

# Eagle GUI Guide

*"A resource guide to development  
of advanced user interfaces  
in Eagle V14"*

Revision history			
Date	Revision	Author	Notes
-	-	M.Codda	First draft
-	-	OB	Several editorial reviews
October 2010	1.10.2	M.Codda	Last update TMDI_CONTEXT_MENU Ribbon Help button
October 2010	1.11.3	OB	Editorial review
3-jan-11	2.00.1	F.Murroni	Index update 9.8.1 Focus-in and Focus-out actions 9.8.2 Set the focus to a control 8.14.1 How to use the PBUTTON command on auto-resizable controls 8.14.2 An interesting example for 8.14.1
12-jan-11	2.00.2	F.Murroni	Page.79 - dock=<i> Page181 - Slider ... The fourth value is the step for the increment GridView – primer “mode=” was wrong GridView – primer “readonly” was missing
19-jan-11	2.00.3	F.M.	PIN=-23 description changed
23-mar-11	2.00.4	F.M.	Spin button to return 1 or -1 for arrow UP/RIGHT or DOWN/LEFT
4-Apr-11	2.01.2	OB	Editorial review and merge of versions
31-May-11	2.01.3	M.Lilli/OB	Review and edit
25-Sep-11	2.01.4	OB	5.1 Command window history title 6.0 Ribbon chapter edits 6.1 New Ribbon MAX MIN settings 6.1.1 New Ribbon Select, Add and Remove Page tabs 6.6 Ribbon Eagle messages handling 9.3 Updated click command options Appendix A configuration new and updated settings

## Contents

<i>1 – Introduction to GUI in Eagle.....</i>	<i>8</i>
1.1 – The importance of a graphic user interface .....	8
1.2 – Developing User Interfaces in Eagle .....	8
1.3 – How to organize a graphic user interface .....	9
1.4 – Adopt the user's perspective .....	9
1.5 – Give the user control .....	9
1.6 – Use real world metaphors.....	10
1.7 – Keep interfaces natural.....	10
1.8 – Keep interfaces consistent .....	10
1.9 – Communicate application actions to the user .....	11
1.10 – Know your target platform.....	11
 <i>2 – The Model for a Graphic Application.....</i>	 <i>12</i>
2.1 – Locator and Pointer .....	13
2.2 – Button Functionality .....	14
2.3 – Menu layout .....	14
2.4 – Messages.....	14
 <i>3 – Eagle's Makeup .....</i>	 <i>15</i>
3.1 – Theme .....	15
3.2 – Docking and Markers .....	21
3.3 – Main Window Features.....	23
3.4 – Mouse shape .....	25
3.5 – Function keys.....	26

3.6 – Splash Window .....	28
<hr/>	
<i>4 – Document Windows.....</i>	<i>32</i>
4.1 – The “Gwindow” Command.....	34
4.2 – “Gwindow” behavior.....	36
4.3 – Multiple Document Interface .....	38
4.4 – Tabbed Document Interface.....	39
4.5 – Tabbed Multiple Document Interface.....	43
4.6 – Floating window .....	44
<hr/>	
<i>5 – Command and Message Windows .....</i>	<i>45</i>
5.1 – The Command window .....	45
5.2 – The Message window.....	49
<hr/>	
<i>6 – Ribbon Bar.....</i>	<i>54</i>
6.1 – Ribbon Bar Structure.....	55
6.1.1 Ribbon Selecting, Inserting and Removing Pages .....	60
6.2 – Ribbon UI philosophy.....	61
6.3 – Polling extensions .....	62
6.4 – Recent documents gallery.....	63
6.5 – Help button .....	66
6.6 – Messages Support.....	66
6.6.1 Restrictions .....	67
<hr/>	
<i>7 – Menu Bar and Popup Menu .....</i>	<i>69</i>
7.1 – Menu Bar .....	69
7.2 – Popup Menu .....	76
<i>8 – Control Bar .....</i>	<i>78</i>

8.1 – How to create a control bar: the panel command .....	79
8.2 – About docking mechanisms .....	83
8.3 – Enabling and freezing panels and controls .....	85
8.4 – Moving inside panels .....	88
8.5 – Coordinates .....	88
8.6 – Destroying panels .....	89
8.7 – Status Bar .....	91
8.8 – Tool Bar .....	94
8.9 – Dialog Bar .....	105
8.10 – FixedSize Bar .....	110
8.11 – Tabbed Bar .....	114
8.12 – User Interface Persistence .....	122
8.13 – Options in a panel .....	126
8.14 – Anchoring and auto-resizable controls .....	130
8.14.1 How to use the PBUTTON command on auto-resizable controls .....	134
8.14.2 An interesting example .....	134
8.15 – Auto-hiding panels .....	135
8.16 – Panels related to a gwindow .....	136
8.17 – Panel without caption .....	139
8.18 – Paged Bar .....	140
 <i>9 – Controls .....</i>	 <i>147</i>
9.1 – Icon Format .....	148
9.2 – Adding and Removing controls .....	149
9.3 – The “click” command .....	150
9.3.1 – Pushed buttons .....	153
9.4 – Set and Get a string .....	155
9.5 – Text Button .....	156

9.6 – Image Button .....	157
9.7 – Label .....	159
9.8 – Edit .....	161
9.8.1 – Focus-in and Focus-out actions .....	164
9.8.2 – Set the Focus to a control .....	164
9.9 – Combo Box .....	165
9.10 – Check Box and Three State Check Box .....	179
9.11 – Radio Group .....	182
9.12 – Slider, Scroll and Zoom Bar .....	184
9.13 – Spin Button and Spin Edit Button .....	188
9.14 – Frame Color .....	190
9.15 – Frame Title .....	192
9.16 – Color Picker .....	194
9.17 – Hyperlink .....	199
9.18 – Progress Bar .....	201
9.19 – Rich Edit .....	202
9.20 – Tree View .....	204
9.20.1 Treeview Appearance Configuration .....	208
9.20.2 Treeview Polling Information and Behaviour .....	210
9.20.3 Treeview Drag & Drop .....	212
9.20.4 Treeview Node Right Click .....	213
9.20.5 Treeview CLICK Behaviour .....	213
9.20.6 Treeview Insert .....	215
9.20.7 Treeview Freezing and Unfreezing .....	216
9.21 – List View .....	217
9.21.1 – Single column .....	219
9.21.2 – Multiple column with header .....	220
9.21.3 – Selection and multiple selection .....	223
9.21.4 – Add a new line .....	224
9.21.5 – Remove a line .....	224

9.21.6 – Standalone list.....	225
9.21.7 – List as button in tab file .....	226
9.22 – Grid View.....	230
9.22.1 – Styles and behavior .....	230
9.22.2 – The gridview command .....	232
9.22.3 – Create and destroy a grid .....	235
9.22.3 – Add columns .....	237
9.22.4 – Insert a row .....	238
9.22.5 – Set content of a cell.....	239
9.22.6 – Fetch the content.....	243
9.22.7 – Cell selection.....	244
9.22.8 – Remove columns and rows.....	246
9.22.9 – Grid polling behavior .....	246
 <i>10 – Dialogs and Message Boxes .....</i>	 <i>250</i>
10.1 – Notify .....	250
10.2 – Prompt command.....	254
10.3 – Font Dialog .....	257
10.4 – File dialog .....	258
10.4.1 – Operating System file dialog.....	258
10.4.2 – Eagle file dialog .....	260
10.5 – Balloon Message.....	265
10.6 – Folder Dialog .....	267
 <i>11 – ActiveX .....</i>	 <i>271</i>
<i>12 – Eagle window inside a panel.....</i>	<i>274</i>
<i>13 – How to customize controls via DLL .....</i>	<i>274</i>
 <i>Appendix A – Configuration file settings .....</i>	 <i>275</i>

## 1 – Introduction to GUI in Eagle

Eagle v.14 offers a new extensible, object-oriented framework that enables you to easily customize and extend user interface (UI) functionality. In the new suite developers are given the opportunity to create the most innovative and up-to-date graphical user interface (GUI) that fits their requirements keeping abreast of the latest in UI trends.

### *1.1 – The importance of a graphic user interface*

A user interface is the bridge between the user and the application. Interface design is not just about making the application look good, it represents the primary set of functions a user will need to learn and use to make the application work productively. There are many different types of interface design and the interface you choose will reflect the target user for your application. If your application is an internal design, the interface may well be a bespoke design, or, if your application is part of a commercial venture you may need to follow a design model current in applications on the target system, i.e. Microsoft Windows. User interface design often takes as much, if not more, thought as the core application functions. This section we give guidelines on how to well design an easy to use product. The effectiveness of an application means that the interaction should be simple and natural. As the designer, you should always keep the user in mind while designing. First and foremost this means, know your user and place yourself whenever possible in their position. Do not assume anything; expect the user to do the unexpected. Remember the user is happy when they can complete a task without having to perform unnecessary complicated input routines or constantly refer to help documentation.

### *1.2 – Developing User Interfaces in Eagle*

The means of programming user interface capabilities to the services and functions of Eagle has significant importance. For that reason, in Eagle v.14 Macrovision has introduced a new GUI Framework, which enables the Eagle programmer to implement Windows applications with a professional and user-friendly interface, rendered with Microsoft Office 2000/XP/2003, Visual Studio 2005 appearance and Office 2007-like ribbon interface. Obviously, as was the case in previous versions, developers can continue to deliver Eagle 3D functions in C/C++ developments using the EPI to functions and datasets, or interfaces based on a web browser technology or Visual Basic application using the EagleActiveX OCX component.

The new graphic tool is fully compatibility with the Eagle v.12 MFC-based version but introduces new controls and functionalities by extending MFC controls rendering and limitations. An application interface which currently works in Eagle v.12 will work on the latest Eagle without changes. Applying the default settings with the default Windows XP style and color scheme it is possible to also maintain the same look and feel for the graphic user interface.



The framework is flexible and capable of adapting to new graphical user interface styles that will be available on new versions of Windows. Microsoft Vista and Windows 7, for instance, introduce new styles and new graphical features.

Eagle windows, panels, lists and all the controls can be viewed with the selected look, using colors and schemes. At the application level, full compatibility is assured and there is no need to port existing applications. If the extended features and appearance of the user interface are desired, then it is necessary to undertake some further localized work on panel creation and icons graphic to implement these features.

Developers integrate can integrate their own controls (DLLs) and components (DLLs and ActiveX) to build more complete and integrated applications that go beyond the 2D/3D graphics supported natively by Eagle.

The Eagle v.14 user interface runs on all current Windows platforms: Vista, XP, 2000 and 2005, 2003 and 2000 Server. The graphic UI style respects the underlying operating system and chosen UI scheme. So, for instance on Windows Vista, the graphic display of Windows controls and windows cosmetics are 100% Vista compliant, with transparency, rounded corners, highly defined fonts, colored shades, enhanced slide bars, and so forth.

### *1.3 – How to organize a graphic user interface*

The implementation of the guidelines illustrated in this section are not always possible, sometimes it is application dependent. Proper use of GUI tools can only make the application better. A good guideline to remember in your mind at all times is; "Every time the user has to consult help documentation, you have failed." By making an application intuitive, you will not only increase the usability of the application but will also gain respect for the application from the user.

### *1.4 – Adopt the user's perspective*

Effective design starts with adopting the user's point of view. This is always difficult to do. Application designers tend to see an application as the implementation of functions, while the user sees an application in terms of its interface. Good design is rooted in an understanding of the user's work. A well designed application solves users' problems, makes their work easier and offers them new capabilities. The two most effective ways to understand the user's work are to involve users in the design and to be a user yourself.

### *1.5 – Give the user control*

Users want and need to be in control of the tools they use to perform their work. The user can be in control when an application is flexible and uses progressive disclosure. A way to increase the user's sense of control is to provide multiple ways of accessing application functions. Flexibility enables user to select the best method of accessing a function, depending to experience level, personal preference or simply habit. The application should also be

configurable. Users like to configure settings and select personal preferences. Use progressive disclosure, meaning that the application should present the vital and most common functions first and in a logical order. The more sophisticated and less frequently used functions should (if space is restricted) be hidden from immediate view, but still be readily available.

### *1.6 – Use real world metaphors*

A good user interface allows the user to transfer skills from real world experiences. Users need to be able to directly manipulate elements of the interface. The direct manipulation model is an object-oriented model. First the user selects an object or a group of objects, and then performs an action on the selected objects. An object-oriented model allows the user to see what elements will be acted on before performing the action. It also allows completion of multiple actions on the selected elements. Another suggestion is to make the application's response as fast as possible. The immediacy of the visual response is crucial to the experience of direct manipulation.

### *1.7 – Keep interfaces natural*

Make navigation easy by providing a straightforward presentation of both the overall work area and the mechanism for moving through it. Moving easily and quickly gives the user a sense of mastery over the application and work. An optimal arrangement of elements on the screen assists the user's decision-making processes and reduces the possibility of errors. Reducing mouse movements is good as it simplifies user's actions. Provide natural shades and colors, minimizing the contrast between screen objects in order to direct the user's attention. Use color as a redundant aspect of the interface, for example to differentiate screen objects.

### *1.8 – Keep interfaces consistent*

Consistency is important. It helps the user transfer familiar skills to new situations. The user can apply the knowledge learned from one application to another, reducing the learning time. Consistency within applications facilitates exploration of new functions. Consistency within applications can be achieved by following these rules:

- similar components should operate similarly;
- the same action should always have the same result
- the function of components shouldn't change based on context
- the position of components shouldn't change based on context
- the position of the mouse pointer should not warp.

Consistency between applications can be achieved by following yet another set of simple rules:

- components should look familiar
- interaction with the applications is similar
- components are organized in a familiar manner between applications.

### *1.9 – Communicate application actions to the user*

Effective applications let the user know what is happening with the application, but without revealing implementation details. Generating a appropriate communication dialogue between the user and the application will increase user satisfaction. Guidelines to reach this target could be:

- User Interface Feedback i.e. give the user feedback (e.g. highlight a selected component, advise the user that a long task is working, etc.)
- anticipate errors (e.g. undo facility, context-sensitive helps, etc.)
- explicit destruction and warning messages

### *1.10 – Know your target platform*

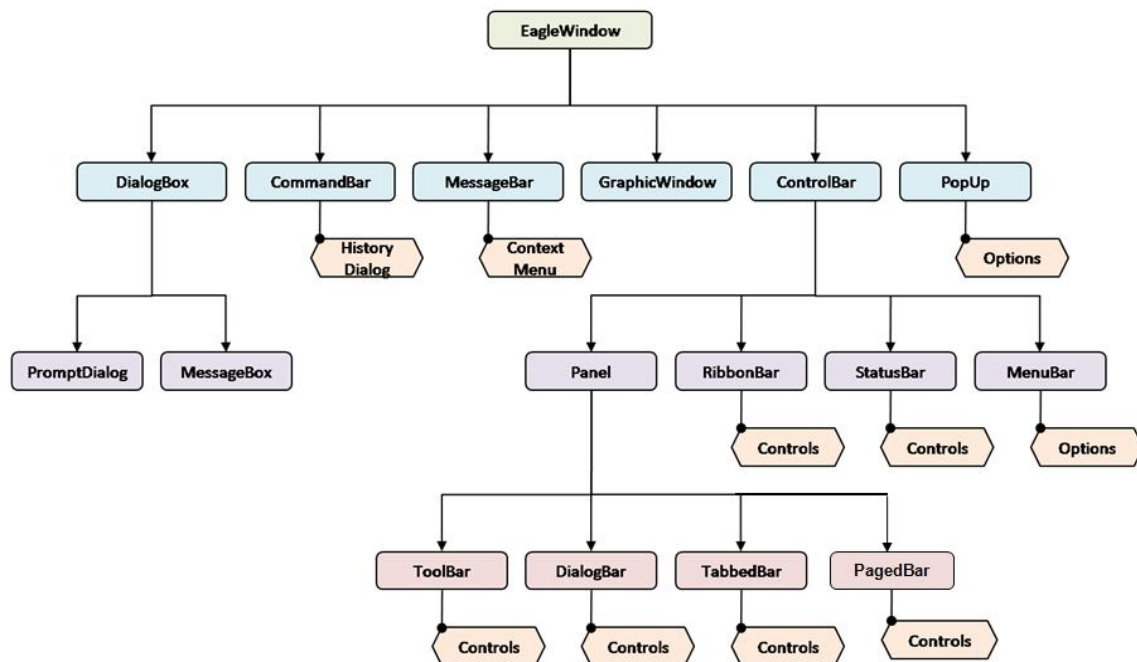
This guide provides a framework of specifications and suggestions for the application designer to implement effective graphic user interfaces that will be consistent with other application user interface models. The target of these guidelines is to establish a common look and feel and a consistent behavior among windowing systems. You will notice that on certain platforms you can utilize interface components (widgets) that may not be available on another windowing system. You should be familiar with the target platform or platforms and make your choice of components accordingly.

Note: You will find that the look and feel of the various images in this guide may vary from your particular system these are operating system or theme specific.

## 2 – The Model for a Graphic Application

A user interface is structured as collection of different items, for example buttons, bars, dialogs and windows. This set of objects can be associated to a model that describes the relationships between every part of a GUI through a hierarchical view.

As the tree-view illustration below shows, the root is the main Eagle's window and starting from here it's possible to find every element used to build an application.



The first level of the structure contains the core components:

- **GraphicWindow** : represents the frames used to hold the graphical environment; a single document interface (SDI) has only one **GraphicWindow** object, whereas a tabbed or multiple document interface (TDI and MDI) may have an object for every frame;
- **CommandBar** : is the command window from which Eagle's instructions are executed;
- **MessageBar** : corresponds to the dialogue window in which Eagle messages are displayed;
- **ControlBar** : embeds the configuration details for all bar types, for instance : toolbar, menubar, Eagle panels (sizable, not sizable, floating, etc.), statusbar and the newer ribbonbar;
- **PopUp** : contains the representation of popup menus;
- **DialogBox** : is the container for items like the Eagle prompt dialog or the Eagle message boxes.

The lower levels of the hierarchy contain the specialization of a particular GUI aspect, such as a toolbar belongs to the "ControlBar" kind but is related directly to the main window.

Many of leaves in the tree illustration have a relation with other objects, for instance:

- The command window has a related to a dialog with the history;
- The message window contains a context menu to execute some shortcut commands;
- All panels, ribbon and statusbar include controls;
- Popup and menubar are collections of options.

At this point let us focus first on the interaction system; there are three objects that are the main interaction tools for the user: the screen, the keyboard and the mouse.

The operation of the mouse implements four actions:

- Locator Action : movement;
- Select Action : select objects, manipulate and activate components of the GUI;
- Adjust Action : relocate components which are not fixed;
- Menu Action : display and choose from popup menus.

The keyboard contains four classes of keys or accelerators (shortcut-keys):

- Standard Keys for the input of characters and numbers;
- Acceptance Keys which will execute the current operation. Normally, this key is the RETURN key;
- Function Keys to assign a predefined string or keystroke sequences to one key;
- Arrow Keys alternative to mouse for moving locator (e.g. text scroll).

It is advisable to pay particular attention to the GUI sections which deal with locator movement, button functionality, GUI components and concepts, paying particular attention to the Panel, Options and Bar commands.

## *2.1 – Locator and Pointer*

During the locator movement the pointer should change its shape as the focus changes; for example, if the pointer is a red arrow when it is inside a graphic window, it changes as soon as it leaves the graphic window and becomes an I-beam shape if it enters a textual window or a black arrow pointing to the upper right corner when an action from a cascade menu must be selected. Also, it should change to a "caution" shape to indicate that an action is expected in another area before an input can be given in the current area. This very much helps the user to quickly and instinctively understand what kind of action they are about to take.

## *2.2 – Button Functionality*

If the first button of the mouse is clicked inside a panel containing components representing a menu, the action will start the function related to that particular component (e.g. a button), while if it is pressed inside the graphic area it means that a graphic object must be selected or de-selected, if it was already selected.

When the user clicks the right button of the mouse, the application should behave differently depending on whether the pointer is on a menu button or inside the graphic area or on the banner of a window. An example of this functionality is to have a context help associated with the right click.

In the second case, if a selection was expected then a menu grouping of all selection options is displayed, otherwise if an input in the co-ordinate system was expected then a menu grouping of all the input options is made. The examples show that different actions can be applied depending on the location of the pointer, be it in the graphics area, window banner, menu, textual window, list, etc.)

## *2.3 – Menu layout*

The layout used to present objects on the screen affects the clarity of the final interface. The main guideline for layout design is to follow the natural use order and the natural scanning order of the users who will be using the application. The components should be positioned so that moving between them is simple and quick while performing the most common tasks. The least amount of pointer movement required, the better it is for the user in terms of productivity and usability. The natural scanning order is most important as it helps the user by arranging small groups of components. The most important and most used commands should be put first. In most cases this order is from left to right and from top to bottom. The Ribbon takes menu layout and consolidation to a new extreme. The ribbon interface model is Microsoft's advanced proposal for logically gathering commands in an application interface based on use and context. Eagle fully supports this method of UI programming.

## *2.4 – Messages*

The application should usually provide a message area for presentation of application messages. Effectively this message area is a log of usage and guidelines. The application should not use the message area for warnings or messages that require immediate action by users. User notification messages should be displayed in DialogBoxes that are implemented in several forms and designed just for that purpose depending to the relevant situation (e.g. error, warning, information, question and working).

## 3 – Eagle’s Makeup

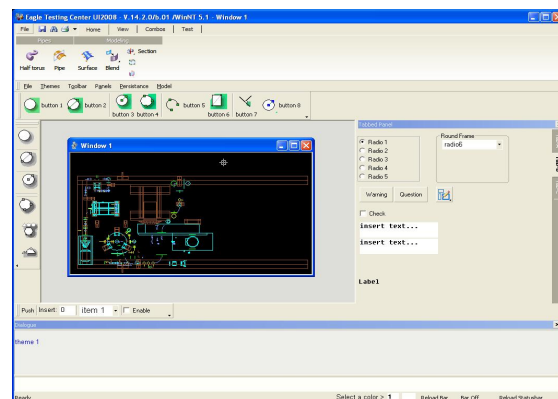
An application should be well defined with a clear and unambiguous user interface and means of identifying items; otherwise users will have difficulty remembering the meaning of component parts. Icons, button graphics and symbols are very useful for visually representing an action or an item without using a large area of the application. Graphics in applications should be kept simple and easy to remember by users, and should identify clearly the objects or concepts they represent.

Designing a GUI with Eagle v.14, the first step is to define the look of the main window. Developers have a powerful suite of settings for this purpose (theme, docking behavior, icon, etc.) to choose the appropriate appearance for your own solution.

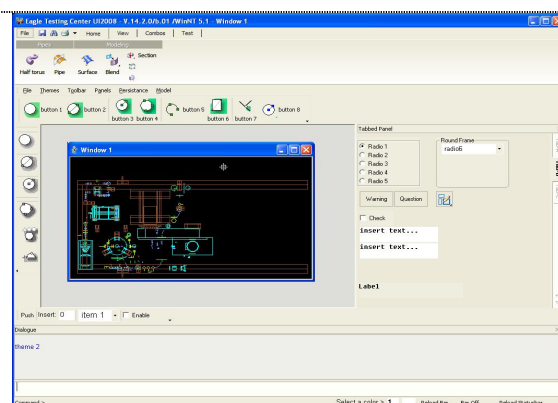
### 3.1 – Theme

The new GUI framework allows the user to customize a large number of components. Firstly we will consider all the available themes. Switching through the various schemas in the following images, we can see that rendering of all the GUI components is dependent on the currently selected theme: the main frame (outer container), the ribbon bar appearance, the foreground and background colors, the caption styles, toolbars, status-bar, the layout, scroll-bars, grippers, and so on. The currently supported themes are as follows:

