EVENTS OF THE GUI WINDOW INTERFACE

Version 1.0

Reference book

Moscow, 2000 year

1 TABLE OF CONTENTS

2 INTRODUCTION

2.1 Field of application

Window events represent events which are sent by the graphical user interface (GUI) to the **GWnd** class objects and its descendants (for more information see the reference book on the standard gui library).

2.2 Requirements to the user training

A user should have computer operation skills and study the current manual. It is also necessary to learn the Pluk language reference book, the reference book on the standard library.

3 EVENTS DESCRIPTION

3.1 Events sent to windows

A window event handler is assigned with the help of the **event** command (for more information see the Pluk language reference book). Handler assignment does not mean its obligatory calling on receiving the event. It is necessary to allow the event to pass to a window object. There are special methods **GWnd::AllowEvent** and **GWnd::ForbidEvent** (for more information see the reference book on the standard gui library) that allows or forbids the event receiving. It is always allowed to receive the following events: **WND_COMMAND** (see 3.1.1, 3.2.1), **WND_HSCROLL** (see. 3.1.4), **WND_VSCROLL** (see 3.1.5), as well as user's (see 3.1.17) and command events (see 3.1.180, 3.2.2).

A handler can return an integer value. –1 value is defined as the default event handling. In some cases (for instance, **WND_UPDATE**, see 3.1.3) default handling allows you to avoid the erroneous situation.

3.1.1 WND_COMMAND

EventHandler(int, int)
param id, notifyCode;
EventHandler(int, int, int)
param id, notifyCode, pos;

This event is sent to a window when a user selects a menu item, performs any actions in a control or presses a key which is interpreted by the keyboard accelerator table.

id	Menu item or control identifier.
notifyCode	Notification code, if the event is from the control (see 3.3). 0, if the event is from the menu. 1, if the event is from the keyboard accelerator.
pos	Scroll bar position, if the event is from the control of the scroll bar type.

Returns –1, if the default handling is required.

Comments

The receiving of this event is always allowed.

The second handler type is called for a control that represents a scroll bar, the first type is called for controls of other types, menu item and the key which is interpreted by keyboard accelerator table.

The default handling of the current event sends additional **WND_COMMAND** events to all objects (derived from the **GControl** class) which are connected to the control from which the notification has been received(see 3.2.1).

3.1.2 WND_SIZE

EventHandler(int, int, int)
param sizeType, width, height;

This event is sent to a window when its size has changed.

sizeType Event type.
width New client region width.
height New client region height.

Returns –1, if the default handling is required.

Comments

Available values of the **sizeType** parameter include:

SIZE_RESTORED The window has changed its size but neither **SIZE_MINIMIZED** nor **SIZE_MAXIMIZED** event has not occurred.
SIZE_MINIMIZED The window was minimized.
SIZE_MAXIMIZED The window was maximized.

3.1.3 WND_UPDATE

EventHandler(void)

This event is sent to a window when the system or an application requests the refreshing of the application window (or part of the window).

Returns –1, if the default handling is required.

Comments

This event can be received according to two reasons. First, if any actions that require window content update occurred in the system, then the **WND_UPDATE** event is placed into the application event queue. Second, the application itself can place the event into the queue by calling the **GWnd::Invalidate** method (for more information see the reference book on the standard gui library). If there are no other events in the queue, then this event will be sent to the window. If it is necessary for the application to update the window immediately, then it should force the event to be extracted from the queue by calling the **GWnd::Update** method.

If the application did not perform drawing in the current event handler, then it should return –1 in order to avoid the erroneous situation .

3.1.4 WND_HSCROLL

EventHandler(int, int)
param notifyCode, pos;

This event is sent to a window when a user scrolls the horizontal window scroll bar.

notifyCode Notification code (see 3.3.50).
pos The current position of the scroll bar slider.

Returns –1, if the default handling is required.

Comments

The receiving of this event is always allowed.

3.1.5 WND_VSCROLL

EventHandler(int, int)
param notifyCode, pos;

This event is sent to a window when a user scrolls the vertical window scroll bar.

notifyCode Notification code (see 3.3.5).
pos The current position of the scroll bar slider.

Returns –1, if the default handling is required.

Comments

The receiving of this event is always allowed.

3.1.6 WND_TIMER

EventHandler(int)
param id;

This event is sent to the application event queue after each time interval determined in the **GWnd::SetTimer** method which is used for setting the window timer (for more information see the reference book on the standard gui library).

id Timer identifier.

