What is an Event-Driven Application?

This document covers the following topics:

Introduction

Event-driven applications represent a new approach to development in addition to the program-driven approach. Natural offers you both. Event-driven programming allows the application to be driven by input received through the graphical user interface.

In program-driven applications, the application controls the portions of code that execute - not an event. Execution starts with the first line of executable code and follows a defined pathway through the application, calling additional programs as instructed in the predetermined sequence.

In event-driven programming, the user's action or a system event triggers the code attached to that event. Thus, the order in which your code executes depends on which events occur, which in turn depends on what the user does. This is the essence of graphical user interfaces and event-driven programming: The user is in charge, and the code responds. Even though event-driven programming is possible in character-oriented interfaces, it is more common in graphical user interfaces.

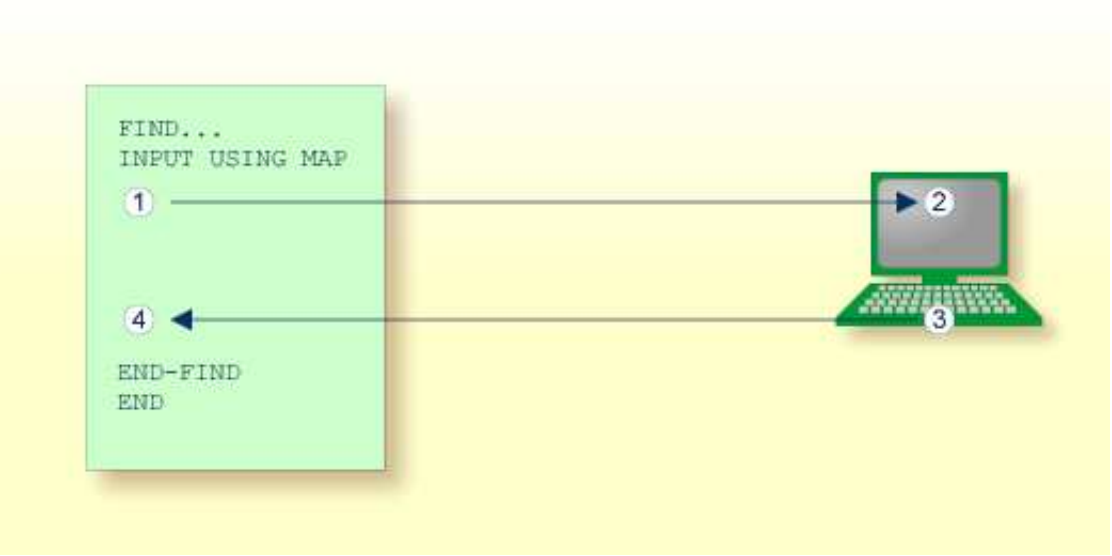
Because you cannot predict what the user will do, your code must make a few assumptions when it executes. For example, the application might assume that the user added text to an edit-area control before pressing the **OK** button.

When you must make assumptions, you should try to structure your application so that these assumptions are always valid. For example, to ensure the user added text, you can disable the button and enable it only when the change event occurs for the edit area control.

Your code can trigger additional events as it performs certain operations. For example, moving the slider in a scroll bar control triggers the change event.

The following diagrams illustrate the difference between program-driven and event-driven applications.