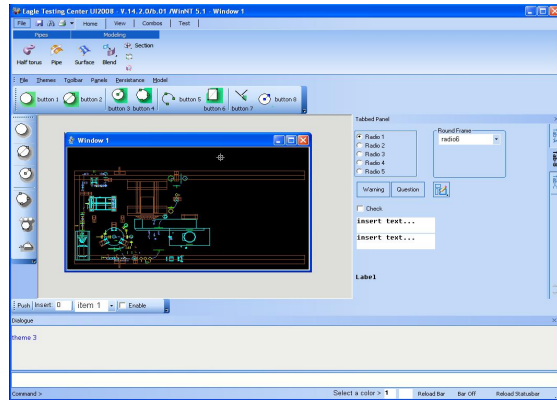
Office 2000



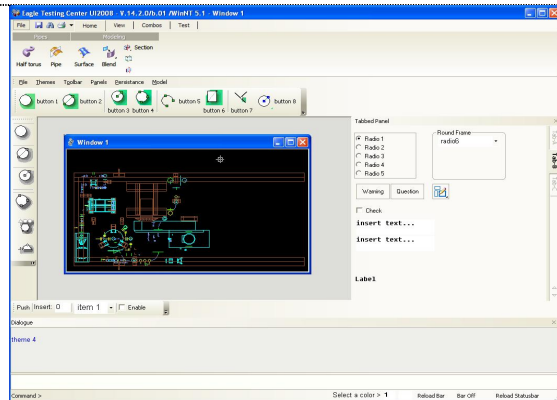
Office Xp



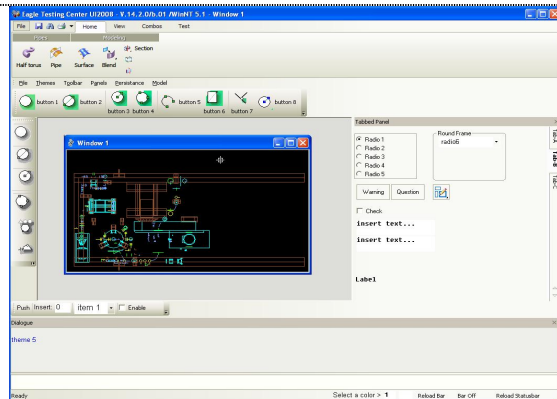
Office 2003



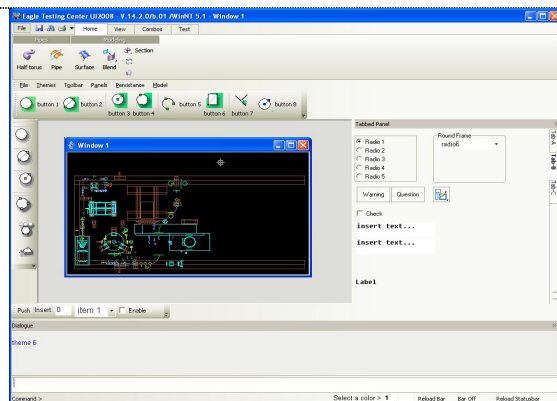
Office 2003 in Windows Classic



Visual Studio 2005

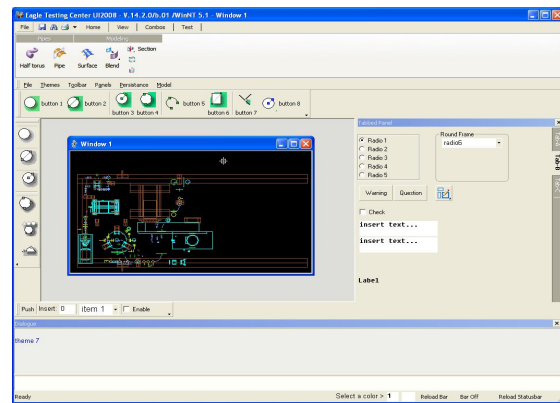


Visual Studio 2008

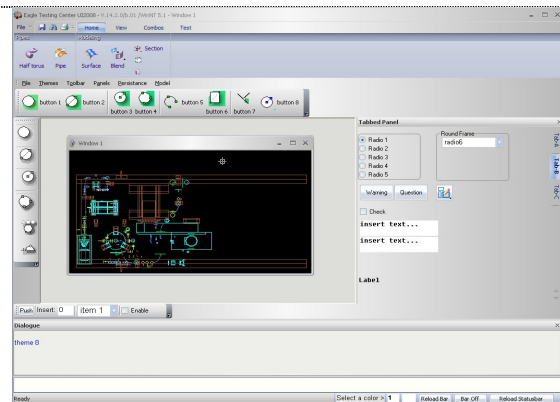




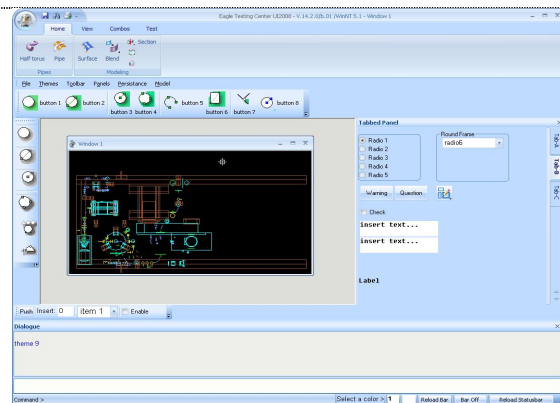
Windows Xp Native



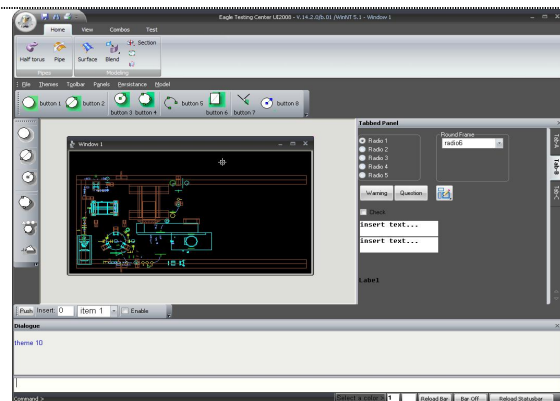
Office 2007 Standard - Release 1



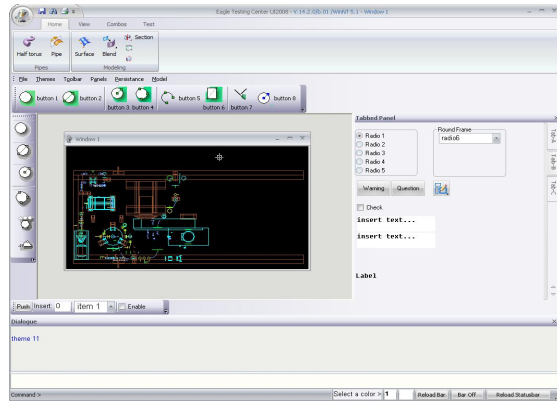
Office 2007 Luna Blue - Release 2



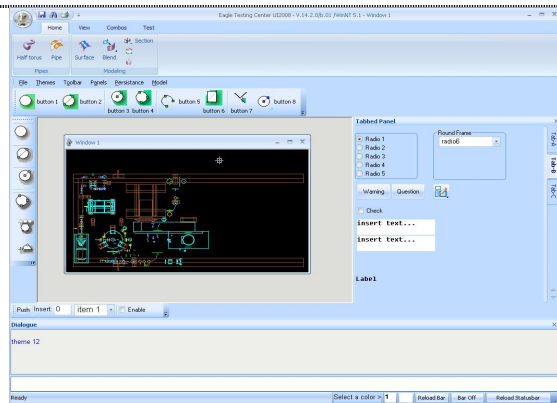
Office 2007 Obsidian - Release 2



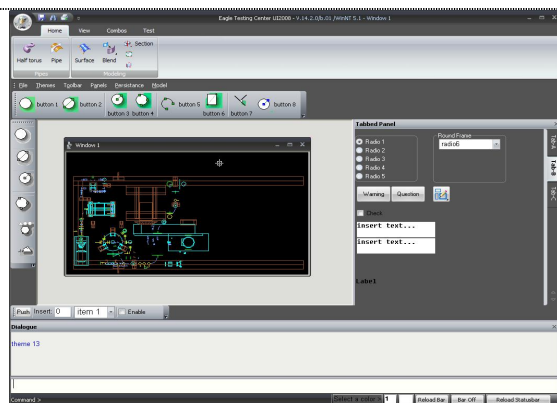
Office 2007 Silver - Release 2



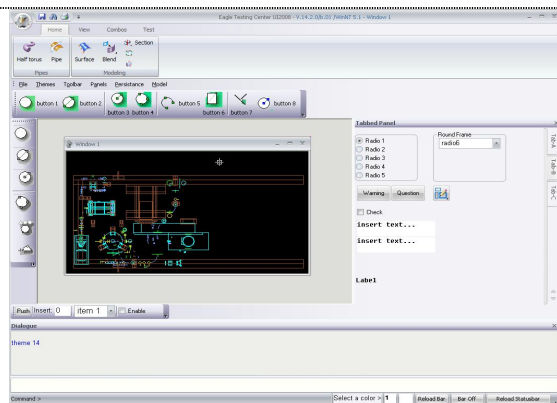
Office 2007 Luna Blue - Release 3



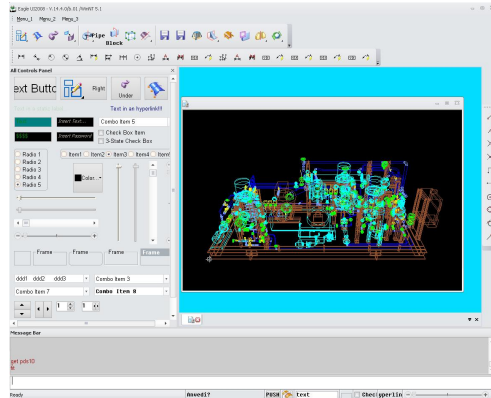
Office 2007 Obsidian - Release 3



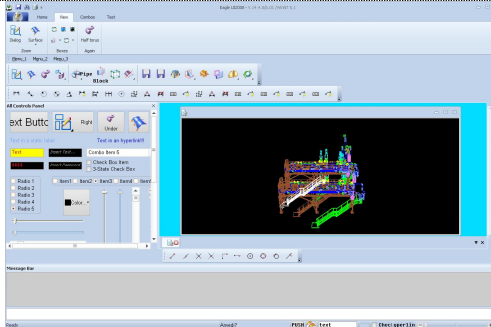
Office 2007 Silver - Release 3



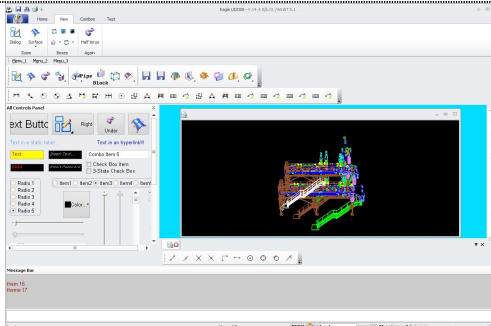
Office 2010 - Release 1



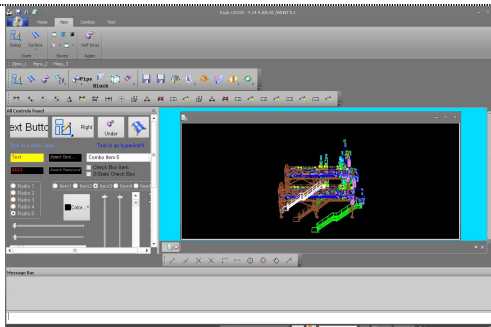
Office 2010 Blue - Release 2



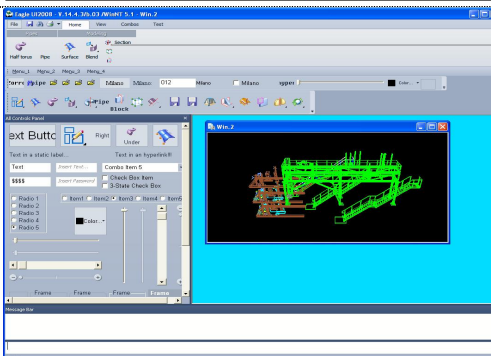
Office 2010 Silver - Release 2



Office 2010 Black - Release 2



Visual Studio 2010



A user can select the default theme at startup by defining the "ROOT\_THEME" entry in the configuration file with the corresponding keyword for the relevant theme. If the option is not specified, the default theme used is the "Office 2007 Luna Blue - Release 2".

INI Sample :

```
ROOT_THEME = NATIVEXP
```

It is also possible to switch theme from within the application by calling the "theme" command with the appropriate index (integer):

Prototype :

```
theme <n>
```

where:

Parameter	Description
<n>	Index (integer) of the theme, according to the next table.

Sample Code :

```
theme 3
```

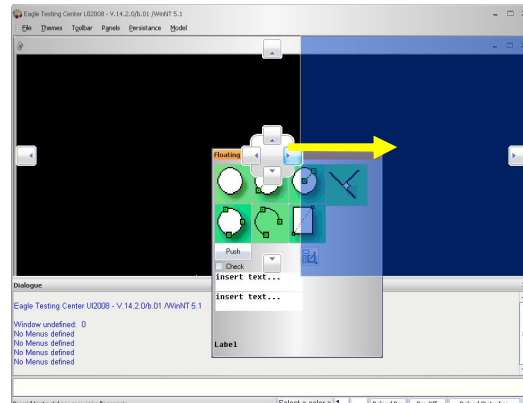
The next table contains the relation between theme, keyword for the ini file and index :

Theme	Index	Keyword in ROOT_THEME
Office 2000	1	OFFICE2000
Office Xp	2	OFFICEXP
Office 2003	3	OFFICE2003
Office 2003 in Windows Classic	4	OFFICE2003NOTHEMES
Visual Studio 2005	5	STUDIO2005
Visual Studio 2008	6	STUDIO2008
Windows Xp Native	7	NATIVEXP
Office 2007 Standard - Release 1	8	OFFICE2007_R1
Office 2007 Luna Blue - Release 2	9	OFFICE2007_R2_LUNABLU
Office 2007 Obsidian - Release 2	10	OFFICE2007_R2_OBSIDIAN
Office 2007 Silver - Release 2	11	OFFICE2007_R2_SILVER
Office 2007 Luna Blue - Release 3	12	OFFICE2007_R3_LUNABLU
Office 2007 Obsidian - Release 3	13	OFFICE2007_R3_OBSIDIAN
Office 2007 Silver - Release 3	14	OFFICE2007_R3_SILVER
Office 2010 - Release 1	15	OFFICE2010_R1
Office 2010 Blue - Release 2	16	OFFICE2010_R2_BLUE
Office 2010 Silver - Release 2	17	OFFICE2010_R2_SILVER
Office 2010 Black - Release 2	18	OFFICE2010_R2_BLACK
Visual Studio 2010	19	STUDIO2010

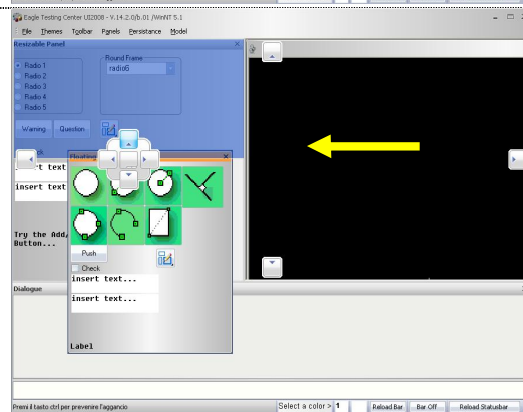
### 3.2 – Docking and Markers

Docking markers, first introduced in Visual Studio 2005, help to see where a dialog bar, you are dragging over the screen, is about to be docked. The docking position of the dialog bar can be relative to the main window or to another docked bar, for instance:

Docking position  
relative to the main window :

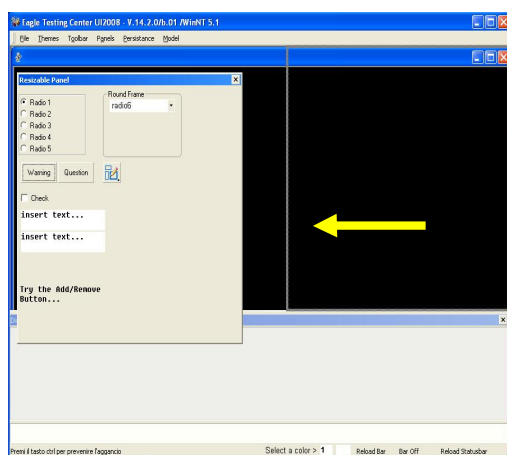


Docking position  
relative to another bar :

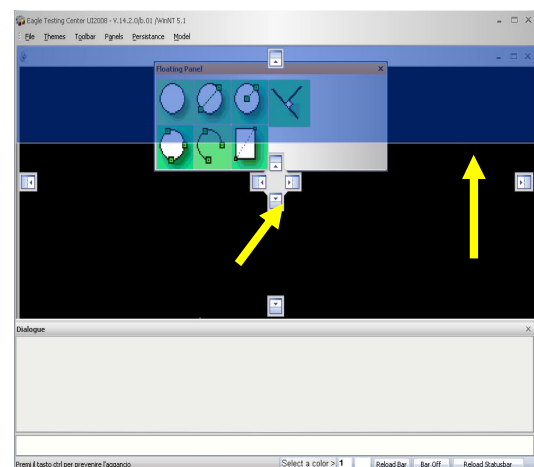


Eagle V.14 supports the following docking marker styles :

- Behavior set by the theme : docking markers follow the current theme, so when using themes from Office 2000 to Windows Xp native markers aren't shown and the docking mechanism for panel is the same as in Windows; on the other hand with Office 2007 themes markers are active.

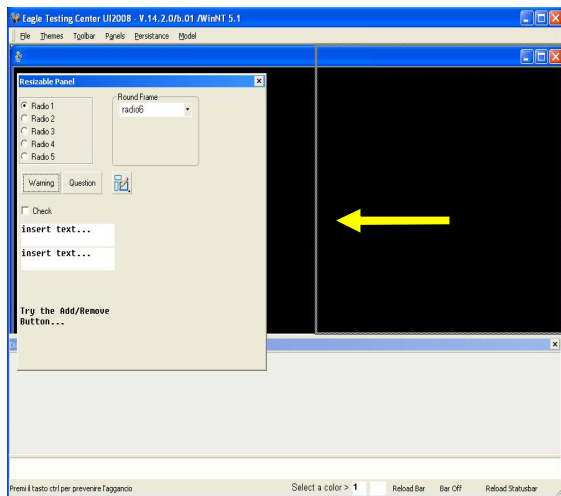


Themes less or equal Window Xp Native

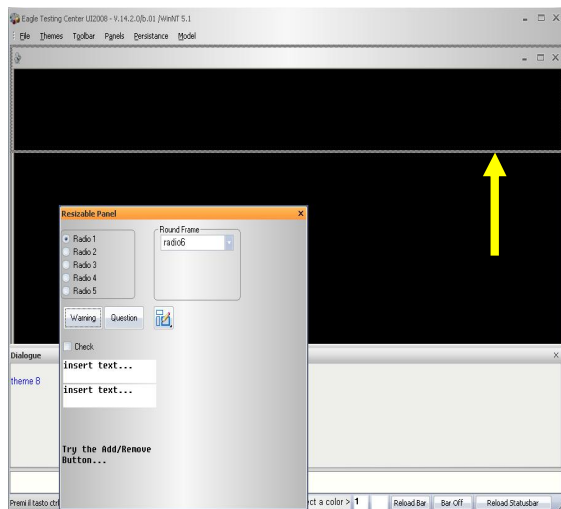


Themes greater or equal to Office 2007 R1

- Mode used by Visual Studio 2003 : for all themes, docking markers follow the style present in “Visual Studio 2003”.

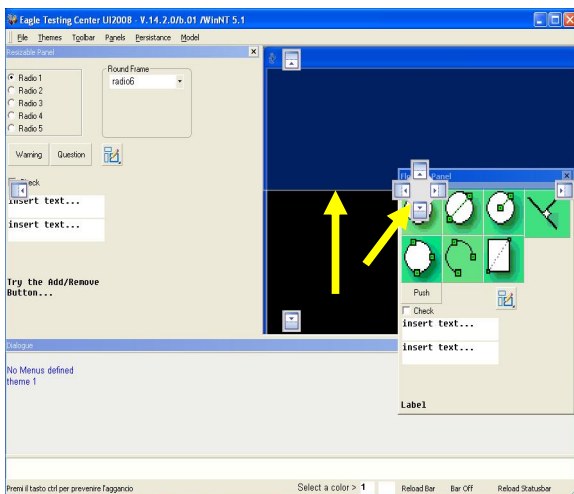


Themes less or equal Window Xp Native

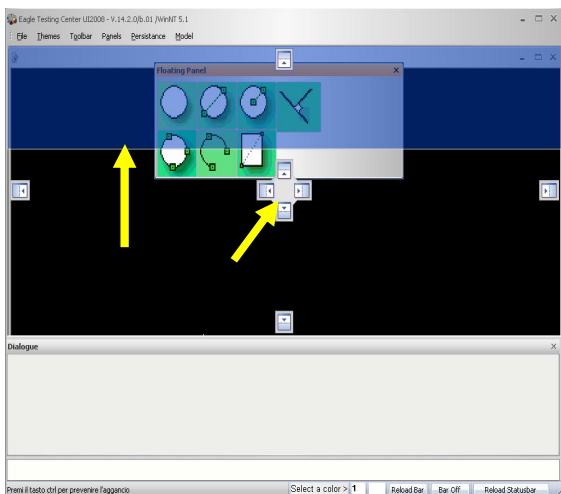


Themes greater or equal to Office 2007 R1

- Mode used by Visual Studio 2005 : for all themes, docking markers follow the style present in “Visual Studio 2005”.

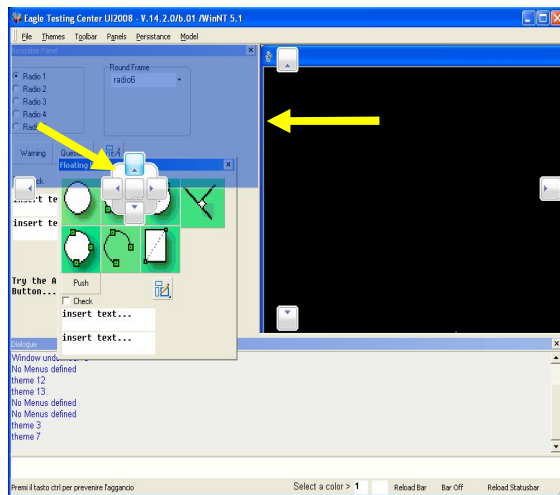


Themes less or equal Window Xp Native

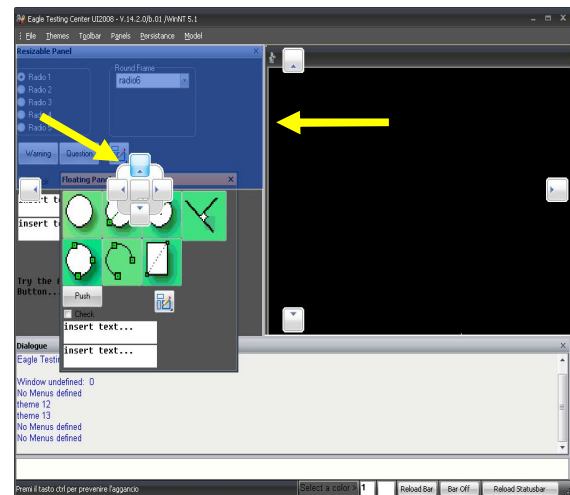


Themes greater or equal to Office 2007 R1

- Mode used by Visual Studio 2008 in Windows XP : for all themes, docking markers follow the style present in “Visual Studio 2008” and used with the behavior of Windows Xp.
- Mode used by Visual Studio 2008 in Windows Vista : for all themes, docking markers follow the style present in “Visual Studio 2008” and used with the behavior of Windows Vista.



Themes less than or equal Window Xp Native



Themes greater than or equal Office 2007 R1

The item "DOCKING\_MARKERS\_TYPE" in the configuration file allows setting docking markers style and behavior (take a look of the next table).

Docking Markers Type	Keyword in DOCKING_MARKERS_TYPE
<i>Regulated by the theme</i>	BYTHEME
<i>Visual Studio 2003</i>	STUDIO2003
<i>Visual Studio 2005</i>	STUDIO2005
<i>Visual Studio 2008 on Windows Xp</i>	STUDIO2008XP
<i>Visual Studio 2008 on Windows Vista</i>	STUDIO2008VISTA

INI Sample :

DOCKING\_MARKERS\_TYPE = STUDIO2005

### 3.3 – Main Window Features

In addition to changing the theme's aspect or the docking behavior, Eagle v.14 allows you to customize a large part of the main window elements using the configuration file settings (usually called INI file) :

- "ROOT\_START\_MAXIMIZED" : to start the main window maximized. The possible values are yes or no (default)

INI Sample :

ROOT\_START\_MAXIMIZED = no | yes



- "ROOT\_BACKGROUND\_COLOR" : to change the color of the background on the main window;

Note : The color will be specified using a "#" followed with three hexadecimal values (one for red, one for green and one for blue), so the range of possible values is : #000000 to #FFFFFF

INI Sample :

```
ROOT_BACKGROUND_COLOR = #777710
```

If ROOT\_VACKGROUND COLOR is set from within and Eagle session the newly set color is triggered when the next THEME command is issued.

Sample Code :

```
env ROOT_BACKGROUND_COLOR=#004466
Theme 9
```

- "ROOT\_MINIMIZEBOX\_ENABLED" : to enable/disable the minimize button;
- "ROOT\_MAXIMIZEBOX\_ENABLED" : to enable/disable the maximize button;
- "ROOT\_CLOSE\_ENABLED" : to enable/disable the close button;
- "ROOT\_RESIZE\_ENABLED" : to enable/disable the resizing for the main window;

INI Sample :

```
ROOT_MINIMIZEBOX_ENABLED = no | yes
ROOT_MAXIMIZEBOX_ENABLED = no | yes
ROOT_CLOSE_ENABLED = no | yes
ROOT_RESIZE_ENABLED = no | yes
```

- "ROOT\_SIZE\_ACTION" : is the action performed when the main window has been resized;
- "ROOT\_CLOSE\_ACTION" : action called before closing the application;
- "ROOT\_CAPTION\_ICON" : used to set the icon on the main window. The icon file must contain two formats, a 16x16 image to representing small icons (i.e. in the caption of the window) and a 32x32 image to represent large icons (i.e. in the "ALT + TAB" system menu).

INI Sample :

```
ROOT_SIZE_ACTION = C:\Actions\sizeroot.cmd
ROOT_CLOSE_ACTION = C:\Actions\closeroot.cmd
ROOT_CAPTION_ICON = C:\Mylcon\Eagle.ico
```



- "TITLE" : to add a title to the Eagle based application;

INI Sample :

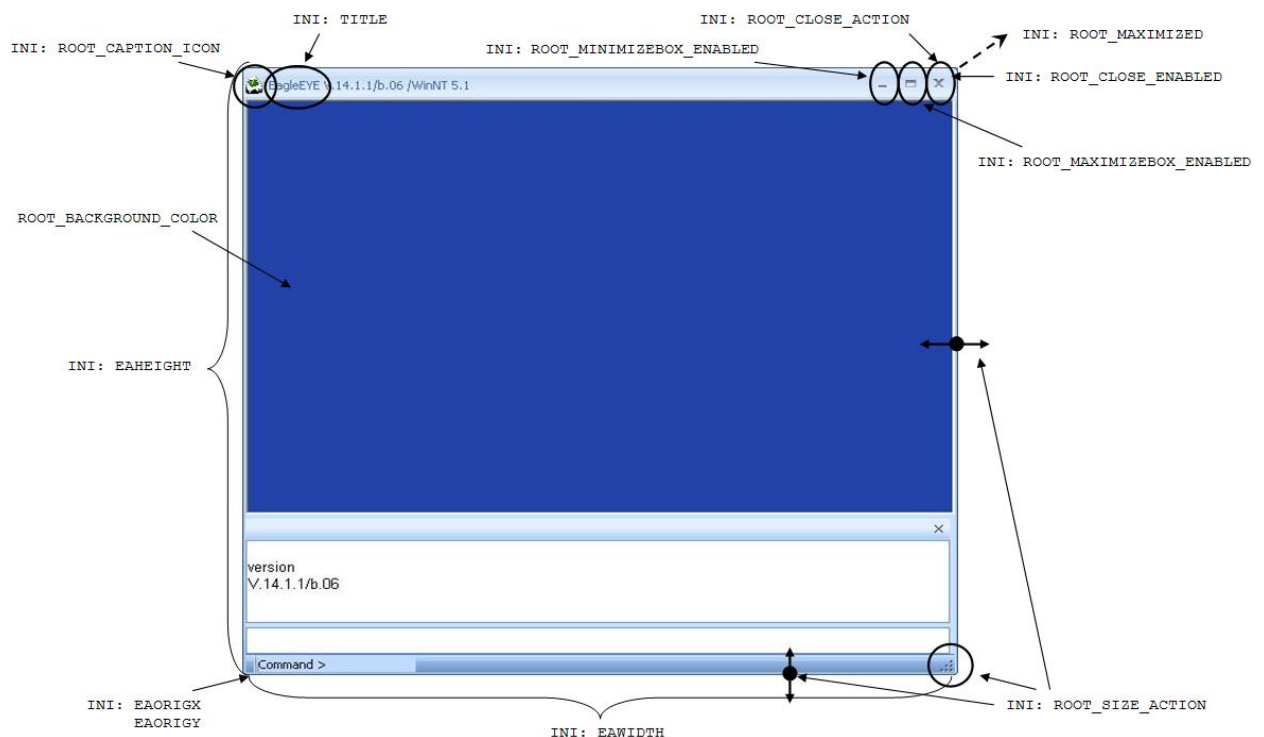
TITLE = MyApplication

- "EAHEIGHT" : represents the window height at start up;
- "EAWIDTH" : is the start up width of the main window.

INI Sample :

EAWIDTH=900  
EAHEIGHT=700

The next figure illustrates what is possible to modify in the main window using the configuration file :



### 3.4 – Mouse shape

In Eagle you can dynamically set the shape of the mouse using the "shape" command; so it's possible to select the cursor index from one of the 12 cursors available under the current Windows installed schema.

This command refers to windowing system cursor and should not be confused with the Eagle internal cursor. They are entirely independent it is possible for both to be visible during an interactive session.

Prototype :	<code>shape cur=&lt;i&gt;</code>
-------------	----------------------------------

where:

Parameter	Description
<code>cur=&lt;i&gt;</code>	cursors are defined 1-12 i.e. on the default Windows schema cur=12 will be the hourglass.

Sample Code :	<code>shape cur=12</code>
---------------	---------------------------

Cursors can be either monochrome or color, and either static or animated. The type of cursor used on a particular computer system depends on the system's display. Old displays such as VGA do not support certain colors or animated cursors, but newer displays (which display drivers use the DIB engine) support them.

Windows provides a set of standard cursors that are available for any application to use at any time. The Windows header files contain identifiers for the standard cursors.

1. Standard arrow
2. Crosshair
3. Text I-beam
4. A square with a smaller square inside its lower right corner
5. Four pointed arrow NSEW
6. Stop symbol
7. Double pointed cursor with arrows facing NE and SW
8. Double pointed cursor with arrows facing N and S
9. Double pointed cursor with arrows facing NW and SE
10. Double pointed cursor with arrows facing W and E
11. Vertical up arrow
12. Hourglass

### 3.5 – Function keys

The function keys define shortcuts to execute commands quickly pressing a single key; normally every keyboard have at least 12 function keys, so Eagle allows to assign a string to each one.

Using the command "funkey" the user is able to define the correspondent command for a function key:

Prototype :	<code>funkey &lt;i&gt;,&lt;text&gt; &lt;file&gt;</code>
-------------	---

where:

Parameter	Description
<b>i</b>	The number of the function key to be assigned, the range of <i> is installation dependent.
<b>text</b>	A sequence of alphanumeric characters. If the text contains a semi-colon, it should be enclosed by single quotes.
<b>file</b>	Assign a range of funkey keys with assignments defined in a text files. The default file extension is TAB.

For example:

Sample Code :

```
funkey 2,comp01
funkey fassgn
```

Note: function key 1 cannot be assigned; it is automatically used by Eagle to open the Help file as defined in the INI file setting USERHELP.

A semi-colon as the last character in a function key assignment will be interpreted as RETURN (newline).

The F1 key is mapped as a common system action for help files and this should be noted when developing applications. There is an entry in the configuration file called USERHELP which when set takes precedence by calling a custom application help file over the standard Eagle Help file.

INI Sample :

```
USERHELP=C:/Program Files/MyApp/MyApp.chm
```

Function key assignments are only available during a polling loop. When a function key is pressed the specified action is executed and, in addition, the VB variable is set to "10 + the key index".

F4 pressed :

```
VB = 14
```

In addition Eagle has a function to display the current string values of function keys :

Prototype :

```
keys { l | p } | { f=<file> }
```

where:

Parameter	Description
<b>l   p</b>	Send the listing to the printer.

**f= <file>**

Send the output to a text file. The default file extension is TAB.

With no parameters list the details in the Message Bar window. For example :

Sample Code :

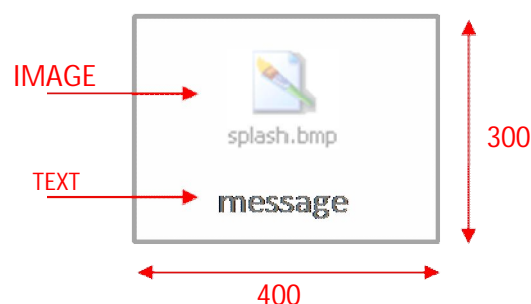
```
keys
keys l
keys f=fkeys
```

### 3.6 – Splash Window

A “splash window” is a message or information dialog about the program contents that appears while an Eagle program is loading and disappears once the application start-up has been completed; For example:



The splash contains an image, normally a bitmap with the dimensions of 400 pixels in width and 300 pixels in height, and text which displays the loading sequence.



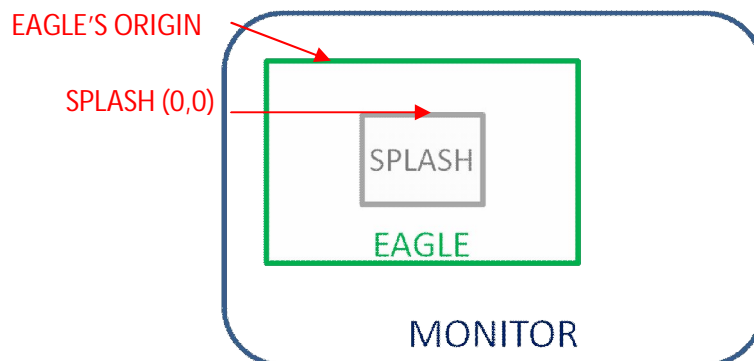
By default, Eagle displays a set of messages in the text field, but developers may add other custom script messages using the “splash” command before closing the window

The following table lists the order of messages:

1	Application Initializing
2	Initializing the Command Manager
3	Loading the Configuration File
4	Creating the Control Bars
5	Configuring Environment and Workspace
6	Applying the Docking Behavior
7	Defining the System Appearance
8	Loading a Startup Command File
9	Add through the “splash” command
10	Complete

In order to load the splash screen the “SPLASH\_WINDOW\_ENABLE” entry in the configuration file must be set to “yes” and to close the dialog, once set and loaded, the “splash off” command must be called.