Returns –1, if the default handling is required.

Comments

If the previous **WND_TIMER** event remains in the queue after the time interval determined in the **GWnd::SetTimer** method, the the new one is not plased into the queue.

If there are no other events in the queue, then this event will be sent to the window.

3.1.7 WND_ACTIVATE

EventHandler(int)
param activeType;

This event is sent when a window is activated or deactivated. First of all this event is sent to the deactivated window and then to the activated one.

activeType Specifies whether the window is activated or deactivated.

Returns –1, if the default handling is required.

Comments

Available values of the **activeType** parameter include:

WA_INACTIVE The window is deactivated.
WA_ACTIVE The window is activated without using the mouse.
WA_CLICKACTIVE The window is activated via the mouse.

3.1.8 WND_KEYDOWN

EventHandler(int)
param virtCode;

This event is sent when a user presses a key.

virtCode Virtual key code (see 3.4).

Returns –1, if the default handling is required.

3.1.9 WND_KEYUP

EventHandler(int)
param virtCode;

This event is sent when a user releases a key.

virtCode Virtual key code (see 3.4).

Returns –1, if the default handling is required.

3.1.10 WND_MOUSEMOVE

EventHandler(int, int)
param x, y;

This event is sent when the mouse pointer is moving.

x x-coordinate of the pointer relatively to the top-left window corner.
y y-coordinate of the pointer relatively to the top-left window corner.

Returns –1, if the default handling is required.

3.1.11 WND_LBUTTONDOWN

EventHandler(int, int)
param x, y;
This event is sent when a user presses the left mouse button.

x	x-coordinate of the pointer relatively to the top-left window corner.
y	y-coordinate of the pointer relatively to the top-left window corner.

Returns –1, if the default handling is required.
Comments
The default handling of the current event brings the window to front and passes the input focus to it.

3.1.12 WND_LBUTTONUP

EventHandler(int, int)
param x, y;
This event is sent when a user releases the left mouse button.

x	x-coordinate of the pointer relatively to the top-left window corner.
y	y-coordinate of the pointer relatively to the top-left window corner.

Returns –1, if the default handling is required.

3.1.13 WND_LBUTTONDBLCLK

EventHandler(int, int)
param x, y;
This event is sent when a user double clicks the left mouse button.

x	x-coordinate of the pointer relatively to the top-left window corner.
y	y-coordinate of the pointer relatively to the top-left window corner.

Returns –1, if the default handling is required.
Comments
The double click of the left mouse button forces the following events to occur: **WND_LBUTTONDOWN**, **WND_LBUTTONUP**, **WND_LBUTTONDBLCLK**, **WND_LBUTTONUP**.

3.1.14 WND_RBUTTONDOWN

EventHandler(int, int)
param x, y;
This event is sent when a user presses the right mouse button.

x	x-coordinate of the pointer relatively to the top-left window corner.
y	y-coordinate of the pointer relatively to the top-left window corner.

Returns –1, if the default handling is required.
Comments
The default handling of the current event brings the window to front and passes the input focus to it.

3.1.15 WND_RBUTTONUP

EventHandler(int, int)
param x, y;
This event is sent when a user releases the right mouse button.

x	x-coordinate of the pointer relatively to the top-left window corner.

y y-coordinate of the pointer relatively to the top-left window corner.

Returns –1, if the default handling is required.

Returns –1, if the default handling is required.

Comments

The default handling of the current event displays the window context menu if such a menu is assigned to the window.

3.1.16 WND_RBUTTONDBLCLK

EventHandler(int, int)
param x, y;

This event is sent when a user double clicks the right mouse button.

x x-coordinate of the pointer relatively to the top-left window corner.

y y-coordinate of the pointer relatively to the top-left window corner.

Returns –1, if the default handling is required.

Comments

The double click of the left mouse button forces the following events to occur: **WND_RBUTTONDOWN**, **WND_RBUTTONUP**, **WND_RBUTTONDBLCLK**, **WND_RBUTTONUP**.

3.1.17 User's event

EventHandler(...)
param [pars];

An event from **WND_USER** to 0x8000 represents a user's event which can be freely used by an application.

pars Parameters that depend on the event.

Returns –1, if the default handling is required.

Comments

The receiving of this event is always allowed.

