure of the device will stay the same all the time in every sampling time interval. Thus sampling mode is CPU intensive.

- **Interrupt-driven Interaction:**

In order to relax the CPU-intensive sampling (polling) loop to save computer resources, an interrupt-driven technique is used. Using this method, one or more devices can input and then continue normal execution until interrupted by some input events; control then passes asynchronously to an interrupt procedure, which responds to the event and finishes what the user wants to do. In interrupt-driven interaction, the sign of a new

event is the trigger. The trigger tells the system when to interrupt and to transfer to an interrupt procedure. Different input devices use different event triggers depending on the properties of events and the purpose of events.

- **Event-driven Interaction:**

The asynchronous transfer of control is difficult and tricky. Many graphics packages (GKS, PHIGS, SRGP) provide an "event-driven" technique to simulate interrupt-driven interaction. Figure 2 shows the principles of event-driven technique and sampling. Using event-driven technique, an application enables devices and continues execution. In the background, these graphics packages monitor the devices and store information about each event in an event queue. An application inspects the event queue and decides

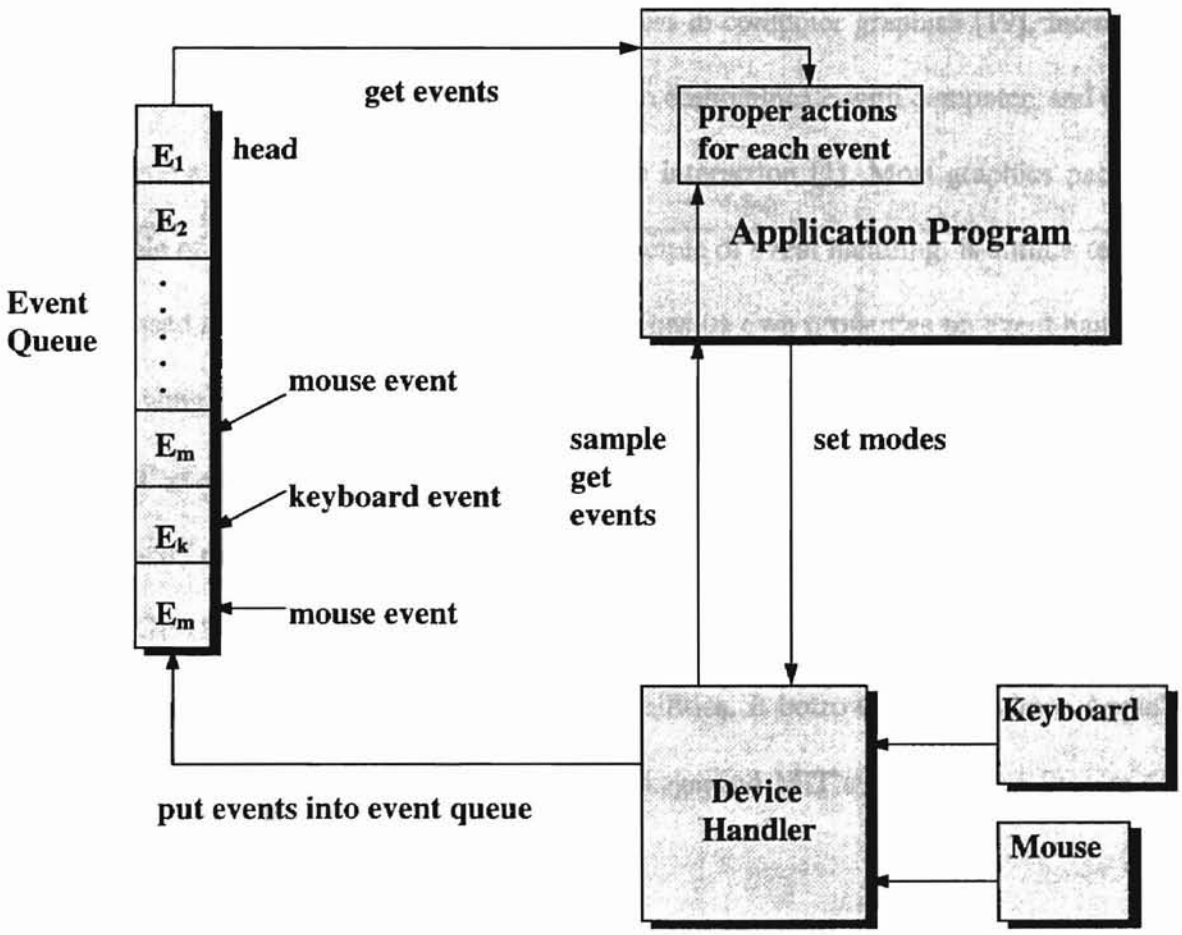


Figure 2 Sampling versus event-driven technique.

whether it is in a wait state or not. If there are one or more events in the queue, the head event is dequeued, and its information is made available to the application. When there is no event in the queue (it is empty), the application can either do other jobs, wait for the next event to occur, or wait for an application-specified maximum-wait-time interval to pass depending on the specification of the application.

In the sampling mode, the device is polled and an event measure is collected, regardless of any user activity. In the event mode (or event-driven mode), the application either gets an event report from a prior user action, waits until a user action occurs, or time-out occurs. The basic difference between sampling and event-driven modes is the response to the user action.

Event handling is the most important aspect in computer graphics [19]. Interactive graphics has now provided us an effective tool to communicate with computer, and thus it has become a major facilitator of man/machine interaction [1]. Most graphics packages are capable of event handing. Basically, the principle of event handling is similar to what we discussed above. But each graphics package has its own properties on event handling. The following is a review of some graphics packages.

2.3 SRGP graphics package

2.3.1 A brief description of SRGP

SRGP (Simple Raster Graphics Package) is designed to be a device independent graphics package and makes use of raster capabilities. It borrows features from Apple's popular QuickDraw integer raster graphics package and MIT's X Windows System for

output, and from GKS (Graphics Kernel System) and PHIGS (Programmer's Hierarchical Interactive Graphics System) for input [1] [2] [3] [5] [7].

2.3.2 Event handling in SRGP

(1) Device types

In order to provide device independence for graphics input, SRGP supports two logical input devices:

- **Locator:**

A device for specifying screen coordinates and the state of one or more associated buttons such as mouse.

- **Keyboard**:

A device for specifying character string input.

SRGP maps the logical devices onto the physical devices to achieve device independence.

(2) Event handing mode

SRGP uses two modes to get information input by user interaction: sample mode and event-driven mode.

(3) Sample mode

In SRGP, user can make input devices active or inactive. SRGP uses sample mode especially for Locator. So, after an application sets Locator to sample mode, it can get the Locator's measure. In SRGP, locatorMeasure is a data structure which includes the cursor position as a screen (x, y) coordinate pair, the number of the button that most recently experienced a transition, and the state of the buttons as an array (UP, DOWN). The most

recent transition lets the application know which button caused the trigger for that event. Using this Locator's measure, the application can perform tasks like getting point coordinates and drawing a line using the coordinates. Figure 3 is a flow chart to describe a typical example using sample Mode. In this example, two sampling loops are used. First sampling loop makes sure that left button of mouse is pressed down before drawing. The second sampling loop finishes drawing when user drags the mouse. Because of the continuous query, sample mode is CPU intensive. The sampling rate is determined essentially by the speed at which the CPU runs the operating system, the package, and the application.

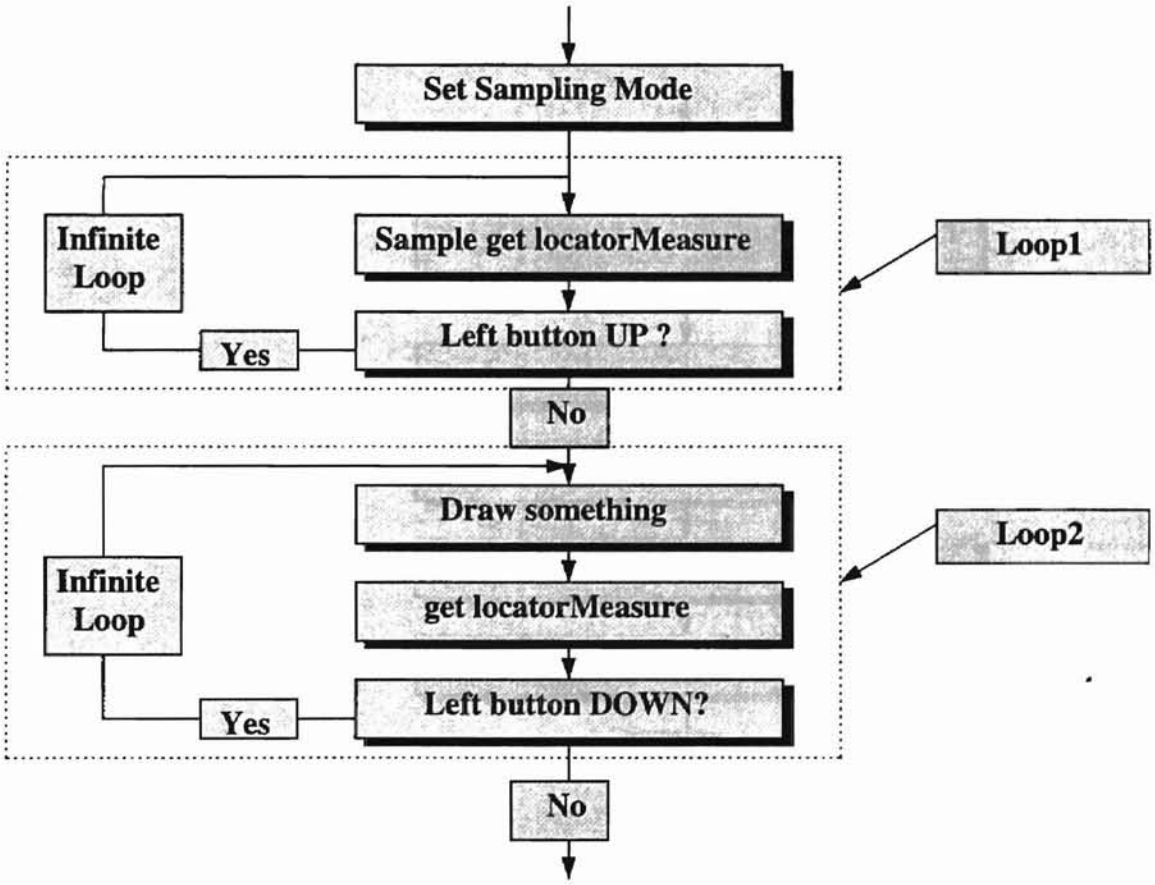


Figure 3 An example use of sample technique

(4) Event mode

Similar to sample mode, user can activate a device in event mode. Then SRGP uses a function to inspect the event queue and get the input device information. The input devices can be LOCATOR, KEYBOARD or NO_DEVICE. There are two processing modes in keyboard device. One is EDIT mode and the other is RAW mode. EDIT mode is used when the application receives strings from the user, who types and edits the string and then presses the return key to trigger the event. When the keyboard interactions must be monitored closely, RAW mode is used. In RAW mode, every key press is an event trigger. Using EDIT mode, the user can type entire strings, correcting them with the backspace key as necessary, and then use the Return (or Enter) key as trigger. This mode

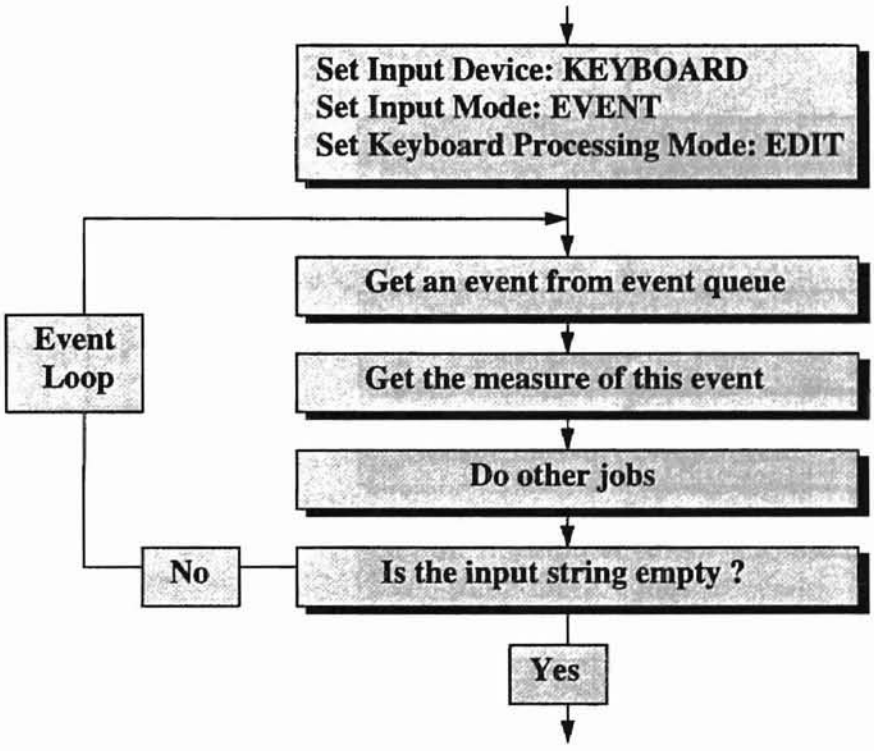


Figure 4 An example of KEYBOARD EVENT with EDIT mode.

just accepts backspace and Return keys and ignores other control keys. The measure is the string as it appears at the time of the trigger. In RAW mode, on the other hand, each character typed, including control characters, is a trigger and is returned individually as the measure. Figure 4 is the flow chart of an example which uses event mode for keyboard device. In this example, user can type any string and then press return key as trigger of this event. Application gets this event from event queue and then gets its measure. This string is included in this measure. User can use the string of the measure to do what they want to so. Then user checks the measure to decide whether go on the event loop or quit.

Figure 5 is a flow chart of the basic usage of event mode for locator device. When using locator (mouse) as input device, press or release of a mouse button is the trigger of

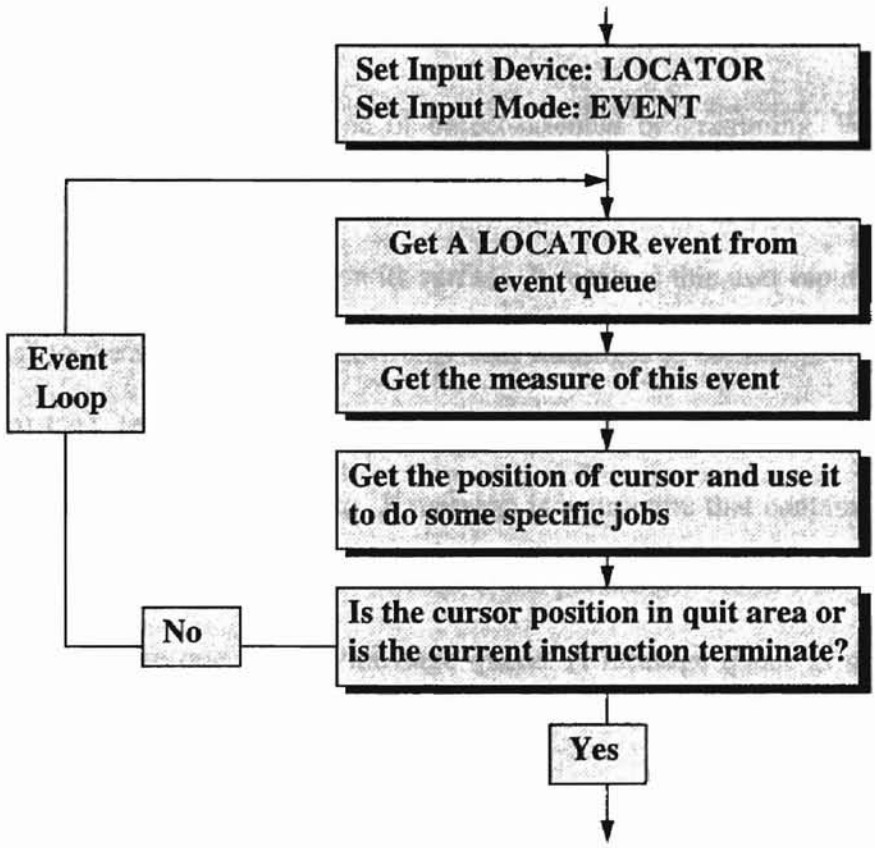


Figure 5 Basic usage of LOCATOR EVENT mode.

locator device. For example, press or release left button of the mouse is an event trigger. An application can get the measure (pointer to locatorMeasure) of a mouse event using a function call. An example is the selection of an area displayed on the display device. The position field of the measure is used to determine which area of the screen the user intended to pick.

In figure 5, the event mode is used for locator (mouse) input device. The application program gets an event from event queue and then gets the measure (or some important information) of this event. Using some data such as cursor position of the measure, the application program can finish some works which are related to the position on the screen such as selection of an object. And the process can repeat in an event queue to process events until the application exits or cursor position selects the area named quit [1].

2.4 Event handling in MS Windows 3.1 SDK

2.4.1 Messages

Windows programming is a type of object-oriented programming. Windows are rectangular areas on the screen. A window receives user input from the keyboard or mouse and displays graphical output on its surface. It receives this user input in the form of "messages" to the window. A window also uses messages to communicate with other windows [20] [21]. Messages are the input to an application. They represent events that the application may need to respond to. A message is a structure that contains a message identifier and message parameters. The content of the parameters varies with the message type [22]. Messages are queued in a message queue. A message queue is similar to an event queue.

2.4.2 Window procedure

A window procedure actually is a subprogram in window program. Usually it is called "WndProc" [20]. In window programming, the real action occurs in the window procedure. Based on the message received, the window procedure determines what the window displays in the client area and how the window responds to user input.

2.4.3 Message queue

Windows 3.1 generates an input message for each input event, such as when the user moves the mouse or presses a key. Windows 3.1 collects input messages in a systemwide message queue and then places the messages in an application message queue. These messages are held in an application's message queue until the application has processed all other messages. Windows 3.1 places messages that belong to a specific application in that application's message queue. The application then reads the messages by using the GetMessage function and dispatches them to the appropriate window procedure by using the DispatchMessage function. A window procedure processes messages to the window. Very often these messages inform a window of user input from the keyboard or mouse.

2.4.4 Detailed description of message queue

When a window has been displayed, Windows 3.1 makes itself ready to read keyboard and mouse input from the user. Windows 3.1 maintains a message queue for each windows program currently running under it. When an input event occurs, Windows 3.1 translates the event into a message that it places in the program's message queue. A

program retrieves these messages from the message queue by executing the following message loop:

```
While (GetMessage (&msg, NULL, 0, 0))
{
    TranslateMessage (&msg);
    DispatchMessage (&msg);
}
return msg.wParam;
```

The GetMessage call that begins the message loop retrieves a message from the message queue. This call passes to Windows 3.1 a pointer to the MSG structure called msg. Windows 3.1 fills in the fields of the message structure with the next message from the message queue. If the message field of the message retrieved from the message queue is anything except WM_QUIT, then GetMessage returns a nonzero value. A WM_QUIT message causes the program to fall out of the message loop. The TranslateMessage call passes the msg structure back to Windows 3.1 for some keyboard translation. The DispatchMessage call passes the msg structure back to Windows 3.1 again. Windows 3.1 then sends the message to the appropriate window procedure for processing. After that window procedure processes the message, it returns to Windows 3.1, and the message loop continues with the next GetMessage call.