The dimension of the window is guided by the size of the background bitmap which is normally 400x300 pixels in size, and its position is always at the center of the Eagle frame. If Eagle starts maximized the splash will be placed on the center of the screen.



The “splash” command can be concentrated into :

Prototype :	splash off   t=< text >
-------------	-------------------------

where:

Parameter	Description
off	Remove the splash from the screen.
t=< text >	Message string to display in the window

For example:

Sample Code :

```
splash off  
splash t='Loading a cmd file'
```

Eagle v.14 permits changing of the aspect of the “splash window” by using entries in the configuration file settings (commonly called INI file but these may be set in other ways such as environment variables) :

- “SPLASH\_WINDOW\_ENABLE” is used to enable the splash screen, the default setting is “no”;

INI Sample :

```
SPLASH_WINDOW_ENABLE = yes | no
```

- “SPLASH\_WINDOW\_BITMAP” is the image to use as the background of the Splash window. If the image is not specified or the file doesn’t exist Eagle uses a default bitmap;

INI Sample :

```
SPLASH_WINDOW_BITMAP = C:\res\splash.bmp
```

- “SPLASH\_WINDOW\_SLEEP” : the minimum time (in milliseconds) to show a message; the default is 0 milliseconds;

INI Sample :

```
SPLASH_WINDOW_SLEEP = 10
```

- “SPLASH\_WINDOW\_TEXT\_COLOR” is used to change the color of the text displayed, the default color is “black”.

Note : The color will be specified using a “#” followed with three hexadecimal values (one for red, one for green and one for blue), so the range of possible values is : #000000 to #FFFFFF

INI Sample :

```
SPLASH_WINDOW_TEXT_COLOR = #777710
```

- “SPLASH\_WINDOW\_FONT\_NAME” is used to set the font name of the text displayed. If not specified the default font will be applied.

INI Sample :

```
SPLASH_WINDOW_FONT_NAME = "Calibri"
```

- “SPLASH\_WINDOW\_FONT\_SIZE” is used to set the size of the text displayed. If not specified the default font will be applied.

INI Sample :

```
SPLASH_WINDOW_FONT_SIZE = 12
```

- “SPLASH\_WINDOW\_FONT\_BOLD” is used to apply the “bold” style to the text displayed, the default is “no”.

INI Sample :

```
SPLASH_WINDOW_FONT_BOLD = no | yes
```

- “SPLASH\_WINDOW\_FONT\_ITALIC” is used to apply the “*italic*” style to the text displayed, the default is “no”.

INI Sample :

```
SPLASH_WINDOW_FONT_ITALIC = no | yes
```

- “SPLASH\_WINDOW\_FONT\_UNDERLINED” is used to apply the “underlined” style to the text displayed, the default is “no”.

INI Sample :

```
SPLASH_WINDOW_FONT_UNDERLINED = no | yes
```

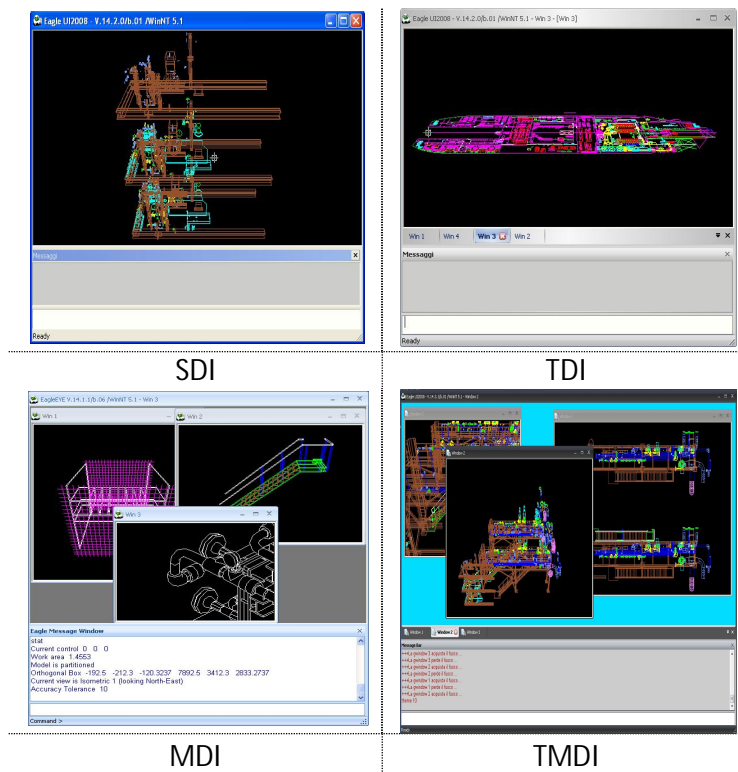
## 4 – Document Windows

A Document Window (in Eagle called “gwindow” or Graphic Window) represents the area where the graphics will be created. This can be a standard Eagle vector and raster graphics window, OpenGL 2D and 3D graphics window, or ActiveX.

Eagle v.14 significantly expands flexibility for application developers. Previously Eagle worked in a SDI (Single Document Mode) much the same as applications like Microsoft Word®, whereas the latest version adds the opportunity to develop applications with Multiple Document Interface (MDI), Tabbed Document Interface (TDI) or Tabbed Multiple Document Interface (TMDI). Typically multiple windows can contain different views of the same model but they may also contain entirely different models or even different ActiveX processes. In this section we will concentrate on the Multiple Window and Multiple Model modes of operation.

Eagle can operate with four document modes:

1. Single Document Interface(SDI), means a Single Document Interface, which is the same behavior of previous Eagle versions;
2. Tabbed Document Interface(TDI), multi-windows layout based on a single container with multiple tabs, one for each document window;
3. Multiple Document Interface(MDI), means Multiple Document Interface, which is the multi-windowed document configuration;
4. Tabbed Multiple Document Interface(TMDI), a hybrid multi-windows layout which has a container with multiple tabs, one for each document window;





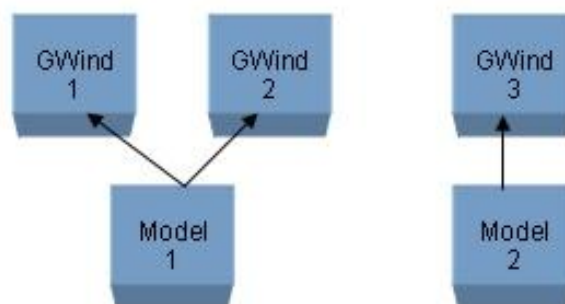
In MDI, TMDI and TDI modes, the application can handle multiple models and multiple windows on each stack (model document representation in memory), while in SDI mode only one model and one window can be used like in previous Eagle versions. The main difference between MDI/TMDI and TDI mode is that, different windows can be visible at the same time in the MDI/TMDI mode, while they are exclusively displayed one at a time, depending on the selected tab, in the TDI mode.

Before starting the application, users can chose to operate in any one of these modes by placing the relevant argument in the line command; "-sdi" to use the single document interface, "-tdi" for the tabbed document interface, "-mdi" to have the multiple document interface and "-tmdi" to have the tabbed multiple document interface. If no argument is specified the default value is "-sdi". When Eagle starts, *gwindow 1* (the default graphics document window) is automatically created as a maximized window with no title.

During the execution, developers are able to know the selected mode through the "system arithmetic information" `sys(46)` :

- `sys(46) = 0` → SDI mode
- `sys(46) = 1` → MDI mode
- `sys(46) = 2` → TDI mode
- `sys(46) = 3` → TMDI mode

Eagle v.14 supports multi-graphic windows and multi-model environments. This means that an application can open more models at the same time, use different stacks (the Eagle term to describe the data in memory) and can display each of them in a single graphic-window or in more than one graphic-window with different viewing parameters.



Essentially, we have a 1-M (One-to-Many) relationship between stacks (models) and graphic-windows. This relationship is not static but can be changed dynamically. Using the `STACK` command enables you to set the active stack on which the program is to operate and the `GWINDOW` command does the same for the active window. The combination of the two commands allows us to create a powerful range of application handling enhancements.

So in the following example by writing the following:

Sample Code :

```
gwindow w=1
stack 1
move p0
get $mymodel
fit
```

we set a relationship between STACK 1 and GWINDOW 1, which means that GWINDOW 1 will show STACK 1 data.

If we need to associate the current stack to another window, then we can simply redefine the current gwindow assignment

Sample Code :

```
gwindow w=2
```

whilst keeping STACK 1 active. In this last case, we now have two windows associated with STACK 1.

This is the general flow of what happens when using the macro language, the part handled by the application code. You also have the possibility to interactively change the active window, therefore consequently the active stack, by clicking the caption bar on top of a graphic-window.

#### 4.1 – The “Gwindow” Command

Using GWINDOW we can also reopen, position, resize, arrange or close a graphic-window. The current or active stack is associated to the new created graphic-window.

Graphic windows can be created, deleted, resized, activated interactively (through standard mouse operations) or by using the GWINDOW command. Each window can be turned into OpenGL mode with the OPENGL ON|OFF command, but there can be only one OpenGL window at the time in the Eagle session. Eagle supports up to 10 graphic windows.

Prototype :

```
gwindow <i>{,POS=x,y}{,WIDTH=<i1>,HEIGHT=<i2>}{,TITLE='<text>'}
{,CASCADE | TILEHORIZONTAL | TILEVERTICAL}
{,MAXIMIZE | MINIMIZE | RESTORE | ICONIZE}
{,DELETE}
{,PROGID='<text>'}
{,S=<i3>}
```

If Eagle starts in SDI (Single Document) mode, the GWINDOW command is not enabled, but the STACK mechanism will still be available.

The TDI mode is, in terms of behavior and use of stacks, basically the same as the MDI or the TMDI mode. There are some GWINDOW options, such as cascading for example, that do not

have any relevance to tabbed windows and if programmed will be ignored. In all other ways the choice of TMDI, MDI or TDI is simply a matter of application preference.

Parameter	Description	SDI	TDI	MDI	TMDI
<i>	A number to identify the graphic or child window.	✗	✓	✓	✓
POS=x,y	Define the position (x and y coordinates) of the child window.	✗	✗	✓	✓
WIDTH=<i1>	Resize the width of the defined graphic window. Unit in pixels.	✗	✗	✓	✓
HEIGHT=<i2>	Resize the height of the defined graphic window. Unit in pixels.	✗	✗	✓	✓
TITLE='<text>'	Define the title text for the defined graphic window. The effect of this option is dependent on the mode in which Eagle is working, MDI (Multiple document interface) mode will display the title in the window title bar, whereas in TDI (Tabbed document interface) mode the title will be displayed in the gwindow tab.	✗	✓	✓	✓
CASCADE	Arrange the current graphic windows by cascading.	✗	✗	✓	✓
TILEHORIZONTAL	Arrange the current graphic windows by horizontal tiling.	✗	✗	✓	✓
TILEVERTICAL	Arrange the current graphic windows by vertical tiling.	✗	✗	✓	✓
MAXIMIZE	Maximize the current graphic windows.	✗	✗	✓	✓
MINIMIZE	Minimize the current graphic windows.	✗	✗	✓	✓
RESTORE	Restore the current graphic windows.	✗	✗	✓	✓
ICONIZE	Iconize the current graphic windows.	✗	✗	✓	✓
DELETE	Delete or close the defined graphic window.	✗	✓	✓	✓
PROGID='<text>'	Set a relationship between a GWIND and an ActiveX prog_id from Windows. In this way existing ActiveX components can be integrated directly inside an Eagle application.	✗	✓	✓	✓
S=<i3>	Set a relationship between a STACK (model data in memory) and a graphic window.	✗	✓	✓	✓
FIX	Parameter to bring the gwindow window inside the Main Frame <i>Note 1</i> .	✗	✗	✓	✓
FLOAT	Command to move the window outside the Main Frame. <i>Note 1</i>	✗	✗	✓	✓

*Note 1:* Refer to [section 4.6](#) below for details for FIXed and FLOATing gwindows.

Sample Code :

```
GWIND 1,pos=200,300,w=200,h=100,t='W1',
progid='ACTIVEXTEST.ActiveXTestCtrl.1'
GWIND 1,delete
GWIND 1
GWIND 1,s=2
GWIND 1,t='title'
GWIND 1,restore
GWIND 1,pos=0,0, w=500, h=500
GWIND 1,maximize
```

Operating with TDI, MDI or TMDI mode, keyboard Windows special keys, such as Alt-tab or Ctrl-tab for switching from one child window to another, are supported. So, graphic windows can be created, activated and closed (deleted) through the command language or mouse interactions.

## 4.2 – “Gwindow” behavior

Eagle v.14 additionally enables control of a large set of actions performed by a Graphic Window, where events can be defined through the configuration file. The list below describes the available features:

- “EAGLE\_DOCUMENT\_ICON” : this item permits setting of the icon for GWindows. The icon must be in a “.ico” file; If the parameter is not specified, no icon is installed;

INI Sample :

```
EAGLE_DOCUMENT_ICON = C:\Mylcon\gwindoc.ico
```

- “DEFAULT\_GWINDOW” : when this variable is defined as “yes”, then at start up the first gwindow is automatically created. If the parameter is not specified then no gwindow is initialized until specifically initiated. The default value is “no”;

INI Sample :

```
DEFAULT_GWINDOW = no | yes
```

- “GWINDOW\_INITIAL\_MAXIMIZED” : when it’s defined “yes”, at start up the first gwindow has created maximized. If the parameter is not specified, the default value is “no”;

INI Sample :

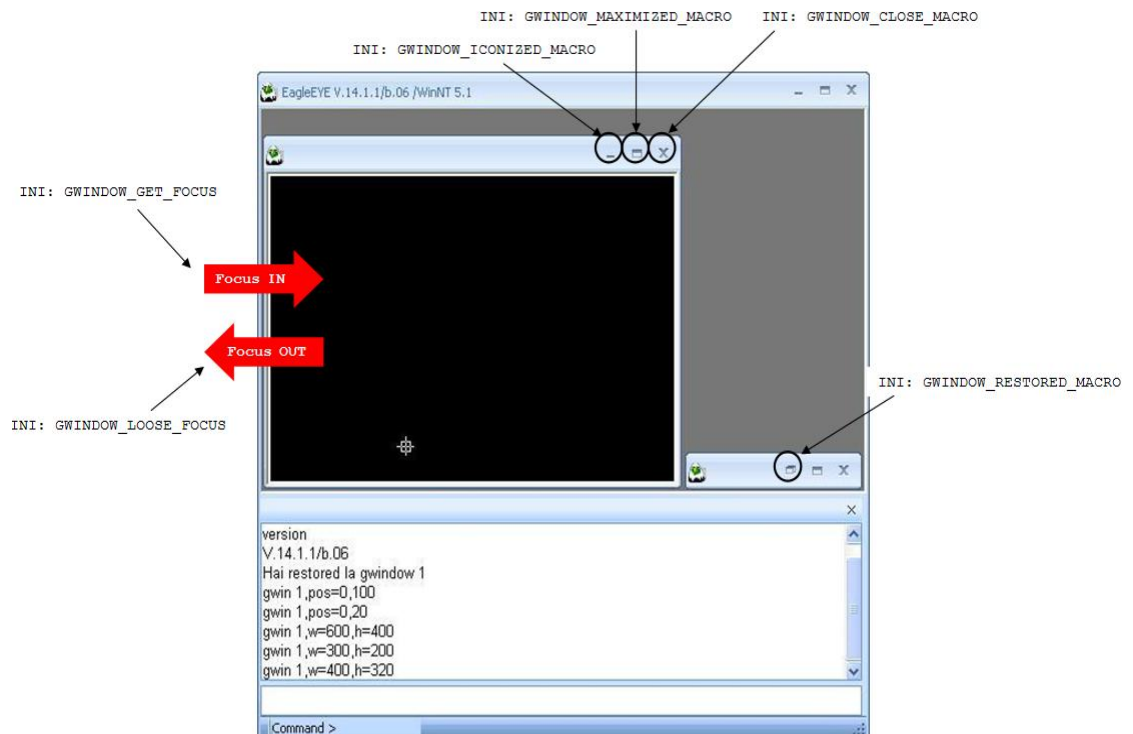
```
GWINDOW_INITIAL_MAXIMIZED = no | yes
```

- “GWINDOW\_CLOSE\_MACRO” : action called when a gwindow has been closed;

- "GWINDOW\_ ICONIZED\_MACRO" : action called when a gwindow has been minimized;
- "GWINDOW\_ RESTORED\_MACRO" : action called when a gwindow has been restored;
- "GWINDOW\_ MAXIMIZED\_MACRO" : action called when a gwindow has been maximized;
- "GWINDOW\_ LOOSE\_FOCUS" : action called when a gwindow has loosen the focus;
- "GWINDOW\_ GET\_FOCUS" : action called when a gwindow has kept the focus;

INI Sample :

```
GWINDOW_CLOSE_MACRO=C:\Actions\closeWnd.cmd
GWINDOW_ICONIZED_MACRO=C:\Actions\minimize.cmd
GWINDOW_RESTORED_MACRO=C:\Actions\restore.cmd
GWINDOW_MAXIMIZED_MACRO=C:\Actions\maximize.cmd
GWINDOW_LOOSE_FOCUS=C:\Actions\loosefocus.cmd
GWINDOW_GET_FOCUS=C:\Actions\getfocus.cmd
```



The figure above illustrates these events in the case of an MDI configuration. Obviously, these actions are only fully available when in MDI mode and are partially available when in the TDI mode. Take a look at the next figure to understand how use these features. If an action doesn't have a related file setting then the event is not handled within Eagle.

INI Parameter	SDI	TDI	MDI	TMDI
EAGLE_DOCUMENT_ICON	✗	✓	✓	✓
GWINDOW_INITIAL_MAXIMIZED	✗	✗	✓	✓
GWINDOW_CLOSE_MACRO	✗	✓	✓	✓
GWINDOW_ICONIZED_MACRO	✗	✗	✓	✓

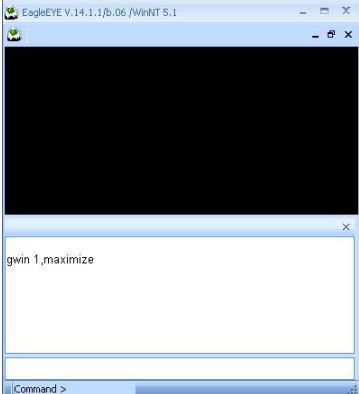
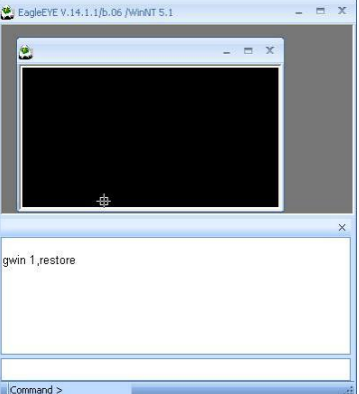
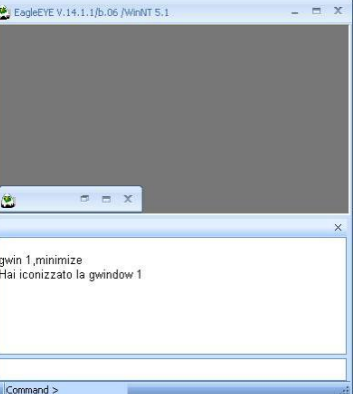
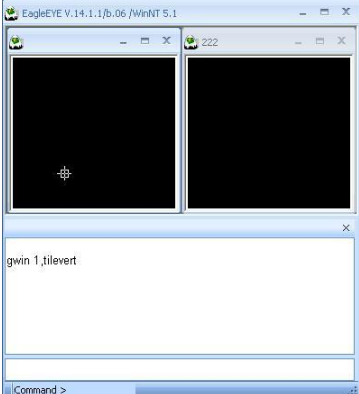
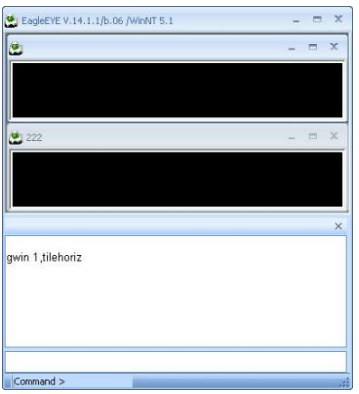
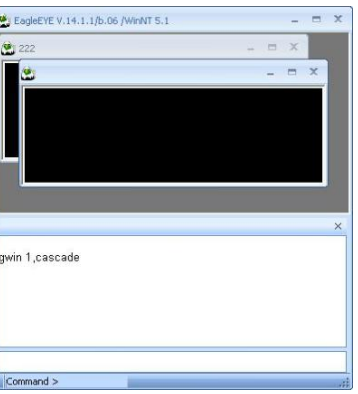
GWINDOW_RESTORED_MACRO	×	×	✓	✓
GWINDOW_MAXIMIZED_MACRO	×	×	✓	✓
GWINDOW_LOOSE_FOCUS	×	✓	✓	✓
GWINDOW_GET_FOCUS	×	✓	✓	✓

### 4.3 – Multiple Document Interface

MDI applications enable you to display multiple documents at the same time, with each document displayed in its own window.

You can change the status of a window using the GWINDOW command, alternatively, just like any application, you can interactively operate with the mouse, resizing, moving, minimizing or maximizing, closing the window as a standard Windows window. Furthermore keyboard Windows special keys, such as Alt-tab or Ctrl-tab for switching from one child window to another, are supported. So, graphic windows can be created, activated and closed (deleted) through the command language or mouse interactions. The behavior in Eagle will be the same.

For the document window, Eagle v.14 supports the same layouts that you can find in a classical document environment (i.e. Microsoft Excel) :

<p>Maximized</p>  <p>gwindow 1, maximize</p>	<p>Restored</p>  <p>gwindow 1, restore</p>	<p>Minimized</p>  <p>gwindow 1, minimize</p>
<p>Tile Vertical</p>  <p>gwindow 1, tilev</p>	<p>Tile Horizontal</p>  <p>gwindow 1, tileh</p>	<p>Cascade</p>  <p>gwindow 1, cascade</p>

#### 4.4 – Tabbed Document Interface

The TDI mode is another way to operate with a multiple documents application where all the frames are contained within a single window, using tabs to navigate between them. It is an interface style most commonly associated with web browsers, web applications, text editors and preference panes.

The main feature of the TDI interface mode is that each gwindow is maximized. Each gwindow is represented by a Tab header that can be positioned on one of the four sides in relation of the value of the INI "TAB\_ORIENTATION" variable.

Moving from one gwindow to another can be achieved in one of the following ways:

1. Press <Ctrl-Tab> ;
2. Execute the command GWINDOW <id>;

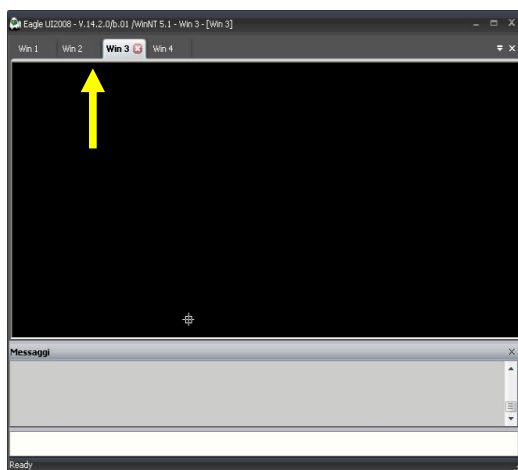


3. Select from the pulldown (reported above) that Eagle presents when the little arrow is pressed.

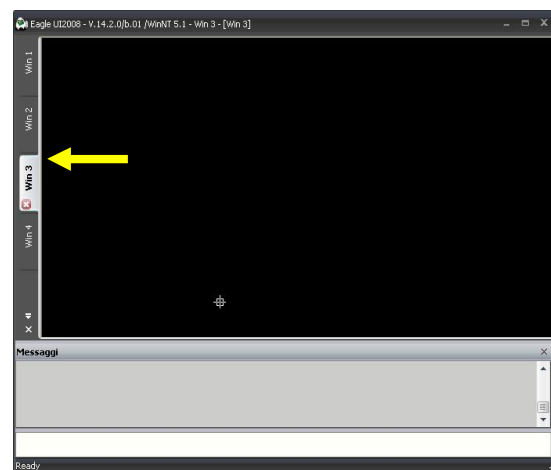
Eagle v.14 allows customizing of the presentation of a tabbed interface with several entries in the configuration file, such as :

- "TAB\_ORIENTATION" : this entry allows setting of the orientation of tabs, it's possible to place the tabs in the top, left, right or bottom side of the main frame

TOP



LEFT





INI Sample :

`TAB_ORIENTATION = top | left | right | bottom`

- “TAB\_BOLD\_SELECTION” : this entry enables having a “BOLD” style for the characters of the tab-page (gwindow) selected; if the parameter is not specified, the default value is “no”;

INI Sample :

`TAB_BOLD_SELECTION = no | yes`

- “TAB\_ENABLE\_DRAGGING” : this entry enables the dragging of the tab pages. If the entry is to “YES”, it’s possible to drag a tab-page /before/after another tab-page; if set to yes dragging of a tab header and change its position is allowed:



If the parameter is not specified, the default value is “no”;

INI Sample :

`TAB_ENABLE_DRAGGING = no | yes`

- “TAB\_ENABLE\_GWINDOW\_CLOSE” : this entry enables closing a tab-page by clicking on the “black x” near the list of tab-pages; When set to “yes” (default), the gwindow close action, if any, is enabled.





If the parameter is not specified, the default value is "no";

INI Sample :

```
TAB_ENABLE_GWINDOW_CLOSE = no | yes
```

- "TAB\_HOVER\_FOCUS" : this entry allows changing of the focused page (gwindow) without having to click on the tab-page but only by moving the mouse pointer over the tab-page. If the parameter is not specified, the default value is "no";

INI Sample :

```
TAB_HOVER_FOCUS = no | yes
```

- "TAB\_ENABLE\_SCROLL" : this entry enables an extended version of the "scroll" buttons when there are a lot of tab-page present and some page are not visible; If set to yes the scroll options to navigate the tabs will be displayed if necessary:





If the parameter is not specified, the default value is "no";

INI Sample :

```
TAB_ENABLE_SCROLL = no | yes
```

- "TAB\_CLOSE\_ON\_PAGE" : this item allows insertion of a "close button" inside a tab-page (near the title of the gwindow). There three possible options: all, active or no :

NO	no close button near the title of the gwindow
	
ACTIVE	close button is displayed only on the active gwindow/page
	

ALL

close button has been added to each pages.



If the parameter is not specified, the default value is “no”;

INI Sample :

```
TAB_CLOSE_ON_PAGE = no | active | all
```

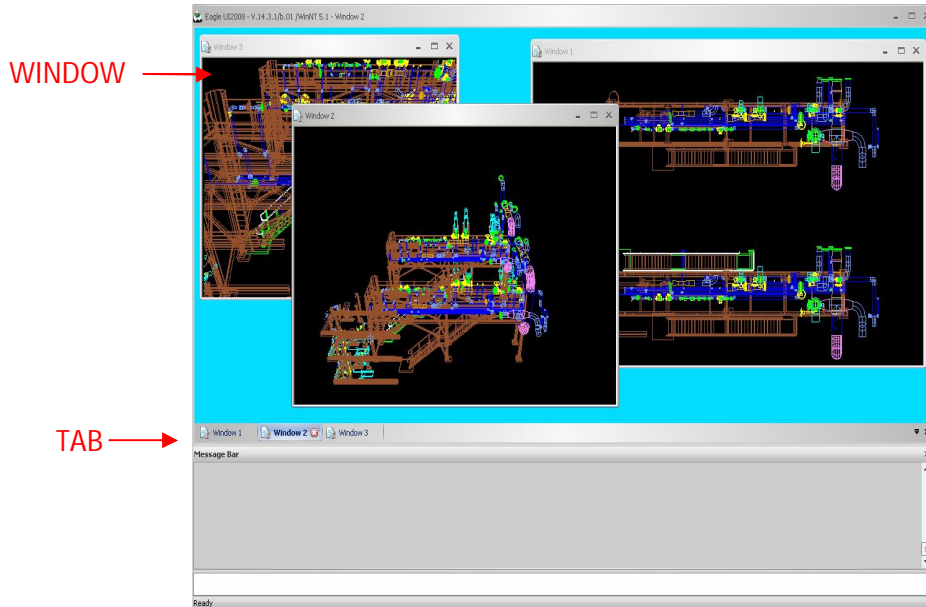
- The “TMDI\_CONTEXT\_MENU” option in the INI file allows definition of a custom popup menu which is displayed when a right button click is pressed on a page. By default the popup does not exist, so the right click performs the same action as the left click. If the “TMDI\_CONTEXT\_MENU” contains an existing menu file (\*.men) the function is enabled and the popup will be displayed when the right button is pressed over a tab-page. Naturally, if the file does not exist the default behavior will be applied.