Program-Driven Applications

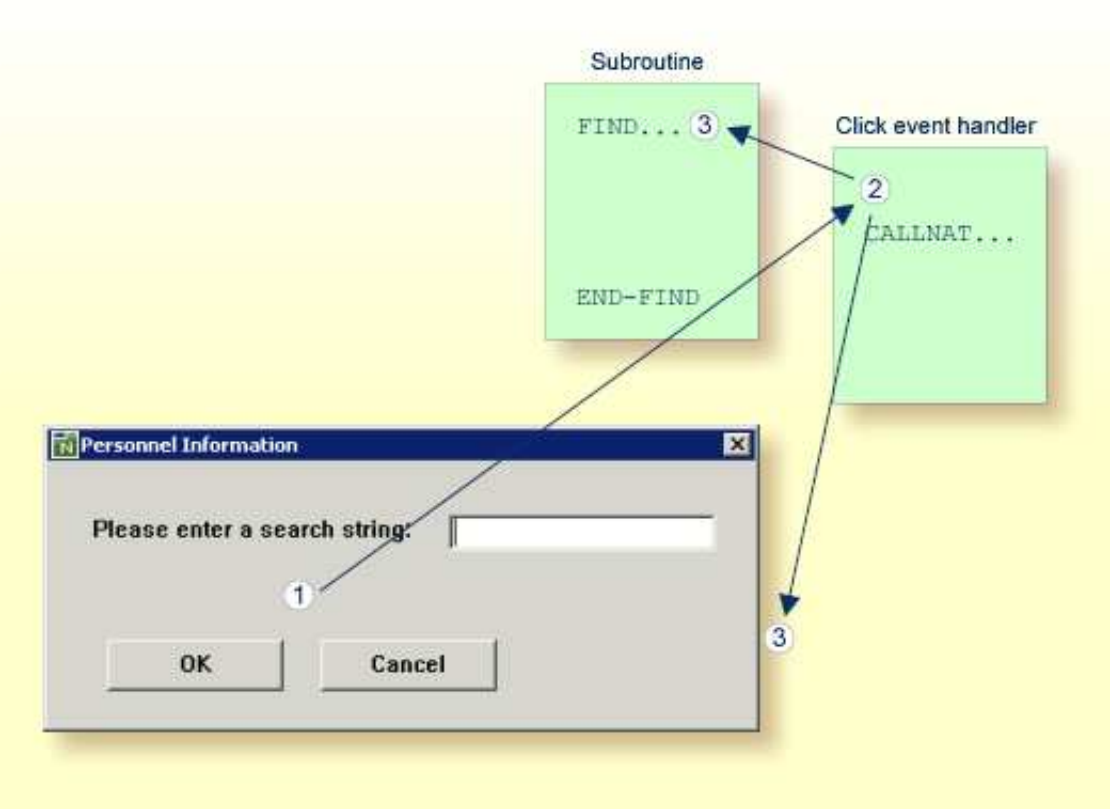


In typical program-driven applications, the following sequence of steps applies:

1. The program sends a screen to the terminal.
2. The user reacts by filling in the data fields.
3. The user then presses ENTER or a function key.
4. The program then decides whether or not the user's entries are valid.

If the data are valid, it processes the results until it reaches an `END` statement.

Event Driven Applications



In typical event-driven applications, the following sequence of steps applies:

1. The user requests an action on the screen.
2. The event handler code reacts in the background according to the context.
3. If certain conditions are fulfilled, the executed event handler code triggers other Natural code (here: a subroutine) or returns control to the screen.

In the program-driven approach, the user interacts with the code through the ENTER and function keys, the user of an event-driven application triggers specific pieces of code (event handlers). Typically, an event-driven application is not executing any code when waiting for user input; in the same situation, the program-driven application might be processing an `INPUT` statement.

What is Happening Here?

Graphical user interface programs require you to write programs that react to isolated events initiated by the user.

An event is an action recognized by a dialog or a dialog element. Event-driven applications execute code in response to an event. Each dialog or dialog element has a predefined set of events. If one of these events occurs, Natural invokes the code in the associated event handler.

You decide if and how the dialogs and dialog elements in your application respond to a particular event. When you want a program to respond to an event, you write event code for that event.

Writing Event-Driven Code

For each dialog or dialog element you create, Natural predefines a set of events to which your program (event handler) can respond. It is easy to respond to events: dialogs and dialog elements have the built-in ability to recognize user actions and execute the code associated with them.