Figure 6 gives the relationship between message queue and a window procedure.

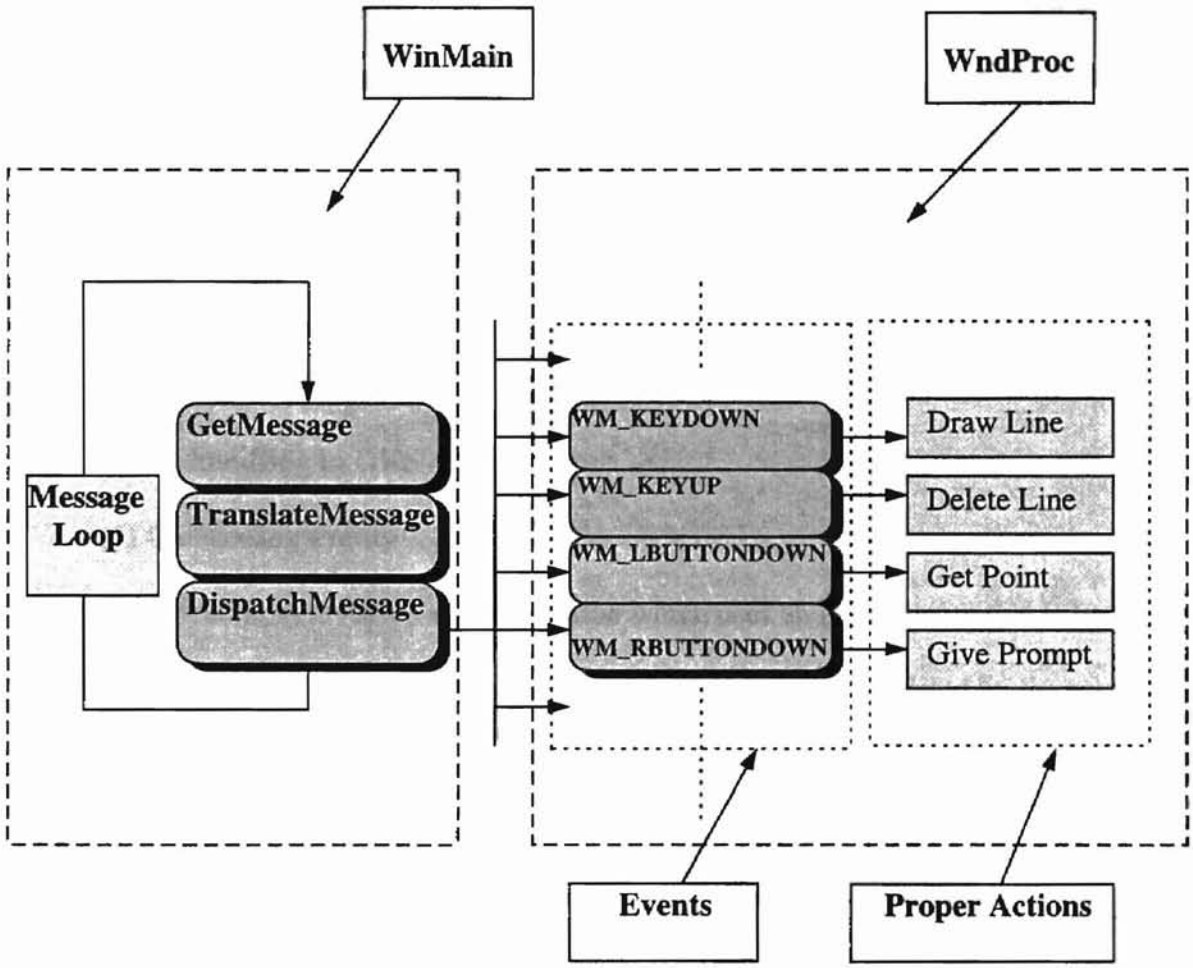


Figure 6 The relationship between message queue and window procedure in Windows 3.1.

In windows programming, pressing a keyboard key or releasing a keyboard key and pushing a mouse key or releasing a mouse key all are messages (or events). Windows 3.1 gets these messages and dispatches them to appropriate window procedure to perform the appropriate task.

2.5 GKS graphics package

2.5.1 A brief description of GKS

Graphics Kernel System (GKS) is an international standard for graphics programming [2]. It consists of a set of 209 graphics functions arranged into a structure with three levels of input and three levels of output [3]. A GKS implementation can conform to any input level and any output level, and the combination of functions defines a valid level of GKS. These functions include some functions which are used for graphical display and interaction. GKS standard supports language independence. It is defined in each computer language by means of a language binding.

2.5.2 Event handling in GKS

(1) Generating events

Graphical input involves a human action which uses an input device that is bound to a computer. The actions involved in using the input device are translated by the graphical system into meaningful events depending on the class of the input device used. Figure 7 describes these ideas.

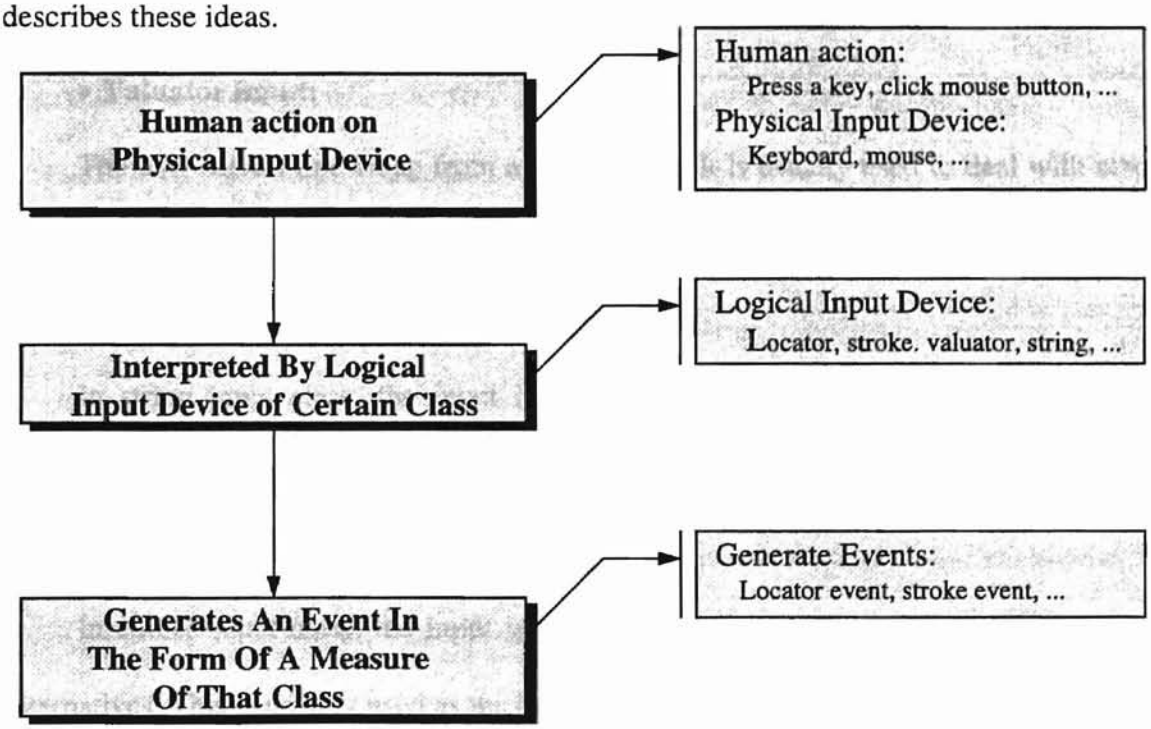


Figure 7 Generating events (adopted from [19]).

(2) Input classes

GKS includes six input classes: locator input, stroke input, valuator input, string input, choice input, and concluding remark [2] [3]. Using the input classes, operator actions on physical devices can be transferred to meaningful events to the graphics system. The data type generated by an event of an input class constructs the measure of that input class.

- **Locator input:**

The user selects a point on the screen by positioning the cursor and pressing a mouse button. A mouse button action is an event. Based on this event, the measure of a locator is generated.

- **Stroke input:**

Like a locator class, except that a sequence of points is returned rather than a single point.

- **Valuator input:**

The user selects one value from a given range. It is usually used to deal with scroll bars.

- **String input:**

In string input class, the input is a character string. The keyboard is the most common physical device used to input the string.

- **Choice input:**

In choice input class, the input is a positive integer representing a selection from alternatives. This is usually used as the basis of a menu selection tool.

• **Pick input:**

The user selects an output primitive on the screen by pointing at the primitive with the cursor and pushing a mouse button.

(3) Input mode

Three input modes are used in GKS: request mode, sample mode, and event mode.

• **Request mode:**

The system suspends until the operator explicitly requests the input event by taking the appropriate action. The request input mode is strictly sequential. This can be described by the process:

RequestInput-->trigger-->measure-->

RequestInput-->trigger-->measure ...

• **Sample mode:**

The input device is sampled and the current value is returned without requiring operation trigger. Sample mode is similar to the request mode in the sense that it is sequential. The process can be described as follows:

SampleInput-->measure-->SampleInput-->measure ...

• **Event mode:**

The event mode of an input device allows a set of events to be simultaneously generated by a single trigger action. For example, pressing a button of a mouse could be interpreted as giving rise to a locator measure, a pick measure and a choice measure at the same time. Figure 8 illustrates this process.

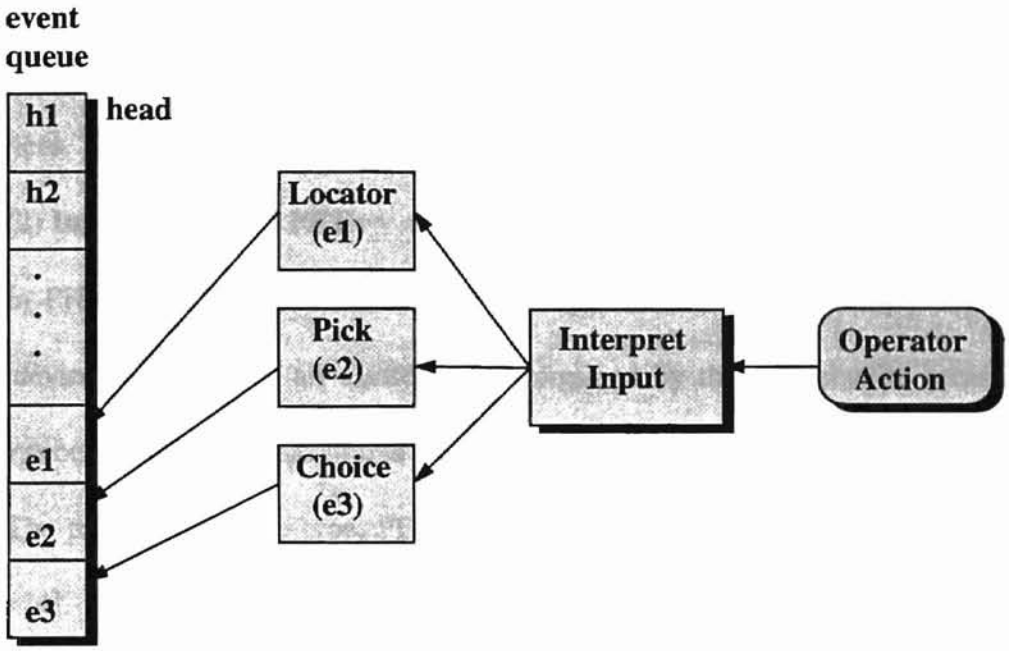


Figure 8 An operator action generates a set of simultaneous events (e1,e2, e3). These are written to the event queue, and have the same simultaneity class (adopted from [19]).

2.6 PHIGS graphics package

2.6.1 A brief description of PHIGS

PHIGS (The Programmer's Hierarchical Interactive Graphics System) is a standard approved by the International Standard Organization (ISO). It can be used in many different computers using different operating systems and window systems. It provides 2D and 3D graphics and graphics-oriented input model [4] [5].

2.6.2 Event handling in PHIGS

(1) Input device classes

In PHIGS, there are six different classes for logical devices: choice, locator, string, stroke, valuator and pick. Using these logical devices, user can represent different types of devices. The meaning of each device class is similar to what we described in GKS.

(2) Input model and PET

In PHIGS, an abstraction of a physical input device forms the input model. The input device properties of all classes are determined by this model. The measure and trigger processes are the basis of this model.

The prompt and echo type, PET, determine what the user will see from the input device [4].

(3) Operating modes of input devices

In PHIGS input model, there are three operating modes which determine how to get input from the device. They are request mode, sample mode, and event mode. The meaning of each mode is similar to what we described in GKS.

- **Request mode:**

Figure 9 gives the flow diagram of a device in request mode.

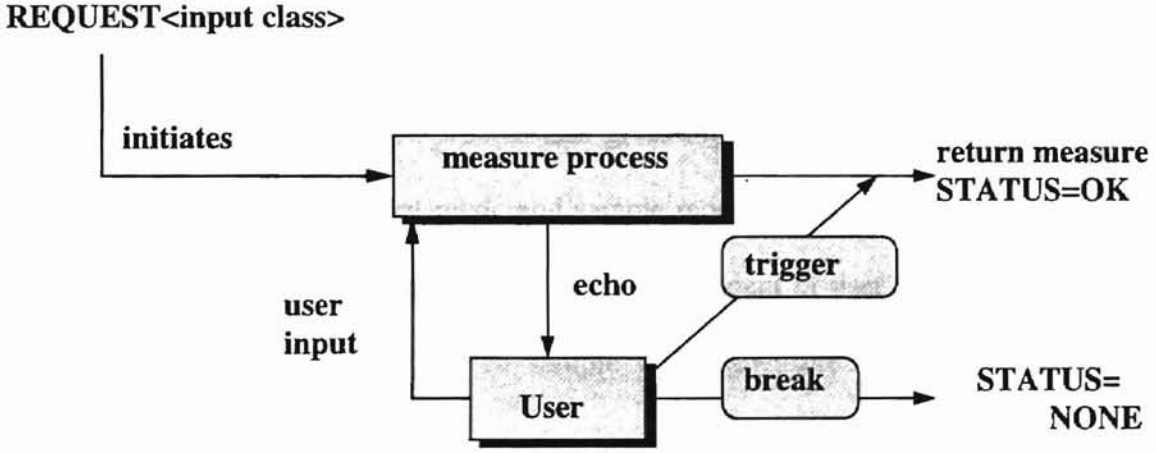


Figure 9 Request mode input (adopted from [4]).

In figure 9, the measure process is initiated as request mode. For each user input, the measure process gives an echo to prompt the user what he/she has input or what he/she will input. Based on the echo that the user got from the measure process, user gives either a trigger or a break signal. If user gives a trigger to respond to the echo prompted by measure process, the measure process will return the measure and set the system STATUS to OK immediately. Otherwise, the measure process will set the system STATUS to NONE to respond.to the break signal given by user. Thus, in request mode, the measure process will return the measure as soon as it receives user trigger.

- **Sample mode:**

Figure 10 gives the flow diagram of a device in sample mode.

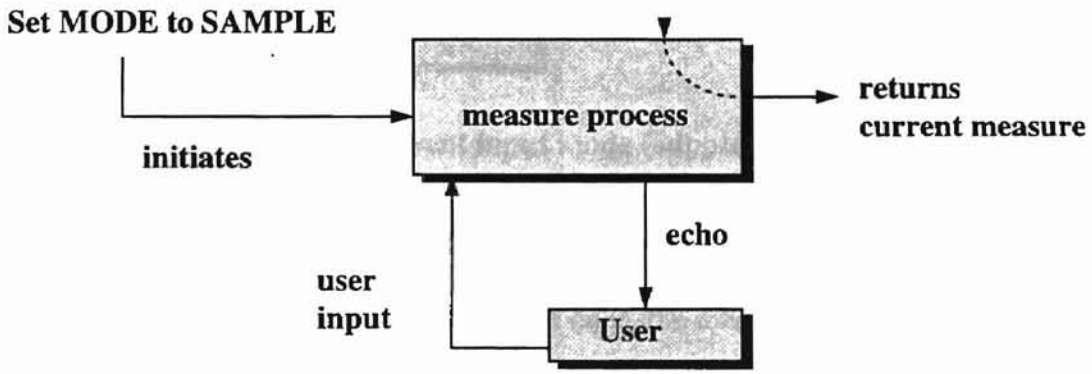


Figure 10 Sample mode input (adopted from [19]).

In sample mode, measure process also gives the echo to user to prompt for input. The difference between request mode and sample mode is that request mode needs user trigger to return measure and sample mode is independent of user trigger i.e. it will get measure in certain time interval. In sample mode, measure process returns current measure within some time interval.

• **Event mode:**

Figure 11 gives the flow diagram of a device in event mode.

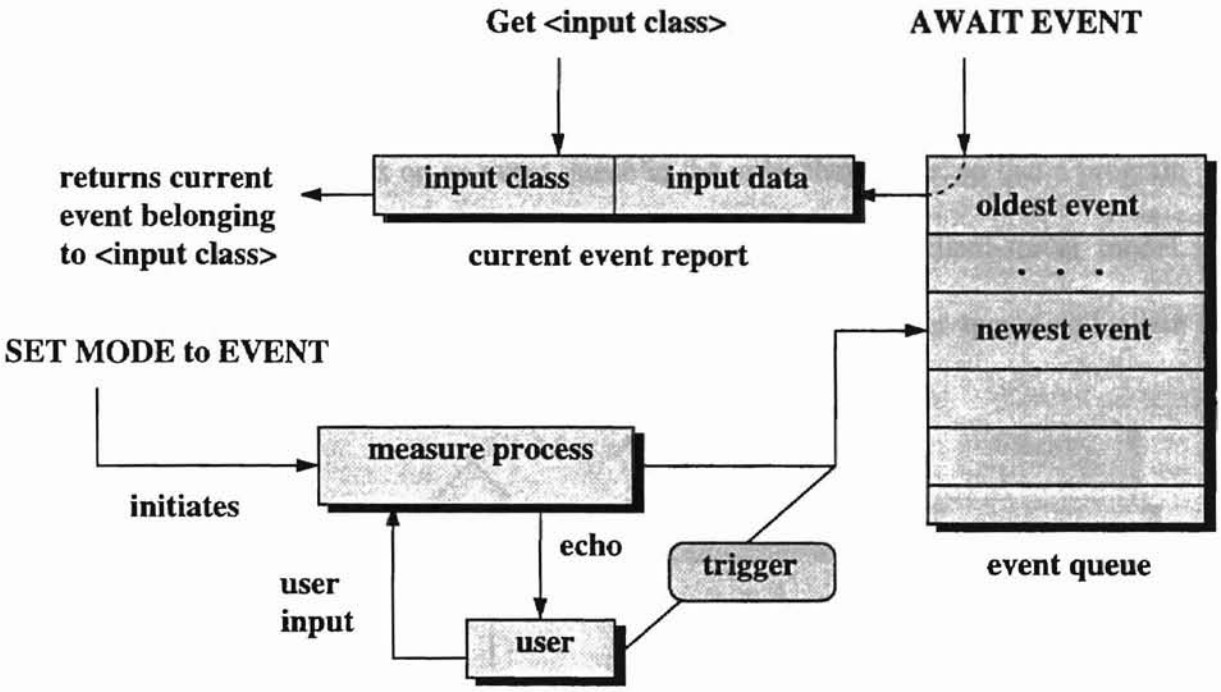


Figure 11 Event input mode (adopted from [19]).