INI Sample :

```
TMDI_CONTEXT_MENU = C:\menu\tmdi_popup.men
```

## 4.5 – Tabbed Multiple Document Interface

The TMDI interface could be considered as a combination of TDI and MDI modes. Actually the TMDI has the same “gwindow” behavior as the MDI style but in addition it also uses tabs to navigate between the various documents.



TMDI also supports the same layouts for the document window that are found in the MDI environment, these are, maximized, restored, minimized, tile vertical, tile horizontal and cascade. For more details see the “Multiple Document Interface” [chapter \(4.3\)](#).

The TMDI tab container has the same features as detailed in the [TDI section](#), consequently it also is possible to further modify the container presentation using the following Configuration file options:

- “TAB\_ORIENTATION” ;
- “TAB\_BOLD\_SELECTION”;
- “TAB\_ENABLE\_DRAGGING”;
- “TAB\_ENABLE\_GWINDOW\_CLOSE”;
- “TAB\_HOVER\_FOCUS”;
- “TAB\_ENABLE\_SCROLL”;
- “TAB\_CLOSE\_ON\_PAGE”.

[Section \(4.4\)](#) Tabbed Document Interface above details more information about the precise the meaning of these options.

The “TMDI\_CONTEXT\_MENU” option in the INI file allows definition of a custom popup menu which is displayed when a right button click is pressed on a page. By default the popup does not

exist, so the right click performs the same action as the left click. If the "TMDI\_CONTEXT\_MENU" contains an existing menu file (\*.men) the function is enabled and the popup will be displayed when the right button is pressed over a tab-page. Naturally, if the file does not exist the default behavior will be applied.

INI Sample :

```
TMDI_CONTEXT_MENU = C:\menu\tmdi_popup.men
```

## 4.6 – Floating window

Starting from Eagle V14 document windows can be moved to the extern of the Main Frame container; naturally this option is available only with the "MDI" or "TMDI" configurations.

In order to allow the new feature, the GWINDOW command has been extended in this way:

Prototype :

```
gwindow <i>, fix | float
```

Where:

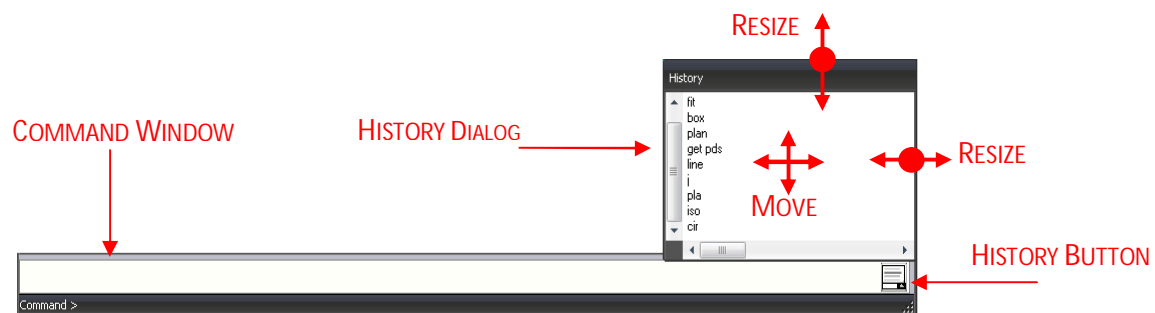
Parameter	Description	SDI	TDI	MDI	TMDI
<i>	A number to identify the graphic or child window.	✗	✓	✓	✓
FIX	Parameter to import the window inside the Main Frame.	✗	✗	✓	✓
FLOAT	Command to move the window outside the Main Frame.	✗	✗	✓	✓

## 5 – Command and Message Windows

The primary relationship between the Eagle language and Graphic User Interface is represented by the command bar and the message bar. These windows facilitate the user with the ability to invoke Eagle's command mode and to control feedback from Eagle's engine.

### 5.1 – The Command window

Just above the status bar and attached to the status bar and the frame window is the command window. Running along the entire length of the bottom of the window, the command window is where macros can be called, Eagle commands can be interactively entered or operating system functions executed. Whilst the sophisticated operation of applications makes the command window redundant for many operations when running applications, the power of this feature cannot be underestimated in providing a quick and easy way to prototype and trial scenarios without having to resort to developing a specific complete applet for every one-off task.



The command window docked to the frame window and is resizable with this window but it cannot be undocked. There is a command function which enables switching on and off of the command window (display or hide the Command Line) and obviously this should be used with caution.

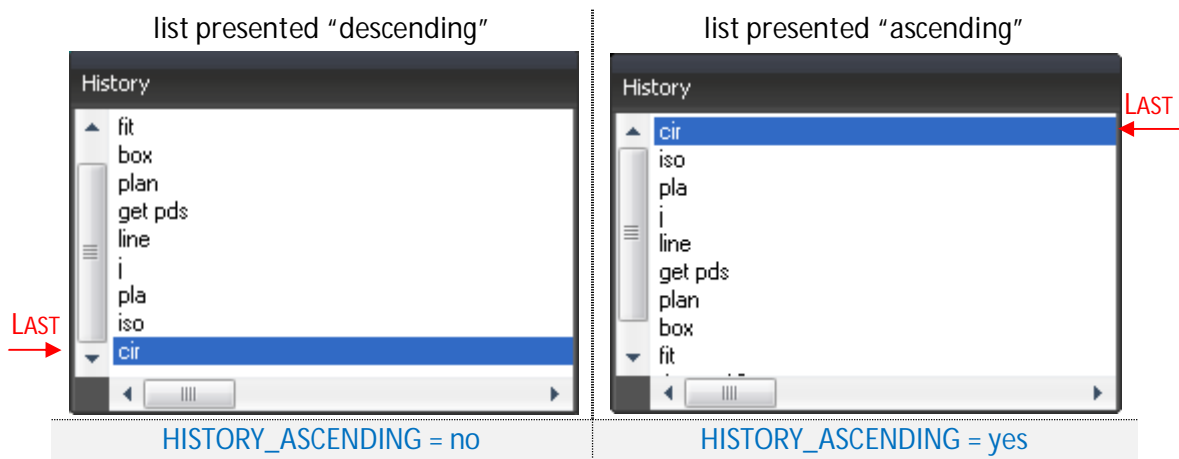
Prototype :	<code>command on   off</code>
Sample Code :	<code>command on</code> <code>command off</code>

The standard Windows context menu is available for Cut & Paste, Select, Edit, etc. on the command line.

The command window has an additional feature called "History Dialog". When the mouse cursor is passed over the extreme right of the command window the "History Button" (a button

with an icon) appears. Clicking on this brings up a history of the commands issued from the command line. The contents are available for copy and paste by double clicking which enters the selected history command into the command line; the panel is also removed at this point. Pressing Esc removes the command history panel by default.

The "History Dialog" is movable, resizable and follows the selected theme. The command list doesn't contain duplicated instructions. In Eagle v.14 a developer can chose how the list is ordered and displayed; using the variable "HISTORY\_ASCENDING", in the configuration file, it's possible to have the last command placed at the top of the list with the scroll bar remaining at the top (ascending) or the last command placed at the bottom of the list and the scroll bar stays on bottom (descending).



The customization of the command window and its derived objects (History Dialog and History Button) can be achieved by modifying this group of variables:

- "COMMAND\_TITLE" : Title displayed on the caption of the command bar.

INI Sample :

```
COMMAND_TITLE = "Command"
```

- "COMMAND\_WINDOW\_TITLE" : When set to YES or NO defines whether or not to use the Eagle command window caption or title bar. Removing the title bar allows development of presentation pre-view type applications. Default value is "no";

INI Sample :

```
COMMAND_WINDOW_TITLE = no | yes
```

- "HISTORY\_BUTTON\_ICON" : this entry sets the icon for the button on the "command bar" that contains the history of executed instructions. Syntax is

HISTORY\_BUTTON\_ICON=<file> where <file> is an ico file. If unset the default icon is used;

INI Sample :

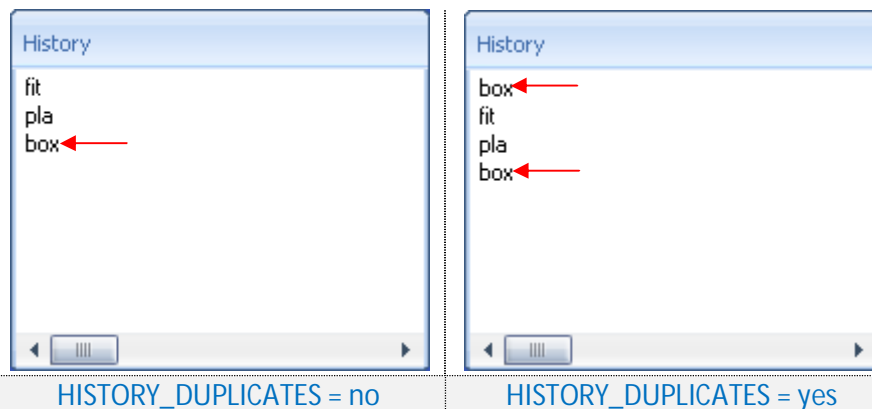
```
HISTORY _ BUTTON _ ICON = C:\Mylcon\ story1.ico
```

- "HISTORY\_ASCENDING" : When set to YES or NO defines if the last command will be placed at the top of the list with the scroll bar remaining at the top (ascending) or the last command will be placed at the bottom of the list and the scroll bar stays on bottom (descending)";

INI Sample :

```
HISTORY_ASCENDING = no | yes
```

- "HISTORY\_DUPLICATES" : This option enables duplication of commands within the history dialog list. When set to YES duplication of executed command executed is enabled, alternatively when set to NO when the same instruction is executed more than once, the first instance in the list will be deleted. The default setting for this option is "no";



INI Sample :

```
HISTORY_DUPLICATES = no | yes
```

- "HISTORY\_TITLE" : set the font used in the command window. The default font is Courier

INI Sample :

```
HISTORY_TITLE = "<Text>"
```

- "COMMAND\_WINDOW\_FONT\_NAME" : This option sets a title for the command window History Dialog List. The default is Hlstory;

INI Sample :

```
COMMAND_WINDOW_FONT_NAME= Courier
```

- "COMMAND\_WINDOW\_FONT\_SIZE" : sets the font size used in the command window. The default is 10;

INI Sample :

```
COMMAND_WINDOW_FONT_SIZE= 10
```

- "COMMAND\_TEXT\_COLOR" : set the text color (#RGB) in the command bar. The color is specified using a "#" followed with three hexadecimal values (one for red, one for green and one for blue), so the range of possible values is : #000000 to #FFFFFF, (the default is #FFFFFF);

INI Sample :

```
COMMAND_TEXT_COLOR=#000099
```

- "COMMAND\_ BACKGROUND\_COLOR" : sets the background color (#RGB) in the command bar. The color is specified using a "#" followed with three hexadecimal values (one for red, one for green and one for blue), so the range of possible values is : #000000 to #FFFFFF, (the default is #000000);



INI Sample :

```
COMMAND_BACKGROUND_COLOR=#FFFFFF
```

## 5.2 – The Message window

The Eagle message window displays system status messages and any reported errors in interactive or application mode transactions. At startup the message window is docked just above the command window. As the data in the window populates the window becomes a scrolling one, so history of messages can be viewed. The number of the visible lines can be set in the INI file or defined through the DIALOG command.

The Eagle command called "DIALOG" displays or hides the message area and change the number of visible dialogue lines;

Prototype :

```
dialogue { on | off | <n> }
```

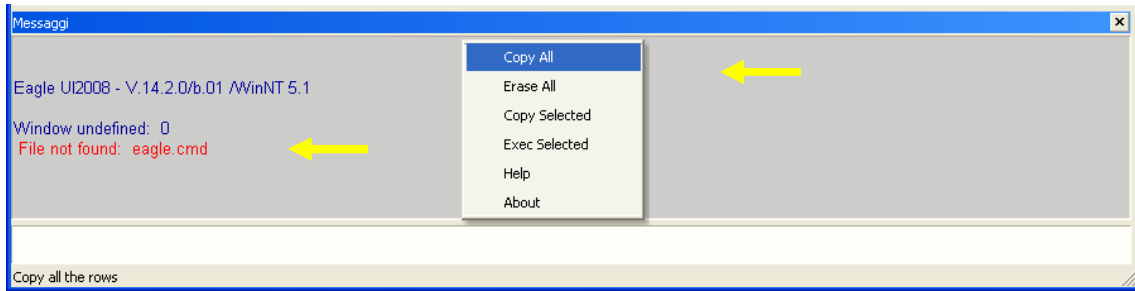
where:

Parameter	Description
-	Toggle the state of the facility.
on	Turn on the message area and display it where last turned off.
off	Turn off message area.
<n>	Set the number of the visible message dialogue lines to n.

Sample Code :

```
dialogue
dialogue off
dialogue 8
```

The message window also has a context menu for selection, redoing and storing of messages into a text file. It is envisaged that this menu will be customizable with user defined commands. Finally, the error and warning messages displayed are done so in different colors. Again it is envisaged that a larger use of text styles will give the message window better communication characteristics.

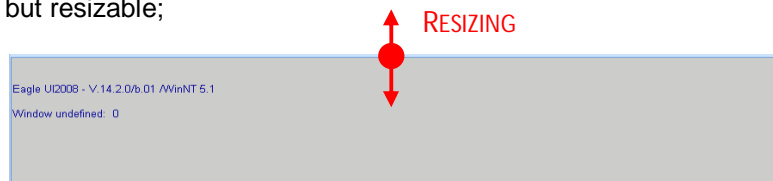


Eagle v.14 offers four different types of message window and it is also possible to modify many aspects of the window using entries in the INI file

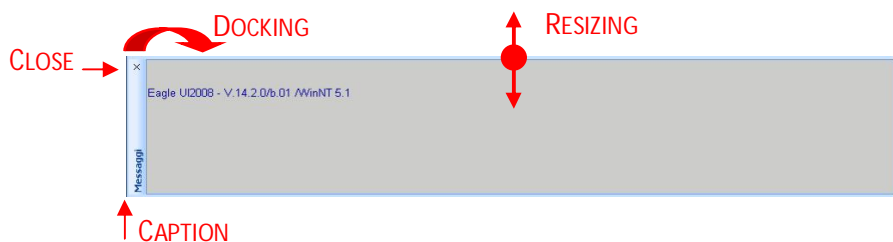
- "MESSAGE\_WINDOW\_TYPE" : select one of the four types of the message bar:
  - **NOSIZE** : message bar without caption, with a left gripper, that is able to be docked/undocked but not resizable;



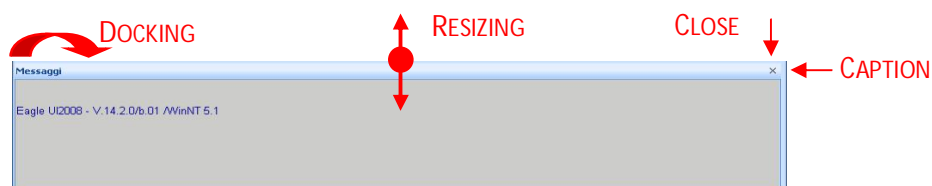
- **FIXED** : message bar without caption, without gripper, not dockable/undockable but resizable;



- **LEFT** : standard message bar with the caption placed on the left side when docked top or bottom; the close button has the same effect of the command "dialog off";



- **TOP** : standard message bar with a standard caption; the close button has the same effect of the command "dialog off";



If the parameter is not specified, the default value is "nosize";

INI Sample :

```
MESSAGE_WINDOW_TYPE = top
```

Next table presents an outline of the differences between the four types of message window :

Type	Caption/Gripper	Title	Close Button	Resize	"dialog" command	Docking
No Size	✓	✗	✗	✗	✓	✓
Fixed	✗	✗	✗	✓	✓	✗
Left	✓	✓	✓	✓	✓	✓
Top	✓	✓	✓	✓	✓	✓

- "MESSAGE\_LINES" : set the number of lines on the Message Bar at start up;

INI Sample :

```
MESSAGE_LINES = 8
```

- "MESSAGE\_WINDOW\_FIXED" : to enable/disable the docking/undocking of the message bar. The default value is "no";

INI Sample :

```
MESSAGE_WINDOW_FIXED = no | yes
```

- "MESSAGE\_READONLY" : to permit or not user written entries on the message window. By default it is read-only but it can be set to read/write mode with the environment variable setting.

INI Sample :

```
MESSAGE_READONLY = no | yes
```

- "MESSAGE\_TITLE" : title displayed on the caption of the message bar. The text is show only if the MESSAGE\_WINDOW\_TITLE is to "YES" and the MESSAGE\_WINDOW\_TYPE is to "TOP" or "LEFT".

INI Sample :

```
MESSAGE_TITLE = MyMessageTitle
```

- "MESSAGE\_WINDOW\_TITLE" : to display or not the "MESSAGE\_TITLE" on the caption of the message bar. Default value is "no";

INI Sample :

```
MESSAGE_WINDOW_TITLE = no | yes
```

- "MESSAGE\_BACKGROUND\_COLOR" : set the background color of the message bar; the color will be specified using a "#" followed with three hexadecimal values (one for red, one for green and one for blue), so the range of possible values is : #000000 to #FFFFFF, (default is #000000);

INI Sample :

```
MESSAGE_BACKGROUND_COLOR=#009900
```

- "MESSAGE\_TEXT\_COLOR" : set the text color in the message bar; the color will be specified using a "#" followed with three hexadecimal values (one for red, one for green and one for blue), so the range of possible values is : #000000 to #FFFFFF, (default is #FFFFFF);

INI Sample :

```
MESSAGE_TEXT_COLOR=#990000
```

- MESSAGE\_WINDOW\_FONT\_NAME" : set font used in the message window. Default is Courier;

INI Sample :

```
MESSAGE_WINDOW_FONT_NAME= Courier
```

- “MESSAGE\_WINDOW\_FONT\_SIZE” : sets the font size used in the message window. The default is 10;

INI Sample :

```
MESSAGE_WINDOW_FONT_SIZE= 10
```

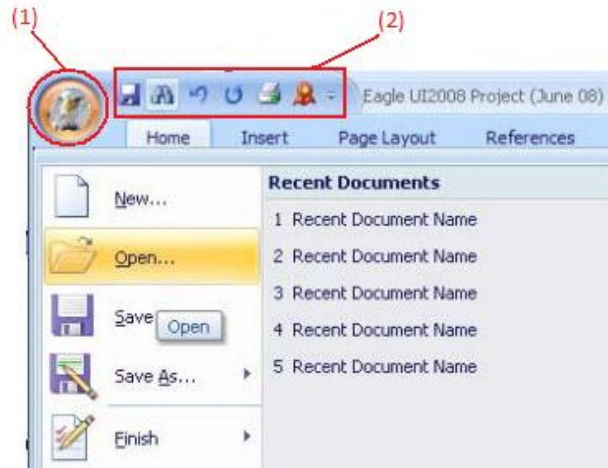
- “MESSAGE\_CONTEXT\_MENU” : customize the popup menu shown in the message area. If the variable is not defined, the default menu will be presented.  
Alternative popup menus are defined using the appropriate Eagle syntax, see the chapter 7.2 to find more details, in this way the menu will be handled like a standard popup.  
The custom defined context menu, described in a “\*.men” file, will then be displayed when the user right clicks the mouse in the message area.

INI Sample :

```
MESSAGE_CONTEXT_MENU = my_menu.men
```

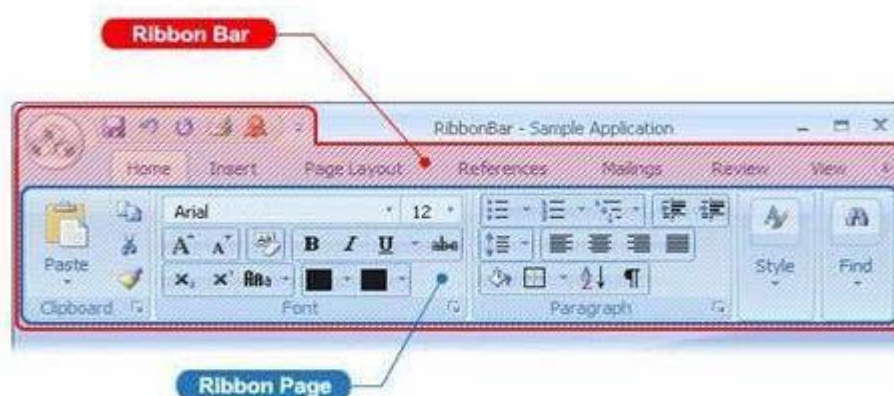
## 6 – Ribbon Bar

The Ribbon UI is a completely new kind of graphical user interface, both in appearance and interaction terms, whose target is to simplify user accessibility. Graphically, the Ribbon displays different semantic areas, where operations are grouped and easy to find. The round button (1) opens a pulldown graphic menu that contains the basic functionality of an application. To the right of the round button, there is a quick access toolbar (2), which shows the most recently used commands.



The Ribbon Bar is a container for Ribbon Pages, as shown in the figure below. These substitute the MenuBar and the ToolBar, presenting a Tabbed version of the two in the same graphical style. The idea is to avoid showing sets of commands which are not frequently or recently used. It is a sort of hiding buttons/options, but, at the same time, having them ready if needed.

Menus and Toolbars were designed for applications with a set of limited features. If the number of toolbars, task panes and the number of options in menus increase too much, then it becomes hard to find functionality and the risk is to use minimal part of the rich set of features of an application. Often, users do not know where commands are or even if they exist.



Office 2007 has introduced some nice ideas, such as:

- The Ribbon
- Galleries
- Contextual Tabs
- Quick Access Toolbar
- Mini Toolbar
- Enhanced StatusBar
- Key Tips and Keyboard Navigation
- Context Menus
- Enhanced Tooltips

For Office 2007 Microsoft used Results-Oriented Design in place of Command-Oriented Design.

Macrovision added to Eagle a set of analysis commands aimed to help the programmer to gather as many information as possible about the application user interface accessibility from users' sessions. This data is then analyzed and will help the GUI programmer to design the layout of better interface which reflects user usage patterns.

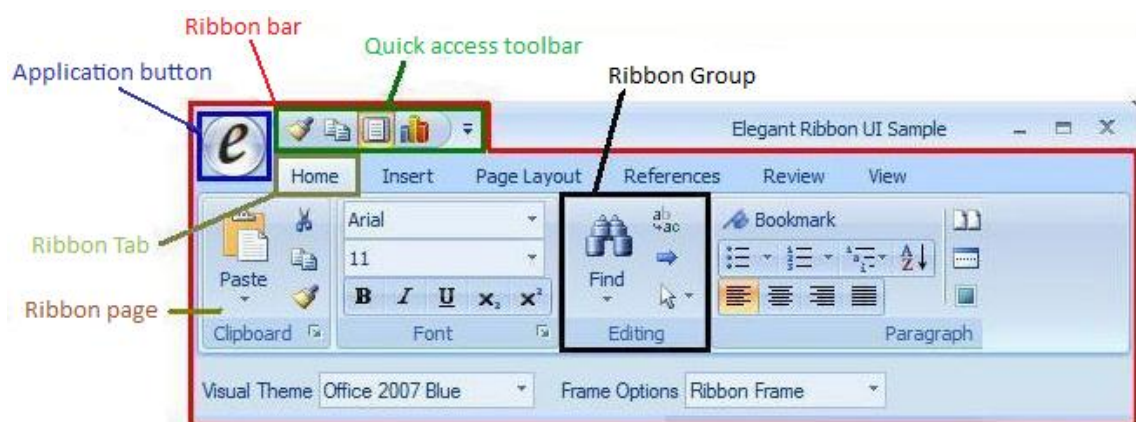
The Ribbon style helps in such reorganization work, avoiding crowded interfaces, continuous selection of different menus searching the right features, etc., defining the following targets for UI design:

- A user's focus must be on their content not on the UI
- Reduce the number of choices presented at any given time
- Increase efficiency
- Embrace consistency
- Give features a permanent home
- Straightforward is better than clever

## 6.1 – Ribbon Bar Structure

A Ribbon UI is structured in several component parts which can be defined as follows:

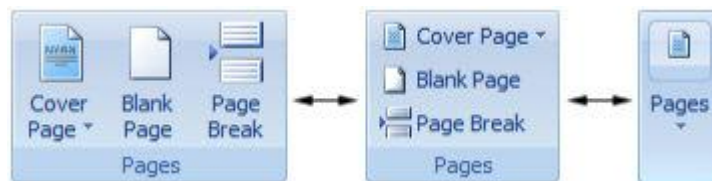
1. Ribbon Bar - the main container contain all menus, pages and controls
2. Application button - main commonly used application configuration and general interaction options
3. Quick Access Toolbar – configurable menu containing use defined options for quick access
4. Ribbon Tab – tab menu which brings up additional ribbon page options by choice or by context usage
5. Ribbon Page – page container on which collection of associated options is displayed
6. Ribbon Group – the grouping of collected associated options



We will now describe each of these component parts.

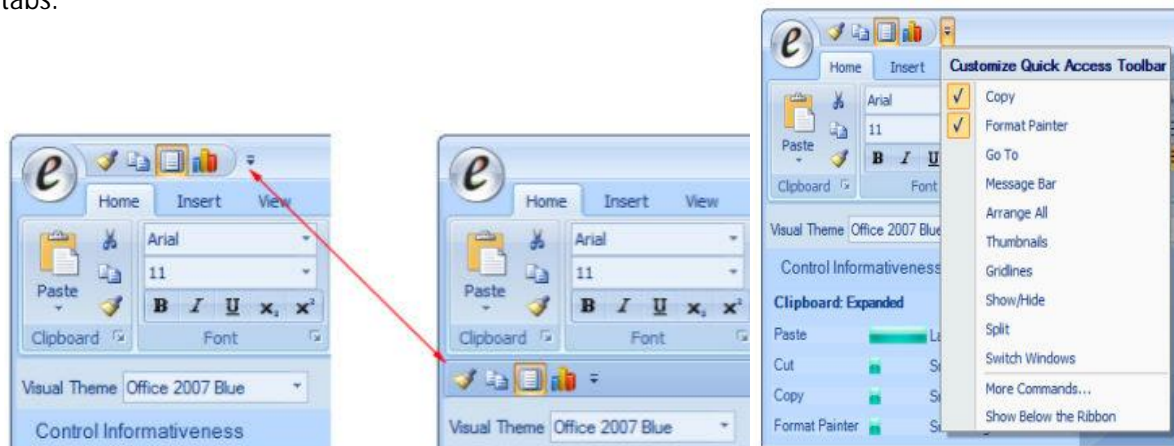
The Ribbon Bar, which is the main container, contains the RibbonPage, the Application Button and the Quick Access Toolbar. The RibbonPage is identified by a RibbonTab, exactly the same as in a tabbed window. Each RibbonPage is composed of RibbonGroups, which contains the functionality access. These can be any type of control that access a program feature directly or through other groups of controls, such as Pulldown Buttons, Dialogs, etc. A RibbonGroup can have a caption holding a title and a button to open the entire corresponding set of associated functionality.

The width of the Ribbon determines the type of display of a RibbonGroup:



The Application Button opens up the application main menu, while the Quick Access Toolbar contains a set of controls that are always available to the user regardless of which tab is currently selected.

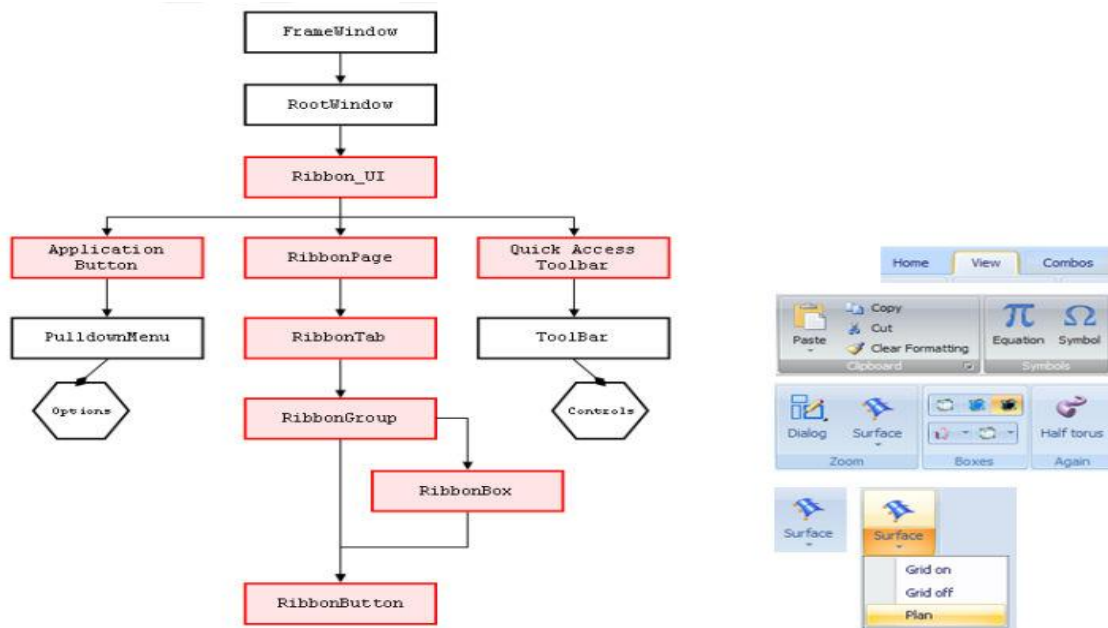
The Quick Access Toolbar can be located above or under the tabs:



The user can customize the location and content of the Quick Access Toolbar using a menu that appears when the user clicks the drop-down arrow on the right as shown in the figure on the above right.

