STIN2024
Pengaturcaraan Logik
Prolog Programming
Wan Hussain Wan Ishak

School of Computing
UUM College of Arts and Sciences
Universiti Utara Malaysia

(P) 04-9285150
(E) hussain@uum.edu.my
(U) http://wanhussain.com
Lecture notes

Windows Programming

- Designing GUI in Prolog
- Object Classes for dialogs
- Message passing mechanism
- Message handling mechanism

STIN2024
Pengaturcaraan Logik
Logic Programming

The Eminent Management University
GUI in Prolog

■ Dialog Editor – toolkit to develop interactive GUI.

■ A dialog consists of:
  ■ Dialog window with GUI elements and its controls.
  ■ Dialog message handler – to interpret the control messages.
GUI in Prolog

- Types of dialog:
  - Modeless dialog
  - Modal dialog
GUI in Prolog

- **Modal dialog**
  - Once active, it wait user to response. Other windows will be inactive mode.
  - Example: use to accept input or for verification purposes (eg: Ok, Yes, No, Cancel, etc.).

- **Modeless dialog**
  - Co-exist with other active windows/dialog.
Designing GUI in Prolog

- Starting editor
- Select Dialog Editor

Scratch window

Toolbox

Dialog Code
Designing GUI in Prolog

Click and draw

Click and Draw
Designing GUI in Prolog

- Change properties

Double Click
Designing GUI in Prolog

- Preview Dialog

1. Click Edit
Designing GUI in Prolog

- Export
- Source Code

Click Export

Source Code

Source Code

Export

Source Code
Designing GUI in Prolog

- Click Import Source Code
- Import
- Click Yes
- Dialog will be created based on the given source code
Object Classes for Dialog

- Two types of classes:

  - **Window class** - A framework for a window (skeleton) in which dialogs can be developed.

  - **Control class** - Objects embedded on a window and to control the behaviour of the dialog.
Object Classes for Dialog

- **Window classes**
  - Attribute for the window class is determined by user.

![Window dialog](image)

**Change Attributes**
Object Classes for Dialog

- Control classes

  - Control classes:
    1) Button class
    2) Edit Class
    3) Listbox Class
    4) Combobox Class
    5) Static Class
    6) Grafix Class
    7) Scrollbar class
Object Classes for Dialog

- **Button class**
Object Classes for Dialog

- **Edit class**

![Example STATIC edit control](image)

*(Example: SINGLE LINE edit box)*

Password

![Example: MULTIPLE LINE dialog box. Horizontal and vertical scrollbars can also be created](image)
Object Classes for Dialog

- Listbox class
Object Classes for Dialog

- **Combobox** class

![Combobox Example](image)

- **Editbox**
- **Listbox**
Message Passing Mechanism

- Message passing – to coordinate windows functions.

- Information passes between processes – windows messages.

- Windows messages: contains – commands, parameters or notification of actions.
Message Handling Mechanism

- Message is passes to Win Prolog to perform user specific actions.

- Message handler will catch the message and do the processing.

- Each dialog has its own handler.
Message Handling Mechanism

How message handling works?

1) A message interrupts the execution of the Prolog program.
2) Program control is passed to a Message Handler – a specified action.
3) Prolog code associated with the Message Handler is executed.
4) Once completed, the control is passed back to the main program (return to the dialog/window).
Message Handling Mechanism

- To define a relation between window and its handlers.
- Window Handlers - window_handler/2.
- Get or set the current message handler for the given window

`window_handler(Window, Handler)`

Window – name of the window created.
Handler – state a name as the handler.
The handler name will become the name for predicate handler/4.

handler(Win, Msg, Data, Result)

Win – name of the window
Data – suitable data
Msg – message receive
Result – variable
Message Handling Mechanism

General message names

- msg_menu
- msg_fuzzy
- msg_double
- msg_vert
- msg_leftup
- msg_char
- msg_sysmenu
- msg_change
- msg_size
- msg_paint
- msg_rightdown
- msg_key
- msg_close
- msg_button
- msg_move
- msg_leftdown
- msg_rightdouble
- msg_drop
- msg_focus
- msg_select
- msg_horz
- msg_leftdouble
- msg_rightup
- msg_mousedmove
Message Handling Mechanism