In event mode, user needs to respond to the echo and gives a trigger to produce an event. Instead of returning the event measure at once, the measure process puts this event into an event queue. Then a function call GetInputClass will get events from event queue. Based on the input class and input data, a current event report will be returned as the measure of the current event. The whole process is described in figure 11.

2.7 Event handling in X-Window system

2.7.1 Events

In X-window, both user input and interaction with other programs are events. Like any other mouse-driven window system, an X client can respond to the different events.

Events, as they occur, are placed on a queue and are processed by clients. Unlike traditional programs, the X-window has functions for receiving events, and then the program branches and performs the appropriate response based on the event's type [7].

2.7.2 Event queue

X-window puts events on an event queue in the order they occur, so that a program can read them and take actions accordingly. X-windows follows client-server model. Application programs are called clients. Figure 12 shows the server queue and client queues.

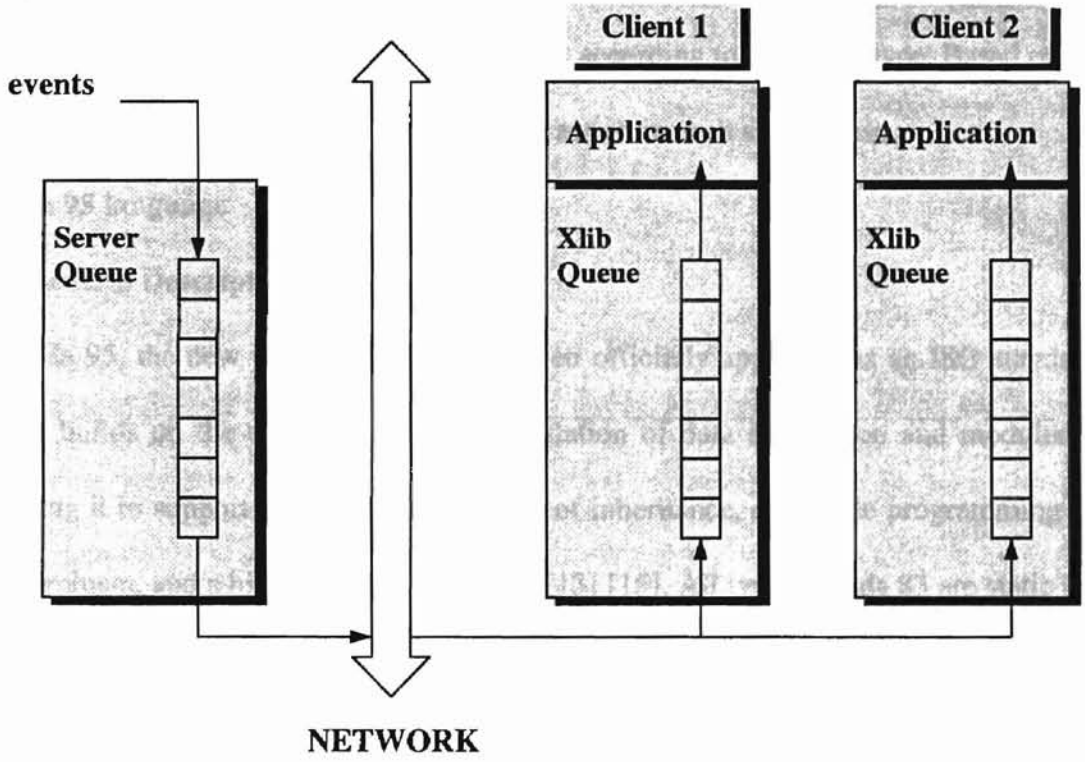


Figure 12 The server's event queue and client's event queue (adopted from [7]).

The server has an event queue which includes all events and dispatches events to the Xlib queue in each client. The server transfers the events periodically from its queue

to the client Xlib queues over the network. The client has an event-receiving loop to process the events in its event queue. The client can remove each event from its client queue at any time and process it according to its type. Peeking the queue, removing one event and putting it back, or clearing all the events from the queue are also done by the client.

2.7.3 Event loop

Every application program has an event loop to receive and process events. Usually this loop is an infinite **while** loop. An event-getting routine is in the beginning of this loop and a **switch** statement follows it to branch according to the event type. Based on the type of event, the client performs the desirable action in each switch case.

2.8 Ada 95 language

2.8.1 General Description of Ada 95

Ada 95, the new version of Ada, has been officially approved as an ISO standard. Ada 95 builds on the excellent Ada 83 foundation of data abstraction and modularity, enhancing it to support type extension as part of inheritance, classwide programming for polymorphism, and a hierarchical library [12] [13] [15]. All types in Ada 83 are static and thus Ada 83 is not classified as a truly Object Oriented language but an Object Based language [17]. However, Ada 95 includes the essential functionality associated with OOP such as polymorphism and type extension [12]. Moreover, Ada 95 has many optional functions to provide additional specialized facilities for six distinct application areas. These areas are Systems Programming, Real-Time Systems, Distributed Systems, Information Systems, Numerics, and Safety and Security [12].

2.8.2 Some important features useful to GNA95GP graphics package

(1) Packages and private types

Packages are the specification of groups which are logically related. It can be used to specify groups of related entities including subprograms that can be called from outside the package, while the inner workings remain concealed and protected from outside users. A package consists of two parts: the specification which gives the interface to the outside world, and the body which gives the hidden details [14] [15].

In order to separate the characteristics used directly by outside program units from other characteristics used within this package, a private type can be declared in a package.

(2) Type extension

It is useful to introduce a new type which is similar in most respects to an existing type but which is a distinct type. Ada 95 supports this two ways.

(a) Derived type:

The primitive operations are inherited by the derived type. They could be replaced and further primitive operations could be added if the derivation is in a package specification.

(b) Tagged type:

Tagged type is a more flexible form of derivation where it is possible to add additional components to a record type as well as additional operations. This gives rise to a possible tree of types where each type contains the components of its parent plus other components as well.

(3) Interfacing with other languages

Ada 95 can communicate with the other languages using binding technology [23] [24]. The ability of interfacing to other languages is a very important aspect in Ada 95. Using parameters and function binding in Ada 95, a subprogram written in another language can be called from Ada [25]. Since many bindings and other external systems are written in C, one of the more important objectives of Ada 95 is to ease the job of having Ada code work with such software [15]. The C interface package in Ada 95, *Interfaces.c*, supports importing C programs into Ada, and exporting Ada subprograms to C. Since system which supported procedures in Windows 3.1 are written in C, the C interface package is helpful to develop graphics applications in Ada 95 to run under Windows 3.1 [14].

Chapter Three

Implementation Issues of Event Handling in GNA95GP

3.1 Software Architecture

3.1.1 Free software for GNA95GP

The software used for developing event handling package of GNA95GP are all free software according to the definition of the free software foundation. They are DJGPP, GNAT, RSXWDK.

(1) DJGPP:

DJGPP is a free environment for developing 32-bits protected mode software in C/C++ under MS-DOS [26]. The newest version of DJGPP is 2.01 [27]. GNA95GP used version 1.12 for Windows 3.1 environment. A new version of GNA95GP is being developed using DJGPP to run under Windows 95 [11].

(2) GNAT:

GNAT is the GNU Ada 95 Translator which works with gcc to compile Ada source[28]. The newest version of GNAT for DOS environment is 3.05 [29]. Based on DJGPP, GNAT compiles an Ada program using gcc and links using gnatbl to make DOS execution file. But when a Windows 3.1 execution file is made using RSXWDK, a new Ada library needs to be built using "ar" command to be rebuilt to avoid link errors.

(3) RSXWDK:

RSXWDK is used to build 32-bit Windows applications with GNU-C/C++ for MS-Windows 3.1 [30]. Now a new version RSXNTDJ is available for Windows 95 and Windows NT [30].

Figure 13 is the software architecture for event handling in GNA95GP. It shows how GNA95GP is related to the application programs. Especially, it shows how the C binding is used.

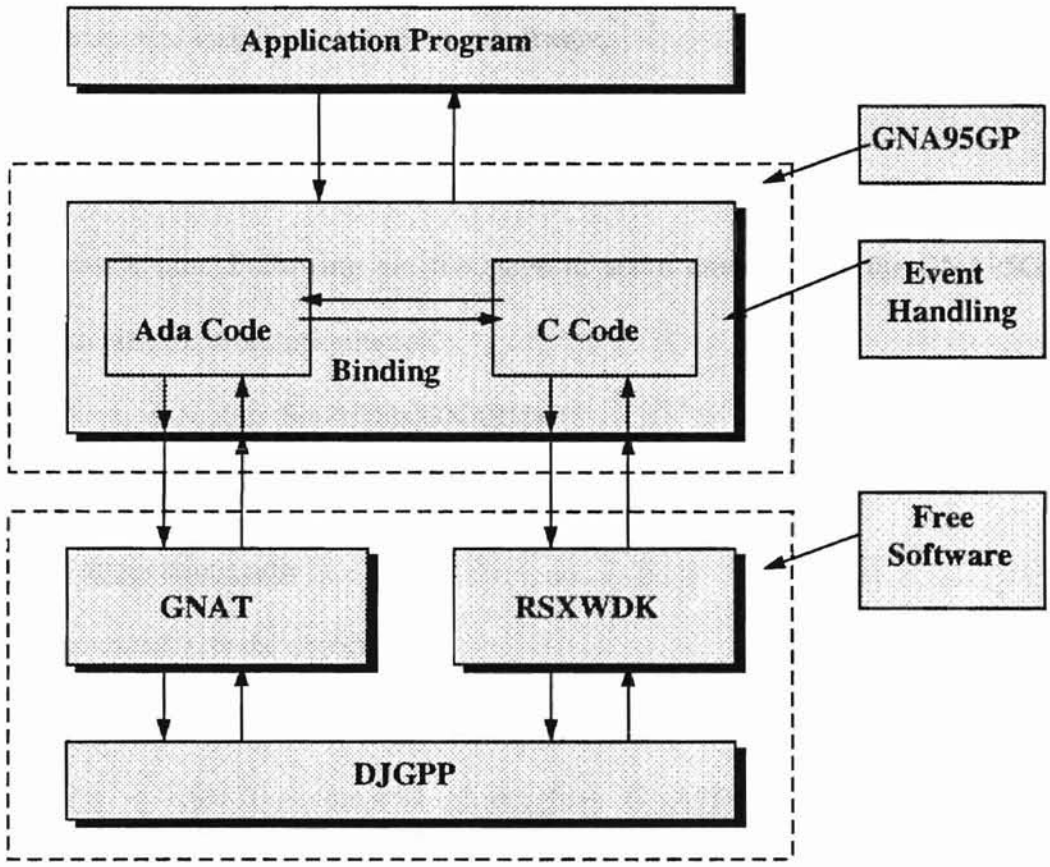


Figure 13 Software architecture for event handling.

3.1.2 The usage of these free software

In order to use GNA95GP correctly, the user need:

(1) The free software packages DJGPP, GNAT, RSXWDK, and GNA95GP can be obtained from INTERNET. The WWW sites for these software are:

DJGPP. http://www.delorie.com/djgpp/.

GNAT: http://www.gnat.com/.

RSXWDK: ftp://ftp.uni-bielefeld.de/pub/systems/msdos/misc/.

GNA95GP: http://www.cs.okstate.edu/gna95gp/. In this site, there are some help files to guide users to install and to use these software.

(2) Installation and environment setting for GNA95GP

The "README" files for DJGPP, GNAT, RSXWDK, and GNA95GP contain installation instruction. Following are the steps to setup environment for GNA95GP. Please refer to [10] for more information.

(3) Make an execution file of this GNA95GP

* Compile command:

gcc -c applications.**ada**

This command gets the object files.

* Link command:

gcc -win -o exec-file-name.**w32 object files GNA95GP library GNAT ada library**

This command gets the **w32** file.

* Get execution file:

rsxwbind exec-file-name.**w32**

After this command, the execution file, exec-file-name, is produced. The user can execute this file under Windows 3.1.

3.2 Design considerations for event handling package

3.2.1 Interface with other packages of GNA95GP

GNA95GP is a graphics package. It includes drawing package, event handling package, and storage package. Each package is parallel with other packages in structure to make them independ of each other and make each package portable. Because GNA95GP supports these three packages under windows environment, the three packages must be coordinated in the same "WinMain". And some objects, such as Windows 3.1 class and some windows handlers should be shared by these three packages i.e. using some global valuables. Even though the three packages of GNA95GP are parallel in structure, they are dependent on each other. Figure 14 is the architecture of GNA95GP:

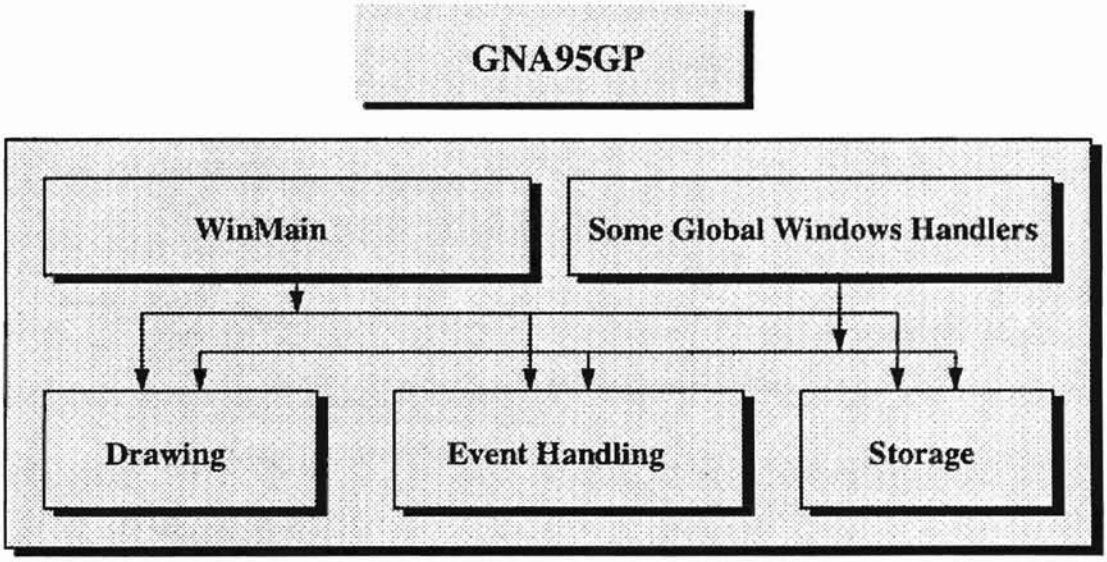


Figure 14 The architecture of GNA95GP

```
/*****************************************************************/
/*                WinMain Program And WndProc For GNA95GP        */
/*****************************************************************/
#include <windows.h>
#include "win_head.h"

long FAR PASCAL _export WndProc (HWND, UINT, UINT, LONG) ;
void UserMain(void);   /* user main program in ada code */

/* the following are globe valuables for class name and some handlers */
HANDLE Gna95gp_hInstance;
int Gna95gp_nCmdShow;
char *szAppName="Gna95gp";
HWND hwndMain;
HDC hdc;
UINT message;
LONG lParam,plParam;

/* the following are globe valuables for event handling package */
int Gna95gp_cur_mode[3];
int Gna95gp_device_at_head_of_queue;
deluxe_locator_measure Gna95gp_cur_locator_measure;
deluxe_locator_measure Gna95gp_get_locator_measure;
deluxe_keyboard_measure Gna95gp_cur_keyboard_measure;
deluxe_keyboard_measure Gna95gp_get_keyboard_measure;
int Gna95gp_cur_locator_button_mask;
int Gna95gp_cur_keyboard_measure_length;
int Gna95gp_cur_keyboard_processing_mode;
int Gna95gp_get_timestamp;
int currentDevice=NO_DEVICE;

/* WinMain */
INT PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpszCmdParam, INT nCmdShow)
    {

    WNDCLASS    wndclass ;

    if (!hPrevInstance)
     {
     wndclass.style       = CS_HREDRAW | CS_VREDRAW ;
     wndclass.lpfnWndProc  = WndProc ;
     wndclass.cbClsExtra   = 0 ;
     wndclass.cbWndExtra   = 0 ;
     wndclass.hInstance    = hInstance ;
     wndclass.hIcon        = LoadIcon (NULL, IDI_APPLICATION) ;
     wndclass.hCursor      = LoadCursor (NULL, IDC_ARROW) ;
     wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
     wndclass.lpszMenuName  = NULL ;
     wndclass.lpszClassName = szAppName ;

     RegisterClass (&wndclass) ;
     }

    Gna95gp_hInstance = hInstance;
    Gna95gp_nCmdShow  = nCmdShow;

    UserMain();

    return 0;
    }

/* WndProc */
long FAR PASCAL _export WndProc (HWND hwnd, UINT message, UINT wParam,
                                                          LONG lParam)
    {

    switch (message)
     {

     case WM_MOUSEMOVE:
          if(Gna95gp_cur_mode[LOCATOR]==INACTIVE)
                    break;
          Gna95gp_cur_locator_measure.position.x=LOWORD(lParam);
          Gna95gp_cur_locator_measure.position.y=HIWORD(lParam);
          currentDevice=NO_DEVICE;
          break;

     case WM_CHAR:
          if(Gna95gp_cur_mode[KEYBOARD]==INACTIVE)
                    break;
          GetCursorPos((POINT FAR *)&Gna95gp_cur_keyboard_measure.position);
          ScreenToClient(hwnd,
                    (POINT FAR *)&Gna95gp_cur_keyboard_measure.position);
          if(Gna95gp_cur_keyboard_processing_mode==RAW)
                    HandleRawModeKeyEvent(wParam,lParam);
          else
                    HandleProcModeKeyEvent(wParam,lParam);
          currentDevice=KEYBOARD;
          break;

     case WM_LBUTTONDOWN:
     case WM_MBUTTONDOWN:
     case WM_RBUTTONDOWN:
     case WM_LBUTTONUP:
     case WM_MBUTTONUP:
     case WM_RBUTTONUP:
          if(Gna95gp_cur_mode[LOCATOR]==INACTIVE)
                    break;
          HandleWinButtonEvent(wParam,lParam);
                    break;
     case WM_DESTROY:
          PostQuitMessage (0) ;
          return 0 ;
     }

    return DefWindowProc (hwnd, message, wParam, lParam) ;
}
```

Figure 15 WinMain program and WndProc function.

3.2.2 The design style of event handling package

In order to give the convenience to the user who is familiar with some graphics packages, the event handling package offers the functions in the style of standard graphics packages. SRGP is a graphics package which is accepted as a tool for graphics courses by many universities [1] [6]. Event handling package follows the style of SRGP.