3.1.18 Command event

EventHandler(int, int)
param id, notifyCode;
EventHandler(int, int, int)
param id, notifyCode, pos;

An event from **CMD_FIRST** to 0x20000 represents a command event. It is received from the controls which are owned by the window.

id Menu item or control identifier.

notifyCode Notification code, if the event is from the control (see 3.3). 0, if the event is from the menu. 1, if the event is from the keyboard accelerator.

pos Scroll bar position, if the event is from the control of the scroll bar type.

Returns –1, if the default handling is required.

Comments

The receiving of this event is always allowed.

The second handler type is called for a control that represents a scroll bar, the first type is called for controls of other types, menu item and the key which is interpreted by keyboard accelerator table.

The event number is determined as **CMD_FIRST** plus menu item or control identifier. For instance, if it is necessary to define a handler for the button which identifier is equal to 101, then you should assign a handler to the **CMD_FIRST** + 101 event. On calling the method the **id** parameter value will be equal to 101.

Command events of both types are generated on executing the **GWnd::OnCommand** method (for more information see the reference book on the standard gui library).

3.2 Events sent to controls

Some control types can have connected to them objects of the classes derived from the **GControl** class (**GListBox**, **GComboBox**, **GEdit**, **GTable** (for more information see the reference book on the standard gui library). If an application requires the default handling for the **WND_COMMAND** event which is sent to a window that owns such a control and the objects of the above-listed types exist and connected with the control, then the **WND_COMMAND** event also will be sent to these objects (but this event will have the auxiliary parameter – reference to the window).

3.2.1 WND_COMMAND

EventHandler(refer object GWnd, int, int)
param wnd, id, notifyCode;

This event is sent to a control as a result of default handling the **WND_COMMAND** event which is sent to the window that owns the control.

wnd	The window that owns the control.
id	Control identifier.
notifyCode	Notification code (see 3.3).

Returns –1, if the default handling is required.

Comments

The receiving of this event is always allowed.

3.2.2 Command event

EventHandler(refer object GWnd, int, int)
param wnd, id, notifyCode;

An event from **CMD_FIRST** to 0x20000 represents a command event. It is received from a control as a result of default handling the **WND_COMMAND** event which is sent to the window that owns the control.

wnd	The window that owns the control.
id	Control identifier.
notifyCode	Notification code (see 3.3).

Returns –1, if the default handling is required.

Comments

The receiving of this event is always allowed.

The event number is determined as **CMD_FIRST** plus menu item or control identifier. For instance, if it is necessary to define a handler for the button which identifier is equal to 101, then you should assign a handler to the **CMD_FIRST** + 101 event. On calling the method the **id** parameter value will be equal to 101.

Command events of both types are generated from the **GControl::OnCommand** method (for more information see the reference book on the standard gui library).

3.3 Control notification codes

A window receives notification codes from controls via the **notifyCode** parameter of command events (see 3.1.1, 3.1.180, 3.2.1, 3.2.2). This code indicates that the state of the control, from which the command event is received, has changed. For instance, when the button with the 101 identifier is pressed, the window that owns this button receives the **CMD_FIRST** + 101 event with the **id** parameter equal to 101 and the **notifyCode** parameter equal to **BN_CLICKED**. Below you can find the notification codes for all control types supported by GUI.

3.3.1 Button

3.3.1.1 BN_CLICKED

It is sent when a user presses a button, moreover, it is sent at the moment when a user releases a mouse button or a key.

3.3.1.2 BN_PUSHED

It is sent when a user presses a button, moreover, it is sent at the moment when a user presses a mouse button or a key.

Comments

Notification is sent only for the button with the extended notification.

3.3.1.3 BN_RCLICKED

It is sent when a user presses the right mouse button, moreover, it is sent at the moment when a user releases the mouse button.

Comments

Notification is sent only for the button with the extended notification.

3.3.1.4 BN_SETFOCUS

It is sent when a button receives the input focus.

Comments

Notification is sent only for the button with the extended notification.

3.3.1.5 BN_RELEASEFOCUS

It is sent when a button looses the input focus.

Comments

Notification is sent only for the button with the extended notification.

3.3.2 Editor

3.3.2.1 EN_SETFOCUS

It is sent when an editor receives the input focus.

3.3.2.2 EN_RELEASEFOCUS

It is sent when an editor looses the input focus.

3.3.2.3 EN_CHANGE

It is sent when a user performed any action that results in text modification within an editor.

Comments

In contrast to the **EN_UPDATE** notification the current notification is sent after an editor has updated itself on screen.

3.3.2.4 EN_UPDATE

It is sent when an editor is going to update the modified text.