Example

test:-
create_example_dialog,  
window_handler(example, example_handler),  
show_dialog(example).  

example_handler((example,1000), msg_button, _, Result):-  
wtext((example,8000), N1),  
wtext((example,8001), N2),  
number_string(Number1, N1),  
number_string(Number2, N2),  
Ht is Number1 + Number2,  
number_string(Ht,H),  
wtext((example,8002),H).
Message Handling Mechanism

Example – source code for a dialog

create_example_dialog:-
    _S1 = [dlg_ownedbyprolog,ws_sysmenu,ws_caption],
    _S2 = [ws_child,ws_visible,ss_center],
    _S3 = [ws_child,ws_visible,ws_tabstop,ws_border,es_center,es_multiline,es_autohscroll,es_autovscroll],
    _S4 = [ws_child,ws_visible,ws_tabstop,bs_pushbutton],
    wdcreate( example, `example Dialog`, 322, 199, 272, 161, _S1 ),
    wccreate( example,10000, static, `Masukkan dua nilai dan tekan butang dibawah`, 20, 20, 230, 20, _S2 ),
    wccreate( example,8002, edit, `0`, 20, 110, 230, 20, _S3 ),
    wccreate( example,8000, edit, `0`, 20, 50, 60, 20, _S3 ),
    wccreate( example,8001, edit, `0`, 190, 50, 60, 20, _S3 ),
    wccreate( example,1000, button, `TAMBAH`, 20, 80, 50, 20, _S4 ),
    wccreate( example,1001, button, `TOLAK`, 80, 80, 50, 20, _S4 ),
    wccreate( example,1002, button, `DARAB`, 140, 80, 50, 20, _S4 ),
    wccreate( example,1003, button, `BAHAGI`, 200, 80, 50, 20, _S4 ).
Message Handling Mechanism
Built-in Predicates: Creating Window

- wdcreate/7 - create a dialog window

wdcreate( Window, Title, Left, Top, Width, Height, Style)

Create a dialog window with the given Window, Title, Left - Top corner, Width - Height dimensions, and Style. The Window argument must be an atom. Style is a list of logical window styles which are combined to create the 32-bit integer which is passed directly to Windows. This predicate can only use the generic window styles. Note: at present all dialogs must include the style 'ws_popup' to allow the dialog to function correctly stand-alone.
Built-in Predicates: Creating Window

- \texttt{wcreate/8 - create a window}

\texttt{wcreate( Window, Class, Title, Left, Top, Width, Height, Style)}

The predicate \texttt{wcreate/8} creates a window with the given Window name, Class, Title, Left - Top corner coordinates, Width - Height dimensions and Style. The window will be created in one of several styles, depending upon the given handle and style: if the handle is an atom, a top level window is created; if the handle is a conjunction of the form \((\text{window,id})\), then a control window with the given ID is created within the given window. The Class argument defines the type of window to be created; it may be either a predefined Windows class or an LPA defined window class. The Style argument is a 32-bit integer that specifies an available window style which is passed directly to Windows. For top level windows (or MDI children), the text argument forms the window title (style permitting) of the window; for control windows, the text is the label of the control (where appropriate).
Built-in Predicates: Creating Window

- \texttt{wccreate/8} - create a control window

\texttt{wccreate( Window, Class, Title, Left, Top, Width, Height, Style)}

Create a control Window with the given Class, Title, Left - Top corner coordinates, Width - Height dimensions, and Style. The Window argument is of the form (Parent,ID), where Parent is the handle of a top-level window and ID is the handle of the control window. The Class argument is one of the predefined control window classes. The Style argument is a list of logical window styles which are combined to create the 32-bit integer which is passed directly to Windows. This predicate can combine any of the generic window styles with the styles for the given class. Note: you should always include the \texttt{ws\_child} style in the Style list.
Built-in Predicates: Creating Window

- wtcreate/6 - create a text window

wtcreate( Name, Title, Left, Top, Width, Height)

Create a "text" window with the given Name, Title, Left - Top corner coordinates, Width and Height dimensions. Name should be an atom which is used from then on to refer to the window. Text windows contain an "edit" field that is automatically resized according to the resizing of the window.
Built-in Predicates: Creating Window