3.3 Implementation of event handling

3.3.1 WinMain program and WndProc function

In **3.2.1**, we mentioned that a "WinMain" function will be used to coordinate the three packages Drawing package, Event handling package, and Storage package of GNA95GP. "WinMain" function and "WndProc" function implementation are given in figure 15. In this program, a function "void UserMain(void)" is declared as the main function. That means that all users who use GNA95GP must have their main procedure name "UserMain".

3.3.2 Binding between C and Ada 95

Event handling package consists of two parts, C codes and Ada codes. Using RSXWDK, C codes can get the event properties and develop event handling functions for Windows 3.1 program. Using GNAT, Ada language can bind with C language such that event handling package can be developed in Ada language [23] [24] [25]. The important and difficult process is how to bind and where to bind to make sure that the communication is effective and to make sure that every event is received. This package follows the standard graphics style for event handling. Figure 16 is an example of real

source code for binding C language with Ada language in event handling. This example is a typical usage of event handling in source codes.

C Code

```
    /*******************/
    /* get coordination X */
    /*******************/
int Win_GetDeluxeLocatorX(void)
{
  int temp;
  temp=Gna95gp_get_locator
            _measure.position.X;
  return temp;
}


    /*******************/
    /* get coordination Y */
    /*******************/
int Win_GetDeluxeLocatorY(void)
{
  int temp;
  temp=Gna95gp_get_locator
            _measure.position.Y;
  return temp;
}
```

Ada Code

```
-------------------------------
--      get coordination X    --
-------------------------------
function win_getDeluxeLocatorX return integer;
pragma import (C, win_getDeluxeLocatorX,
                  "Win_GetDeluxeLocatorX");
function Gna95gp_getDeluxeLocatorX return integer is
  X: integer;
begin
  X:=win_getDeluxeLocatorX;
  return X;
end Gna95gp_getDeluxeLocatorX;


-------------------------------
--      get coordination Y    --
-------------------------------
function win_getDeluxeLocatorY return integer;
pragma import (C, win_getDeluxeLocatorY,
                  "Win_GetDeluxeLocatorY");
function Gna95gp_getDeluxeLocatorY return integer is
  Y: integer;
begin
  Y:=win_getDeluxeLocatorY;
  return Y;
end Gna95gp_getDeluxeLocatorY;
```

Figure 16 Binding between C and Ada 95.

This piece of code, shown in figure 16, highlights the key point for binding between C and Ada 95 languages in GNA95GP. The C code calls Windows 3.1 functions (Windows SDK) supported by RSXWDK and gives the graphical features for GNA95GP. Then Ada 95 code uses the feature of binding Ada with other languages (i.e. using **pragma import** function of Ada 95) to get these windows information to develop Ada 95 graphical package. From figure 16, we know that the names of functions to be converted

from C language to Ada 95 language are not necessarily the same, but the first function name in **pragma import** must match exactly with the function name in C code.

3.3.3 Functions supported in the event handling package

(1) Functions in event handling of GNA95GP to set parameters

Figure 17 shows the functions for setting parameters, creating a window and getting an event.

```
--------------------------------------------------------------------------------
--          Creating a Window, setting parameters and getting events          --
--------------------------------------------------------------------------------
• procedure Gna95gp_CreatWindow ( name: String; width: integer; height: integer );
• procedure Gna95gp_setInputMode ( device: inputDevice; mode: inputMode );
• function Gna95gp_waitEvent ( max_wait_time: integer ) return inputDevice;
• procedure Gna95gp_setLocatorButtonMask ( mask: buttonMask );
• procedure Gna95gp_setKeyboardProcessingMode ( mode: keyboardMode );
• procedure Gna95gp_setLocatorMeasure ( value: point );
• procedure Gna95gp_setKeyboardMeasure ( str: String );
```

Figure 17 Creating a window, setting parameters, and getting events.

(2) Functions in event handling of GNA95GP to get mouse measure in event mode

Figure 18 shows the functions supported by event handling package for getting mouse measure in event mode.

```
--------------------------------------------------------------------------------
--                  Get mouse measure in event mode                           --
--------------------------------------------------------------------------------
• function Gna95gp_getLocator return locator_measure_Ptr;
• function Gna95gp_getLButtonChord return buttonStatus;
• function Gna95gp_getMButtonChord return buttonStatus;
• function Gna95gp_getRButtonChord return buttonStatus;
• function Gna95gp_getShiftChord return buttonStatus;
• function Gna95gp_getControlChord return buttonStatus;
• function Gna95gp_getMetaChord return buttonStatus;
• function Gna95gp_getLastTran return integer;
• function Gna95gp_getDeluxeLocatorX return integer;
• function Gna95gp_getDeluxeLocatorY return integer;
• function Gna95gp_getDeluxeLocator return deluxe_locator_measure_Ptr;
```

Figure 18 Functions to get mouse measure in event mode.

(3) Functions in event handling of GNA95GP to get keyboard measure in event mode

Figure 19 shows the functions supported by event handling package for getting keyboard measure in event mode.

```
--------------------------------------------------------------------------------
--                    Get keyboard measure in event mode                      --
--------------------------------------------------------------------------------
• procedure Gna95gp_getKeyboard ( buffer: String );
• function Gna95gp_getShiftKey return buttonStatus;
• function Gna95go_getControlKey return buttonStatus;
• function Gna95gp_getAltKey return buttonStatus;
• function Gna95gp_getKeyboardX return integer;
• function Gna95gp_gerKeyboardY return integer;
• function Gna95gp_getDeluxeKeyboard return deluxe_keyboard_measure_Ptr;
```

Figure 19 Functions to get keyboard measure in event mode.

(4) Functions in event handling of GNA95GP to get mouse measure in sample mode

Figure 20 shows the functions supported by event handling package for getting mouse measure in sample mode.

```
--------------------------------------------------------------------------------
--                    Get mouse measure in sample mode                        --
--------------------------------------------------------------------------------
• function Gna95gp_sampleDeluxeLocatorX return integer;
• function Gna95gp_sampleDeluxeLocatorY return integer;
• function Gna95gp_sampleLButtonChord return buttonStatus;
• function Gna95gp_sampleMButtonChord return buttonStatus;
• function Gan95gp_sampleRButtonChord return buttonStatus;
• function Gan95gp_sampleShiftChord return buttonStatus;
• function Gan95gp_sampleControlChord return buttonStatus;
• function Gan95gp_sampleMetaChord return buttonStatus;
• function Gan95gp_sampleLastTran return integer;
• function Gan95gp_sampleLocator return locator_measure_Ptr;
• function Gan95gp_sampleDeluxeLocator return deluxe_locator_measure_Ptr;
```

Figure 20 Functions to get mouse measure in sample mode.

(5) Functions in event handling of GNA95GP to get keyboard measure in sample mode

Figure 21 shows the functions supported by event handling package for getting keyboard measure in sample mode.

```
--------------------------------------------------------------------------
--                  Get keyboard measure in sample mode                 --
--------------------------------------------------------------------------
• procedure Gna95gp_sampleKeyboard ( buffer: String );
• function Gna95gp_sampleShiftKey return buttonStatus;
• function Gan95gp_sampleControlKey return buttonStatus;
• function Gan95gp_sampleAltKey return buttinStatus;
• function Gan95gp_sampleKeyboardX return integer;
• function Gan95gp_sampleKeyboardY return integer;
• function Gna95gp_sampleDeluxeKeyboard return deluxe_keyboard_measure_Ptr;
```

Figure 21 Functions to get keyboard measure in sample mode.

(6) Functions in event handling of GNA95GP to handle I/O

In order to make GNA95GP work effectively, an I/O package is developed to handle input and output in windows environment. Figure 22 lists these functions.

```
-- Some important input and output functions

• procedure Gna95gp_num_to_string ( str: String; num: intrger );
• procedure Gna95gp_flo_to_string( str: String; num: float );
• procedure Gna95gp_char ( x: integer; y: integer; text: String; size: integer );
• procedure Gna95gp_textCoord ( x: integer; y: integer; text: String );
• function Gna95gp_fopen_read ( fname: String ) return integer;
• function Gna95gp_file_read ( hRfile: integer; str: String; num: integer ) teturn integer;
• function Gna95gp_fopen_write ( fname: String ) return integer;
• function Gna95gp_file_write ( hWfile: integer; str: String; num: integer ) return integer;
• procedure Gna95gp_fclose ( hfile: integer );
```

Figure 22 I/O functions of GNA95GP under Windows environment.

3.3.4 Declaration of data types, structures and parameters list

(1) Declaration of data types and structures

Figure 23 shows the type declarations specific to event handling in GNA95GP.

```
-- DECLARATION FOR DATA TYPE AND STRUCTURE --
-- OF EVENT HANDLING --

subtype XCoord is integer range 0 .. 1024;
subtype YCoord is integer range 0 .. 800;
type point is
  record
    x: XCoord;
    y: YCoord;
  end record;
type locator_measure is
  record
    position: point;
    button_chord: buttonChord ( 1 .. 3 );
    button_of_last_transition: integer;
  end record;
type Gna95gp_timestamp is
  record
    seconds: integer;
    ticks: integer;
  end record;
type deluxe_locator_measure is
  record
    positon: point;
    button_chord: buttonChord ( 1 .. 3 );
    button_of_last_transition: integer;
    modifier_chord: buttonChord ( 1 .. 3 );
    timestamp: Gna95gp_timestamp;
  end record;
type inputDevice is ( NO_DEVICE, LOCATOR, KEYBOARD );
type inputMode is ( INACTIVE, SAMPLE, EVENT );
type buttonStatus is ( UP, DOWN );
type buttonChord is array ( integer range<> ) of buttonStatus;
type buttonMask is ( LEFT_BUTTON_MASK, MIDDLE_BUTTON_MASK,
                     RIGHT_BUTTON_MASK );
type keyboardMode is ( EDIT, RAW );
type locator_measure_Ptr is access locator_measure;
type deluxe_locator_measure_Ptr is access deluxe_locator_measure;
type deluxe_keyboard_measure_Ptr is access deluxe_keyboard_measure;
```

Figure 23 Declaration of data types and structures.

(2) Parameters list

Parameters list specific to event handling in GNA95GP is described in table 1.

Table 1 Parameters list for event handling in GNA95GP

Parameters Name	Meaning
NO_DEVICE	No input device is set.
LOCATOR	Input device is set to LOCATOR.
KEYBOARD	Input device is set to KEYBOARD.
INACTIVE	Set no input mode.
SAMPLE	Input mode is set to SAMPLE mode.
EVENT	Input mode is set to EVENT mode.
UP	Button status is UP.
DOWN	Button status is DOWN.
LEFT_BUTTON_MASK	Left button of mouse is set.
MIDDLE_BUTTON_MASK	Middle button of mouse is set.
RIGHT_BUTTON_MASK	Right button of mouse is set.
EDIT	Keyboard mode is set to EDIT mode.
RAW	Keyboard mode is set to RAW mode.

3.3.5 Using event handling

Event handling package supports two input devices, namely LOCATOR (mouse) and KEYBOARD. Both devices are supported with SAMPLE mode and EVENT mode.

3.3.5.1 SAMPLE MODE

*** Activating a device and setting the mode**

The following function is used to activate or deactivate a device, taking a device and a mode as parameters:

```
procedure Gna95gp_setInputMode (LOCATOR/KEYBOARD: inputDevice;
                                INACTIVE/SAMPLE/EVENT: inputMode);
```

Thus, to set the locator to sample mode, the following procedure call can be used:

```
Gna95gp_setInputMode (LOCATOR, SAMPLE);
```

Initially LOCATOR and KEYBOARD are inactive. Also setting one input device does not affect another input device.

*** The locator's measure**

The data type of locator_measure is shown in figure 23. After activating the locator in SAMPLE mode using Gna95gp_setInputMode function, users can get the current measure calling the function **Gna95gp_sampleLocator**. Based on the measure received, users can make use of the coordinates (x, y) or mouse button information to complete actions such as drawing a line.

Sample mode can be used for LOCATOR and KEYBOARD devices. Usually keyboard device is operated in event mode.

3.3.5.2 EVENT MODE

*** Get an event**

Similar to sample mode, user can activate or deactivate a device in event mode. After function **Gna95gp_setInputMode** has activated a device (LOCATOR or KEYBOARD), the program can inspect the event queue by entering the wait state using:

```
function Gna95gp_waitEvent (max_wait_time: integer) return inputDevice;
```

This function returns immediately if the event queue is not empty; otherwise, the parameter specifies the maximum amount of time (measured in 1/60 seconds) the function should wait for an event to fill the queue. A negative max_wait_time such as -1

will make this function wait indefinitely. And a value of zero makes it return immediately regardless of the status of the queue. The return value of this function is LOCATOR or KEYBOARD or NO_DEVICE depending on the head event in event queue.

*** The keyboard device**

The keyboard device has two processing modes, EDIT mode and RAW mode. Different processing modes have different event triggers for keyboard device. EDIT mode is used when the application receives strings from the user, who types and edits the string and then presses the return key to trigger the event. When the keyboard interactions must be monitored closely, RAW mode is used. In RAW mode, every key press is an event trigger. Using EDIT mode, the user can type entire strings, correcting them with the backspace key as necessary, and then use the Return (or Enter) key as trigger. This mode just accepts "backspace" and "return" keys and ignores other control keys. The measure is the string as it appears at the time of the trigger. In RAW mode, on the other hand, each character typed, including control characters, is a trigger and is returned individually as the measure. The application uses the following procedure to set the processing mode:

procedure **Gna95gp_setKeyboardProcessingMode** (EDIT/RAW: **keyboardMode**);

When function **Gna95gp_waitEvent** returns the device code KEYBOARD, the application obtains the measure associated with the event by calling the procedure:

procedure **Gna95gp_getKeyboard** (buffer: **String**);

When the keyboard device is active in RAW mode, its measure is always exactly one character in length. In this case, the first character of the measure string is the RAW measure.

*** The locator device**

Pressing or releasing of a mouse button is the trigger of locator device. When the function **Gna95gp_waitEvent** returns the device code LOCATOR, the application obtains the measure associated with the event by calling:

```
function Gna95gp_getLocator return locator_measure_Ptr;
```

An application can get the measure (pointer to locatorMeasure) of a mouse event. Typically, the position field of the measure is used to determine in which area of the screen the user designated the point. For example, if the locator cursor is within a rectangular region where a menu button is displayed, the event should be thought of as a request for some action. If the locator cursor is in the main drawing area, the point might be inside a previously drawn object to indicate it should be selected, or in an empty region to indicate where a new object should be placed.

An event can be pressing a button or releasing a button. And events can come from different actions of different mouse buttons. A mouse button can be set to trigger a locator event by calling the procedure:

```
procedure Gna95gp_setLocatorButtonMask (mask: buttonMask);
```

Button mask can be LEFT_BUTTON_MASK, or MIDDLE_BUTTON_MASK, or RIGHT_BUTTON_MASK. The default locator mask is LEFT_BUTTON_MASK.

3.4 Create a window

A function to create a window for GNA95GP drawing objects and processing events is supported by event handling package. This function is:

```
procedure Gna95gp_CreatWindow (name: String;  width: integer;
                                               heigtht: integer);
```

This procedure is to create a window. The name of this window and its dimensions given as parameters.

3.5 Name of event handling package

The package name "**devices**" refers to event handling package. Thus in UserMain program, the statement "**with devices; use devices;**" makes event handling package available.

3.6 I/O package for input and output

3.6.1 Usage of I/O package

Since GNA95GP supports graphical functions under windows environment, an I/O package is needed to handle input and output in windows environment.

Windows 3.1 input and output are different from DOS. Usually, the method for Windows 3.1 program output places all the results into a string, and then puts the string on the windows screen. A description of I/O package is given below in table 2.

Table 2 I/O package in GNA95GP

Function/procedure name	Behavior
procedure **Gna95gp_num_to_string** (str: **String;** num: **integer**);	Puts an integer number into a string.
procedure **Gna95gp_flo_to_string** (str: **String;** num: **float**);	Puts a floating point number into a string.
procedure **Gna95gp_char** (x: **integer;** y: **integer**; text: **String**; size: **integer**);	Outputs a string to screen. The integer variables x,y are the coordinates to position this string. The point (0,0) is the upper left corner of the base window. The size variable is the length of this string. If this string just has a character, the size is 1.
procedure **Gna95gp_textCoord** (x: **integer;** y: **integer**; text: **String**);	Prints a string starting at the position (x,y).
function **Gna95gp_fopen_read** (fname: **String**) return **integer;**	Opens a file to read. The string "fname" refers to the name of the input file. The integer value returned is the file handler for this file.
function **Gna95gp_file_read** (hRfile: **integer**; str: **String**; num: **integer**) return **integer;**	This function reads a string whose length is an integer, "num" , from an opened file whose file handler is "hRfile" and returns an integer value. When the return value is greater than zero, that means the read function worked correctly. Otherwise, errors occurred.
function **Gna95gp_fopen_write** (fname: **String**) return **integer;**	Opens a file to write. The string "fname" refers to the name of the output file. The integer value returned is the file handler for this opened file.
function **Gna95gp_file_write** (hWfile: **integer**; str: **String**; num: **integer**) return **integer;**	This function writes a string whose length is an integer, "num", from an opened file whose file handler is "hWfile" and returns a integer value. When the value returned is greater than zero, that means the write function worked correctly. Otherwise, errors occurred.
procedure **Gna95gp_fclose** (hfile: **String**);	Closes the file whose file handler is "hfile".

3.6.2 Name of I/O package

The package name "**io**" refers to I/O package. Thus in user's UserMain program, "**with io; use io**;" makes I/O package visible.

3.7 Examples of event handling usage

Two example programs are used to demonstrate the usage of event handling package. The first program uses the functions of event handling package to handle keyboard RAW mode, locator inactive mode, locator sample mode, locator event mode, button mask, button chord, et. al. Appendix I gives the source code of this program. The second program uses the event handling package to get locator measure, and the coordinates of locator point, and to draw lines using these points. Appendix II gives the source code of this program. Appendix III gives the complete source code listing for this event handling package.

Chapter Four

Summary

GNA95GP focuses on the development of tools used in undergraduate computer course. It is a free software which is modeled after graphics packages SRGP and PHIGS to provide a high-level programming interface for students to write graphics applications using Ada 95 language. It is designed to support dynamic and interactive Ada 95 graphics applications which run under Windows 3.1 in IBM PCs.

Event handling package of GNA95GP developed and implemented event handling feature to Ada 95 2D graphics package. It provides users an easy way to write interactive programs using the Ada 95 language. In order to be convenient for those users who are familiar with a graphics package to learn this package, this event handling package has been developed in the standard graphics package style. This event handling package will be helpful for developing graphics applications in Ada 95 programming language. More information is available at the WWW site ***http://www.cs.okstate.edu/gna95gp/.***

Bibliography

[1] Foley, J. D., Van Dam, A. et. al. ***Introduction to Computer Graphics.*** Reading, Massachusetts: Addison-Wesley Publishing Company, 1994.

[2] Hopgood, F, R, A., Duce, D, A., Gallop, J,R., Sutcliffe, D,C., ***Introduction to the Graphical Kernel System***. Orlando, Florida: Academic Press, Inc., 1986.

[3] Enderle, G., Kansy, K., Pffaff, G., ***Computer Graphics Programming: GKS-The Graphics Standard***. New York: Springer-Verlag Heidelberg, 1987.