The Ribbon UI added to Eagle in the UI Framework extends the hierarchy structure of Eagle as follows:





The new Eagle command that is used to create this type of graphical user interface is called the RIBBON command, the syntax of which is as follows:

Prototype :

**ribbon** **f=<text>** | **select pos=<pos>** | **minimize** | **maximize** | **off**

where:

Parameter	Description
<b>f='text'</b>	Specify the xml file that contains the description of the ribbon bar to activate.
<b>minimize</b>	Temporarily hides the created ribbon bar.
<b>maximize</b>	Shows the temporarily hidden ribbon bar.
<b>off</b>	Remove the ribbon bar from the screen.

for example:

Sample Code :

```

ribbon f=filename.xml
ribbon minimize
ribbon maximize
ribbon off
  
```

The following example shows the structure of the layout file simplified for the sake of clarity and the XML schema associated with it.

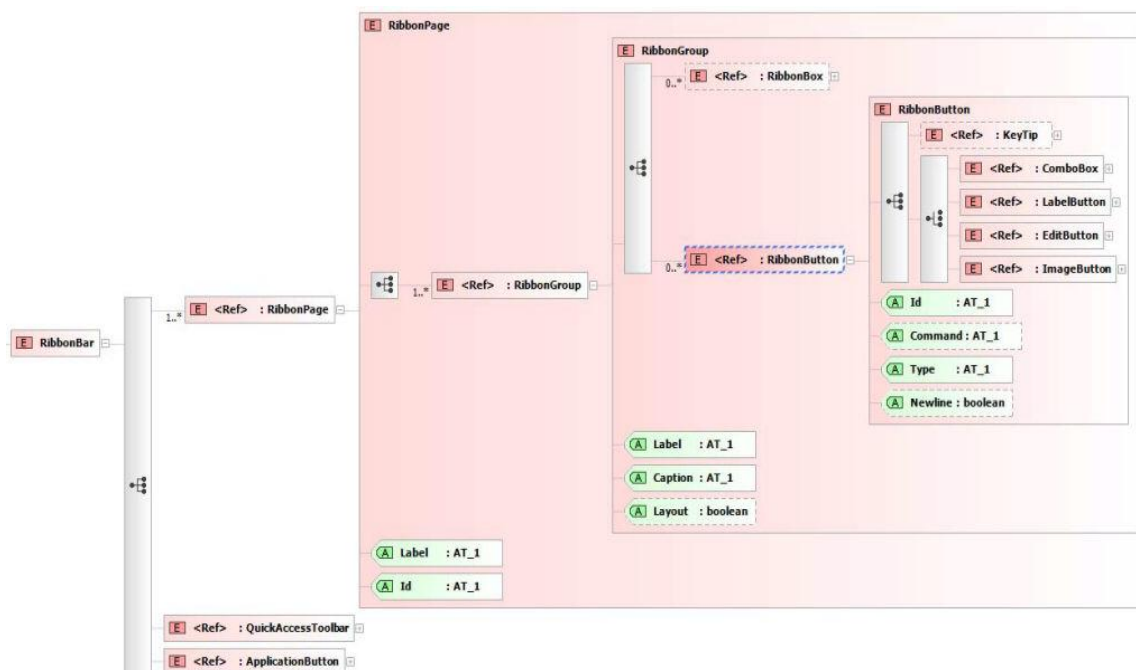
The file format is based on XML and it complies to the Eagle GAML schema defined in the GAML.xsd file, the structure of which is partially described for the ribbon in the diagram below:

```

4 <RibbonBar xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="C:\LocalRepository\MacroVision\UI2008\Ribbon.xsd">
5   <RibbonPage Id="1" Label="Home">
6     <RibbonGroup Label="Pipes" Caption="Bottom">
7       <RibbonButton Id="1" Type="4" Command="pip col=6;n150,e300,s150;">
8         <ImageButton Size="big" Label="Half torus" Image="\escape\ncg\opengl\se3.bmp"></ImageButton>
9       </RibbonButton>
10      <RibbonButton Id="2" Type="4" Command="pipe col=6;u200;">
11        <ImageButton Size="big" Label="Pipe" Image="\escape\ncg\opengl\se55.bmp"></ImageButton>
12      </RibbonButton>
13    </RibbonGroup>
14    <RibbonGroup Label="Modeling" Caption="Bottom">
15      <RibbonButton Id="1" Type="4" Command="get pds">
16        <KeyTip Title="Load a plant" Image="\escape\ncg\opengl\se2.bmp" Help="Press F1 for more help." HelpImage="\escape\ncg\opengl\help.bmp">
17          Create a surface item blending
18          two given surfaces.
19        </KeyTip>
20        <ImageButton Size="big" Label="Surface" Image="\escape\ncg\opengl\se2.bmp"></ImageButton>
21      </RibbonButton>
22      <RibbonButton Id="2" Type="4" Command="fit">
23        <KeyTip Title="Blend two surfaces" Image="\escape\ncg\opengl\se44.bmp" Help="Press F1 for more help." HelpImage="\escape\ncg\opengl\help.bmp">
24          Create a surface item blending
25          two given surfaces.
26        </KeyTip>
27        <ImageButton Size="big" Label="Blend" Image="\escape\ncg\opengl\se44.bmp"></ImageButton>
28      </RibbonButton>
29      <RibbonButton Id="3" Type="4" Command="shade">
30        <ImageButton Size="small" Label="Section" Image="\escape\ncg\opengl\se51.bmp"></ImageButton>
31      </RibbonButton>
32      <RibbonButton Id="4" Type="4" Command="tell T,D">
33        <ImageButton Size="small" Label="" Image="\escape\ncg\opengl\se52.bmp"></ImageButton>
34      </RibbonButton>
35      <RibbonButton Id="5" Type="4" Command="stat 3">
36        <ImageButton Size="small" Label="" Image="\escape\ncg\opengl\se53.bmp"></ImageButton>
37      </RibbonButton>
38    </RibbonGroup>
39  </RibbonPage>

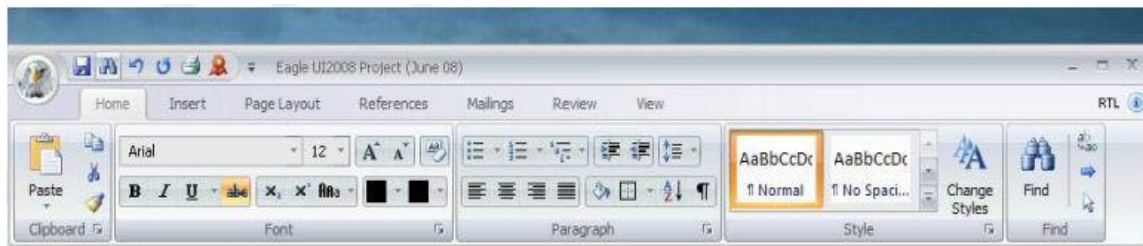
```

Note: the "text string" displayed in the above XML file can be treated in exactly the same way as in Eagle TAB files, which means that there is full support for the Message Function (e.g.: 'm(11,27)') and Eagle pathnames (e.g.: '@application/utils/recover.cmd' or '-NUPASDIR/open.bmp').

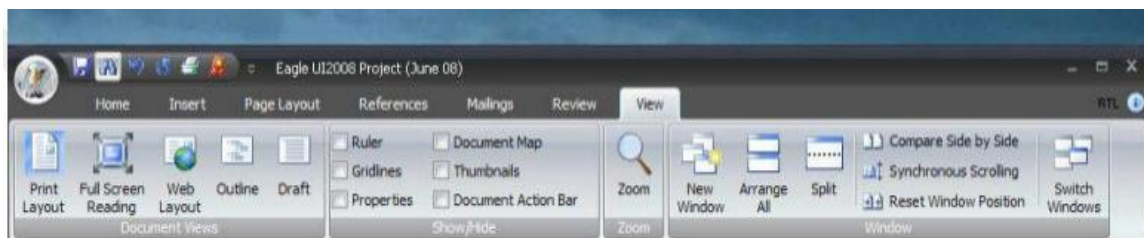


(The listing above represents only part of the file for practicality purposes)

The ribbon interface layout starts as a static map of the interface, but during the processing of the application, it can be changed dynamically. For instance, the Quick Access Toolbar changes accordingly to the most used features or to the application's strategy.

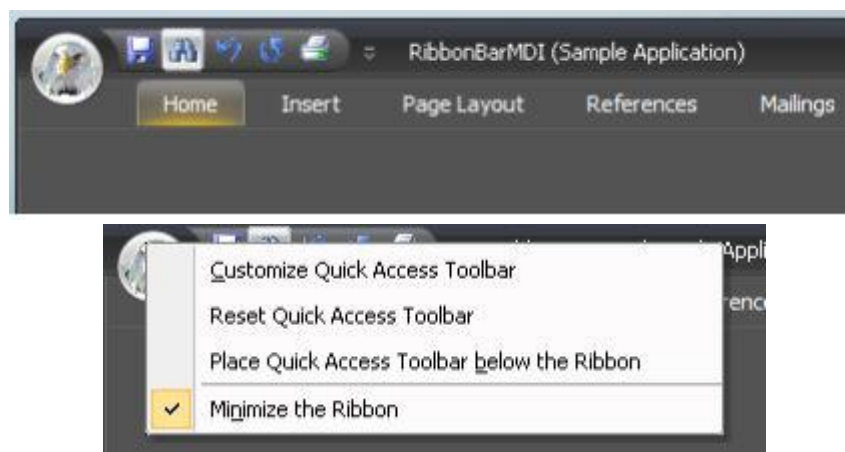


Office 2007 Silver theme



Office 2007 Obsidian theme

The above images show the ribbon as always being visible, however it is also possible to minimize it and reduce the real estate taken up by the user interface even further.



Minimized ribbon

In this case the ribbon tabs remain visible but when selected will open up the relevant corresponding ribbon page. The ribbon will then be made invisible once more when it is selected again. This behavior is achieved by right-clicking on the application button and selecting the "Minimize the Ribbon" option from the pulldown menu as illustrated in the above lower image.

### 6.1.1 Ribbon Selecting, Inserting and Removing Pages

The new Ribbon Eagle command has been extended to provide dynamic updating of the Ribbon Tabs, groups and buttons. To accommodate this functionality, the ability to select, add and remove Ribbon Pages is implemented. RIBBON command syntax is extended as follows:

Prototype :

```
ribbon      Select, pos=<i>
            insert, pos=<i> ,f=<newpage.xml> |
            remove, pos=<i>
```

where:

Parameter	Description
<b>select</b>	Select a tab page from the ribbon bar where POS is the page position. <b>POS=</b> defines the position of the tab page to select
<b>insert</b>	Inserts a new page where <b>POS=</b> defines the position of the new page <b>F=</b> the xml file which describes the defined page
<b>remove</b>	Removes a page from the ribbon bar where <b>POS=</b> the position of the page to remove.

for example:

Sample Code :

```
ribbon select, pos=3
ribbon insert, pos=3,f=newpage.xml
ribbon remove,pos=3
```

The new page must be defined in an XML file containing the specification of the new page;

```
<RibbonBar xmlns:xs="http://www.w3.org/2001/XMLSchema-instance">
  <RibbonPage Label="Inserted page" Id="2" Mnemonic="V">
    ...
  </RibbonPage>
</RibbonBar>
```

If the XML file contains more pages, they are also inserted as well.

It is important to note that the original XML file used to install the RibbonBar is changed and its content is updated with the inserted or removed pages. This reflects the persistency characteristic of the new XML-based Eagle graphic user interface.

The above example in GAML becomes:

```
<Gaml xmlns:xs="http://www.w3.org/2001/XMLSchema-instance">
<RibbonBar>
  <RibbonPage Label="Inserted page" Id="2" Mnemonic="V">
    ...
```

```

    ...
    </RibbonPage>
  </RibbonBar>
</Gaml>

```

The First line can include the schema for run-time validation purposes:

```

<Gaml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="C:\tmp\gaml.xsd">

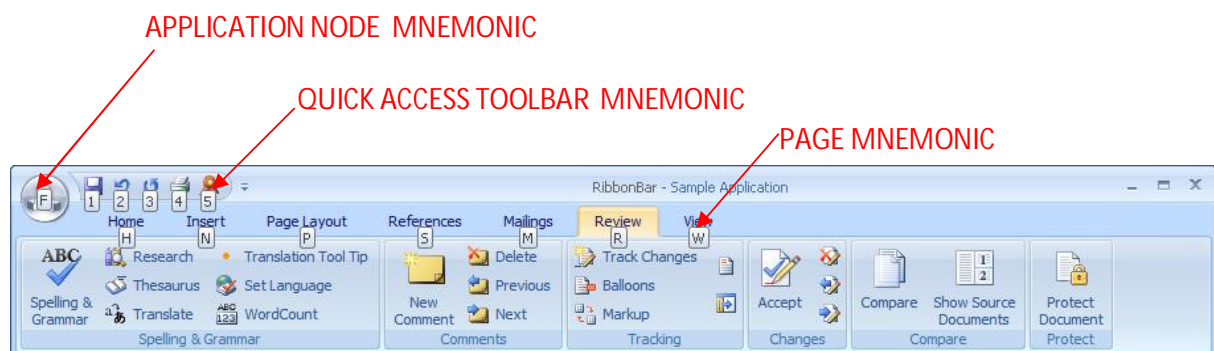
```

## 6.2 – Ribbon UI philosophy

In this paragraph we will outline some characteristics of a Ribbon-based user interface, considering the factors that enrich the interactive nature of an interface. These ideas are those which have been embraced by the proposers of the Ribbon interface. Most of our research derives from the considerable user and developer studies Microsoft undertook during the design of the Office 2007 user interface.

### Mnemonics

A mnemonic is a readily recognizable character (or set of characters) that the user can type to quickly execute a command or an action. In the ribbon bar the relevant mnemonics are shown in a tooltip near to the related command option when the ALT key is pressed. A mnemonic could be considered as being the same as a shortcut key or accelerator, that is, a quick access way for the user can to activate a specific action ribbon choice.



The user presses ALT and the relevant mnemonic key to navigate the ribbon page activating a tab group or to execute an action/command. Pressing the ALT and the mnemonic key also displays a label associated with a tab group which is then used to navigate to the tab group. The user presses the mnemonic key again when within a menu or tab group to activate or toggle the appropriate choice from within the same menu or tab group.

Mnemonics keys are available for the follow:

- Tab page;
- Controls inside pages;
- Quick access toolbar buttons;
- Application node;

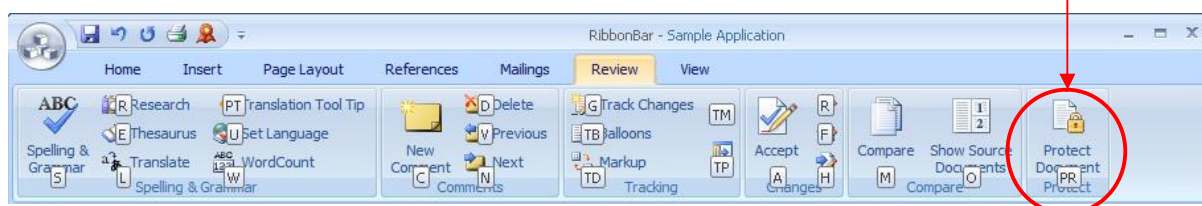


- Buttons inside the application menu.

The mnemonic can have one or more keys and the developer can set the appropriate tips in the xml configuration file.

In practical terms if we consider for example the previous image, if the user needs to navigate to the “Project Document” button (which is the last control in the “Review” page), the correct sequence of keys will be as follows:

ALT → R → P and R



### 6.3 – Polling extensions

Since we have new GUI objects, we also have new GUI events, which are reported by the Eagle POLLING command with new values for the system variables the command sets.

WT Variable:

The RibbonBar is a totally new GUI objects and it is represented with the value 12 in the WT variable.

MN Variable:

The value given to the MN variable is the RibbonPage Tab id or one of the following values for the special case:

MN Value	Description
$\geq 1$	RibbonPage
0	Quick Access Toolbar
-1	File Menu Options.
-2	Recent Documents gallery option

BN Variable:

The value given to the BN variable is the RibbonButton id or the index of the selected option for the special case, starting from 1.

With this mapping a mainloop function is able to detect if the GUI event comes from

- a RibbonBar object (WT=12)
- a RibbonPage (MN  $\geq 1$ ) and which button (BN),

- the Quick Access Toolbar (MN = 0) and which option (BN),
- the FileMenu (MN = -1) and which option (BN).

## 6.4 – Recent documents gallery

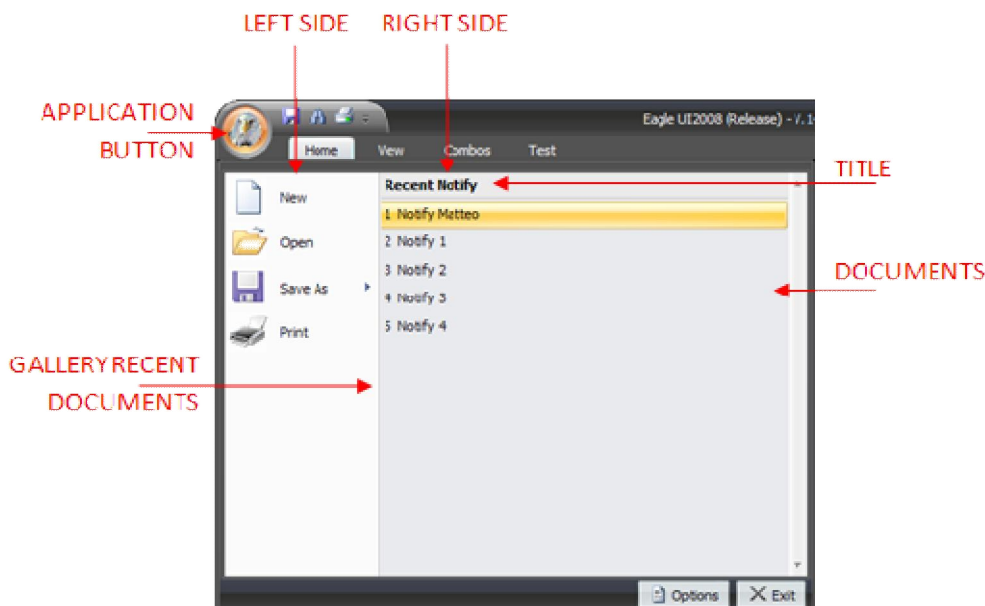
The Application menu, which is accessible through the Application Menu Button, is divided in two main parts. The left side is comprised of buttons (such as “New”, “Open” or “Save”) and the right side contains a nested submenu.

The items in the right hand section can be customized to contain a specific menu depending on the button selected in the left side. This means that the button is “expandable” and can group together complimentary application functionalities using a single easy-to-use entry point.

In addition to “expandable buttons” (such as “Print”), the Application menu has a default gallery that displays “Recent Documents”.

The structure of the “Recent Documents” menu enables definition of:

- the title;
- the size, the maximum number of elements present in the gallery;
- the list of elements.



In order to accommodate the gallery of Recent Documents the XML file, representing the ribbon interface, has been extended and a new “Ribbon.xsd” schema file has been created. The new part has been included in the “ApplicationButton” structure, located after the “FileMenu” definition:

```
...
<ApplicationButton Image="EagleRibbon.bmp" ... >
...
    <FileMenu Exit= ...>
```

```

    ...
  </FileMenu>
  <Gallery Title="Gallery Items" Size="12">
    <GalleryItem Id="3" Text="File 1" Action="do ..." />
    <GalleryItem Id="2" Text="File 2" Action="do ..." />
    <GalleryItem Id="1" Text="File 3" Action="..." />
  </Gallery>
  ...
</ApplicationButton>

```

As shown in the “xml” sample above, the “*Gallery*” element must be used to define the Recent Document menu. The Gallery consists of two attributes, a label for the title (Title) and the number of element shown in the “Recent Documents” menu (Size), and within this is an unbounded list of “*GalleryItem*” objects.

The “*GalleryItem*” type describes single items present in the “Recent Documents” menu using the following attributes:

- Id : index of the item inside the menu;
- Text : label displayed for the item;
- Action : command executed when the item is selected.

The “ribbon” command has been extended to manage the gallery and to modify its characteristics at runtime, meaning the developer doesn’t need to access the XML file directly. The changes applied to the ribbon’s gallery are persisted in the XML file used for creating the ribbon itself.

The new command options are:

- ribbon gallery, size=<value> : change the maximum number of options
- ribbon gallery, title=’<string>’ : change the title
- ribbon gallery, add=’<string>, action=’<string>’ : add on top a new option with an associated action.

The syntax of the new command is as follows:

Prototype :

```

ribbon gallery  size=<value>      |
                  title=’<string>’ |
                  add=’<string>, action=’<string>’

```

where:

Parameter	Description
gallery	Primer to identify procedures related to the “Recent Documents” menu



<b>size</b>	Maximum number of items in "Recent Documents" gallery
<b>title</b>	Label displayed as title of "Recent Documents" gallery
<b>add</b>	Set the label of a new item in "Recent Documents" gallery
<b>action</b>	Set the action performed by a new item in "Recent Documents" gallery

For example:

Sample Code :

```
ribbon gallery, size = 10
ribbon gallery, title = 'Recent models'
ribbon gallery, add = 'My model.mod', action='get mymodel.mod'
```

When the user adds a new item to the gallery list, the "Id" of the other elements in the XML file are increased by one unit and the new item will be appended to the list with an Id = 1, for instance:

Sample  
Code :

```
<Gallery Title="Recent Notify" Size="12">
  <GalleryItem Id="3" Text="Notify 3" Action="do noti3.cmd"/>
  <GalleryItem Id="2" Text="Notify 2" Action="do noti2.cmd"/>
  <GalleryItem Id="4" Text="Notify 4" Action="do noti4.cmd"/>
  <GalleryItem Id="1" Text="Notify 1" Action="do noti1.cmd"/>
</Gallery>
```

```
ribbon gallery, add = 'Notify Matt', action='do matt'
```

```
<Gallery Title="Recent Notify" Size="12">
  <GalleryItem Id="4" Text="Notify 3" Action="do noti3"/>
  <GalleryItem Id="3" Text="Notify 2" Action="do noti2"/>
  <GalleryItem Id="5" Text="Notify 4" Action="do noti4"/>
  <GalleryItem Id="2" Text="Notify 1" Action="do noti1"/>
  <GalleryItem Id="1" Text="Notify Matteo'" Action="do matt"/>
</Gallery>
```

Note, the order of "GalleryItem" objects is not relevant because the presentation is guided by the "Id" value. The most recent GalleryItem is given the id=1 meaning it is displayed at the top of the list and so on for all other items.

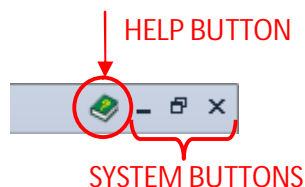
When an option is selected the related action is executed and polling returns the following values:

- vb = 1
- mn = -2

- bn = index of the option selected (1 based)
- wt = 12
- ln, rn = 0

## 6.5 – Help button

The ribbon bar includes an additional button in the right side, near the system's mdi buttons:



This object, called "help button", could be used to execute a specific action in order to support the user displaying a help document, web page or something else.

The Help button is enabled by setting the "RIBBON\_HELP\_ICON" to with an existing icon file (\*.ico); the icon binder has to contain a "16x16" image. When the "RIBBON\_HELP\_ICON" option is not present in the configuration file or the defined icon file does not exist the button is not displayed.

Furthermore a developer can set tooltip text by defining the "RIBBON\_HELP\_TEXT" and an action with the "RIBBON\_HELP\_ACTION" entry. By default the help button doesn't have tooltip text or an associated action defined.

INI Sample :

```
RIBBON_HELP_ICON=C:\test\ico_help.ico
RIBBON_HELP_TEXT = Help
RIBBON_HELP_ACTION = C:\test\help.cmd
```

## 6.6 – Messages Support

Starting with v.14.5.1.b03, Eagle RibbonBar supports text/messages taken from the Eagle Message File.

Anywhere it is possible to specify "text" using a string (e.g.: Label="Straight Pipe"), it is now possible to use the `m(set,line)` function instead.

A few examples follow:

Sample Code :

```
<RibbonPage Label="m(1,1)"
<RibbonGroup Label="m(1,1)"
<ImageButton Image="..." Label="m(1,1)" Size="big"/>
<KeyTip Title="m(1,1)" Image="..." Help="m(1,1)"
HelpImage="-opengl/help.bmp">
```

```

        m(1,1)
        m(1,1)
        Eagle is the queen of birds.
        m(1,1)
        m(1,1)

</KeyTip>

<DropDownMenu Id="1">
<Option Id="1" Label="Model File" Image="..." Command=" "
Mnemonic="O31"/>
    <Option Id="2" Label="m(1,1)" Image="..." Command=" "
Mnemonic="O32">

<Gallery Title="m(1,1)" Size="5">

```

### 6.6.1 Restrictions

Usually Eagle supports two syntaxes for messages:

1. m(set,line)
2. mess(set,line)

In the actual implementation of the RibbonBar only the first syntax is supported.

Furthermore, whilst Eagle is able to “inject” a text message in place of a `m(set,line)` phrase using syntax like:

Example :

```

string s1
s1 = '(E) The previous command returned m(1,1).'
The previous command returned Syntax error.

```

The RibbonBar however does not support that method; you must use “text” or a “message”:

```
<Option Id="1" Label="Ambient Light"    or
```

```
<Option Id="1" Label="m(15,62) "
```

Whilst the syntax:

```
<Option Id="1" Label="Ambient m(15,58) "
```

is invalid.

The `<KeyTip>` element is an exception to that rule. In fact, it accepts the following “partially-mixed” syntax:

Sample Code :

```
<KeyTip Title="Access outcome" Image="clock.ico"  
Help="Press F1 for more help."  
    HelpImage="help.bmp">  
    m(15,24)  
    m(15,25)  
    m(15,26)  
    since you are the administrator.  
</KeyTip>
```

Output:



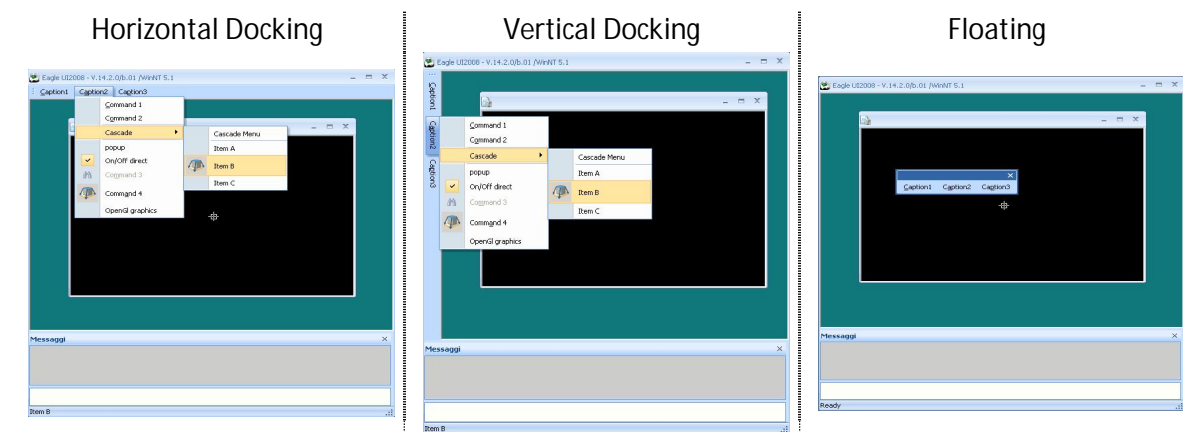
## 7 – Menu Bar and Popup Menu

This chapter introduces the “menu” components, namely the elements that display a list of options using pull down entities. “Menu Bar” and “Popup Menu” belong into this group; in fact a menu bar is composed of a number of pull down menus activated by buttons. The behavior of a pull down menu is exactly the same as a popup menu.