- wtcreate/6 - example

```prolog
?- wtcreate( test, 'Test', 50, 50, 20, 30).
yes
?- boleh tulis teks
```
Built-in Predicates: Creating Window

- `wucreate/6` - create a user MDI window

```
wucreate( Name, Title, Left, Top, Width, Height)
```

Create a "user" MDI window with the given Name, Title, Left - Top corner coordinates, Width and Height dimensions. Name should be an atom which is used from then on to refer to the window. User windows are created with a system menu, a hide button, a maximize button and are re-sizeable. They do not contain any other controls. Note: if you put any control items in a user MDI window you must write your own code to handle the re-sizing of the window.
Built-in Predicates: Creating Window

**wucreate/6** – example:

```
?- wucreate( test, 'Test', 20, 20, 20, 20).
yes
```

![Console screenshot](image)
Built-in Predicates: Button Class

- wbttssel/2 - get or set selection state of a button

wbttssel(Window, Status)

Get or set the selection status of the given radio or checkbox "button". The Window argument is the handle of the button. The Status argument is a button status value.

Value | Status
--- | ---
0 | radio button deselected or checkbox unchecked
1 | radio button selected or checkbox checked

example: wbttssel((example,100), Status)
Built-in Predicates: Listbox Class

- wlbxadd/3 - add an item to a list box

wlbxadd(Window, Position, String)

Add a String to the "listbox" control Window at the given Position. If the position is given as -1, the item is inserted or appended to the list box depending upon the list box style. Entries in a listbox are numbered from 0.

example: wlbxadd((user_dialog,4000), 0, `item1`).
Built-in Predicates: Listbox Class

- `wlbxdel/2` - delete an item from a list box

```
wlbdel(Window, Position)
```

Delete the item at the given Position in the given "listbox" control Window. Entries in a listbox are numbered from 0.

Example: `wlbxdel((user_dialog,4000), 0)`.
**Built-in Predicates: Listbox Class**

- **wlbxfnd/4 - find a string in a list box**

  \[\text{wlbxfnd}(\text{Window}, \text{Start}, \text{String}, \text{Position})\]

  Return the Position of a partial match String in the given "listbox" control Window, starting search one place after the given Start. Entries in a listbox are numbered from 0. If String is the empty string `\` then \text{wlbxfnd}/4 will return the Position of the entry following the Start.

  example: \text{wlbxfnd}(\text{(example,4000), 0, String, Position}).
Built-in Predicates: Listbox Class

- **wlbxget/3** - get an item from a list box

\[ \text{wlbxget(\text{Window, Position, String})} \]

Get the String at the given Position in the given "listbox" control Window. Entries in a listbox are numbered from 0. This predicate will fail if the listbox has a number of entries that is less than or equal to the given position.

Example: \( \text{wlbxget((example,4000), 2, String)} \).
Built-in Predicates: Listbox Class

- **wlbxsel/3 - get or set selection in a list box**

```
wlbxsel(Window, Position, State)
```

Set or get the selection State of the item at the given Position in the given "listbox" control Window. If the Position is given as -1, and the listbox is a multi-choice list box, the selection state is applied to all items. Entries in a listbox are numbered from 0. The State argument is a variable or a listbox selection state value.

example: `wlbxsel((example,4000), 2, 1)`.
**Built-in Predicates: Editbox Class**

**wedtsel/3** - get or set selection in an "edit" control window

`wedtsel( Window, First, Second)`

Sets the text selection area in the given "edit" control Window to start and finish at the given First and Second values, or returns the existing values. This predicate causes a direct side effect on the window, whose cursor moves to the position specified. Note that the start and finish positions can be given in either order: the flashing caret is positioned at the end specified by the First value. Windows does not provide the caret position when retrieving the selection: the smallest value is always returned in the First parameter.

example: `wedtsel((example,800), 11, 15)`.
Built-in Predicates: Editbox Class

- `wedtfnd/6` - find a text string in an "edit" control window

```
wedtfnd( Window, Start, End, String, StartMatch, EndMatch)
```

Search the given "edit" control Window for the given text String within the given Start and End points. The start and finish of the first matching string is returned as a pair of integers, StartMatch and EndMatch. As a special case, the search text may be specified as an empty string. In this case, the start and finish of the next space-delimited token is returned. No side effects are caused by this predicate, which is used for the gathering of information only. The returned parameters may be passed directly into `wedtsel/3` if it is desired to move the selection to the found string.
Built-in Predicates: Editbox Class