[4] Blake, J. W. ***PHIGS And PHIGS PLUS***. London: Academic Press Limited, 1993.

[5] Gaskins, T. ***PHIGS Programming Manual***. Sebastopol, CA: O'Reilly & Associates, Inc. 1992.

[6] James, D., Andries, V., and Steven, K., ***Introduction to Computer Graphics,*** Addison-Wesley Publishing Company, Inc., February, 1994.

[7] Nye, A. ***Xlib Programming Manual for Version 11.*** O'Reilly & Associates, Inc., July, 1993.

[8] Li, L., Kuang, S. and George, K. M. ***A 2D Graphics Package in Ada 95.*** Proceedings of the 10th Annual ASEET (Ada Software Engineering and Education Team) Symposium. Prescott, Arizona. 1996.

[9] Lan, Li ***Design And Implementation Of A 2D Graphics Package in Ada 95***. MS thesis, Department Of Computer Science, Oklahoma State University, Dec. 1996.

[10] George, K. M., Li, Lan, Kuang, Shan and Huang, Yang. ***Ada 95 2D Graphics Package For PC***. web site: http://www.cs.okstate.edu/gna95gp/.

[11] Huang, Yang. ***Upgrade And Enhance GNA95GP Graphics Package For Windows 95***. MS thesis in progress, Computer Science Department, Oklahoma State University.

[12] Barnes, J. ***Programming In Ada 95***. Workingham, England: Addison-Wesley Publishing Company, 1996.

[13] Booch, G., ***Software Engineering with Ada.*** Menlo Park, CA: The Benjamin/Commings Publishing Company, Inc., 1987.

[14] ***Ada 9X Rationale -- The language, The Standard Libraries.*** Cambridge, Massachusetts: Intermetrics, Inc., 1994.

[15] ***Ada 95 Rationale.*** Cambridge, Massachusetts: Intermetrics, Inc., 1995.

[16] Nicholas J. De Lillo ***A First Course in Computer Science with Ada.*** Homewood, IL: Richard D. Irwin, Inc., 1993

[17] ***Reference Manual for the ADA Programming Language.*** New York: United States Department of Defense, American National Standards Institute, Inc.,1983.

[18] Skansholm, J. ***ADA From The Beginning.*** Workingham, England: Addison-Wesley Publishing Company, 1988.

[19] Salmon, R.; Slater, M. ***Computer Graphics Systems & Concepts***. Workingham, England: Addison-Wesley Publishers Limited, 1987.

[20] Petzold, C. ***Programming Windows 3.1***. Redmond, Washington: Microsoft Press, 1992.

[21] Kruglinski, David J. ***Inside Visual C++***. Redmond, Washington: Microsoft Press, 1994.

[22] ***Microsoft Windows Programmer's Reference Volume 1.*** SDK books online, Microsoft Windows, 1992.

[23] ***Ada Bindings,*** web site: http://sw-eng.falls-church.va.us/AdaIC/source-code/.

[24] ***Ada Bindings,*** web site: http://sw-eng.falls-church.va.us/AdaIC/source-code/bindings/binding95/binding95.txt.

[25] Gart, M. ***Interfacing Ada To C -- Solutions to Four Problems.*** TRI-Ada'95 Conference Proceedings, pp. 28-34, November, 1995.

[26] ***DJGPP***, Delorie software, web site: http://www.delorie.com/.

[27] ***DJGPP***, official distribution area, web site: http: //serviceftp. flashnet. it:80/ pub /simtelnet/gnu/djgpp/.

[28] ***GNAT***, AdaCore, web site: http://www.gnat.com/.

[29] ***GNAT***, software distribution area, web site: http://cs.nyu.edu/pub/gnat/.

[30] ***RSX*** Windows Development Kit, web site: ftp://ftp.uni-bielefeld.de/pub/systems /msdos /misc/.

[31] ***Categories of Free and Non-Free Software,*** web site: http://www.fsf.org /philosophy /categories.html.

Appendix I Example One of Event Handling Usage

```
------------------------------------------------------------------------------------------------------
--                This program places the keyboard in RAW EVENT mode                                --
------------------------------------------------------------------------------------------------------
--                                                                                                   --
--        You use the keyboard to control the locator's mode:                                        --
--        i -- inactive                                                                              --
--        s -- sample                                                                                --
--        e -- event                                                                                 --
--                                                                                                   --
--        You can control button mask in this way:                                                   --
--        1 -- toggles the status of the LEFT button in the button mask                              --
--        2 -- toggles the status of the MIDDLE button in the button mask                            --
--        3 -- toggles the status of the RIGHT button in the button mask                             --
--                                                                                                   --
--        You quit by typing:                                                                        --
--        q -- QUIT                                                                                  --
--                                                                                                   --
------------------------------------------------------------------------------------------------------

-----------------------------------------
-- Including Gna95gp packages  --
-----------------------------------------
with init;
with objtype;
with Output;
with attribute;
with attrtype;
with devices;  -- this package hnadling events
use devices;
use Output;
use objtype;
use init;
with demo;  -- a subfunction of this ecample program
use demo;

pragma suppress(All_Checks);
-----------------------------------------
--          Main Procedure          --
-----------------------------------------
procedure UserMain is

  s: String(1..80);
  keym:deluxe_keyboard_measure_Ptr;

begin

  gnat95gp_init;
  Gna95gp_CreatWindow("ADA WINDOWS",640,480);  -- create a window
  Gna95gp_setInputMode(LOCATOR,INACTIVE);      -- inactive locator
  Gna95gp_setInputMode(KEYBOARD,EVENT);        -- set keyboard as event mode
  k_mode:=RAW;
  Gna95gp_setKeyboardProcessingMode(k_mode);   -- set keyboard as RAW mode
  attribute.gna95gp_setcolor(4);
  DisplayLocatorMeasure;
  attribute.gna95gp_setcolor(6);


   loop   -- loop to get events --

   if(mode=SAMPLE) then
       timeout:=20;  -- time used for sample mode --
```

```
else
    timeout:=-1;   -- time <0 means event mode --
end if;

device:=Gna95gp_waitEvent(timeout);                    -- get an event

case device is

  when NO_DEVICE =>

    measure:=Gna95gp_sampleDeluxeLocator;              -- when sample mode
    DisplayLocatorMeasure;

  when LOCATOR =>

    measure:=Gna95gp_getDeluxeLocator;                 -- whne locator mode
    DisplayLocatorMeasure;

  when KEYBOARD =>                                     -- when keyboard mode

    Gna95gp_getKeyboard(s);                            -- get keyboard input
    keym:=Gna95gp_getDeluxeKeyboard;

    if(keym.buffer(1)='k') then
       Gna95gp_textCoord(250,250,"Show Deluxe Keyboard");
    end if;

    if(s(1)='1') then
       locatorButtonMask:=LEFT_BUTTON_MASK;

      if(l=0) then
         l:=1;
       else
         l:=0;
       end if;

       Gna95gp_setLocatorButtonMask(locatorButtonMask); -- set left button mask
       DisplayLocatorMeasure;                          -- display results
     end if;

   if(s(1)='2') then
       locatorButtonMask:=MIDDLE_BUTTON_MASK;

       if(m=0) then
         m:=1;
       else
         m:=0;
       end if;

       Gna95gp_setLocatorButtonMask(locatorButtonMask); -- middle button mask
       DisplayLocatorMeasure;                          -- display results
    end if;

  if(s(1)='3') then
    locatorButtonMask:=RIGHT_BUTTON_MASK;

    if(r=0) then
      r:=1;
    else
      r:=0;
    end if;

    Gna95gp_setLocatorButtonMask(locatorButtonMask);   -- set right button mask
    DisplayLocatorMeasure;                             -- display results
  end if;
```

```
      if(s(1)='i') then
        Gna95gp_setInputMode(LOCATOR,INACTIVE);      -- set locator inactive
        mode:=INACTIVE;
        DisplayLocatorMeasure;                        -- display results
      end if;

      if(s(1)='s') then
        Gna95gp_setInputMode(LOCATOR,SAMPLE);    -- set locator sample mode
        mode:=SAMPLE;
        DisplayLocatorMeasure;                        -- display results
      end if;

      if(s(1)='e') then
        Gna95gp_setInputMode(LOCATOR,EVENT);     -- set locator event mode
        mode:=EVENT;
        DisplayLocatorMeasure;                       -- display results
      end if;

      if(s(1)='q') then                             -- quit program
        Gna95gp_end;
      end if;

    end case;
  end loop;

end UserMain;

pragma export (C,UserMain,"UserMain");
```

```
-------------------------------------------------------------------------------
--                    The ads file of function demo                          --
-------------------------------------------------------------------------------

with devices;
use devices;


-----------------------------------------------
-- Declare data structure and give initialize --
-----------------------------------------------
package demo is

  pragma suppress(All_Checks);

  device:inputDevice;
  mode:inputMode:=INACTIVE;
  k_mode:keyboardMode;
  measure:deluxe_locator_measure_Ptr;
  timeout:integer;
  l:integer:=1;
  m:integer:=0;
  r:integer:=0;

  locatorButtonMask: buttonMask :=LEFT_BUTTON_MASK;

  procedure DisplayLocatorMeasure;

end demo;
```

```
------------------------------------------------------------
--      Subfunction Demo.adb of Example One       --
------------------------------------------------------------
with devices;
use devices;

package body demo is

  pragma suppress(All_Checks);

  procedure DisplayLocatorMeasure is
    x:integer;
    y:integer;
    s1:String(1..3);
    s2:String(1..3);
  begin

    if(mode=INACTIVE) then
      Gna95gp_textCoord(10,20,"LOCATOR MODE: INACTIVE ");
    end if;

    if(mode=SAMPLE) then
      Gna95gp_textCoord(10,20,"LOCATOR MODE: SAMPLE   ");
    end if;

    if(mode=EVENT) then
      Gna95gp_textCoord(10,20,"LOCATOR MODE: EVENT    ");
    end if;

    if(mode=INACTIVE) then
      Gna95gp_textCoord(10,35,"                                                  ");
      Gna95gp_textCoord(10,50,"                                                ");
      Gna95gp_textCoord(10,65,"                                                  ");
      Gna95gp_textCoord(10,80,"                                                    ");
      Gna95gp_textCoord(10,95,"                                                  ");
    end if;

    Gna95gp_textCoord(10,305,"            INSTRUCTIONS");
    Gna95gp_textCoord(10,320,"------------------------------------------------------------");
    Gna95gp_textCoord(10,335,"Press 'i' to set INACTIVE mode.");
    Gna95gp_textCoord(10,350,"Press 'e' to set EVENT mode.");
    Gna95gp_textCoord(10,365,"Press 's' to set SAMPLE mode.");
    Gna95gp_textCoord(10,380,"Press '1' to set LEFT button mask.");
    Gna95gp_textCoord(10,395,"Press '2' to set MIDDLE button mask.");
    Gna95gp_textCoord(10,410,"Press '3' to set RIGHT button mask.");
    Gna95gp_textCoord(10,425,"Press 'q' to quit.");


    if(mode=SAMPLE or mode=EVENT) then


      Gna95gp_textCoord(10,35,"Button mask: ");
      if(l=1) then
        Gna95gp_textCoord(96,35,"  LEFT    ");
      else
        Gna95gp_textCoord(96,35,"  -------- ");
      end if;
      if(m=1) then
        Gna95gp_textCoord(145,35,"  MIDDLE    ");
      else
        Gna95gp_textCoord(145,35,"  ----------   ");
      end if;
      if(r=1) then
        Gna95gp_textCoord(220,35,"  RIGHT                ");
      else
```

```
  Gna95gp_textCoord(220,35,"   ----------                ");
end if;

Gna95gp_textCoord(10,50,"Position:   ");
x:=measure.position.x;
Gna95gp_num_to_string(s1,x);
Gna95gp_textCoord(70,50,"                                                  ");
if(x<10) then
  Gna95gp_char(70,50,s1,1);
elsif(x<100 and x>=10) then
  Gna95gp_char(70,50,s1,2);
else
  Gna95gp_char(70,50,s1,3);
end if;

Gna95gp_textCoord(95,50,",");

y:=measure.position.y;
Gna95gp_num_to_string(s2,y);
if(y<10) then
  Gna95gp_char(100,50,s2,1);
elsif(y<100 and y>=10) then
  Gna95gp_char(100,50,s2,2);
else
  Gna95gp_char(100,50,s2,3);
end if;


Gna95gp_textCoord(10,65,"Button chord:   ");
Gna95gp_textCoord(100,65,"                                                  ");

if(measure.button_chord(1)=DOWN) then
  Gna95gp_textCoord(100,65,"1");
else
  Gna95gp_textCoord(100,65,"0");
end if;

Gna95gp_textCoord(110,65,",");

if(measure.button_chord(2)=DOWN) then
  Gna95gp_textCoord(120,65,"1");
else
  Gna95gp_textCoord(120,65,"0");
end if;

Gna95gp_textCoord(130,65,",");

if(measure.button_chord(3)=DOWN) then
  Gna95gp_textCoord(140,65,"1");
else
  Gna95gp_textCoord(140,65,"0");
end if;


Gna95gp_textCoord(10,80,"Button of last transition:  ");
Gna95gp_textCoord(170,80,"                                  ");
Gna95gp_num_to_string(s1,measure.button_of_last_transition);
Gna95gp_char(170,80,s1,1);


Gna95gp_textCoord(10,95,"Modifier chord:  ");

if(measure.modifier_chord(1)=DOWN) then
  Gna95gp_textCoord(120,95,"1");
else
```

```
        Gna95gp_textCoord(120,95,"0");
      end if;

      Gna95gp_textCoord(130,95,",");

      if(measure.modifier_chord(2)=DOWN) then
        Gna95gp_textCoord(140,95,"1");
      else
        Gna95gp_textCoord(140,95,"0");
      end if;

      Gna95gp_textCoord(150,95,",");

      if(measure.modifier_chord(3)=DOWN) then
        Gna95gp_textCoord(160,95,"1");
      else
        Gna95gp_textCoord(160,95,"0");
      end if;


    end if;

  end DisplayLocatorMeasure;

end demo;
```

Appendix II Example TWO of Event Handling Usage

```
-------------------------------------------------
--        Include event handling package       --
-------------------------------------------------
with devices;
use devices;
with WinType;
use WinType;
with head;
use head;

---------------------------------
-- Ada_main.adb procedure  --
---------------------------------
procedure UserMain is

   pragma suppress(All_Checks);
   device: inputDevice;                              -- declare inputDevice type
   x1:integer;
   y1:integer;
   x2:integer;
   y2:integer;
   timeout: integer;                                 -- timeout for getting events
   measure: locator_measure_Ptr;                     -- measure for processing events

begin

   Gna95gp_CreatWindow("ADA WINDOWS",640,480);       -- create a window
   Gna95gp_setInputMode(LOCATOR,EVENT);              -- set locator event mode
   Gna95gp_setInputMode(KEYBOARD,INACTIVE);          -- set keyboard inactive
   x1:=0;
   y1:=0;

   loop                                  -- event loop

         timeout:=-1;                    -- timeout < 0 means event mode

    device:=Gna95gp_waitEvent(timeout);              -- get an event

    if (device = LOCATOR) then
         measure:=Gna95gp_getDeluxeLocator_Ptr;      -- get the measure of this event
         x2:=measure.position.x;                     -- get x coordination
         y2:=measure.position.y;                     -- get y coordination
      Gna95gp_line(x1,y1,x2,y2);                     -- draw a line
         x1:=x2;
         y1:=y2;
     end if;
   end loop;

end UserMain;

pragma export (C,UserMain,"UserMain");
```