Comments

This notification is sent after an editor has formatted the text but before it displays this text on screen.

3.3.2.5 EN_MAXTEXT

It is sent when the number of inserted characters exceeds the restriction on the number of characters within an editor. It is also sent when the editor does not support horizontal (vertical) text scrolling and the insertion exceeds the editor width (height).

3.3.3 List

3.3.3.1 LBN_SELCHANGE

It is sent when the selection within a list has changed.

Comments

Notification is not sent when the selection has changed by calling one of the following methods: **GListBox::SetSel**, **GListBox::SetStrSel** (for more information see the reference book on the standard gui library).

For multi-selection lists the event is sent each time a user presses an arrow key even if the selection does not change.

3.3.3.2 LBN_DBLCLK

It is sent when a user double clicks the left mouse button on the list line.

3.3.3.3 LBN_SETFOCUS

It is sent when a list receives the input focus.

3.3.3.4 LBN_RELEASEFOCUS

It is sent when a list looses the input focus.

3.3.4 List combined with an editor control

3.3.4.1 CBN_SELCHANGE

It is sent when the selection within a list has changed.

Comments

Notification is not sent when the selection has changed by calling one of the following methods: **GComboBox::SetSel**, **GComboBox::SetStrSel** (for more information see the reference book on the standard gui library).

3.3.4.2 CBN_DBLCLK

It is sent when a user double clicks the left mouse button on the list line.

Comments

Notification is not sent for the dropdown list as the single click results in closing the list.

3.3.4.3 CBN_SETFOCUS

It is sent when a list receives the input focus.

3.3.4.4 CBN_RELEASEFOCUS

It is sent when a list looses the input focus.

3.3.4.5 CBN_EDITCHANGE

It is sent when a user performed any action that results in text modification within a list editor.

Comments

In contrast to the **CBN_EDITUPDATE** notification the current notification is sent after a list editor has updated itself on screen.

3.3.4.6 CBN_EDITUPDATE

It is sent when a list editor is going to update the modified text.

Comments

This notification is sent after a list editor has formatted the text but before it displays this text on screen.

3.3.4.7 CBN_DROPDOWN

It is sent when a list is dropped down.

Comments

Notification is sent only for the dropdown list.

3.3.4.8 CBN_CLOSEUP

It is sent when a list is closed.

Comments

Notification is sent only for the dropdown list.

3.3.5 Scroll bar

3.3.5.1 SB_LINEUP

It is sent when a user moves the scroll bar slider upward on one step.

3.3.5.2 SB_LINELEFT

It is sent when a user moves the scroll bar slider to the left on one step.

3.3.5.3 SB_LINEDOWN

It is sent when a user moves the scroll bar slider downward on one step.

3.3.5.4 SB_LINERIGHT

It is sent when a user moves the scroll bar slider to the right on one step.

3.3.5.5 SB_PAGEUP

It is sent when a user moves the scroll bar slider upward on one page step.

3.3.5.6 SB_PAGELEFT

It is sent when a user moves the scroll bar slider to the left on one page step.

3.3.5.7 SB_PAGEDOWN

It is sent when a user moves the scroll bar slider downward on one page step.

3.3.5.8 SB_PAGERIGHT

It is sent when a user moves the scroll bar slider to the right on one page step.

3.3.5.9 SB_THUMBTRACK

It is sent when a user moves the scroll bar slider using the mouse.

3.3.5.10 SB_ENDSCROLL

It is sent when a user releases a mouse button on the scroll bar slider.

3.3.6 Table

3.3.6.1 TN_SELCHANGE

It is sent when the selection within a table has changed.

Comments

Notification is not sent when the selection has changed by calling the **GTable::SetSel** method (for more information see the reference book on the standard gui library).

For multi-selection tables the event is sent each time a user presses an arrow key even if the selection within the table does not change.

In the current notification code handler the calling of the **GTable::GetNotifyPos** method (for more information see the reference book on the standard gui library) returns the position of the cell which is selected y a user (for a single-selection table) or position of the last cell for which a user set or discarded selection (for a multi-selection table).

3.3.6.2 TN_DBLCLK

It is sent when a user double clicks the left mouse button on a table cell.

In the current notification code handler the calling of the **GTable::GetNotifyPos** method (for more information see the reference book on the standard gui library) returns the position of the cell which was double clicked by a user.

3.3.6.3 TN_RCLICKED

It is sent when a user releases the right mouse button on a table cell.