- `wedtlin/4` - get offsets a line in an "edit" control window

`wedtlin(Window, Offset, Start, Finish)`

Returns the Start and Finish of the line of text containing the given character offset in the given "edit" control Window, or tests the given values for correctness. The offsets returned include everything on the given line, but not the carriage return/line feed. No side effects are caused by this predicate, which is used for the gathering of information only.
Built-in Predicates: Editbox Class

- **wedtpxy/4** - convert between linear offset and x, y coordinates in "edit" windows

```
wedtpxy( Window, Offset, X, Y)
```

Returns the X and Y coordinates that are the equivalent of a given character Offset, or returns the character Offset of the given X and Y values, or tests the given values for correctness. The values are computed for the given "edit" Window. No side effects are caused by this predicate, which is used for the gathering of information only.
Built-in Predicates: Editbox Class

- **wcount/4** - get char, word and line counts for the given window

\[ \text{wcount( Window, Characters, Words, Rows) } \]

Returns the number of Characters, Words and Rows in the given Window. No side effects are caused by this predicate, which is used for the gathering of information only.
Built-in Predicates: Editbox Class

**wtext/2 - get or set the window text**

```prolog
wtext(Window, Text)
```

Replace the text of the given Window to the given Text, or get the current Text. For top level and MDI child windows, the text is the window title (style permitting); for "button" and "static" control windows it is the window label, and for "edit" control windows and the "edit" control components of "combobox" windows, it is the entire window contents. Note that, unlike wedittxt/2, this predicate works with all types of window, but instead of replacing the current selection it replaces the entire text.

Example: `wtext((example,8002),`item1`)`. 
Built-in Predicates: Message Box

- **msgbox/4 - display the message box**

```lisp
msgbox(Title, Message, Style, Button)
```

Display a standard Windows message box with a given title, message and style returning the users response to the dialog. The Title argument is a string that sets the message box's window caption. The Message argument is either an atom or string that is the message to be shown to the user. The Style argument is a message box style value that dictates which combination of predefined buttons, icons and modality is used in the message box. The Button argument is a variable that gets bound to an integer indicating which button was used to terminate the dialog.

The predicate succeeds whichever button is clicked, or when RETURN is pressed.
Built-in Predicates: Message Box

Example:

```
?- msgbox('Contoh', 'Message Dialog', 1, Button).
```

![Message Box Example](image)
Built-in Predicates: Message Box

- `message_box/3` - create a message box and return a response

```
message_box(Buttons, Message, Response).
```

Display a Message in a window with the specified Buttons (in the domain \{ok, okcancel, yesno, yesnocancel\}) and return the selected button in Response. The returned selected button may be one of the following: "ok", "cancel", "yes" or "no".
Built-in Predicates: Message Box

Example:

```prolog
?- message_box(okcancel, 'Cubaaab', Response).
```
Built-in Predicates: About Box

- abtbox/3 - display the about box

\[ \text{abtbox(Title, Message, Font)} \]

Displays the "about" dialog box with the given title (window caption) and message, using the given font.

The predicate succeeds if the 'OK' button is clicked or the <return> key is pressed, or fails if the dialog is closed with 'Close' system menu option or the <escape> key is pressed.
Built-in Predicates: About Box

Example:

```prolog
?- abtbox('About Box', 'Contoh About Box', []).
```
Built-in Predicates: About Box

**abtbox/3 styles**

<table>
<thead>
<tr>
<th>Value</th>
<th>Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>fixed IBM PC font with large Prolog bitmap</td>
</tr>
<tr>
<td>1</td>
<td>proportional Windows font with large Prolog bitmap</td>
</tr>
<tr>
<td>2</td>
<td>fixed IBM PC font without bitmap</td>
</tr>
<tr>
<td>3</td>
<td>proportional Windows font without bitmap</td>
</tr>
</tbody>
</table>

Styles 4-7 have the same attributes as above except that the window displayed is wider. In this format a smaller LPA bitmap is displayed on the left hand side of the window and the text is displayed on the right.