You do not have to write code for all events. When you do want a dialog object to respond to an event, you write event code that Natural executes in response to that event.

In a typical event-driven application, the following series of actions takes place:

- A dialog or dialog element recognizes an action as an event. The action can be caused by the user (such as a click or keystroke).
- If there is event code corresponding to the event, it is executed.
- The application waits for the next event.

The event code you write to respond to events can perform calculations, get input, and manipulate parts of the interface. Using Natural, you manipulate dialogs or dialog elements by changing the values of their attribute settings.

Vorsicht:
Avoid creating cascading events in your code caused by events occurring repeatedly. For example, when the user drags the slider in the scroll-bar control, the current `SLIDER` attribute setting is automatically changed and the change event is triggered. If the code attached to the change event also changes the current `SLIDER` attribute setting, then the change event is triggered again, the current `SLIDER` attribute setting is again adjusted, the change event is once again triggered, and so on. At this rate, you quickly run out of memory.

Components of an Event Driven Application

The following topics are covered below:

- Dialogs
- Dialog Elements
- Attributes
- Event Handlers
- Data Areas - Global, Local, Parameter
- Inline Subroutines

Dialogs

The dialog is the central Natural object in an event-driven application. An event-driven application is started by running or executing the base dialog. This may open other dependent dialogs when the `OPEN DIALOG` statement is specified. As opposed to program-driven applications, these dialogs are usually modeless, that is, all open dialogs can be processed concurrently by the end user. The application

terminates when the base dialog is closed.

You create a dialog with the dialog editor. Just like the map editor, the dialog editor assembles a Natural object from the specification of the dialog window and its dialog elements, the global data area (GDA), the local data areas (LDAs), the parameter data areas (PDAs), the subroutines and the specified event handler sections.

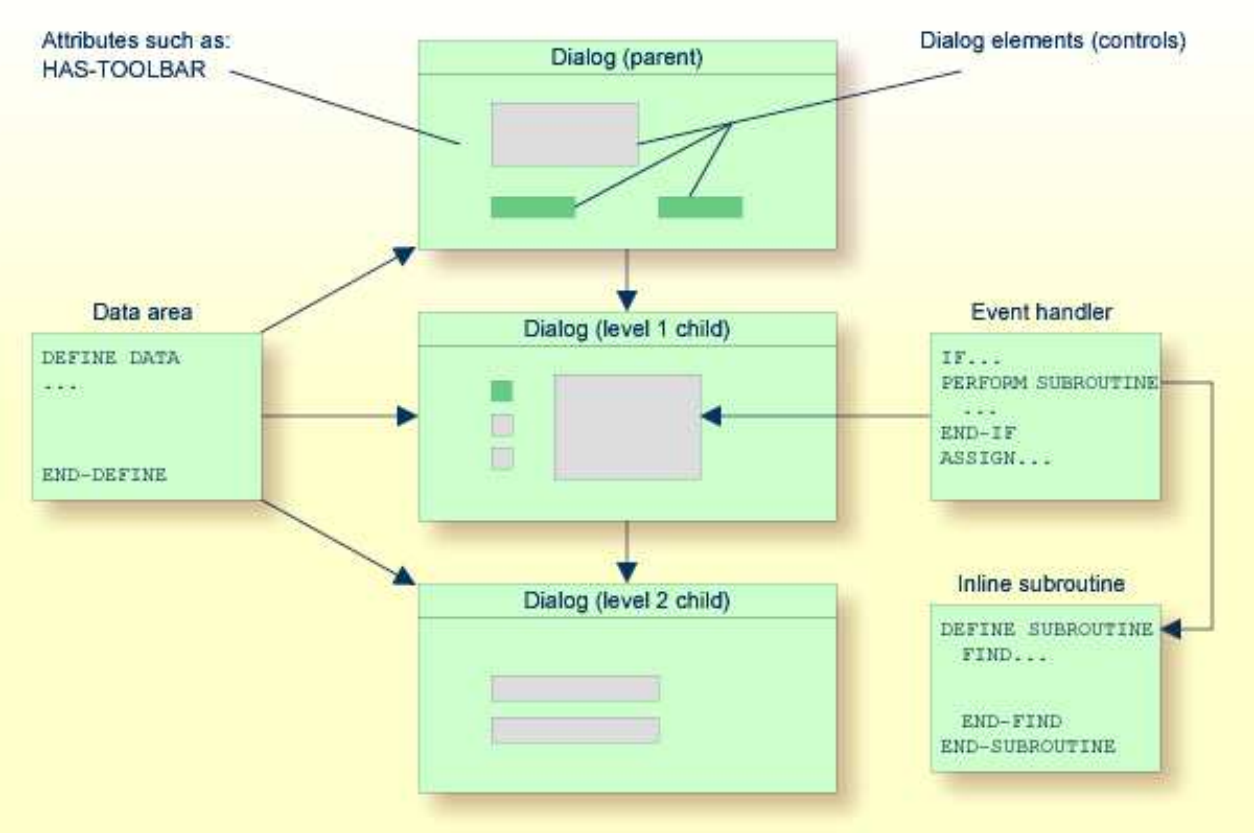
At runtime of the dialog, there is a difference between the runtime instance identified by the system variable `*DIALOG-ID` and the GUI instance (handle) of the dialog window (the default handle name is `#DLG$WINDOW`).