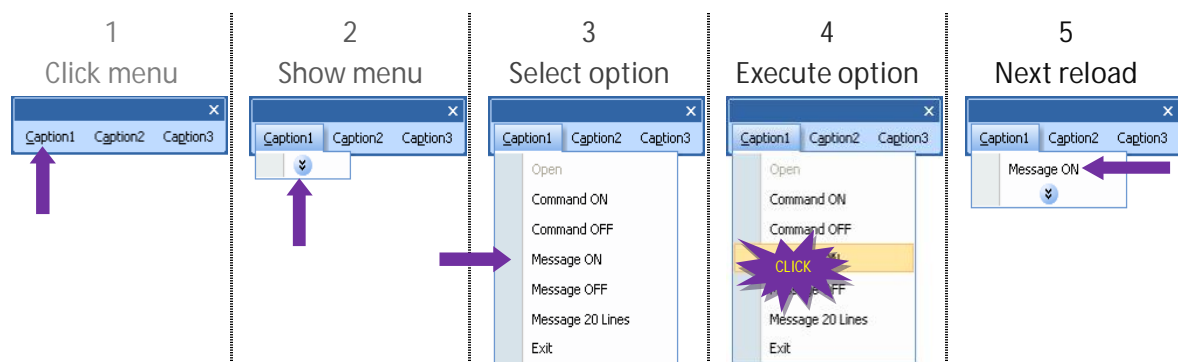
### 7.1 – Menu Bar

A menu bar is a region where computer menus are housed. Its purpose is to hold specific menus which provide access to functions such as opening files, interacting with an application, or help. Menu bars are typically present in graphical user interfaces with windows, so in Eagle v.14 this element has been implemented and updated to follow the evolution of GUI, but menu bar additionally provides many features not supported by default:

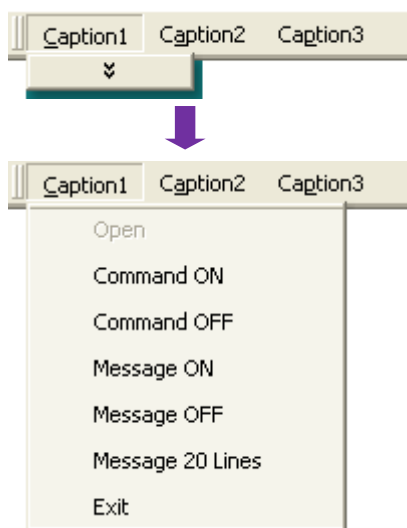
- The new “Menu Bar” can be docked to any side of the main frame window or can float anywhere on the screen. In fact in Eagle v.14 the “Menu Bar” becomes a “Control Bar” and consequently can be undocked from its default position as a toolbar and docked in another position or remain floating;



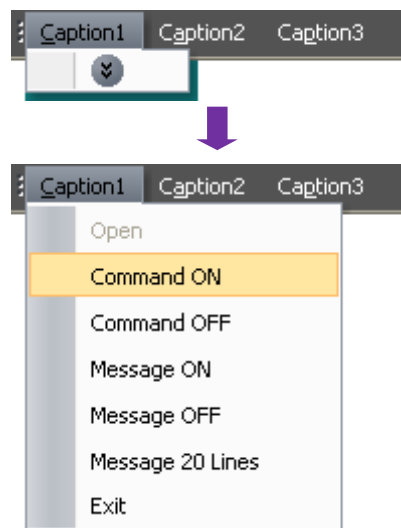
- Rarely used menu items can be initially hidden and are only shown after clicking on the chevron button or after a short delay;



- Pop-up menu is automatically scrolled when the menu height is greater than the screen size;
- Behavior and style of the menu bar follow the current theme;



Themes less or equal to Window Xp Native



Themes greater or equal to Office 2007 R1

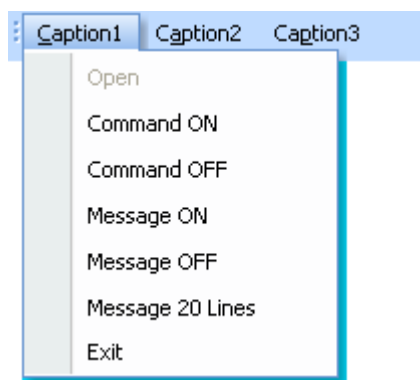
- Pop-up menu shadow, which is correctly displayed on all supported operating systems.

Using the “BAR\_EXPANDABLE” entry in the configuration file developers can choose between the new (accordion style) or classical menu behavior. If the entry is omitted, the new style is the default presentation.

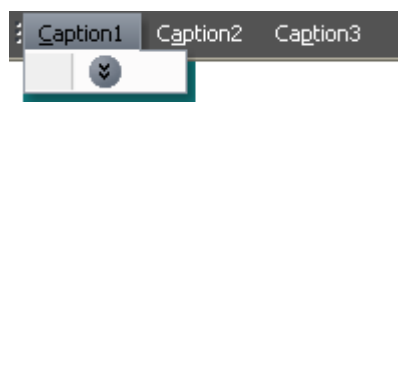
INI Sample :

BAR\_EXPANDABLE = no | yes

If the new features behavior is enabled, then all menus and popups in the application will be displayed with hidden items, the chevron button and exhibit the display behavior that consists of a short delay before displaying the least used options. If the BAR\_EXPANDABLE is set to “no” every menu will be displayed using the standard presentation without any of the new elements.



Classic



New

Similarly it's also possible to enable or disable the undocking mechanism of the menu bar by using the "BAR\_DOCKING\_ENABLED" configuration setting. If this option is set to "yes" then the bar can be undocked, otherwise the bar always remains docked at the top of the main frame. The default setting for this option is "yes".

INI Sample :

```
BAR_DOCKING_ENABLED = no | yes
```

The "Menu Bar" is created and destroyed with the "bar" command (BAR ON and BAR OFF).

Prototype :

```
bar    on|off {,f=<file>}
```

where:

Parameter	Description
on	To define a new bar menu or redefine an existing one.
off	To dismount an installed bar menu.
f = <file>	The name of the file (extension .TAB) where the list of all the cascade menus is stored.

Sample Code :

```
bar off
bar on, f=mybar.tab
```

A menu bar is created by defining entries in a ".tab" file that contains a catalog of ".men" files:

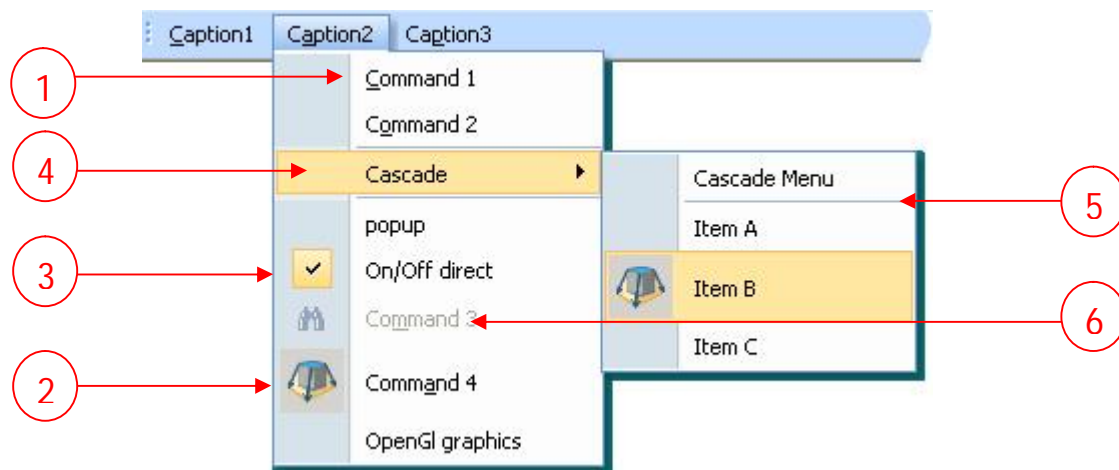
File Sample:

```
mybar.tab
file.men
edit.men
search.men
project.men
window.men
```

This files .tab contains five cascade menu; these menu can have options and submenus.

A file which describes the menu ("men"), can contain different kinds of objects and the next table shows the list of items and the related code templates:

	Description	Code Template
1	simple menu item	<code>index,'text' : command;</code>
2	option with an icon	<code>index,'text@file.bmp': command;</code>
3	check item that shows the enabled/disabled status	<code>index,'text#on': command;</code> or <code>index,'text#off': command;</code>
4	cascade submenu	<code>index,'Text'&gt; file.men</code>
5	separator	<code>index, "&amp;</code>
6	disabled element (one of previous); the option starts frozen	<code>-index,'text': command;</code> Note : NEGATIVE INDEX
7	List of active document windows at the end of the menu	<code>999, "</code>



To set the title of the menu (for instance "Caption2"), the first row of the menu file will consist of a string with the text for the title.

Furthermore the Menu Bar allows definition of "shortcut-keys" to quickly access an option from keyboard accelerators. In order to use shortcut-keys you must introduce a "&" before the character that identifies the accelerator (i.e. '*C&aption2*' or *1,'&Command 1' : vane*).

Just by adding # symbol followed by the name of the string variable that represents the status (eg.: turnit). If the value of the turnit variable is 'on' then, when displayed, the menu will show the checked symbol, otherwise if the value is 'off' then nothing will be shown.

File Sample:

```
33,'OpenGL graphics#turnit': do turnit
```

Clarification of the layout concepts contained in the the ".men" file illustration above is presented below:

File Sample:

```
cap2.men 'C&aption2'
1,'&Command 1' : vane
2,'C&ommand 2': plan
3,"&
```

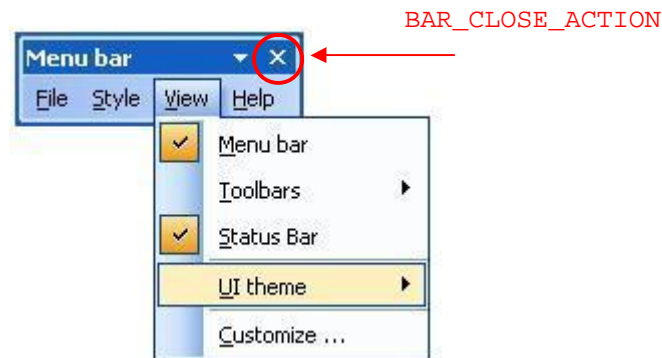


```

4,'Cascade'> sub1.men
5,"&
6,'popup':popup cap1
7,'On/Off direct#on': tell'on/off...dir!'
-8,'Co&mmand 3@binocolus.bmp': get $test;
9,'Comm&and 4@se12.bmp': fit;
10,'OpenGL graphics#TURNIT': do turnit
  
```

Up to sixteen options can be specified. If an option in an active bar menu is changed, i.e. a string variable referring to an option is altered, it is no longer necessary to issue the bar off command and then re-issue a bar on. Just issue the bar on, f=<filename>.tab again and the bar menu will be updated.

In Eagle v.14, when a Menu Bar is floating, it is also possible to use the close button in the same way as other control bars. So when the "X" button in the caption is pressed, Eagle calls the event defined in the INI file entry "BAR\_CLOSE\_ACTION".



INI Sample :

```
BAR_CLOSE_ACTION = C:\MyFolder\closebar.cmd
```

The bar menu or parts of the bar menu can be frozen or unfrozen using the FREEZE and UNFREEZE commands :

Prototype :

```
freeze mb , p=<panel>, b=<button>
```

where:

Parameter	Description
<b>p</b>	[Optional] Is the menu number in a range from 1 to the amount of items in the menu bar.
<b>b</b>	[Optional and need "p"] Is the index of the option inside the popup menu specified using the "p" parameter.

Prototype :

unfreeze mb , p=&lt;panel&gt;, b=&lt;button&gt;

where:

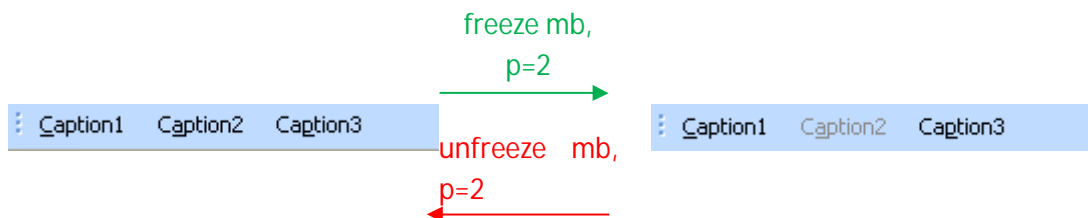
Parameter	Description
p	[Optional] Is the menu number in a range from 1 to the amount of items in the menu bar.
b	[Optional and need "p"] Is the index of the option inside the popup menu specified using the "p" parameter.

Examining the prototypes for the freeze and unfreeze command, we can understand that it's possible to freeze/unfreeze :

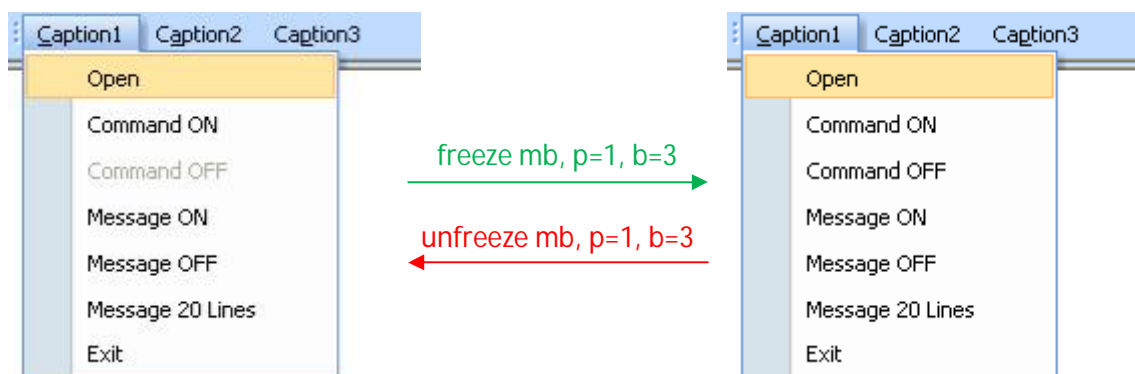
- the entire menu bar : freeze mb or unfreeze mb;



- a single menu (p) : freeze mb, p=index or unfreeze mb, p=index;



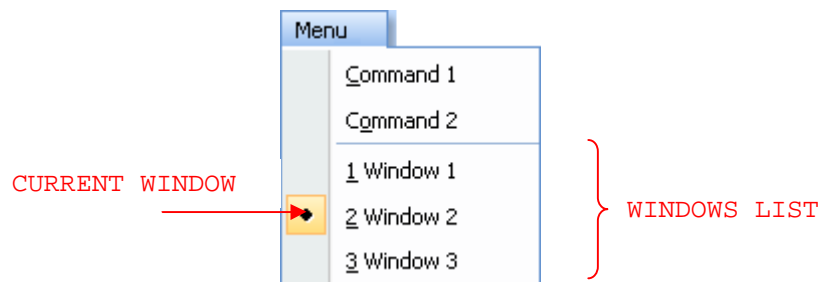
- a specific option (b) in a menu (p) : freeze mb, p=index, b=index or unfreeze mb, p=index, b=index.



Sample Code :

```
freeze mb
unfreeze mb
freeze mb, p=2
unfreeze mb, p=2
freeze mb, p=1, b=3
unfreeze mb, p=1, b=3
```

Eagle V14 allows appending the list of the current active document windows; not available when in SDI mode; to a menu of the menubar.

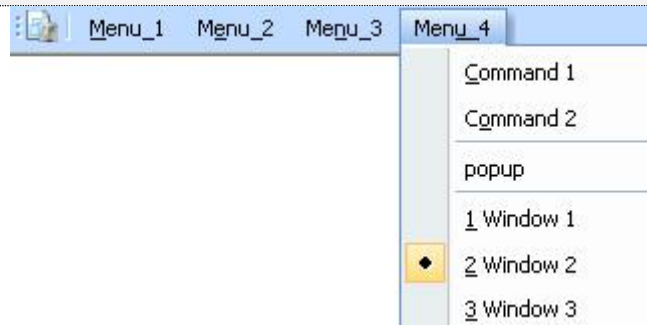


When the special option 999 is added to a "\*.men" file, the window list will be applied at the end of the menu. If there is more than one defined, only the last will be considered. A marker will be displayed adjacent to the active window enabling the user to change the current document by selecting another appropriate document item from the list; For example:

Sample Code :

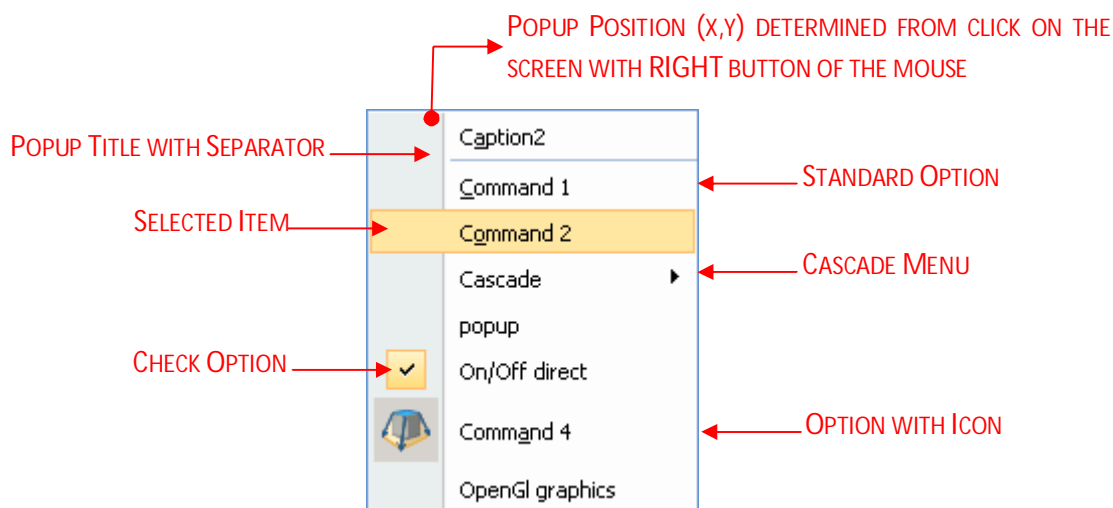
File ".men" of the menu 4 that contains the list document windows

```
'Men&u_4'
1,'&Command 1': vane
2,'C&ommand 2': plan
3,"&
999,"
5,"&
6,'popup':popup cap1
```



## 7.2 – Popup Menu

The other component introduced in this section is the "Popup Menu". In v14, as in previous versions, you can call and activate a multiple choice Window based "recursive" popup menu. Recursion means that an option in the menu file can activate a sub-popup menu in a recursive fashion, using appropriate widgets available in the windowing system. The configuration is located in the appropriate ".men" file (the same used in bar menu) with each option followed by a desired action, so like in "Menu Bar" it is possible to add icons and status symbols (Images and markers) to the text descriptive options in popup menus:



The left button of the mouse is used to select an option from the menu. If the left button is pressed outside the menu it is then deselected thereby enabling the user to move the menu using right button. Sub menus are activated moving the cursor above the options "marked" with an arrow on the right. During the "moving" phase all the exposed widgets are available for selection. Left clicking outside a popup menu causes the popup to exit.

Prototype :

popup <file>

Parameter	Description
<file>	The name of the file containing the menu. The default extension is ".men".

where:

As we have seen before for the menu file takes the format:

File Format:

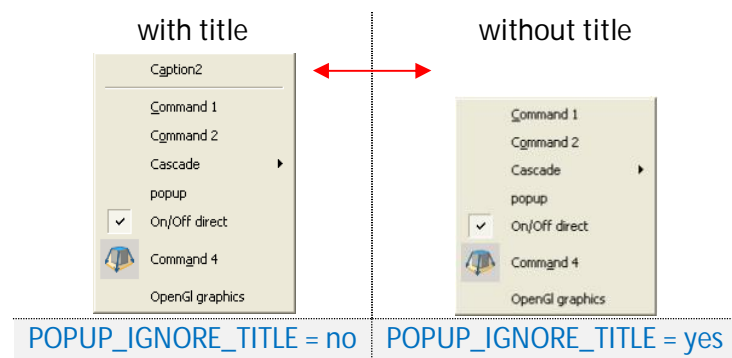
```
'title'
index, 'text':<commands>
index, 'text'><sub-menu file>
index, 'sub title'&
```

Obviously, if "index" is -1 then the option is disabled, i.e. it is half-toned and not selectable.

The first row, without index, represents the title for the popup menu; only one line is accepted, further lines are ignored. You can chose to allow no title in popup menus set through the environment variable called "POPUP\_IGNORE\_TITLE". Default value is "no", so popup with title.

INI Sample :

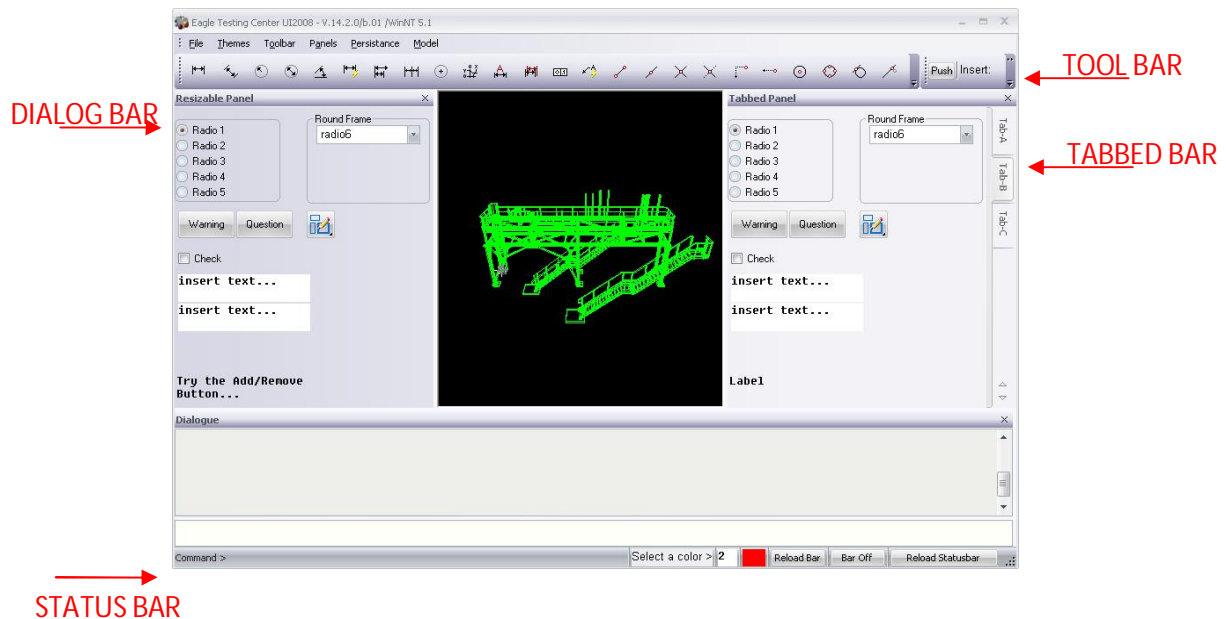
POPUP\_IGNORE\_TITLE = no | yes



Note that if the same popup file (<file>.men) is called recursively, the error/warning message is: File not found <file>.men. In the end, like bar menu, also the "Popup Menu" follows the current selected theme.



## 8 – Control Bar

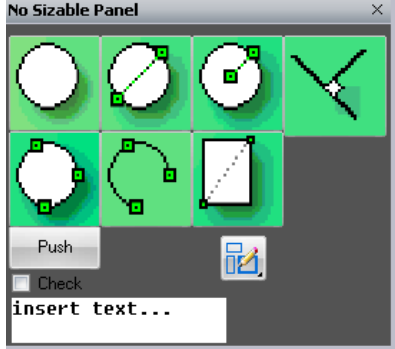
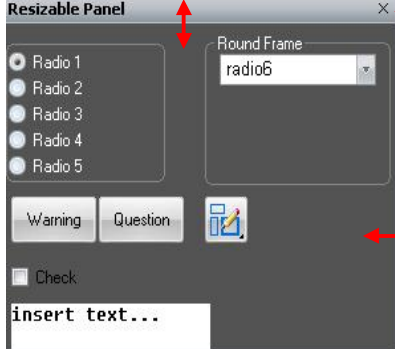
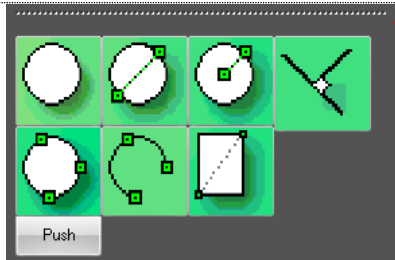
A "Control bar" is the general name for tool bars, status bars, and dialog bars. In Eagle v.14 all these components also have common functionality and are windows that display rows of controls from which users can select options, execute commands, or obtain program information. Types of control bars include tool bars (docked, floating and pull-down), dialog bars (fixed, sizable, not sizable, floating, tabbed or modal), and status bars. As might be expected the rendering appearance of all control bars is influenced by the currently selected THEME.



Control bars greatly enhance a program's usability by providing quick, one-step command actions. Toolbars, status bars, and dialog bars have some common functionality, so it's possible to think about a unique element, "Control Bar" (also called "panel" in Eagle) that creates an area on the screen which can contain menu buttons.

A panel provides the functionality for positioning the bars in its parent frame window. A control bar object uses information about its parent window's client rectangle to position itself. Then it alters the parent's remaining client-window rectangle so that the client view or MDI client window fills the rest of the client window. The next table reports a list of the all "Control Bar" layouts permitted in Eagle v.14:

Type	Sample	Notes
Tool Bar		Classical tool bar layout with the chevron menu for the hidden controls
Status Bar		Fixed panel at the bottom of the Main Window that reports generically information about the application status

Non Sizable Dialog Bar		Dialog, not resizable, that contains a panel with some controls inside.
Sizable Dialog Bar		Dialog resizable using the mouse cursor that contains a panel with some controls inside.
Fixed Size Dialog Bar		This panel type has the behavior (the docking mechanism, for instance) of a toolbar bar so it can include a dialog panel in the same way as a dialog bar.

Note that elements like Tabbed Bar aren't listed because they use the dialog layout to define the presentation of the "Tab Container".

### 8.1 – How to create a control bar: the panel command

Eagle "panel" and "options" commands are the base instructions used to define aspect and behavior of control bars. The "panel" command creates the container and "option" command is used to describe the controls inside the control bar. In this section we introduce the first command and the second will be explained in another chapter when controls will be introduced.

The "panel" command has a complex prototype, because the same instruction can be used to create many different kinds of objects (tool bar, status bar, dialog bar, and so on):

Prototype :	<p>panel {on off}, p=&lt;i&gt;, u=&lt;i&gt;, r=&lt;i&gt;, c=&lt;i&gt;, vr=&lt;i&gt;, vc=&lt;i&gt;, t=&lt;text&gt;, j=&lt;n1&gt;,&lt;n2&gt;, mod=&lt;i&gt;, pin=&lt;i&gt;, name=&lt;i&gt;, att=&lt;i&gt;, ob=&lt;i&gt;, ot=&lt;i&gt;, ol=&lt;i&gt;, or=&lt;i&gt;, open, clos, dock=&lt;i&gt;</p>
	or
	<p>panel {on off}, p=&lt;i&gt;, u=&lt;i&gt;, r=&lt;i&gt;, c=&lt;i&gt;, vr=&lt;i&gt;, vc=&lt;i&gt;, t=&lt;text&gt;, j=&lt;n1&gt;,&lt;n2&gt;, mod=&lt;i&gt;, pin=&lt;i&gt;, name=&lt;i&gt;, att=&lt;i&gt;, open, clos,</p>

	container
	combined with:
panel	{on off}, p=<i>, u=<i>, r=<i>, c=<i>, vr=<i>, vc=<i>, t=<text>, pageof=<i>, open, clos, tabbed

where:

Parameter	Description
on	Initiate a new panel.
off	Remove a panel from the screen.
p=<i>	To specify the panel index (ID). A value between 1 and 96.
u=<i>	Is the unit in pixel of the menu grid ( default is 16).
r=<i>	Number of rows. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
c=<i>	Number of columns. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
vr=<i>	Number of visible rows. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
vc=<i>	Number of visible columns. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
t=<text>	Panel title.
j=<n1>,<n2>	<b>Floating Control Bar (Standard or Tab Container):</b> Position (x, y) on the screen in normalized units between 0 and 1 or in pixel unit. If j is not specified the default values are the center of the application frame.
	<b>Docked Tool Bar:</b> Docking position (x, y) specified with J=<x_value>,<y_value>, defined in pixels. If j is not specified the default values are (-1, -1) = new row/column
	<b>Cascade toolbars:</b> Using J parameter there is the possibility of managing the layout on "pulldown" toolbars. These are horizontal or vertical toolbars that appear attached the button from which they have been called. The n1 determinates if the pulldown perpendicular (-2) or parallel (-3) to the calling button. The n2 defines the number of rows/columns.
	Not used in Tab Page.
mod=<i>	<b>Standard Control Bar:</b> Is 0 if the panel widget is child of the non graphic widget hierarchy; this means that the panel widget is still present also after a DIALOG OFF command. It is 1 if the panel widget is child of the graphic widget hierarchy; this means that the panel widget is iconized when a



	<p>DIALOG OFF command is executed and de-iconized as soon the graphic area is restored via a DIALOG ON command.</p> <p><b>Tab Container :</b></p> <p>Specifies the type of container; the value between 1 and 16.</p> <p>Not used in Tab Page.</p>
<b>pin=&lt;i&gt;</b>	<p>To specify the type of panel, can be of one of the following types:</p> <ul style="list-style-type: none"> <li>• floating dialog</li> <li>• fixed dialog</li> <li>• dockable/floating dialog</li> <li>• dockable/floating toolbar</li> <li>• status bar</li> <li>• floating (only) toolbar</li> <li>• floating (only) dialog</li> <li>• pulldown toolbar</li> </ul> <p>Permitted values are: 0,1,-2,-3,-4,-5,-6,-7,-8,-9,-10,-11,-14,-17,-18,-19,-20,-21,-22,-37,-38,-39,-40, and the differences between each possibilities will be presented in the next sections.</p> <p>Not used in Tab Page.</p>
<b>name=&lt;i&gt;</b>	<p>If equal to 1 then the resource name is, <i>panel_&lt;i&gt;</i>, where "i" is the panel index, otherwise, equal to 0, the resource name is <i>panel</i>.</p> <p>Not used in Tab Page.</p>
<b>att=&lt;i&gt;</b>	<p>If set to -1 panel PIN option setting will define whether or not a panel is floating/dockable and to which extremity of the container windows it should be docked.</p> <p>Else if equal to -999 and the panel is a toolbar, the bar will be created as floating from the start, but then dockable, you have to specify ATT=-999 and J=x,y (the starting position).</p> <p>Not used in Tab Page.</p>
<b>dock = &lt;i&gt;</b>	<p>This parameter, if specified, enables blocking of vertical or horizontal docking when the docking mechanism is the same as the Tool Bar:</p> <ul style="list-style-type: none"> <li>0 = no block for docking, docking everywhere</li> <li>1 = block horizontal docking, only vertical docking permitted</li> <li>2 = block vertical docking, only horizontal docking permitted</li> </ul> <p>Usable only in Tool Bar and FixedSize Bar</p>
<b>pageof=&lt;i&gt;</b>	<p>Represents the id of the Tab Container in which the Tab Page will be added.</p> <p>Not used in Standard Panels and Tab Container.</p>
<b>open</b>	<p>To SHOW a hidden panel.</p>
<b>clos</b>	<p>To HIDE the panel without destroying it. The panel is held in memory for successive use. (This corresponds to an unmanaged widget call). When using this primer no other primers are available.</p>

<b>container</b>	To set a specific panel as "Tab Container", according with PIN value. <b>Not used in Standard Panels and Tab Page.</b>
<b>tabbed</b>	To set a specific panel as "Tab Page". <b>Not used in Standard Panels and Tab Container.</b>
<b>col1</b>	Used when the Tab style is "OneNote"(mod=2,6,10,14),, to set the first color for the background of the Tab Page (more details below). <b>Not used in Standard Panels and Tab Container.</b>
<b>col2</b>	Used when the Tab style is "OneNote" (mod=2,6,10,14), to set the second color for the background of the Tab Page (more details below). <b>Not used in Standard Panels and Tab Container.</b>

The first prototype allows creation of simple panels like tool bar, status bar and dialog bar, whereas the second permits definition of a tabbed bar with a combination of two different commands, one for the container and the other for the pages (more details in the "Tabbed Bar" section).