Appendix III Source Code of Event Handling Package

```
/*****************************************************************/
/*                        Windows Main Program                   */
/*****************************************************************/
#include <windows.h>
#include "win_head.h"   /* head file for windows */

long FAR PASCAL _export WndProc (HWND, UINT, UINT, LONG) ;
void UserMain(void);   /* ada main program */

HANDLE Gna95gp_hInstance;  /* handler of Gna95gp */
int Gna95gp_nCmdShow;
char *szAppName="Gna95gp";
HWND hwndMain;       /* windows handler */
HDC hdc;             /* handler of device content */
UINT message;
LONG lParam,plParam;

int Gna95gp_cur_mode[3];  /* array of current mode */
int Gna95gp_device_at_head_of_queue;  /* variable for head event in event queue */
deluxe_locator_measure Gna95gp_cur_locator_measure;  /* current locator measure */
deluxe_locator_measure Gna95gp_get_locator_measure;  /* get locator measure */
deluxe_keyboard_measure Gna95gp_cur_keyboard_measure;    /*  current keyboard
                                                              measure*/
deluxe_keyboard_measure Gna95gp_get_keyboard_measure;  /*get keyboard measure*/
int Gna95gp_cur_locator_button_mask;          /* current locator mask */
int Gna95gp_cur_keyboard_measure_length;      /* length of keyboard measure */
int Gna95gp_cur_keyboard_processing_mode;     /* current keyboard processing mode */
int Gna95gp_get_timestamp;       /* tme stamp */
int currentDevice=NO_DEVICE;

/* WinMain */
INT PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInstance,
                    LPSTR lpszCmdParam, INT nCmdShow)
    {

    WNDCLASS    wndclass ;

    if (!hPrevInstance)
          {
          wndclass.style         = CS_HREDRAW | CS_VREDRAW ;
          wndclass.lpfnWndProc   = WndProc ;
          wndclass.cbClsExtra    = 0 ;
          wndclass.cbWndExtra    = 0 ;
          wndclass.hInstance     = hInstance ;
          wndclass.hIcon         = LoadIcon (NULL, IDI_APPLICATION) ;
          wndclass.hCursor       = LoadCursor (NULL, IDC_ARROW) ;
          wndclass.hbrBackground = GetStockObject (WHITE_BRUSH) ;
          wndclass.lpszMenuName  = NULL ;
          wndclass.lpszClassName = szAppName ;

          RegisterClass (&wndclass) ;
          }

    Gna95gp_hInstance = hInstance;
    Gna95gp_nCmdShow  = nCmdShow;

    UserMain();   /** binding c program with ada program **/

    return 0;
    }

/* WndProc */
```

```
long FAR PASCAL _export WndProc (HWND hwnd, UINT message, UINT wParam,
                                                    LONG lParam)
    {

    switch (message)
          {

          case WM_MOUSEMOVE:
             if(Gna95gp_cur_mode[LOCATOR]==INACTIVE)
                 break;
             Gna95gp_cur_locator_measure.position.x=LOWORD(lParam);
             Gna95gp_cur_locator_measure.position.y=HIWORD(lParam);
             currentDevice=NO_DEVICE;
             break;

          case WM_CHAR:
             if(Gna95gp_cur_mode[KEYBOARD]==INACTIVE)
                 break;
             GetCursorPos((POINT FAR *)&Gna95gp_cur_keyboard_measure.position);
             ScreenToClient(hwnd,
                          (POINT FAR *)&Gna95gp_cur_keyboard_measure.position);
             if(Gna95gp_cur_keyboard_processing_mode==RAW)
                 HandleRawModeKeyEvent(wParam,lParam);
             else
                 HandleProcModeKeyEvent(wParam,lParam);
             currentDevice=KEYBOARD;
             break;

          case WM_LBUTTONDOWN:
          case WM_MBUTTONDOWN:
          case WM_RBUTTONDOWN:
          case WM_LBUTTONUP:
          case WM_MBUTTONUP:
          case WM_RBUTTONUP:
             if(Gna95gp_cur_mode[LOCATOR]==INACTIVE)
                 break;
             HandleWinButtonEvent(wParam,lParam);
                 break;
          case WM_DESTROY:
             PostQuitMessage (0) ;
             return 0 ;
          }

    return DefWindowProc (hwnd, message, wParam, lParam) ;
    }



/*****************************************************************/
/*   Windows Devices For Gna95gp Event Handling In Windows Program (C)   */
/*****************************************************************/
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "win_head.h"

/* globle variables */
extern HANDLE Gna95gp_hInstance;
extern int Gna95gp_nCmdShow;
extern char *szAppName;
extern HWND hwndMain;
```

```
extern HDC hdc;
extern LONG lParam;
extern LONG plParam;
extern UINT message;

extern int Gna95gp_cur_mode[3];
extern int Gna95gp_device_at_head_of_queue;
extern deluxe_locator_measure Gna95gp_cur_locator_measure;
extern deluxe_locator_measure Gna95gp_get_locator_measure;
extern deluxe_keyboard_measure Gna95gp_cur_keyboard_measure;
extern deluxe_keyboard_measure Gna95gp_get_keyboard_measure;
extern int Gna95gp_cur_locator_button_mask;
extern int Gna95gp_cur_keyboard_measure_length;
extern int Gna95gp_cur_keyboard_processing_mode;
extern int Gna95gp_get_timestamp;
extern int currentDevice;

int win_num;
int num_of_setinputmode;
boolean screenRepaintNeeded=false;

/* init input module */
void Gna95gp_initInputModule(void)
{
  Gna95gp_cur_mode[LOCATOR]=INACTIVE;
  Gna95gp_cur_mode[KEYBOARD]=INACTIVE;
  Gna95gp_cur_keyboard_processing_mode=EDIT;
  Gna95gp_cur_locator_measure.button_chord[LEFT_BUTTON]=UP;
  Gna95gp_cur_locator_measure.button_chord[MIDDLE_BUTTON]=UP;
  Gna95gp_cur_locator_measure.button_chord[RIGHT_BUTTON]=UP;
  Gna95gp_cur_locator_measure.button_chord[LEFT_BUTTON]=UP;
  Gna95gp_cur_locator_measure.position=
    Gna95gp_getPoint(Gna95gp_max_xcoord>>1,
                     Gna95gp_max_ycoord>>1);
  Gna95gp_cur_locator_button_mask=LEFT_BUTTON_MASK;
  Gna95gp_cur_keyboard_measure.buffer=(char
*)malloc(MAX_STRING_SIZE*sizeof(char));
  Gna95gp_cur_keyboard_measure.buffer_length=MAX_STRING_SIZE;
  Gna95gp_cur_keyboard_measure.buffer[0]='\0';

}

/* windows begin */
void win_begin(char *name,int width,int height,int size,int callnum)
{
  if(callnum!=win_num){
  win_num=callnum;
  name[size]='\0';
  hwndMain=CreateWindow(szAppName,name,
                WS_OVERLAPPEDWINDOW,
                CW_USEDEFAULT,
                CW_USEDEFAULT,
                width,
                height,
                NULL,
                NULL,
                Gna95gp_hInstance,
                NULL);
  SetWindowPos(hwndMain,NULL,0,0,width,height,SWP_NOMOVE);
  hdc=GetDC(hwndMain);
  SelectObject(hdc,GetStockObject(NULL_BRUSH));
  SetBkMode(hdc,TRANSPARENT);
  SetTextAlign(hdc,TA_BASELINE);
  ReleaseDC(hwndMain,hdc);
```

```
  ShowWindow(hwndMain,Gna95gp_nCmdShow);
  UpdateWindow(hwndMain);

  Gna95gp_initInputModule();
  }
}

/* windows draw line */
void Win_Drawline(int x1,int y1,int x2,int y2)
{
  MoveTo(hdc,x1,y1);
  LineTo(hdc,x2,y2);
}

/* windows draw mark */
void Win_Drawmark(int x,int y)
{
  MoveTo(hdc,x+2,y+2);
  LineTo(hdc,x-2,y+2);
  LineTo(hdc,x-2,y-2);
  LineTo(hdc,x+2,y-2);
  LineTo(hdc,x+2,y+2);
}

/* windows get x coordination */
int Win_Xc(void)
{
  int X;

  X=LOWORD(lParam);
  return X;
}

/* windows get y coordination */
int Win_Yc(void)
{
  int Y;

  Y=HIWORD(lParam);
  return Y;
}

/* windows get a point */
POINT Gna95gp_getPoint(int x,int y)
{
  POINT Z;

  Z.x=x;
  Z.y=y;

  return Z;
}

/* windows deactive a device */
void Gna95gp_deactiveDevice(int device)
{
  switch(device){

    case LOCATOR:
      Gna95gp_cur_locator_measure.position=
        Gna95gp_getPoint(Gna95gp_max_xcoord>>1,
                  Gna95gp_max_ycoord>>1);
      break;

    case KEYBOARD:
```

```
        Gna95gp_cur_keyboard_measure.buffer[0]='\0';
        break;

    }
}

/* windows active a device */
void Gna95gp_activeDevice(int device)
{
  switch(device){

    case LOCATOR:
      break;
    case KEYBOARD:
      break;

    }
}

/* set input mode */
void Win_SetInputMode(inputDevice device,inputMode mode,int callnum)
{
  if(num_of_setinputmode!=callnum){
    num_of_setinputmode=callnum;
    if(mode==Gna95gp_cur_mode[device])
      return;

    if(mode==INACTIVE){
      Gna95gp_deactiveDevice(device);
      Gna95gp_cur_mode[device]=INACTIVE;
      return;
    }
    else{
      Gna95gp_cur_mode[device]=mode;
      Gna95gp_activeDevice(device);
    }
  }
}

/* windows handle RAW event */
int win_handleRawEvents(boolean inwaitevent,boolean forever)
{
  MSG msg;

  currentDevice=NO_DEVICE;

  if(inwaitevent&&forever){
    do{
      if(GetMessage(&msg,NULL,0,0)){
        message=msg.message;
        lParam=msg.lParam;
        TranslateMessage(&msg);
        DispatchMessage(&msg);
      }
      else exit(0);
    }while(currentDevice==NO_DEVICE);
  }
  else{
    do{
      if(PeekMessage(&msg,NULL,0,0,PM_REMOVE)){
        if(msg.message!=WM_QUIT){
          message=msg.message;
          lParam=msg.lParam;
          TranslateMessage(&msg);
          DispatchMessage(&msg);
```

```
        }
        else exit(0);
      }
      else break;
    }while(currentDevice==NO_DEVICE);
  }

  return currentDevice;
}

/* windows wait for an event */
int Win_WaitEvent(int max_wait_time)
{
  Time expiretime=0;
  inputDevice return_value;
  boolean do_continue_wait;
  boolean forever=(max_wait_time<0);

  expiretime=GetCurrentTime()+max_wait_time;
  do_continue_wait=true;

  return_value=NO_DEVICE;

  do{
    return_value=(inputDevice)win_handleRawEvents(TRUE,forever);

    if((return_value!=NO_DEVICE)||(max_wait_time==0))
      do_continue_wait=false;
    else
      if(max_wait_time>0){
        do_continue_wait=(GetCurrentTime()<expiretime);
      }
  }while(do_continue_wait);

  Gna95gp_device_at_head_of_queue=return_value;

  return return_value;
}

/* handle windows button  event */
void HandleWinButtonEvent(WORD wParam,LONG lParam)
{
  int i;

  Gna95gp_cur_locator_measure.position.x=LOWORD(lParam);
  Gna95gp_cur_locator_measure.position.y=HIWORD(lParam);


  if(wParam & MK_LBUTTON)
    Gna95gp_cur_locator_measure.button_chord[LEFT_BUTTON]=DOWN;
  else
    Gna95gp_cur_locator_measure.button_chord[LEFT_BUTTON]=UP;

  if(wParam & MK_MBUTTON)
    Gna95gp_cur_locator_measure.button_chord[MIDDLE_BUTTON]=DOWN;
  else
    Gna95gp_cur_locator_measure.button_chord[MIDDLE_BUTTON]=UP;

  if(wParam & MK_RBUTTON)
    Gna95gp_cur_locator_measure.button_chord[RIGHT_BUTTON]=DOWN;
  else
    Gna95gp_cur_locator_measure.button_chord[RIGHT_BUTTON]=UP;


  for(i=0;i<3;i++)
```

```
      if(Gna95gp_get_locator_measure.button_chord[i]
        !=Gna95gp_cur_locator_measure.button_chord[i])
        Gna95gp_cur_locator_measure.button_of_last_transition=i;


    if(wParam & MK_SHIFT)
      Gna95gp_cur_locator_measure.modifier_chord[SHIFT]=DOWN;
    else
      Gna95gp_cur_locator_measure.modifier_chord[SHIFT]=UP;

    if(wParam & MK_CONTROL)
      Gna95gp_cur_locator_measure.modifier_chord[CONTROL]=DOWN;
    else
      Gna95gp_cur_locator_measure.modifier_chord[CONTROL]=UP;

    if(GetKeyState(VK_MENU)<0)
      Gna95gp_cur_locator_measure.modifier_chord[META]=DOWN;
    else
      Gna95gp_cur_locator_measure.modifier_chord[META]=UP;


    if((Gna95gp_cur_mode[LOCATOR]==EVENT) &&
      ((Gna95gp_cur_locator_button_mask
      >>Gna95gp_cur_locator_measure.button_of_last_transition)&1)){
      Gna95gp_get_locator_measure=Gna95gp_cur_locator_measure;
      currentDevice=LOCATOR;
    }
    else
      currentDevice=NO_DEVICE;

}

/* windows get deluxe locator measure */
deluxe_locator_measure *Win_GetDeluxeLocator(void)
{
  deluxe_locator_measure *measure;

  *measure=Gna95gp_get_locator_measure;
  if(Gna95gp_get_locator_measure.button_chord[0]==DOWN){
    hdc=GetDC(hwndMain);
    TextOut(hdc,30,30,"LDOWN",5);
    ReleaseDC(hwndMain,hdc);
  }
  return measure;
}

/* windows get deluxe locator x  coordination*/
int Win_GetDeluxeLocatorX(void)
{
  int temp;

  temp=Gna95gp_get_locator_measure.position.x;
  return temp;
}

/* windows get deluxe locator y coordination */
int Win_GetDeluxeLocatorY(void)
{
  int temp;

  temp=Gna95gp_get_locator_measure.position.y;
  return temp;
}

/* windows set locator measure */
```

```
void Win_SetLocatorMeasure(int x,int y)
{
  win_handleRawEvents(false,false);
  Gna95gp_cur_locator_measure.position.x=x;
  Gna95gp_cur_locator_measure.position.y=y;
}

/* windows get deluxe locator x coordination in smaple mode */
int Win_SampleDeluxeLocatorX(void)
{
  int temp;

  temp=Gna95gp_cur_locator_measure.position.x;
  return temp;
}

/* windows get deluxe locator y coordination in sample mode */
int Win_SampleDeluxeLocatorY(void)
{
  int temp;

  temp=Gna95gp_cur_locator_measure.position.y;
  return temp;
}

/* windows get left button chord */
int Win_GetLButtonChord(void)
{
  int temp;

  temp=Gna95gp_get_locator_measure.button_chord[0];
  return temp;
}

/* windows get middle button chord */
int Win_GetMButtonChord(void)
{
  int temp;

  temp=Gna95gp_get_locator_measure.button_chord[1];
  return temp;
}

/* windows get right button chord */
int Win_GetRButtonChord(void)
{
  int temp;

  temp=Gna95gp_get_locator_measure.button_chord[2];
  return temp;
}

/* windows get shift chord */
int Win_GetShiftChord(void)
{
  int temp;

  temp=Gna95gp_get_locator_measure.modifier_chord[0];
  return temp;
}

/* windows get control chord */
int Win_GetControlChord(void)
{
  int temp;
```

```
  temp=Gna95gp_get_locator_measure.modifier_chord[1];
  return temp;
}

/* windows get alt chord */
int Win_GetMetaChord(void)
{
  int temp;

  temp=Gna95gp_get_locator_measure.modifier_chord[2];
  return temp;
}

/* windows get last transition */
int Win_GetLastTran(void)
{
  int temp;

  temp=Gna95gp_get_locator_measure.button_of_last_transition;
  return temp;
}

/* windows get left button chord in sample mode */
int Win_SampleLButtonChord(void)
{
  int temp;

  temp=Gna95gp_cur_locator_measure.button_chord[0];
  return temp;
}

/* windows get middle button chord in sample mode */
int Win_SampleMButtonChord(void)
{
  int temp;

  temp=Gna95gp_cur_locator_measure.button_chord[1];
  return temp;
}

/* windows get right button chord in sample mode */
int Win_SampleRButtonChord(void)
{
  int temp;

  temp=Gna95gp_cur_locator_measure.button_chord[2];
  return temp;
}

/* windows get shift chord in sample mode */
int Win_SampleShiftChord(void)
{
  int temp;

  temp=Gna95gp_cur_locator_measure.modifier_chord[0];
  return temp;
}

/* windows get control chord in sample mode */
int Win_SampleControlChord(void)
{
  int temp;

  temp=Gna95gp_cur_locator_measure.modifier_chord[1];
```

```
  return temp;
}

/* windows get alt chord in sample mode */
int Win_SampleMetaChord(void)
{
  int temp;

  temp=Gna95gp_cur_locator_measure.modifier_chord[2];
  return temp;
}

/* windows get last transition */
int Win_SampleLastTran(void)
{
  int temp;

  temp=Gna95gp_cur_locator_measure.button_of_last_transition;
  return temp;
}

/* windows set locator button mask */
void Win_SetLocatorButtonMask(int value)
{
  Gna95gp_cur_locator_button_mask^=value;
}

/* windows handle keyboard raw mode in event mode */
void HandleRawModeKeyEvent(WORD wParam,LONG lParam)
{
  char far *src,*dest;

  Gna95gp_cur_keyboard_measure.buffer[0]=(char)wParam;
  Gna95gp_cur_keyboard_measure.buffer[1]='\0';

  if(GetKeyState(VK_SHIFT)<0)
    Gna95gp_cur_keyboard_measure.modifier_chord[SHIFT]=DOWN;
  else
    Gna95gp_cur_keyboard_measure.modifier_chord[SHIFT]=UP;

  if(GetKeyState(VK_CONTROL)<0)
    Gna95gp_cur_keyboard_measure.modifier_chord[CONTROL]=DOWN;
  else
    Gna95gp_cur_keyboard_measure.modifier_chord[CONTROL]=UP;

  if(GetKeyState(VK_MENU)<0)
    Gna95gp_cur_keyboard_measure.modifier_chord[META]=DOWN;
  else
    Gna95gp_cur_keyboard_measure.modifier_chord[META]=UP;

  dest=Gna95gp_get_keyboard_measure.buffer;
  src =Gna95gp_cur_keyboard_measure.buffer;

strcpy(Gna95gp_get_keyboard_measure.buffer,Gna95gp_cur_keyboard_measure.buffer);

  memcpy(Gna95gp_get_keyboard_measure.modifier_chord,
    Gna95gp_cur_keyboard_measure.modifier_chord,
    sizeof(Gna95gp_get_keyboard_measure.modifier_chord));

  Gna95gp_get_keyboard_measure.position=
    Gna95gp_cur_keyboard_measure.position;
}

/* windows handle procedure for keyboard event mode */
void HandleProcModeKeyEvent(WORD wParam,LONG lParam)
```

```
{

  switch(wParam){
    case VK_RETURN:
      if(Gna95gp_cur_mode[KEYBOARD]!=EVENT){
        Gna95gp_cur_keyboard_measure.buffer[0]='\0';
        Gna95gp_cur_keyboard_measure_length=0;
      }
      break;
    case VK_BACK:
      if(Gna95gp_cur_keyboard_measure_length>0){
        Gna95gp_cur_keyboard_measure_length-=1;
        Gna95gp_cur_keyboard_measure.buffer
          [Gna95gp_cur_keyboard_measure_length]='\0';
      }
      break;
    default:
      if((isprint(wParam))&&
        (Gna95gp_cur_keyboard_measure_length<MAX_STRING_SIZE)){
        Gna95gp_cur_keyboard_measure.buffer
          [Gna95gp_cur_keyboard_measure_length]=(char)wParam;
        Gna95gp_cur_keyboard_measure_length++;
        Gna95gp_cur_keyboard_measure.buffer
          [Gna95gp_cur_keyboard_measure_length]='\0';
      }
      break;
  }
}

/* windows set keyboard processign mode */
void Win_SetKeyboardProcessingMode(int value)
{

  win_handleRawEvents(false,false);

  if(Gna95gp_cur_mode[KEYBOARD]!=INACTIVE)
    Gna95gp_deactiveDevice(KEYBOARD);

  Gna95gp_cur_keyboard_processing_mode=value;

  if(Gna95gp_cur_mode[KEYBOARD]!=INACTIVE)
    Gna95gp_activeDevice(KEYBOARD);

}

/* windows get keyboard measure */
void Win_GetKeyboard(char *buff)
{
  strcpy(buff,Gna95gp_get_keyboard_measure.buffer);
}
/* windows get shift key */
int Win_GetShiftKey(void)
{
  int temp;

  temp=Gna95gp_get_keyboard_measure.modifier_chord[0];
  return temp;
}

/* windows get control key */
int Win_GetControlKey(void)
{
  int temp;