Whenever you want to work with more than one dialog in your application, you must decide how the base dialog window relates to the other dialogs. First you have to decide whether the application should be MDI (Multiple Document Interface) or not.

If you have opted for an MDI application, the base dialog must be of the type "MDI frame window" and the dependent dialogs must be of the type "MDI child window" and "Standard window".

If you have opted for non-MDI, the application may contain only dialogs of the type "Standard window".

Dialogs of type "Standard window" can have the styles "Popup", "Modal" or "Dialog Box".



Dialog Elements

Almost all dialog elements are graphical elements inside a dialog that allow the end user to interact with the event-driven application. After a dialog has been opened with the dialog editor and its attributes have been set (see below), the programmer will go on to "draw" the dialog elements inside the window; usually, this comprises a menu control, possibly a toolbar, and other elements, such as push-button controls, input-field controls.

"Drawing" a dialog element means that you select the type of dialog element from the dialog editor’s menu or toolbar, and use the mouse to place it at the desired location. It is also possible to define a grid where the dialog elements can be placed more conveniently by aligning them to the grid.

Attributes

Attributes are the properties of dialogs and dialog elements. After creating a dialog or dialog element, you double-click with the mouse on it and the window with the corresponding attributes appears. You can then set the attributes to a value; if not, they remain at the system default value. The attributes window also contains a push-button control that opens up the event handler window.

Event Handlers

The event handlers represent the Natural code that is triggered when an event occurs. A click event occurs, for example, when the end user clicks on a push-button control. Inside the event handler window, you must first select the type of event from the list of events available for the dialog or dialog element (the one whose attributes have just been set). Then, the code window is enabled and Natural code can be entered.

Data Areas - Global, Local, Parameter

- A global data area (GDA) is used to share data fields between Natural objects within the application. One GDA per application may be specified.
- A local data area (LDA) contains the data fields private to the dialog.
- A parameter data area (PDA) is always present in dialogs. It is used to pass parameters to a dialog in the `OPEN DIALOG` or `SEND EVENT` statements. In these statements, parameters are passed either by specifying their name (`WITH` clause), or by listing parameters one after the other. You can use the dialog editor PDA window to type in your PDA in free-form style or to include PDAs defined externally.

Inline Subroutines

An inline subroutine defines standard code to be used for a frequently needed task called by a number of event handlers. You access an inline subroutine window via the "Inline Subroutines" push button control in the dialog window.