Note the difference with the previous Eagle versions, where the command "panel" has been changed to remove the restriction of having only one tabbed panel (called p=96). Now it is possible to create more than one tabbed bar without restriction to a fixed id (more details in the "Tabbed Bar" section).

The complete list of "pin" values is presented in the following table:

PIN	Type	Creation Docking Status	Resizable	Dockable	Note
0	<i>Floating Panel (legacy)</i>	Floating	✗	✗	deprecated in v.14
1	<i>Fixed Panel (legacy)</i>	Fixed	✓	✗	deprecated in v.14
-23	<i>Fixed panel</i>	Fixed	✗	✗	no caption, position relative to Main window
-2	<i>Tool Bar</i>	Left	✓	✓	
-3	<i>Tool Bar</i>	Bottom	✓	✓	
-4	<i>Tool Bar</i>	Right	✓	✓	
-5	<i>Tool Bar</i>	Top	✓	✓	
-6	<i>Status Bar</i>	Fixed	frame	always	size depends on Main Frame
-7	<i>Dialog Bar Not Sizable</i>	Left	✗	✓	
-8	<i>Dialog Bar Not Sizable</i>	Bottom	✗	✓	
-9	<i>Dialog Bar Not Sizable</i>	Right	✗	✓	
-10	<i>Dialog Bar Not Sizable</i>	Top	✗	✓	
-11	<i>Floating Only Panel</i>	Float Only	✗	✗	deprecated in v.14
-14	<i>Floating Only Tool Bar</i>	Float Only	✓	✗	
-17	<i>Dialog Bar Sizable</i>	Left	✓	✓	
-18	<i>Dialog Bar Sizable</i>	Bottom	✓	✓	
-19	<i>Dialog Bar Sizable</i>	Right	✓	✓	
-20	<i>Dialog Bar Sizable</i>	Top	✓	✓	
-21	<i>Floating Only Dialog Bar</i>	Float Only	✓	✗	
-22	<i>Modal Dialog Bar</i>	Float Only	✓	✗	modal

-38	<i>Fixed Size Bar</i>	Bottom	✗	✓	behavior like toolbar
-39	<i>Fixed Size Bar</i>	Right	✗	✓	behavior like toolbar
-40	<i>Fixed Size Bar</i>	Top	✗	✓	behavior like toolbar

Sample Code :

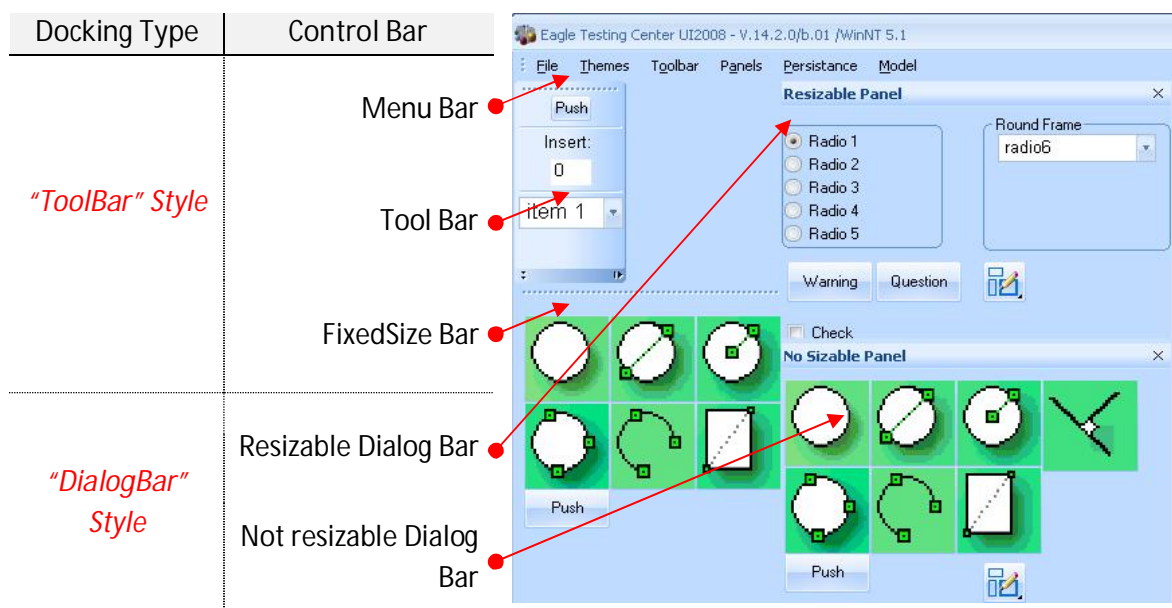
```
panel on,p=1,u=32,r=10,c=2,t='main menu',j=1,1
panel on,p=4,r=1,c=20,j=0,1
panel off,p=1
panel on,p=1,u=1,r=400,vr=200,c=200,vc=120,j=0,0
```

## 8.2 – About docking mechanisms

In previous sections the concepts of docking and layout have been introduced, now it is necessary to understand the differences between the docking mechanisms for every type of layout. Before presenting the docking types, we can observe that the “Status Bar” is the only layout element that is always fixed, consequently it doesn’t have any docking mechanism.

The expression “docking” corresponds to the action made after a panel has been dragged from one side of the screen to another. In most standard applications there are two different ways of docking, one for tool bars and another for dialogs. Usually the first is used for toolbars and the second for dialogs. Eagle v.14 provides classical docking styles and also introduces a panel that combines the behavior of toolbars with the features of dialogs (“FixedSize Dialog Bar”, pin=-37,-38,-39,-40), however we can consider it’s docking mechanism as being the same as a tool bar.

As you can see from the picture below, the most important distinction appears only when the control bars are docked, in the first case there is a gripper facilitating dragging and docking features, whereas the second case a caption enables docking/undocking of the bar.

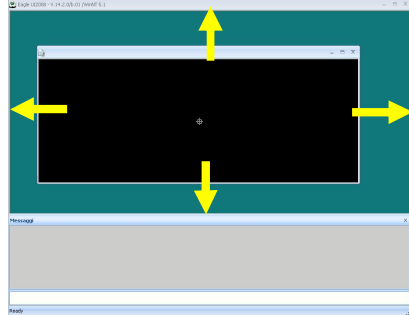
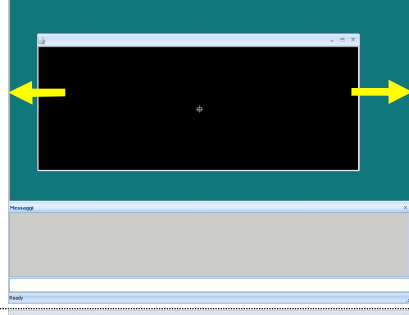
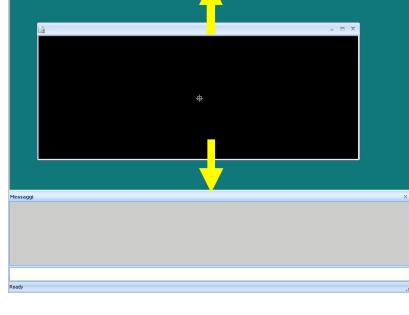


The first method, “toolbar docking”, facilitates docking of a bar attached to the borders of main frame, dragging of the bar inside the docking area, easily changing the docking site,

showing/hiding a part of the bar, undocking using a double click over the gripper and docking through double clicking on the caption.

When a “Tool Bar” or a “FixedSize Bar” are in the floating state, to re-dock the bar it’s possible to double click on the caption to re-dock the bar in the last docked position, or drag the bar to the site where we want to dock it.

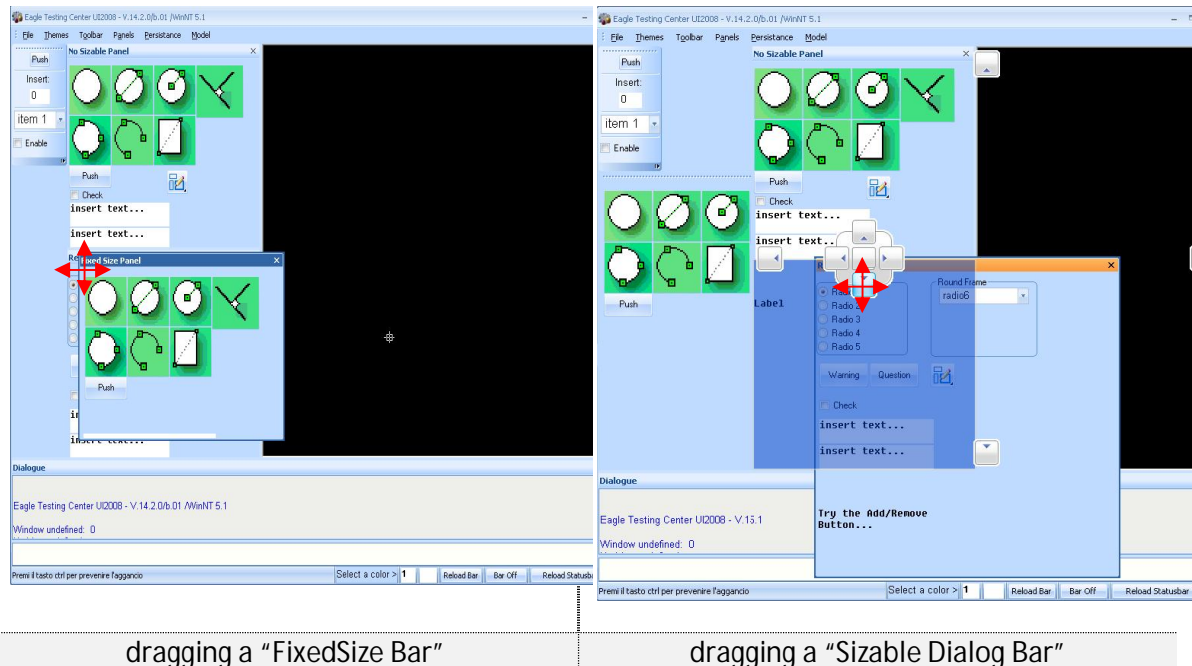
Sometimes if the bar is docked horizontally or vertically, the size or a particular layout of the bar could be result in a negative effect on the resulting GUI presentation. To prevent this situation, Eagle v.14 has introduced a new parameter in the “panel” command to block horizontal or vertical docking:

Type	Effects	Sample
0 (default)	no block for docking, docking everywhere	
1	block horizontal docking, only vertical docking permitted	
2	block vertical docking, only horizontal docking permitted	

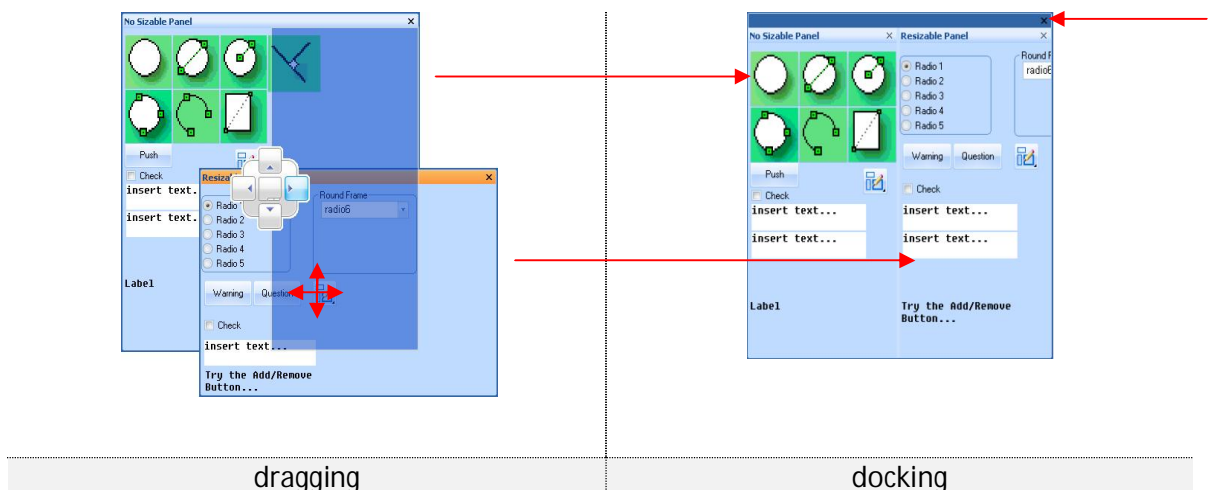
Naturally, this feature is only suitable for structures suitable for “toolbar docking” because they fit the internal dimension in relation to the docking site.

The second docking method, “dialog-bar docking”, is developed to easily control the position of dialog bars, so the application drives the user to dock the bar in the correct place relative to the main frame or next to another dialog bar.

Dragging this type of bar some markers or lines are displayed to illustrate where the bar will be docked when the mouse button is released. As we have seen in the “[Eagle Make Up](#)” chapter, this is accomplished according to the theme and/or style chosen in the configuration file.



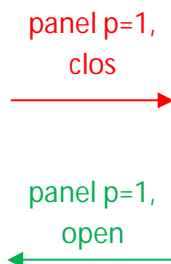
The option to dock a dialog bar relative to another dialog bar is powerful, in fact we can also dock a bar inside a floating dialog bar, just by dragging it, and the result is a “virtual” control bar that contains both dialog bars: VIRTUAL BAR



### 8.3 – Enabling and freezing panels and controls

Examining the panel command, we can see two primers “open” and “clos” that allow to showing and hiding a panel without having to recreate or destroy it:





Additionally there are others ways to enable/disable an entire bar, or a part of it, without completely hiding a panel. The “freeze” and “unfreeze” commands can be used to set one or more controls inside a control bar to a frozen or unfrozen state.

Prototype :

**freeze** p=<panel>, b=<button>

where:

Parameter	Description
<b>p</b>	Is the panel number in a range from 1 to 96.
<b>b</b>	The button number in the panel, as defined in the corresponding options file (*.tab, will be presented in the next sections). If it is not specified then all the buttons of the panel are frozen. <b>Note: not used if p represent a Tab Container</b>

The button number (b) coincides with the option number in the corresponding TAB file which we want to freeze.

Sample Code :

```
freeze p=1, b=3
freeze p=10
```

To restore sensitivity to a frozen GUI object, i.e. make it once available for selection by the user, we need to call the “unfreeze” command:

Prototype :

**unfreeze** p=<panel>, b=<button>

where:

Parameter	Description
<b>p</b>	Is the panel number in a range from 1 to 96.
<b>b</b>	The button number in the panel, as defined in the corresponding options file (*.tab, will be presented in the next sections). If it is not specified then all the

buttons of the panel are unfrozen.

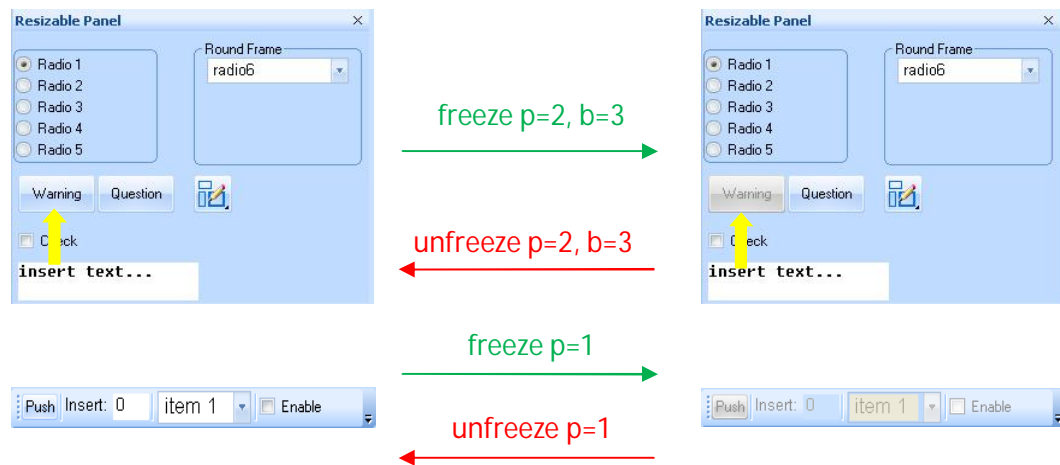
**Note: not used in Tab Container**

The button number (b) coincides with the option number in the corresponding TAB file which we want to freeze.

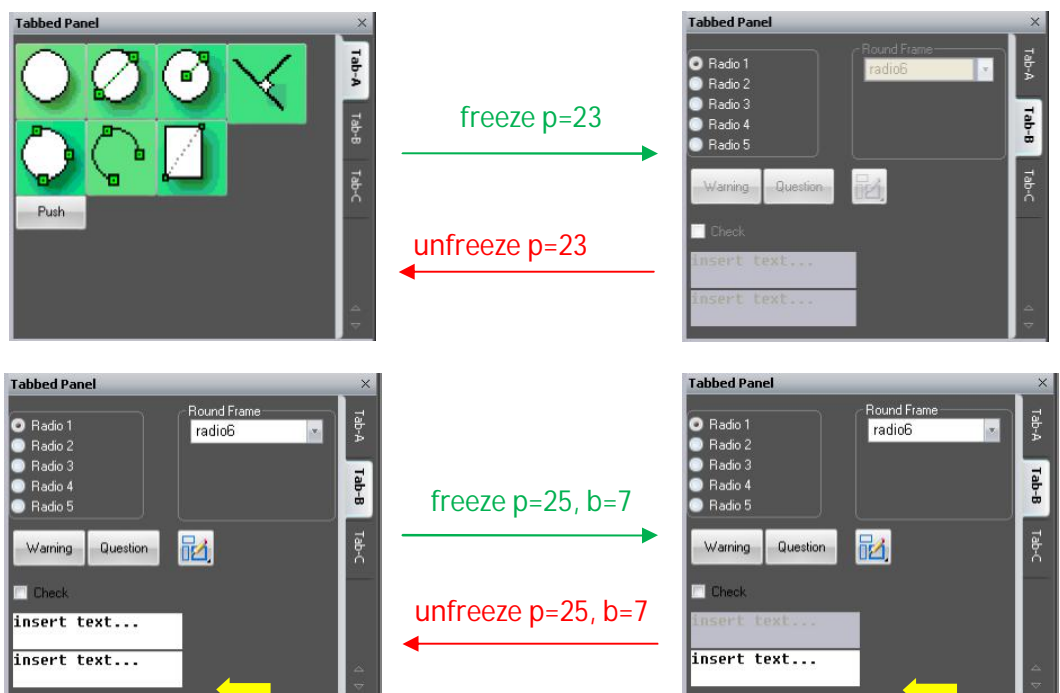
Sample Code :

```
unfreeze p=1, b=3
unfreeze p=10
```

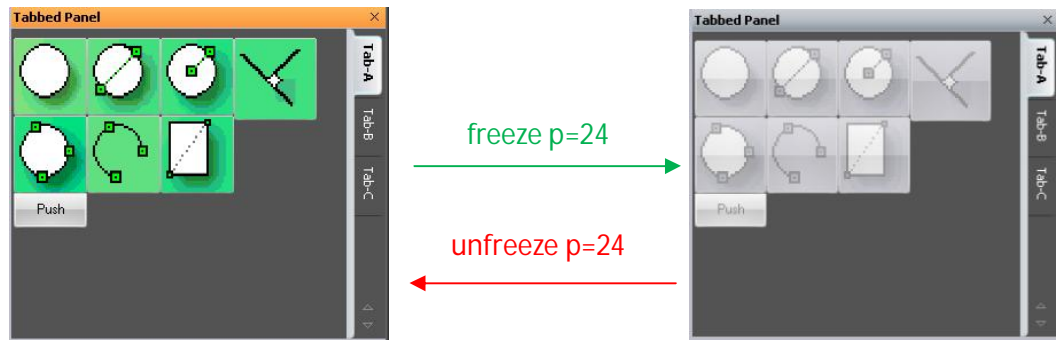
The next illustrations explain the effects of the freeze and unfreeze commands:



In the "Tab Container" panel, using these commands, it is possible to freeze/unfreeze the full tabbed bar, to freeze/unfreeze a single "Tab Page" or a single control in a page. The commands must be applied to the appropriate "Tab Page".







## 8.4 – Moving inside panels

When a control bar has the focus (for instance in the Dialog Bar the caption is highlighted), or when a button inside a bar holds the focus, it's possible to navigate through controls in the panel using the shortcut-keys <TAB> or <ALT+TAB>.

With <TAB> the focus will be made in the forward direction (the next appropriate button) and with the <ALT+TAB> the focus will be moved backward in position (the previous control to the current one).

The order of controls in the navigation "list" is determined by the order of creation: if button 'A' is created earlier than 'B', then 'A' is positioned before 'B' in the navigation list.

## 8.5 – Coordinates

Another feature, occupying the space in the middle ground between Eagle's core and the graphic user interface, is represented by the "coordinate" command. The instruction dynamically displays the current mouse position in world coordinates and/or the distance from the previous point and/or the angle formed with the previous point in a text field of a panel.

Prototype :

`coordinate on | off, p=<n>, p=<n>, t=<n>`

where:

Parameter	Description
<code>on</code>	Enable coordinate functionality.
<code>off</code>	Disable coordinate functionality.
<code>p=&lt;n&gt;</code>	Refers to the panel number.
<code>b=&lt;n&gt;</code>	Refers to the button number in the panel.
<code>t=&lt;n&gt;</code>	Permitted value >= 100 If T=100 then the COORDINATE command calls the hook function in PLNGHOOK.DLL (DYNAINPUT) to calculate the buffer to display. The



T-value minus 100 represents the last parameter passed to the hook function "sense", which can be used to change the way to transform the values or to format the output string.

Note: When the command COORDINATE is executed with a T-value greater than 100, the function PlngHookInitialize() from the PLNGHOOK.DLL is called with parameter equal to (T-value - 100), to allow a custom initialization of the hook function

Sample Code :

coordinate on, p=6, b=1 → where b is an edit, label or sunken label field

GUI Results :

230.00, 100.00, 0.00	470.00, 240.00, 100.00	-380.00, 460.00, 100.00
b = Edit	b = Label	b = Sunken Label

To enable coordinates in a text field it is required that the "STATUSBAR\_COORDINATES" entry in the configuration file is set to "no"; the default value is "no".

INI Sample :

STATUSBAR\_COORDINATES = no | yes

If the flag is "yes", the coordinates will be displayed in combination with the polling hint.

## 8.6 – Destroying panels

A control bar can be destroyed and removed from the memory, using the "off" entry of panel command.

Sample Code :

panel off, p=10

Panels can however also show the close button on the caption bar:

- 1 Tool Bar, when it is floating
- 2 FixedSize Bar, when it is floating

MENU\_CLOSE\_ACTION



- 3 Non Sizable Dialog Bar, always (docked or floating)
- 4 Sizable Dialog Bar, always (docked or floating)
- 5 Tabbed Bar, always (docked or floating)



BAR\_CLOSE\_ACTION

You can handle the events of the close button using entries in the INI file. The “BAR\_CLOSE\_ACTION” entry controls the closing actions for toolbars whereas; the “MENU\_CLOSE\_ACTION” entry contains the instructions to run when the close button is pressed for others control bars. The table below shows the correspondence between bar types and the relevant configuration file setting:

Panel	INI file – Close Action
1 Tool Bar when is floating	<a href="#">BAR_CLOSE_ACTION</a>
2 FixedSize Bar when is floating	<a href="#">MENU_CLOSE_ACTION</a>
3 Non Sizable Dialog Bar always (docked or floating)	<a href="#">MENU_CLOSE_ACTION</a>
4 Sizable Dialog Bar always (docked or floating)	<a href="#">MENU_CLOSE_ACTION</a>
5 Tabbed Bar always (docked or floating)	<a href="#">MENU_CLOSE_ACTION</a>

INI Sample :

```
BAR_CLOSE_ACTION = C:\MyFile\OnToolbarClose.cmd
MENU_CLOSE_ACTION = C:\MyFile\OnMenuClose.cmd
```




The execution of the “close button” of a Panel event causes the polling to exit and returns a BN (button number) value set to – 1 to inform the application about the occurrence of a special event.

## 8.7 – Status Bar

A “Status Bar” is an information area typically found at the bottom of windows in a graphical user interface and sometimes it is divided into sections, each of which shows different information. Its job is primarily to display information about the current state of its window, although some status bars have extra functionality. For example, many web browsers have clickable sections that pop up a display of security or privacy information. So, also in Eagle v.14, status bar is placed at the bottom of the graphic window to monitor the value of some parameters of interest for the current session.

The bar is activated at the end of the startup process. Most of the code that relates to updating the information in the status bar is spread throughout the application at relevant points which deal with the status information.

The “Status Bar” is not so very different from previous versions of Eagle, with the exception of the rendering style that will correspond to the currently defined theme. The Status Bar also has extended support to include insertion of all types of control (in earlier Eagle versions, not every type of button could be added to the status bar):

Windows Xp	
Visual Studio	
Office 2007	

In Eagle, the status bar is considered like a generic panel (toolbar, dialog bar, etc...). So, to install a new status bar in the application, you as the developer must first invoke the “panel” command using the value PIN = -6. When Eagle starts the main frame has an “empty” bar with only a prompt message (string in the left-bottom angle) and the gripper (in the right-bottom angle);



to add a customized status, you must invoke the “panel” command using the same approach shown in the next example:

**Sample Code:**

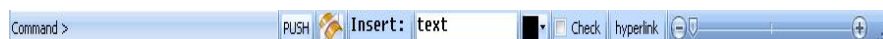
```

numeric id, column, row, pixel, type

id      = 1    →    Panel Identifier
type    = -6   →    Panel Type, Status Bar is always -6
pixel    = 1    →    Unit in pixel to estimate the effective height
column  = 1    →    Size for width; must be specified and >0
row     = 24   →    Height in pixel ('r' x 'u' = 24 x1)

panel p = id, on ,u = pixel, c = column, r = row, pin = type

option p = id, f = status.tab
  
```

**GUI Result :**


The size of a control bar depends on the values of units in pixel ("u ="), rows ("r =") and columns ("c ="). In status bar the columns size must be >0 but Eagle v.14 has auto-fitting algorithms to show the bar in the best possible way, so any "c" value >0 will produce the same outcome.

Note that unit pixels data also effects controls inside the bar, so the combination [u=4, r=6] isn't the same as having [u=1, r=24] if the control has been defined thinking u=1.

The schema below illustrates how to use the command "panel" to work with status bar objects.