  temp=Gna95gp_get_keyboard_measure.modifier_chord[1];
```

```
  return temp;
}

/* windows get alt key */
int Win_GetAltKey(void)
{
  int temp;

  temp=Gna95gp_get_keyboard_measure.modifier_chord[2];
  return temp;
}

/* windows get keyboard x coordination */
int Win_GetKeyboardX(void)
{
  int temp;

  temp=Gna95gp_get_keyboard_measure.position.x;
  return temp;
}

/* windows get keyboard y coordination */
int Win_GetKeyboardY(void)
{
  int temp;

  temp=Gna95gp_get_keyboard_measure.position.y;
  return temp;
}

/* windows set keyboard measure */
void Win_SetKeyboardMeasure(char *str)
{
  win_handleRawEvents(false,false);
  strcpy(Gna95gp_cur_keyboard_measure.buffer,str);
  Gna95gp_cur_keyboard_measure_length=strlen(str);
}

/* windows get keyboard measure in sample mode */
void Win_SampleKeyboard(char *buff)
{
  strcpy(buff,Gna95gp_cur_keyboard_measure.buffer);
}

/* windows get shift key in sample mode */
int Win_SampleShiftKey(void)
{
  int temp;

  temp=Gna95gp_cur_keyboard_measure.modifier_chord[0];
  return temp;
}

/* windows get control key in sample mode */
int Win_SampleControlKey(void)
{
  int temp;

  temp=Gna95gp_cur_keyboard_measure.modifier_chord[1];
  return temp;
}

int Win_SampleAltKey(void)
{
  int temp;
```

```
  temp=Gna95gp_cur_keyboard_measure.modifier_chord[2];
  return temp;
}

/* windows get keyboard x coordination in sample mode */
int Win_SampleKeyboardX(void)
{
  int temp;

  temp=Gna95gp_cur_keyboard_measure.position.x;
  return temp;
}

/* windows get keyboard y coordination in sample mode */
int Win_SampleKeyboardY(void)
{
  int temp;

  temp=Gna95gp_cur_keyboard_measure.position.y;
  return temp;
}

/* windows end */
void Win_End(void)
{
  PostQuitMessage(0);
}
```

```
/*****************************************************************/
/*                  Windows (C) Functions For Gna95gp I/O        */
/*****************************************************************/
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "win_head.h"

/* globle variables */
extern HANDLE Gna95gp_hInstance;
extern int Gna95gp_nCmdShow;
extern char *szAppName;
extern HWND hwndMain; /*hwnd*/
extern HDC hdc;
extern LONG lParam;
extern LONG plParam;
extern UINT message;
extern int Gna95gp_cur_mode[3];
extern int Gna95gp_device_at_head_of_queue;
extern deluxe_locator_measure Gna95gp_cur_locator_measure;
extern deluxe_locator_measure Gna95gp_get_locator_measure;
extern deluxe_keyboard_measure Gna95gp_cur_keyboard_measure;
extern deluxe_keyboard_measure Gna95gp_get_keyboard_measure;
extern int Gna95gp_cur_locator_button_mask;
extern int Gna95gp_cur_keyboard_measure_length;
extern int Gna95gp_cur_keyboard_processing_mode;
extern int Gna95gp_get_timestamp;
extern int currentDevice;
```

```
/* Output a string on the window screen pointed at (x,y) */
void Win_TextCoord(int x,int y,char *text,int textSize)
{
  hdc=GetDC(hwndMain);
  TextOut(hdc,x,y,text,textSize);
  ReleaseDC(hwndMain,hdc);
}

/* Put a integer number into a string */
void Win_Num_To_String(char *str,int num)
{
  sprintf(str,"%d",num);
}

/* Put a float number into a string */
void Win_Flo_To_String(char *str,float num)
{
  sprintf(str,"%f",num);
}

/* Open a file to read */
int Win_Fopen_Read(LPSTR filename,int size)
{
  filename[size]='\0';
  return _lopen(filename,READ);
}

/* Read a string which length is given to a file */
int Win_File_Read(int hRfile,char *str,int num)
{
  return _lread(hRfile,str,num);
}

/* Close a file */
void Win_Fclose(int hRfile)
{
  _lclose(hRfile);
}

/* Open a file to write something to it */
int Win_Fopen_Write(char *filename,int size)
{
  int temp;

  filename[size]='\0';
  temp=_lopen(filename,WRITE);
  if(temp==-1)
    temp=_lcreat(filename,0);
  _llseek(temp,0L,2);
  return temp;
}

/* Write a string to a file */
int Win_File_Write(int hWfile,char *str,int num)
{
  str[num]='\n';
  str[num+1]='\0';
  return _lwrite(hWfile,str,num+1);
}
```

```
--------------------------------------------------------------------------------
--        Data Structure And Declaretation For Event Handling In Ada Language  --
--------------------------------------------------------------------------------
with WinType;
use WinType;
package devices is

  pragma suppress(All_Checks);

  subtype XCoord is integer range 0..1024;
  subtype YCoord is integer range 0..800;
  type point is
   record
     x: XCoord;
     y: YCoord;
   end record;

  type inputDevice is (NO_DEVICE,LOCATOR,KEYBOARD);
  type inputMode is (INACTIVE,SAMPLE,EVENT);

  type buttonStatus is (UP,DOWN);
  type buttonChord is array (integer range<>) of buttonStatus;

  type buttonMask is
(LEFT_BUTTON_MASK,MIDDLE_BUTTON_MASK,RIGHT_BUTTON_MASK);

  type locator_measure is
   record
     position: point;
     button_chord: buttonChord (1..3);
     button_of_last_transition: integer;
   end record;

  type Gna95gp_timestamp is
   record
     seconds: integer;
     ticks: integer;
   end record;

  type deluxe_locator_measure is
   record
     position: point;
     button_chord: buttonChord (1..3);
     button_of_last_transition: integer;
     modifier_chord: buttonChord (1..3);
     timestamp: Gna95gp_timestamp;
   end record;

  type charPtr is access String;
  type deluxe_keyboard_measure is
   record
     buffer: String(1..30);
     buffer_length: integer;
     position: point;
     modifier_chord: buttonChord (1..3);
     timestamp: Gna95gp_timestamp;
   end record;

  function Gna95gp_setPoint(x:integer;y:integer) return point;

  procedure Gna95gp_setInputMode(device: inputDevice;
                                 mode: inputMode);

  procedure Gna95gp_setLocatorButtonMask(mask: buttonMask);
```

```
function Gna95gp_getDeluxeLocatorX return integer;
function Gna95gp_getDeluxeLocatorY return integer;

type keyboardMode is (EDIT,RAW);
procedure Gna95gp_setKeyboardProcessingMode(mode: keyboardMode);
procedure Gna95gp_setLocatorMeasure(value: point);
procedure Gna95gp_setKeyboardMeasure(str : String);

type locator_measure_Ptr is access locator_measure;
type deluxe_locator_measure_Ptr is access deluxe_locator_measure;
type deluxe_keyboard_measure_Ptr is access deluxe_keyboard_measure;

function Gna95gp_sampleDeluxeLocatorX return integer;
function Gna95gp_sampleDeluxeLocatorY return integer;
function Gna95gp_sampleLButtonChord return buttonStatus;
function Gna95gp_sampleMButtonChord return buttonStatus;
function Gna95gp_sampleRButtonChord return buttonStatus;
function Gna95gp_sampleShiftChord return buttonStatus;
function Gna95gp_sampleControlChord return buttonStatus;
function Gna95gp_sampleMetaChord return buttonStatus;
function Gna95gp_sampleLastTran return integer;
function Gna95gp_sampleLocator return locator_measure_Ptr;
function Gna95gp_sampleDeluxeLocator return deluxe_locator_measure_Ptr;

procedure Gna95gp_sampleKeyboard(buffer:String);
function Gna95gp_sampleShiftKey return buttonStatus;
function Gna95gp_sampleControlKey return buttonStatus;
function Gna95gp_sampleAltKey return buttonStatus;
function Gna95gp_sampleKeyboardX return integer;
function Gna95gp_sampleKeyboardY return integer;
function Gna95gp_sampleDeluxeKeyboard return deluxe_keyboard_measure_Ptr;

function Gna95gp_waitEvent(max_wait_time: integer) return inputDevice;
function Gna95gp_getLocator return locator_measure_Ptr;

procedure Gna95gp_getKeyboard (buffer:String);

function Gna95gp_getShiftKey return buttonStatus;
function Gna95gp_getControlKey return buttonStatus;
function Gna95gp_getAltKey return buttonStatus;
function Gna95gp_getKeyboardX return integer;
function Gna95gp_getKeyboardY return integer;
function Gna95gp_getDeluxeKeyboard return deluxe_keyboard_measure_Ptr;

function Gna95gp_getLButtonChord return buttonStatus;
function Gna95gp_getMButtonChord return buttonStatus;
function Gna95gp_getRButtonChord return buttonStatus;
function Gna95gp_getShiftChord return buttonStatus;
function Gna95gp_getControlChord return buttonStatus;
function Gna95gp_getMetaChord return buttonStatus;
function Gna95gp_getLastTran return integer;
function Gna95gp_getDeluxeLocator return deluxe_locator_measure_Ptr;

procedure Gna95gp_CreatWindow(name:String;width:integer;
                    heigth:integer);

procedure Gna95gp_line(x1:integer;y1:integer;
                x2:integer;y2:integer);
procedure Gna95gp_drawmark(x:integer;y:integer);
function Gna95gp_Getmessage return HANDLE;
function Gna95gp_getx return integer;
function Gna95gp_gety return integer;
function Gna95gp_getpx return integer;
function Gna95gp_getpy return integer;
procedure Gna95gp_end;
```

```
end devices;
```

```
-------------------------------------------------------------------------------
--          The Device Body For Event Handling In Ada Language          --
-------------------------------------------------------------------------------
with WinType;
use WinType;

package body devices is

pragma suppress (All_Checks);

  -- windows begin procedure
  procedure Win_begin(name:String;width:integer;heigth:integer;
                      size:integer;callnum:integer);
  pragma import (C,Win_begin,"win_begin"); -- binding with C (windows) language

  -- create a window
  procedure Gna95gp_CreatWindow(name:String;width:integer;
                                heigth:integer) is
    size:integer;
  begin
    size:=name'Last-name'First+1;
    Win_begin(name,width,heigth,size,1);
  end Gna95gp_CreatWindow;

  -- set a point function
  function Gna95gp_setPoint(x:integer;y:integer) return point is
    p:point;
  begin
    p.x:=x;
    p.y:=y;
    return p;
  end Gna95gp_setPoint;

  -- draw line
  procedure win_drawline(x1:integer;y1:integer;
                         x2:integer;y2:integer);
  pragma import (C,win_drawline,"Win_Drawline");

  -- draw line in Gna95gp
  procedure Gna95gp_line(x1:integer;y1:integer;
                         x2:integer;y2:integer) is
  begin
    win_drawline(x1,y1,x2,y2);
  end Gna95gp_line;

  -- draw mark
  procedure win_drawmark (x:integer;y:integer);
  pragma import (C,win_drawmark,"Win_Drawmark");

  -- draw in Gna95gp
  procedure Gna95gp_drawmark (x:integer;y:integer) is
  begin
    win_drawmark(x,y);
  end Gna95gp_drawmark;

  -- get a message from windows
  function Gna95gp_Getmessage return HANDLE is
```

```
   message: HANDLE;
 begin
   pragma import (C,message,"message");
   return message;
 end Gna95gp_Getmessage;

-- get a x coordination
 function win_xc return integer;
 pragma import (C,win_xc,"Win_Xc");

 -- get x coordination in Gna95gp
 function Gna95gp_getx return integer is
   X:integer;
 begin
   X := win_xc;
   return X;
 end Gna95gp_getx;

 -- get y coordiantion
 function win_yc return integer;
 pragma import (C,win_yc,"Win_Yc");

-- get y coordination in Gna95gp
 function Gna95gp_gety return integer is
   Y:integer;
 begin
   Y := win_yc;
   return Y;
 end Gna95gp_gety;

-- set input mode
 procedure win_setInputMode(device: integer;
                            mode:   integer;
                            callnum:integer);
 pragma import (C,win_setInputMode,"Win_SetInputMode");

 -- set input mode in Gna95gp
 procedure Gna95gp_setInputMode(device: inputDevice;
                                mode: inputMode) is
 begin
  if device=LOCATOR then
    if mode=INACTIVE then
      win_setInputMode(1,0,1);
    elsif mode=SAMPLE then
      win_setInputMode(1,1,2);
    elsif mode=EVENT then
      win_setInputMode(1,2,3);
    end if;
  elsif device=KEYBOARD then
    if mode=INACTIVE then
      win_setInputMode(2,0,4);
    elsif mode=SAMPLE then
      win_setInputMode(2,1,5);
    elsif mode=EVENT then
      win_setInputMode(2,2,6);
    end if;
  end if;
 end Gna95gp_setInputMode;

 -- wait an event
 function win_waitEvent(maxtime: integer) return integer;
 pragma import (C,win_waitEvent,"Win_WaitEvent");

 -- wait an event in Gna95gp
 function Gna95gp_waitEvent(max_wait_time: integer)
```

```
                    return inputDevice is
  device: inputDevice;
  temp  : integer;

begin

  temp:=win_waitEvent(max_wait_time);

  if temp=0 then
    device := NO_DEVICE;
  elsif temp=1 then
    device := LOCATOR;
  elsif temp=2 then
    device := KEYBOARD;
  end if;

  return device;

end Gna95gp_waitEvent;

-- get deluxe locator x coordination
function win_getDeluxeLocatorX return integer;
pragma import(C,win_getDeluxeLocatorX,"Win_GetDeluxeLocatorX");

-- get deluxe locator x coordination in Gna95gp
function Gna95gp_getDeluxeLocatorX return integer is
  x: integer;
begin
  x:=win_getDeluxeLocatorX;
  return x;
end Gna95gp_getDeluxeLocatorX;

-- get deluxe locator y coordination
function win_getDeluxeLocatorY return integer;
pragma import(C,win_getDeluxeLocatorY,"Win_GetDeluxeLocatorY");

-- get deluxe locator y coordination in Gna95gp
function Gna95gp_getDeluxeLocatorY return integer is
  y: integer;
begin
  y:=win_getDeluxeLocatorY;
  return y;
end Gna95gp_getDeluxeLocatorY;

-- set locator measure
procedure win_setLocatorMeasure(x:integer;y:integer);
pragma import(C,win_setLocatorMeasure,"Win_SetLocatorMeasure");

-- set locator measure in Gna95gp
procedure Gna95gp_setLocatorMeasure(value: point) is
begin
  win_setLocatorMeasure(value.x,value.y);
end Gna95gp_setLocatorMeasure;

-- get left button chord
function win_getLButtonChord return integer;
pragma import (C,win_getLButtonChord,"Win_GetLButtonChord");

-- get left button chord in Gna95gp
function Gna95gp_getLButtonChord return buttonStatus is
  lbutton_chord: integer;
  lstatus:buttonStatus;
begin
  lbutton_chord:=win_getLButtonChord;
  if(lbutton_chord=0) then
```

```
    lstatus:=UP;
  end if;
  if(lbutton_chord=1) then
    lstatus:=DOWN;
  end if;
  return lstatus;
end Gna95gp_getLButtonChord;

-- get middle button chord
function win_getMButtonChord return integer;
pragma import (C,win_getMButtonChord,"Win_GetMButtonChord");

-- get middle button chord in Gna95gp
function Gna95gp_getMButtonChord return buttonStatus is
  mbutton_chord: integer;
  mstatus:buttonStatus;
begin
  mbutton_chord:=win_getMButtonChord;
  if(mbutton_chord=0) then
    mstatus:=UP;
  end if;
  if(mbutton_chord=1) then
    mstatus:=DOWN;
  end if;
  return mstatus;
end Gna95gp_getMButtonChord;

-- get right button chord
function win_getRButtonChord return integer;
pragma import (C,win_getRButtonChord,"Win_GetRButtonChord");

-- get right button chord in Gna95gp
function Gna95gp_getRButtonChord return buttonStatus is
  rbutton_chord: integer;
  rstatus:buttonStatus;
begin
  rbutton_chord:=win_getRButtonChord;
  if(rbutton_chord=0) then
    rstatus:=UP;
  end if;
  if(rbutton_chord=1) then
    rstatus:=DOWN;
  end if;
  return rstatus;
end Gna95gp_getRButtonChord;

-- get shift chord
function win_getShiftChord return integer;
pragma import (C,win_getShiftChord,"Win_GetShiftChord");

-- get shift key in Gna95gp
function Gna95gp_getShiftChord return buttonStatus is
  shift_chord: integer;
  shift_status:buttonStatus;
begin
  shift_chord:=win_getShiftChord;
  if(shift_chord=0) then
    shift_status:=UP;
  end if;
  if(shift_chord=1) then
    shift_status:=DOWN;
  end if;
  return shift_status;
end Gna95gp_getShiftChord;
```

```
-- get control chord
function win_getControlChord return integer;
pragma import (C,win_getControlChord,"Win_GetControlChord");

-- get control key chord in Gna95gp
function Gna95gp_getControlChord return buttonStatus is
 control_chord: integer;
 control_status:buttonStatus;
begin
 control_chord:=win_getControlChord;
 if(control_chord=0) then
  control_status:=UP;
 end if;
 if(control_chord=1) then
  control_status:=DOWN;
 end if;
 return control_status;
end Gna95gp_getControlChord;
-- get alt key chord
function win_getMetaChord return integer;
pragma import (C,win_getMetaChord,"Win_GetMetaChord");

-- get alt key chord in Gna95gp
function Gna95gp_getMetaChord return buttonStatus is
 meta_chord: integer;
 meta_status:buttonStatus;
begin
 meta_chord:=win_getMetaChord;
 if(meta_chord=0) then
  meta_status:=UP;
 end if;
 if(meta_chord=1) then
  meta_status:=DOWN;
 end if;
 return meta_status;
end Gna95gp_getMetaChord;

-- get last transition
function win_getLastTran return integer;
pragma import (C,win_getLastTran,"Win_GetLastTran");

-- get last transition in Gna95gp
function Gna95gp_getLastTran return integer is
 last_tran: integer;
begin
 last_tran:=win_getLastTran;
 return last_tran;
end Gna95gp_getLastTran;

-- get locator measure in Gna95gp
function Gna95gp_getLocator return locator_measure_Ptr is
 measure: locator_measure_Ptr;
begin

 measure.position.x:=Gna95gp_getDeluxeLocatorX;
 measure.position.y:=Gna95gp_getDeluxeLocatorY;
 measure.button_chord(1):=Gna95gp_getLButtonChord;
 measure.button_chord(2):=Gna95gp_getMButtonChord;
 measure.button_chord(3):=Gna95gp_getRButtonChord;
 measure.button_of_last_transition:=Gna95gp_getLastTran;
 return measure;

end Gna95gp_getLocator;
-- get deluxe locator measure in Gna95gp
function Gna95gp_getDeluxeLocator return deluxe_locator_measure_Ptr is
```

```
  measure: deluxe_locator_measure_Ptr;
begin
  measure.position.x:=Gna95gp_getDeluxeLocatorX;
  measure.position.y:=Gna95gp_getDeluxeLocatorY;
  measure.button_chord(1):=Gna95gp_getLButtonChord;
  measure.button_chord(2):=Gna95gp_getMButtonChord;
  measure.button_chord(3):=Gna95gp_getRButtonChord;
  measure.modifier_chord(1):=Gna95gp_getShiftChord;
  measure.modifier_chord(2):=Gna95gp_getControlChord;
  measure.modifier_chord(3):=Gna95gp_getMetaChord;
  measure.button_of_last_transition:=Gna95gp_getLastTran;
  return measure;
end Gna95gp_getDeluxeLocator;

-- get deluxe locator x coordination
function win_sampleDeluxeLocatorX return integer;
pragma import(C,win_sampleDeluxeLocatorX,"Win_SampleDeluxeLocatorX");

 -- get deluxe locator x coordination in Gna95gp