In the current notification code handler the calling of the **GTable::GetNotifyPos** method (for more information see the reference book on the standard gui library) returns the position of the cell on which a user released the right mouse button.

3.3.6.4 TN_SETFOCUS

It is sent when a table receives the input focus.

3.3.6.5 TN_RELEASEFOCUS

It is sent when a table looses the input focus.

3.3.6.6 TN_COLUMNCLICK

It is sent when a user presses the button situated in the table column header.

Comments

In the current notification code handler the calling of the **GTable::GetNotifyPos** method (for more information see the reference book on the standard gui library) returns the index of the column at which header a user pressed the button (in the second element of the returned vector).

3.3.6.7 TN_COLUMNENDTRACK

It is sent when a user has moved the bound between table columns using the mouse.

Comments

In the current notification code handler the calling of the **GTable::GetNotifyPos** method (for more information see the reference book on the standard gui library) returns the index of the column which is situated at the left from the moved bound (in the second element of the returned vector).

3.3.6.8 TN_BEGINEDIT

It is sent when a user begins the table cell editing.

In the current notification code handler the calling of the **GTable::GetNotifyPos** method (for more information see the reference book on the standard gui library) returns the position of the cell which is being edited.

3.3.6.9 TN_ENDEDIT

It is sent when a user successfully completed the table cell editing.

Comments

The current notification is sent after a user finished the cell editing by pressing the ***Enter*** key or by activating table or another window.

In the current notification code handler the calling of the **GTable::GetNotifyPos** method (for more information see the reference book on the standard gui library) returns the position of the cell whose editing is completed.

3.3.6.10 TN_CANCELEDIT

It is sent when a user unsuccessfully completed the table cell editing.

Comments

The current notification is sent after a user finished the cell editing by pressing the ***Cancel*** key.

In the current notification code handler the calling of the **GTable::GetNotifyPos** method (for more information see the reference book on the standard gui library) returns the position of the cell whose editing is completed.

3.4 Virtual key codes

Virtual key codes are used for hardware-independent key identification when handling the **WND_KEYDOWN** (see 3.1.8) and **WND_KEYUP** (see 3.1.9). Virtual code of numeric keys is

equal to the ASCII code of the appropriate number. Virtual code of literal keys is equal to the ASCII code of the appropriate upper-case letter. System keys have the following virtual codes:

Code	Key
VK_LBUTTON	Left mouse button
VK_RBUTTON	Right mouse button
VK_ESCAPE	***Esc***
VK_F1	***F1***
VK_F2	***F2***
VK_F3	***F3***
VK_F4	***F4***
VK_F5	***F5***
VK_F6	***F6***
VK_F7	***F7***
VK_F8	***F8***
VK_F9	***F9***
VK_F10	***F10***
VK_F11	***F11***
VK_F12	***F12***
VK_PRINT	***Print Screen***
VK_SCROLL	***Scroll Lock***
VK_PAUSE	***Pause Break***
VK_BACK	***Backspace***
VK_TAB	***Tab***
VK_CAPITAL	***Caps Lock***
VK_RETURN	***Enter***
VK_SHIFT	***Shift***
VK_CONTROL	***Ctrl***
VK_ALT	***Alt***
VK_SPACE	***Space***
VK_INSERT	***Insert***
VK_HOME	***Home***
VK_PRIOR	***Page Up***
VK_DELETE	***Delete***
VK_END	***End***
VK_NEXT	***Page Down***
VK_UP	***Up Arrow***
VK_LEFT	***Left Arrow***
VK_DOWN	***Down Arrow***
VK_RIGHT	***Right Arrow***
VK_NUMPAD0	0 (on numeric keypad)
VK_NUMPAD1	1 (on numeric keypad)
VK_NUMPAD2	2 (on numeric keypad)
VK_NUMPAD3	3 (on numeric keypad)
VK_NUMPAD4	4 (on numeric keypad)
VK_NUMPAD5	5 (on numeric keypad)
VK_NUMPAD6	6 (on numeric keypad)

VK_NUMPAD7	7 (on numeric keypad)
VK_NUMPAD8	8 (on numeric keypad)
VK_NUMPAD9	9 (on numeric keypad)
VK_NUMLOCK	Num Lock
VK_DIVIDE	/ (on numeric keypad)
VK_MULTIPLY	* (on numeric keypad)
VK_SUBTRACT	– (on number panel)
VK_ADD	+ (on number panel)
VK_DECIMAL	. (on number panel)