function Gna95gp_sampleDeluxeLocatorX return integer is
  x: integer;
begin
  x:=win_sampleDeluxeLocatorX;
  return x;
end Gna95gp_sampleDeluxeLocatorX;

 -- get deluxe locator y coordination
function win_sampleDeluxeLocatorY return integer;
pragma import(C,win_sampleDeluxeLocatorY,"Win_SampleDeluxeLocatorY");

 -- get deluxe locator y coordination in Gna95gp
function Gna95gp_sampleDeluxeLocatorY return integer is
  y: integer;
begin
  y:=win_sampleDeluxeLocatorY;
  return y;
end Gna95gp_sampleDeluxeLocatorY;

-- get left button chord in sample mode
function win_sampleLButtonChord return integer;
pragma import (C,win_sampleLButtonChord,"Win_SampleLButtonChord");

-- get left button chord in sample mode in Gna95gp
function Gna95gp_sampleLButtonChord return buttonStatus is
  lbutton_chord: integer;
  lstatus:buttonStatus;
begin
  lbutton_chord:=win_sampleLButtonChord;
  if(lbutton_chord=0) then
    lstatus:=UP;
  end if;
  if(lbutton_chord=1) then
    lstatus:=DOWN;
  end if;
  return lstatus;
end Gna95gp_sampleLButtonChord;

-- get middle button chord in sample mode
function win_sampleMButtonChord return integer;
pragma import (C,win_sampleMButtonChord,"Win_SampleMButtonChord");

-- get middle button chord in sample mode in Gna95gp
function Gna95gp_sampleMButtonChord return buttonStatus is
  mbutton_chord: integer;
  mstatus:buttonStatus;
```

```
begin
  mbutton_chord:=win_sampleMButtonChord;
  if(mbutton_chord=0) then
    mstatus:=UP;
  end if;
  if(mbutton_chord=1) then
    mstatus:=DOWN;
  end if;
  return mstatus;
end Gna95gp_sampleMButtonChord;

-- get right button chord in sample mode
function win_sampleRButtonChord return integer;
pragma import (C,win_sampleRButtonChord,"Win_SampleRButtonChord");

-- get right button chord in sample mode in Gna95gp
function Gna95gp_sampleRButtonChord return buttonStatus is
  rbutton_chord: integer;
  rstatus:buttonStatus;
begin
  rbutton_chord:=win_sampleRButtonChord;
  if(rbutton_chord=0) then
    rstatus:=UP;
  end if;
  if(rbutton_chord=1) then
    rstatus:=DOWN;
  end if;
  return rstatus;
end Gna95gp_sampleRButtonChord;

-- get sample shift chord
function win_sampleShiftChord return integer;
pragma import (C,win_sampleShiftChord,"Win_SampleShiftChord");

-- get sample shift chord in Gna95gp
function Gna95gp_sampleShiftChord return buttonStatus is
  shift_chord: integer;
  shift_status:buttonStatus;
begin
  shift_chord:=win_sampleShiftChord;
  if(shift_chord=0) then
    shift_status:=UP;
  end if;
  if(shift_chord=1) then
    shift_status:=DOWN;
  end if;
  return shift_status;
end Gna95gp_sampleShiftChord;

-- get sample control chord
function win_sampleControlChord return integer;
pragma import (C,win_sampleControlChord,"Win_SampleControlChord");

-- get sample control chord in Gna95gp
function Gna95gp_sampleControlChord return buttonStatus is
  control_chord: integer;
  control_status:buttonStatus;
begin
  control_chord:=win_sampleControlChord;
  if(control_chord=0) then
    control_status:=UP;
  end if;
  if(control_chord=1) then
    control_status:=DOWN;
  end if;
```

```
  return control_status;
end Gna95gp_sampleControlChord;

-- get sample alt chord
function win_sampleMetaChord return integer;
pragma import (C,win_sampleMetaChord,"Win_SampleMetaChord");

-- get sample alt chord in Gna95gp
function Gna95gp_sampleMetaChord return buttonStatus is
  meta_chord: integer;
  meta_status:buttonStatus;
begin
  meta_chord:=win_sampleMetaChord;
  if(meta_chord=0) then
    meta_status:=UP;
  end if;
  if(meta_chord=1) then
    meta_status:=DOWN;
  end if;
  return meta_status;
end Gna95gp_sampleMetaChord;

-- get sample last transition
function win_sampleLastTran return integer;
pragma import (C,win_sampleLastTran,"Win_SampleLastTran");

-- get sample last transition in Gna95gp
function Gna95gp_sampleLastTran return integer is
  last_tran: integer;
begin
  last_tran:=win_sampleLastTran;
  return last_tran;
end Gna95gp_sampleLastTran;

-- get sample locator measure in Gna95gp
function Gna95gp_sampleLocator return locator_measure_Ptr is
  measure: locator_measure_Ptr;
begin

  measure.position.x:=Gna95gp_sampleDeluxeLocatorX;
  measure.position.y:=Gna95gp_sampleDeluxeLocatorY;
  measure.button_chord(1):=Gna95gp_sampleLButtonChord;
  measure.button_chord(2):=Gna95gp_sampleMButtonChord;
  measure.button_chord(3):=Gna95gp_sampleRButtonChord;
  measure.button_of_last_transition:=Gna95gp_sampleLastTran;
  return measure;

end Gna95gp_sampleLocator;
-- get sample deluxe locator measure in Gna95gp
function Gna95gp_sampleDeluxeLocator return deluxe_locator_measure_Ptr is
  measure: deluxe_locator_measure_Ptr;
begin
  measure.position.x:=Gna95gp_sampleDeluxeLocatorX;
  measure.position.y:=Gna95gp_sampleDeluxeLocatorY;
  measure.button_chord(1):=Gna95gp_sampleLButtonChord;
  measure.button_chord(2):=Gna95gp_sampleMButtonChord;
  measure.button_chord(3):=Gna95gp_sampleRButtonChord;
  measure.modifier_chord(1):=Gna95gp_sampleShiftChord;
  measure.modifier_chord(2):=Gna95gp_sampleControlChord;
  measure.modifier_chord(3):=Gna95gp_sampleMetaChord;
  measure.button_of_last_transition:=Gna95gp_sampleLastTran;
  return measure;
end Gna95gp_sampleDeluxeLocator;

-- set locator button mask
```

```
procedure win_setLocatorButtonMask (value: integer);
pragma import (C,win_setLocatorButtonMask,"Win_SetLocatorButtonMask");

-- set locator button mask in Gna95gp
procedure Gna95gp_setLocatorButtonMask (mask: buttonMask) is
begin

  if(mask=LEFT_BUTTON_MASK) then
    win_setLocatorButtonMask(1);
  end if;

  if(mask=MIDDLE_BUTTON_MASK) then
    win_setLocatorButtonMask(2);
  end if;

  if(mask=RIGHT_BUTTON_MASK) then
    win_setLocatorButtonMask(4);
  end if;

end Gna95gp_setLocatorButtonMask;

-- set keyboard processing mode
procedure win_setKeyboardProcessingMode(value:integer);
pragma import (C,win_setKeyboardProcessingMode,
              "Win_SetKeyboardProcessingMode");

-- set keyboard processing mode in Gna95gp
procedure Gna95gp_setKeyboardProcessingMode(mode: keyboardMode) is
begin

  if(mode=RAW) then
    win_setKeyboardProcessingMode(1);
  end if;

  if(mode=EDIT) then
    win_setKeyboardProcessingMode(0);
  end if;

end Gna95gp_setKeyboardProcessingMode;

-- get keyboard measure
procedure win_getKeyboard(buffer:String);
pragma import (C,win_getKeyboard,"Win_GetKeyboard");

-- get keyboard measure in Gna95gp
procedure Gna95gp_getKeyboard (buffer:String) is
begin

  win_getKeyboard(buffer);

end Gna95gp_getKeyboard;

-- get keyboard x coordination
function win_getKeyboardX return integer;
pragma import(C,win_getKeyboardX,"Win_GetKeyboardX");

-- get keyboard x coordination in Gna95gp
function Gna95gp_getKeyboardX return integer is
  x: integer;
begin
  x:=win_getKeyboardX;
  return x;
end Gna95gp_getKeyboardX;

-- get keyboard y coordination
```

```
function win_getKeyboardY return integer;
pragma import(C,win_getKeyboardY,"Win_GetKeyboardY");

-- get keyboard y coordination in Gna95gp
function Gna95gp_getKeyboardY return integer is
  y: integer;
begin
  y:=win_getKeyboardY;
  return y;
end Gna95gp_getKeyboardY;

-- set keyboard measure
procedure win_setKeyboardMeasure(str : String);
pragma import(C,win_setKeyboardMeasure,"Win_SetKeyboardMeasure");

-- set keyboard measure in Gna95gp
procedure Gna95gp_setKeyboardMeasure(str : String) is
begin
  win_setKeyboardMeasure(str);
end Gna95gp_setKeyboardMeasure;

-- get shift key
function win_getShiftKey return integer;
pragma import (C,win_getShiftKey,"Win_GetShiftKey");

-- get shift key in Gna95gp
function Gna95gp_getShiftKey return buttonStatus is
  shift_chord: integer;
  shift_status:buttonStatus;
begin
  shift_chord:=win_getShiftKey;
  if(shift_chord=0) then
    shift_status:=UP;
  end if;
  if(shift_chord=1) then
    shift_status:=DOWN;
  end if;
  return shift_status;
end Gna95gp_getShiftKey;

-- get control key
function win_getControlKey return integer;
pragma import (C,win_getControlKey,"Win_GetControlKey");

-- get control key in Gna95gp
function Gna95gp_getControlKey return buttonStatus is
  control_chord: integer;
  control_status:buttonStatus;
begin
  control_chord:=win_getControlKey;
  if(control_chord=0) then
    control_status:=UP;
  end if;
  if(control_chord=1) then
    control_status:=DOWN;
  end if;
  return control_status;
end Gna95gp_getControlKey;

-- get alt key
function win_getAltKey return integer;
pragma import (C,win_getAltKey,"Win_GetAltKey");

-- get alt key in Gna95gp
function Gna95gp_getAltKey return buttonStatus is
```

```
  alt_chord: integer;
  alt_status:buttonStatus;
begin
  alt_chord:=win_getAltKey;
  if(alt_chord=0) then
    alt_status:=UP;
  end if;
  if(alt_chord=1) then
    alt_status:=DOWN;
  end if;
  return alt_status;
end Gna95gp_getAltKey;

-- get deluxe keyboard measure in Gna95gp
function Gna95gp_getDeluxeKeyboard return deluxe_keyboard_measure_Ptr is
  measure: deluxe_keyboard_measure_Ptr;
begin
  Gna95gp_getKeyboard(measure.buffer);
  measure.position.x:=Gna95gp_getKeyboardX;
  measure.position.y:=Gna95gp_getKeyboardY;
  measure.modifier_chord(1):=Gna95gp_getShiftKey;
  measure.modifier_chord(2):=Gna95gp_getControlKey;
  measure.modifier_chord(3):=Gna95gp_getAltKey;
  return measure;
end Gna95gp_getDeluxeKeyboard;

-- get smaple keyboard measure
procedure win_sampleKeyboard(buffer:String);
pragma import (C,win_sampleKeyboard,"Win_SampleKeyboard");

-- get sample keyboard measuare in Gna95gp
procedure Gna95gp_sampleKeyboard (buffer:String) is
begin

  win_sampleKeyboard(buffer);

end Gna95gp_sampleKeyboard;

-- get sample keyboard x coordination
function win_sampleKeyboardX return integer;
pragma import(C,win_sampleKeyboardX,"Win_SampleKeyboardX");

-- get sample keyboard x coordination in Gna95gp
function Gna95gp_sampleKeyboardX return integer is
  x: integer;
begin
  x:=win_sampleKeyboardX;
  return x;
end Gna95gp_sampleKeyboardX;

-- get sample keyboard y coordination
function win_sampleKeyboardY return integer;
pragma import(C,win_sampleKeyboardY,"Win_SampleKeyboardY");

-- get sample keyboard y coordination in Gna95gp
function Gna95gp_sampleKeyboardY return integer is
  y: integer;
begin
  y:=win_sampleKeyboardY;
  return y;
end Gna95gp_sampleKeyboardY;

-- get sample shift key
function win_sampleShiftKey return integer;
pragma import (C,win_sampleShiftKey,"Win_SampleShiftKey");
```

```
-- get sample shift key in Gna95gp
function Gna95gp_sampleShiftKey return buttonStatus is
  shift_chord: integer;
  shift_status:buttonStatus;
begin
  shift_chord:=win_sampleShiftKey;
  if(shift_chord=0) then
    shift_status:=UP;
  end if;
  if(shift_chord=1) then
    shift_status:=DOWN;
  end if;
  return shift_status;
end Gna95gp_sampleShiftKey;

-- get sample control key
function win_sampleControlKey return integer;
pragma import (C,win_sampleControlKey,"Win_SampleControlKey");

-- get sample control key in Gna95gp
function Gna95gp_sampleControlKey return buttonStatus is
  control_chord: integer;
  control_status:buttonStatus;
begin
  control_chord:=win_sampleControlKey;
  if(control_chord=0) then
    control_status:=UP;
  end if;
  if(control_chord=1) then
    control_status:=DOWN;
  end if;
  return control_status;
end Gna95gp_sampleControlKey;

-- get sample alt key
function win_sampleAltKey return integer;
pragma import (C,win_sampleAltKey,"Win_SampleAltKey");

-- get sample alt key in Gna95gp
function Gna95gp_sampleAltKey return buttonStatus is
  alt_chord: integer;
  alt_status:buttonStatus;
begin
  alt_chord:=win_sampleAltKey;
  if(alt_chord=0) then
    alt_status:=UP;
  end if;
  if(alt_chord=1) then
    alt_status:=DOWN;
  end if;
  return alt_status;
end Gna95gp_sampleAltKey;

-- get sample deluxe keyboard in Gna95gp
function Gna95gp_sampleDeluxeKeyboard return deluxe_keyboard_measure_Ptr is
  measure: deluxe_keyboard_measure_Ptr;
begin
  Gna95gp_sampleKeyboard(measure.buffer);
  measure.position.x:=Gna95gp_sampleKeyboardX;
  measure.position.y:=Gna95gp_sampleKeyboardY;
  measure.modifier_chord(1):=Gna95gp_sampleShiftKey;
  measure.modifier_chord(2):=Gna95gp_sampleControlKey;
  measure.modifier_chord(3):=Gna95gp_sampleAltKey;
  return measure;
```

```
  end Gna95gp_sampleDeluxeKeyboard;

  -- end windows
  procedure win_end;
  pragma import (C,win_end,"Win_End");

  -- end windows in Gna95gp
  procedure Gna95gp_end is
  begin
    win_end;
  end Gna95gp_end;

end devices;
```

```
-------------------------------------------------------------------------------------------------
--          Procedures And Functions  Declarations Of  I/O In Ada Language              --
-------------------------------------------------------------------------------------------------
with WinType;
use WinType;
package io is

  pragma suppress(All_Checks);

  procedure Gna95gp_num_to_string (str: String; num: integer);
  procedure Gna95gp_flo_to_string (str:String;  num: float);
  procedure Gna95gp_char(x:integer;y:integer;text:String;size:integer);
  procedure Gna95gp_textCoord(x:integer;y:integer;text:String);
  function  Gna95gp_fopen_read(fname: String) return integer;
  function  Gna95gp_file_read(hRfile:integer;str:String;num:integer)
                    return integer;
  procedure Gna95gp_fclose(hfile:integer);
  function  Gna95gp_fopen_write(fname: String) return integer;
  function  Gna95gp_file_write(hWfile:integer;str:String;num:integer)
                    return integer;

end io;
```

```
-------------------------------------------------------------------------------------------------
--     This Package Is Used For Input And Output Of Gna95gp In Ada Language   --
-------------------------------------------------------------------------------------------------
with WinType;
use WinType;

package body io is

pragma suppress (All_Checks);

  -- String Output
  procedure win_textcoord(x:integer;y:integer;text:String;
              textSize:integer);
  pragma import (C,win_textcoord,"Win_TextCoord");

  procedure Gna95gp_textCoord (x:integer;y:integer;text:
                    String) is
    size:integer;
```

```
begin
  size := text'Last-text'First+1;
  win_textCoord(x,y,text,size);
end Gna95gp_textCoord;

-- Integer write into a String
procedure win_num_to_string (str:String; num:integer);
pragma import (C,win_num_to_string,"Win_Num_To_String");

procedure Gna95gp_num_to_string (str:String;num:integer) is
begin
  win_num_to_string(str,num);
end Gna95gp_num_to_string;

-- Float write into a String
procedure win_flo_to_string (str:String; num:float);
pragma import (C,win_flo_to_string,"Win_Flo_To_String");

procedure Gna95gp_flo_to_string (str:String;num:float) is
begin
  win_flo_to_string(str,num);
end Gna95gp_flo_to_string;

-- Character output
procedure Gna95gp_char (x:integer;y:integer;text:
                String;size:integer) is
begin
  win_textcoord(x,y,text,size);
end Gna95gp_char;

-- Open a file for read
function win_fopen_read(fname: String;size:integer)
                    return integer;
pragma import (C,win_fopen_read,"Win_Fopen_Read");

function Gna95gp_fopen_read(fname:String) return integer is
  temp:integer;
  size:integer;
begin
  size:=fname'Last-fname'first+1;
  temp:=win_fopen_read(fname,size);
  return temp;
end Gna95gp_fopen_read;

-- Read string from a file
function win_file_read(hRfile:integer;str:String;num:integer)
                return integer;
pragma import (C,win_file_read,"Win_File_Read");

function Gna95gp_file_read(hRfile:integer;str:String;num:integer)
                return integer is
  temp:integer;
begin
  temp:=win_file_read(hRfile,str,num);
  return temp;
end Gna95gp_file_read;

-- File close
procedure win_fclose(hfile:integer);
pragma import (C,win_fclose,"Win_Fclose");

procedure Gna95gp_fclose(hfile:integer) is
begin
  win_fclose(hfile);
end Gna95gp_fclose;
```

```
  -- File open for write string
  function win_fopen_write(fname: String;size:integer)
                  return integer;
  pragma import (C,win_fopen_write,"Win_Fopen_Write");

  function Gna95gp_fopen_write(fname:String) return integer is
    temp:integer;
    size:integer;
  begin
     size:=fname'Last-fname'first+1;
     temp:=win_fopen_write(fname,size);
     return temp;
  end Gna95gp_fopen_write;

  -- Write a string into a file
  function win_file_write(hWfile:integer;str:String;num:integer)
                  return integer;
  pragma import (C,win_file_write,"Win_File_Write");

  function Gna95gp_file_write(hWfile:integer;str:String;num:integer)
                  return integer is
    temp:integer;
  begin
     temp:=win_file_write(hWfile,str,num);
     return temp;
  end Gna95gp_file_write;

end io;
```

VITA

Shan Kuang

Candidate of for the Degree of

Master of Science

Thesis: EVENT HANDLING IN GNA95GP GRAPHICS PACKAGE

Major Field: Computer Science

Biographical:

Personal Data: Born in Shenyang City, Liaoning Province, China October 8, 1963, the son of Zhenggen Kuang and Guilan Zhou.

Education: Graduated from No.2 High School of Jinzhou, Dalian City, Liaoning Province, China in July 1981; received Bachelor of Science degree in Mechanical Engineering from College of Armored Force, Beijing, China in July 1986; received Master of Science degree in Mechanical Engineering from China Mining University, Beijing, China in July 1989. Completed the requirements for the Master of Science degree in Computer Science at Oklahoma State University in May 1997.

Experience: Graduate research assistant, Computer Science Department, Oklahoma State University 1995-1997; Engineer, Department of Environmental Control and Life Support System, Institute of Space Medico-Engineering 1989-1994; Graduate research assistant, Department of Mechanical Engineering, China Mining University 1986-1989.