**Prototype :**

```

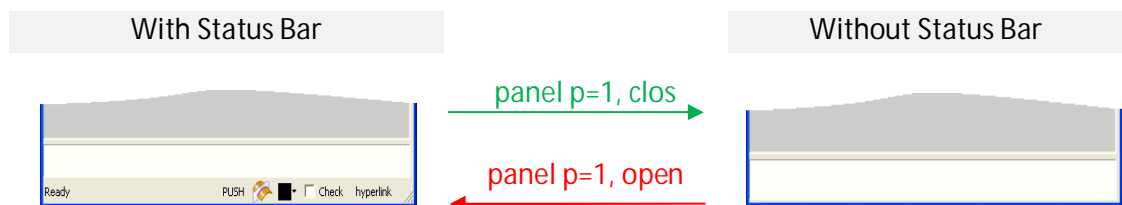
panel {on|off}, p=<i>, u=<i>, r=<i>, c=<i>, vr=<i>, vc=<i>, t=<text>,
j=<n1>,<n2>, mod=<i>, pin=<i>, name=<i>, att=<i>, ob=<i>,
ot=<i>, ol=<i>, or=<i>, open, clos
  
```

where:

Parameter	Description
<b>on</b>	Creates status bar.
<b>off</b>	Removes bar from the screen, and restores the "empty" status bar
<b>p=&lt;i&gt;</b>	To specify the panel index (ID). A value between 1 and 96.
<b>u=&lt;i&gt;</b>	Is the unit in pixel of the menu grid ( default is 16).
<b>r=&lt;i&gt;</b>	Width of the status bar. Value must be >0, but each values are adjusted with the auto-fitting algorithm
<b>c=&lt;i&gt;</b>	Height of the status bar.
<b>vr=&lt;i&gt;</b>	Visible width of the status bar. Value must be >0, but each values are adjusted with the auto-fitting algorithm

<b>vc=&lt;i&gt;</b>	Visible height of the status bar.
<b>pin=-6</b>	-6 is the only permitted pin type
<b>open</b>	To SHOW a hidden panel.
<b>clos</b>	To HIDE the panel without destroying it. The panel is held in memory for successive use. (This corresponds to an unmanaged widget call). When using this primer no other primers are available.

Using the “clos” parameter, also allows to Eagle to completely hide this component , but it is necessary to create a status bar (even an empty one) first , for example:



In the left corner the “Status Bar” is displayed with the polling hint as well as any other Eagle prompt messages, for instance:

Text >

My Polling Message

In order to enable the display of “hint” information it’s necessary to set, “POLLING\_HINT\_IN\_STATUSBAR” in the INI file to “yes”. It is also possible to display the polling hint coordinates and this is achieved by setting the “STATUSBAR\_COORDINATES” entry to “yes”.




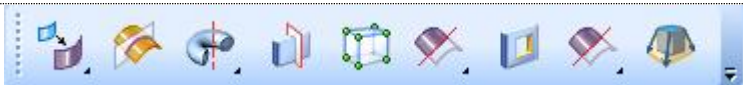
INI Sample :

POLLING\_HINT\_IN\_STATUSBAR = no | yes

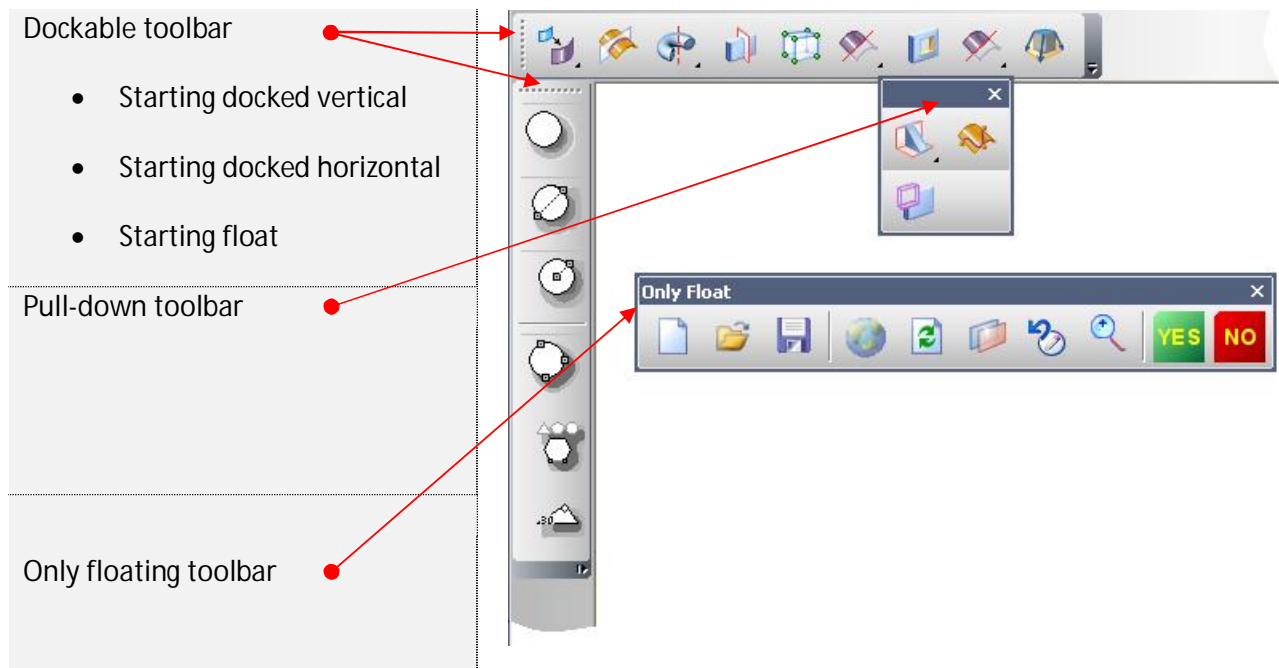
## 8.8 – Tool Bar

A toolbar is a set of tools that are grouped together into an area on the main window of an application. Typically, the tools are arranged into a horizontal strip called a bar, hence the term toolbar is the shortening of the original term tool bar. Vertical and floating toolbars are also possible, though they're not as common. The tools on a toolbar provide quick and convenient access to commonly-performed operations. Buttons are the most common kind of tool on a toolbar, though menus and drop-down lists are also often found. The tools on a toolbar may or may not be customizable by the user.

Eagle v.14 also provides toolbars in which buttons, icons, menus or other input or output elements are placed. This panel is placed just below the bar menu, the look is effected by the current selected theme and normally includes the frequently used features of our application.

Office 2000	
Office XP	
Office 2007 – Release 1	
Office 2007 Luna Blue – Release 3	

Eagle v.14 allows creation of various types of toolbar:



Every type of toolbar must be created using the “panel” command, in the same way presented in the samples below :

Sample Code :

```

numeric id, column, row, pixel, type, attach, xPos, yPos
string title

id      = 9           →   Panel Identifier
type    = -5          →   Panel Type, Docked at Top of the Frame
pixel    = 1          →   Unit in pixel to estimate the effective height
column  = 28          →   Size for width; must be specified and >0
row      = 32          →   Height in pixel
attach   = -1          →   Docked bar
title    = 'MyToolbar' →   Title
xPos     = 10          →   Bar positioned with x=10
yPos     = 50          →   Bar positioned with y=50

panel p = id, on ,u = pixel, c = column, r = row, pin = type, att = attach, t=title,
j=xPos, yPos

option p = id, f = toolbar.tab
  
```

GUI Result :



From the sample above, it's clear that the “panel” command can be reduced to :

Prototype :

```

panel {on|off}, p=<i>, u=<i>, r=<i>, c=<i>, vr=<i>, vc=<i>, t=<text>,
j=<n1>,<n2>, mod=<i>, pin=<i>, name=<i>, att=<i>, dock=<i>,
open, clos
  
```

where:

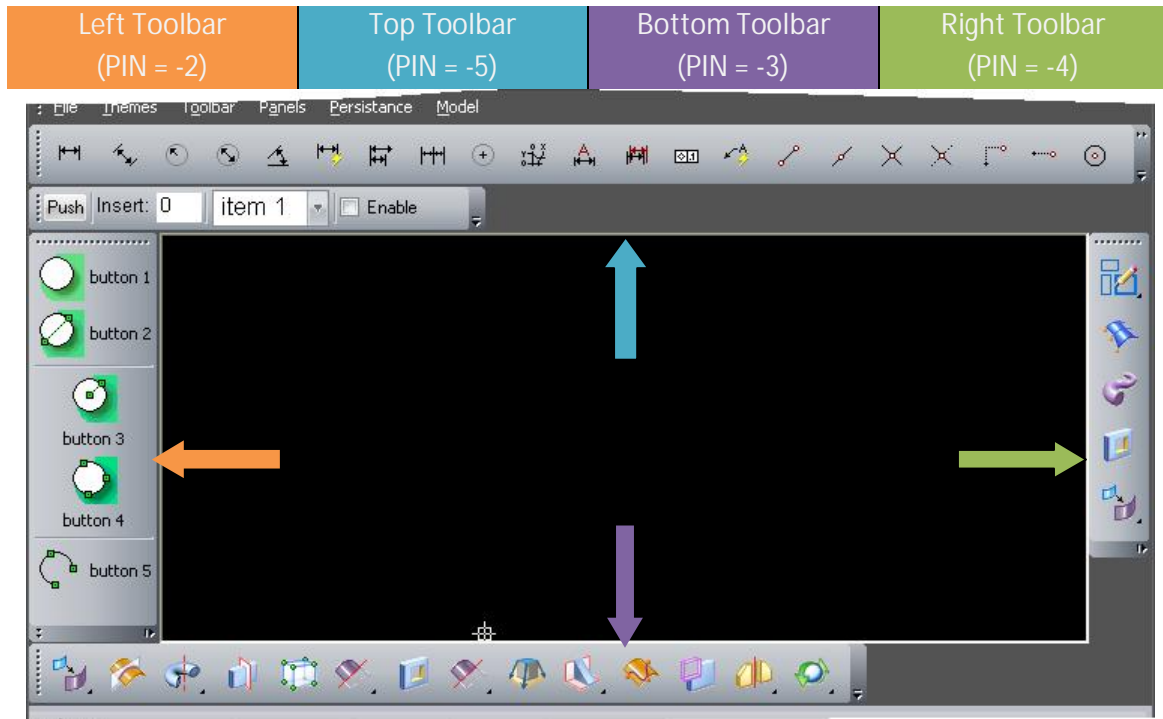
Parameter	Description
<b>on</b>	Initiate a new panel.
<b>off</b>	Remove a panel from the screen.
<b>p=&lt;i&gt;</b>	To specify the panel index (ID). A value between 1 and 96.
<b>u=&lt;i&gt;</b>	Is the unit in pixel of the menu grid ( default is 16).
<b>r=&lt;i&gt;</b>	Number of rows. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
<b>c=&lt;i&gt;</b>	Number of columns. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).

<b>vr=&lt;i&gt;</b>	<b>Floating Tool Bar:</b> Used to specify the number of rows defining the layout of a floating toolbar
<b>vc=&lt;i&gt;</b>	<b>Floating Tool Bar:</b> Used to specify the number of columns defining the layout of a floating toolbar
<b>t=&lt;text&gt;</b>	Panel title.
<b>j=&lt;n1&gt;,&lt;n2&gt;</b>	<b>Floating Tool Bar:</b> Position (x, y) on the screen in normalized units between 0 and 1 or in pixel unit. If j is not specified the default values are the center of the application frame.
	<b>Docked Tool Bar:</b> Docking position (x, y) specified with J=<x_value>,<y_value>, defined in pixels. If j is not specified the default values are (-1, -1) = new row/column
	<b>Cascade toolbars:</b> Using J parameter there is the possibility of managing the layout on "pulldown" toolbars. These are horizontal or vertical toolbars that appear attached the button from which they have been called. The n1 determinates if the pulldown perpendicular (-2) or parallel (-3) to the calling button. The n2 defines the number of rows/columns.
<b>mod=&lt;i&gt;</b>	Is 0 if the panel widget is child of the non graphic widget hierarchy; this means that the panel widget is still present also after a DIALOG OFF command. It is 1 if the panel widget is child of the graphic widget hierarchy; this means that the panel widget is iconized when a DIALOG OFF command is executed and de-iconized as soon the graphic area is restored via a DIALOG ON command.
<b>pin=&lt;i&gt;</b>	To specify the type of panel, can be of one of the following types: <ul style="list-style-type: none"> <li>• dockable/floating toolbar</li> <li>• floating (only) toolbar</li> <li>• pulldown toolbar</li> </ul> Permitted values are: -2,-3,-4,-5,-14.
<b>name=&lt;i&gt;</b>	If equal to 1 then the resource name is, <i>panel_&lt;i&gt;</i> , where "i" is the panel index, otherwise, equal to 0, the resource name is <i>panel</i> .
<b>att=&lt;i&gt;</b>	If set to -1 panel PIN option setting will define whether or not a panel is floating/dockable and to which extremity of the container windows it should be docked. Else if equal to -999 and the panel is a toolbar, the bar will be created as floating from the start, but then dockable, you have to specify ATT=-999 and J=x,y (the starting position).
<b>dock = &lt;i&gt;</b>	This parameter, if specified, allow to block the vertical or the horizontal docking when the docking mechanism is the same of in Tool Bar: <ul style="list-style-type: none"> <li>0 = no block for docking, docking everywhere</li> <li>1 = block horizontal docking, only vertical docking permitted</li> <li>2 = block vertical docking, only horizontal docking permitted</li> </ul>

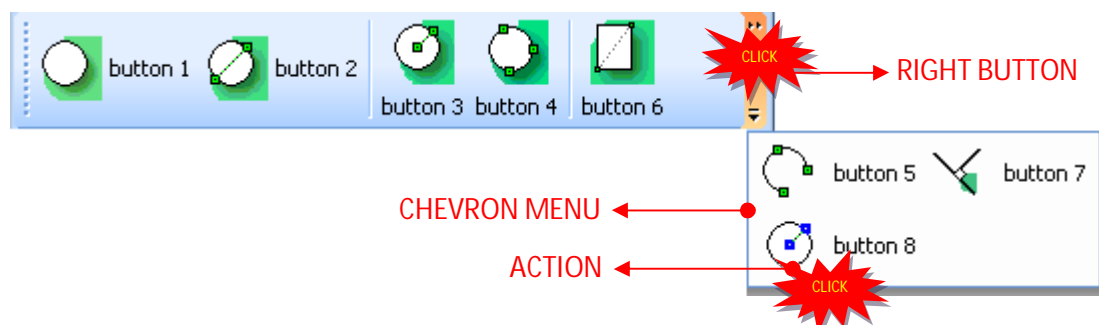


open	To SHOW an hidden panel.
close	To HIDE the panel without destroying it. The panel is held in memory for successive use. (This corresponds to an unmanaged widget call). When using this primer no other primers are available.

Clearly, the most commonly used "Tool Bar" is the docking type which obviously can be docked vertically (TOP or BOTTOM) and horizontally (RIGHT or LEFT).




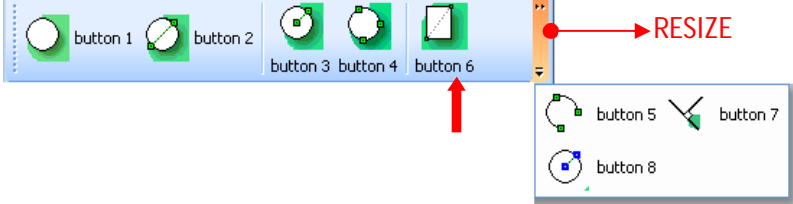
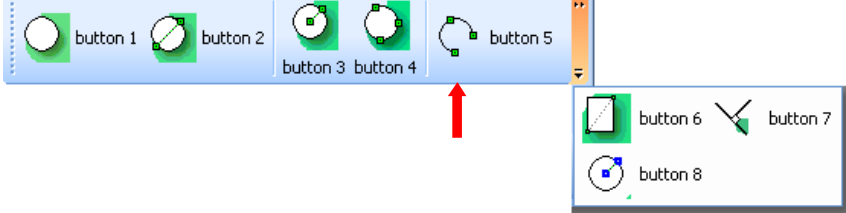
When a toolbar is docked Eagle v.14 provides a method to auto-fit the best layout. Sometimes, if there isn't enough space, not every button is displayed, for instance when there are more bars in the same row/column or when the Main Window has been reduced in width. In this situation, the controls are not completely hidden, but in the same way of the most recent Microsoft releases, there is a button in the right side (called "right button") that, if pressed, shows a "chevron menu" which contains all the hidden buttons.



Observe in the illustrated image, that the system has inserted "button 5" in the chevron menu and has shown "button 6" in the tool bar. The current dimensions of the toolbar only allow

showing of a control with a size that is less than size of “button 5”, so the first available suitably sized control in the chevron menu is “button 6” and not “button 5”.

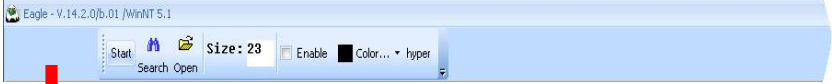

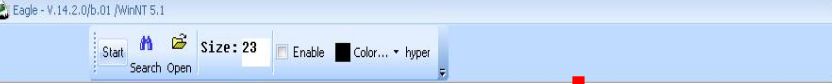
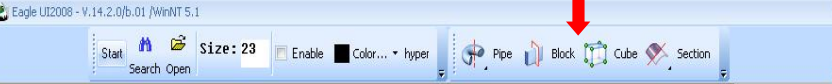
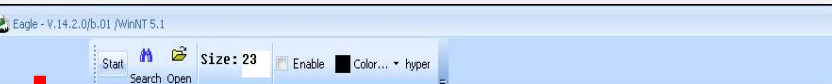





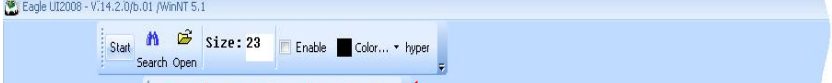


Naturally if any button in the chevron hasn’t a suitable dimension, the space will remain empty. If now we increase the size of toolbar so that it can contain “button 5”, application will now put this control in the bar and “button 6” is now placed in the chevron menu instead. The following table illustrates how the “chevron menu” works:

1	<p>toolbar completely visible large enough no chevron menu needed</p>	
2	<p>main window resized a parts of the toolbar goes in the chevron menu (this time button 6 instead of button 5)</p>	
3	<p>toolbar resized there is enough space to display the button 5 (which means button 6 goes to the chevron menu)</p>	

Of course, pressing a button in the chevron menu performs the same action as the corresponding control in the toolbar.

During the creation of a docked toolbar, using the “j” parameter of the “panel” command, it’s possible to manage the position of toolbars and consequently we can insert a new toolbar before or after another one or in a new row or column.

The next illustrations describe the docking types available using the j parameter, in the case of horizontal docking. Naturally when toolbars are in vertical we can find the same behavior:


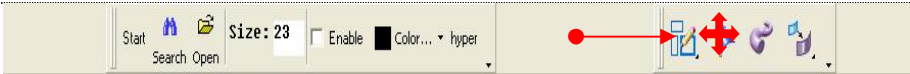
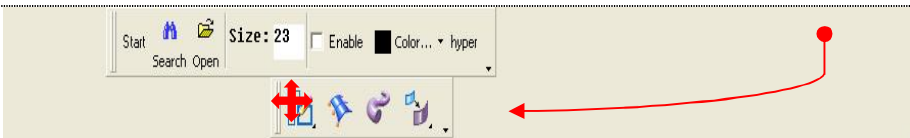
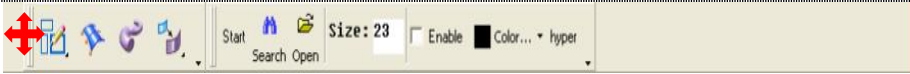
Insert a toolbar before		
1	panel p=1,on,..., j=30, 0	
2	panel p=2,on,..., j=0, 0	
Insert a toolbar after		
1	panel p=1,on,..., j=30, 0	
2	panel p=3,on,..., j=50, 0	
Insert a toolbar between two		
1	panel p=1,on,..., j=30, 0	
2	panel p=2,on,..., j=0, 0	
3	panel p=3,on,..., j=50, 0	
Without specify the "j" parameter		
1	panel p=1,on,..., j=30, 0	
2	panel p=1,on,...	
Insert a toolbar in a new row, specifying an exact position		
1	panel p=1,on,..., j=30, 0	
2	panel p=1,on,..., j=40, 100	
Setting j = -1, -1		
1	panel p=1,on,..., j=30, 0	
2	panel p=1,on,..., j=-1, -1	

Eagle v.14 introduced a new feature in the "panel" function, the setting of the "dock" parameter. This parameter makes it possible to block docking in a specific orientation whether it is vertical or horizontal:




- dock = 0 : no block → toolbar dockable everywhere;
- dock = 1 : horizontal block → toolbar dockable vertically;
- dock = 2 : vertical block → toolbar dockable horizontally;

if the parameter isn't defined in the command, the default value is set to "0", every docking orientation is permitted.

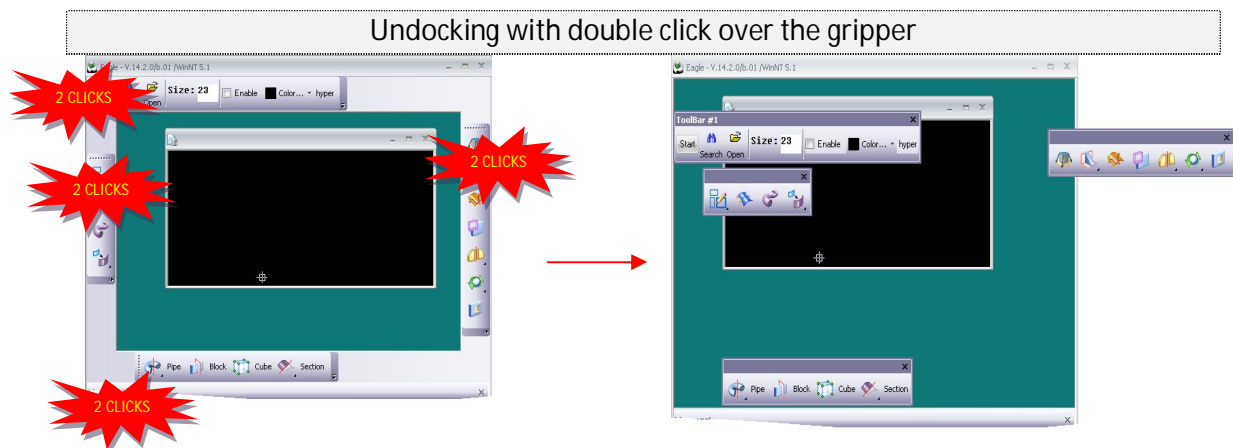
When a "Tool Bar" is docked, it can be moved inside the docking space using the "dragging mechanism", see diagram below for more details:

Start condition	
1 Move toolbar 2 – RIGHT	
2 Move toolbar 2 – UNDER toolbar 1	
3 Move toolbar 2 – BEFORE toolbar 1	

A toolbar can also be in a floating state. It is possible to have a floating toolbar undocking a docked bar, or creating the toolbar just floating. To undock a toolbar the user can drag it out from the docking space or with a double click over the gripper, in the same way to re-dock it's possible to use dragging or a double click over the caption; using the double click the toolbar will be positioned in the previous docked state:

Undock to float	
Dock with double click	
Previous position	

Note if the toolbar is undocked using the double click with the left button of the mouse, it will be placed in the first free space near the docking position :



There are two ways of creating a toolbar which is immediately available in floating state, this is achieved by using:

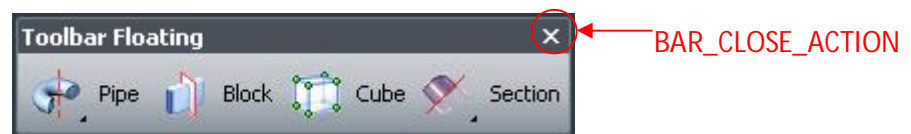
- a normal bar (pin = -2, -3, -4 or -5) with the attach parameter -999 (att=-999)

→ Toolbar starts floating but is dockable;

- the pin type -14

→ Toolbar is always floating, but not dockable.

As we have seen in another section, when the toolbar is floating, inside the caption there is a "close button" present; we can handle the event of closing the toolbar by using the INI file entry "BAR\_CLOSE\_ACTION".



INI Sample :

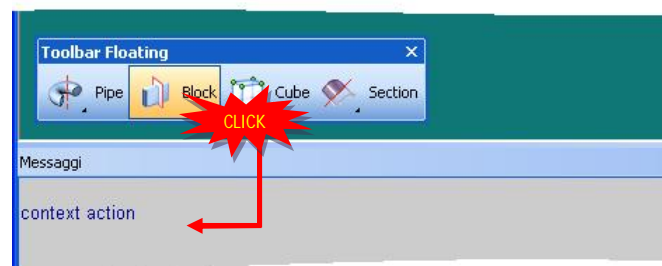
```
BAR_CLOSE_ACTION = C:\MyFile\OnToolbarClose.cmd
```

In addition, Eagle v.14 allows you to attach a "context" action that will be executed after a click of the right button inside the toolbar area. This action can be defined by using the "TOOLBAR\_CONTEXT\_ACTION" in the configuration file:

INI Sample :

```
TOOLBAR_CONTEXT_ACTION = C:\MyFile\ContexToolbarAction.cmd
```

For instance, it's possible to print a string "context menu" in the Message Bar :



Right click also returns two values in the polling loop :

- mn : toolbar identifier;
- bn : button identifier or 0 when the click has been made in a static label or in the panel area.

The last type of toolbar is the “pull-down”, that is a floating bar related to a button inside another toolbar. Pressing this “pull down” button means the pull-down toolbar will be displayed near the control.

To use a pull-down toolbar it’s necessary that the panel is created as a floating one, so by using the pin -14 or setting the “att” parameter to -999.

There are two feasible design possibilities, parallel or orthogonally to the bar caller. In relation to the layout it’s also possible to choose the number of rows/columns for the pull-down. These settings can be applied using the “j = n1, n2” parameters, where :

- n1 sets if the bar is parallel or orthogonally to the caller:

n1 = -2 → perpendicular;

n1 = -3 → parallel;

- n2 sets the number of rows/columns, this is expressed as a positive integer.

When the toolbar caller is floating, it is considered in the same manner as docked toolbar docked to the top of the frame. The next table shows a list of examples:

**Bar Caller : TOP DOCKED TOOLBAR**

**Pull-down : j = -2, 1**

panel p=15,on,u=1,r=32,c=28,pin=-5,att=-999,t='Pulldown',j=-2,1

*The pulldown has 1 column and is perpendicular to the caller button*



**Bar Caller : LEFT DOCKED TOOLBAR**

**Pull-down : j = -2, 2**

panel p=15,on,u=1,r=32,c=28,pin=-5,att=-999,t='Pulldown',j=-2,2

*The pulldown has 2 rows and is parallel to the caller button*

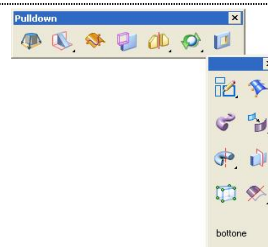


**Bar Caller : FLOATING TOOLBAR**

**Pull-down : j = -2, 2**

panel p=15,on,u=1,r=32,c=28,pin=-5,att=-999,t='Pulldown',j=-2,2

*The pulldown has 2 columns and is perpendicular to the caller button*



**Bar Caller : BOTTOM DOCKED TOOLBAR**

**Pull-down : j = -3, 3**

panel p=15,on,u=1,r=32,c=28,pin=-5,att=-999,t='Pulldown',j=-3,3

*The pulldown has 3 rows and is perpendicular to the caller button*





Bar Caller : **RIGHT DOCKED TOOLBAR**

Pull-down : **j = -3, 1**

panel p=15,on,u=1,r=32,c=28,pin=-5,att=-999,t='Pulldown',j=-3,1

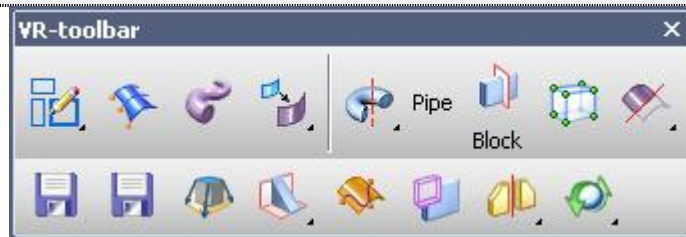


*The pulldown has 1 column and is parallel to the caller button*

In Eagle.v14 it's also possible to define a specific layout for a floating toolbar (a panel created with att=-999 or pin=-14). Developers have control over this layout option using the "vr" and "vc" attributes in the panel command which are used to respectively specify the number of rows or the number of columns for a floating toolbar.

The nature of operation of toolbars means that, if both parameters are defined, the number of columns is skipped and the layout will be controlled only by the "vr" attribute.

panel p=3,on,u=1,r=32,c=1,vr=2,pin=-5,att=-999,j=300,200,t='VR-toolbar'



panel p=5,on,u=1,r=32,c=1,vc=2,pin=-2,att=-999,j=300,200



When the floating toolbar has docked, all the controls are moved into a single row or column depending on the location of the docking site, however if the same toolbar is undocked once again to the floating position the originally defined layout will be restored.

Note that using either if these settings means that the floating layout of the toolbar is fixed and the user will not be able to resize the toolbar using the mouse.

Eagle also provides a feature to lock/unlock a toolbar. This is achieved using the "lock" command, which means that a locked docked toolbar cannot be undocked, a locked undocked toolbar cannot be closed, while conversely unlocked toolbars can be docked/undocked and closed.

Prototype :

**lock**      **p=<i> {,on|off}**

where:

Parameter	Description
<b>p</b>	Panel number (identifier for the toolbar) to which the lock should apply
<b>on</b>	Lock a toolbar (default if neither ON nor OFF are specified).
<b>off</b>	Unlock a locked toolbar.

For instance, the dragging is not allowed :



Sample Code :

```
lock p=10, on
lock p=8
lock p=8, off
```



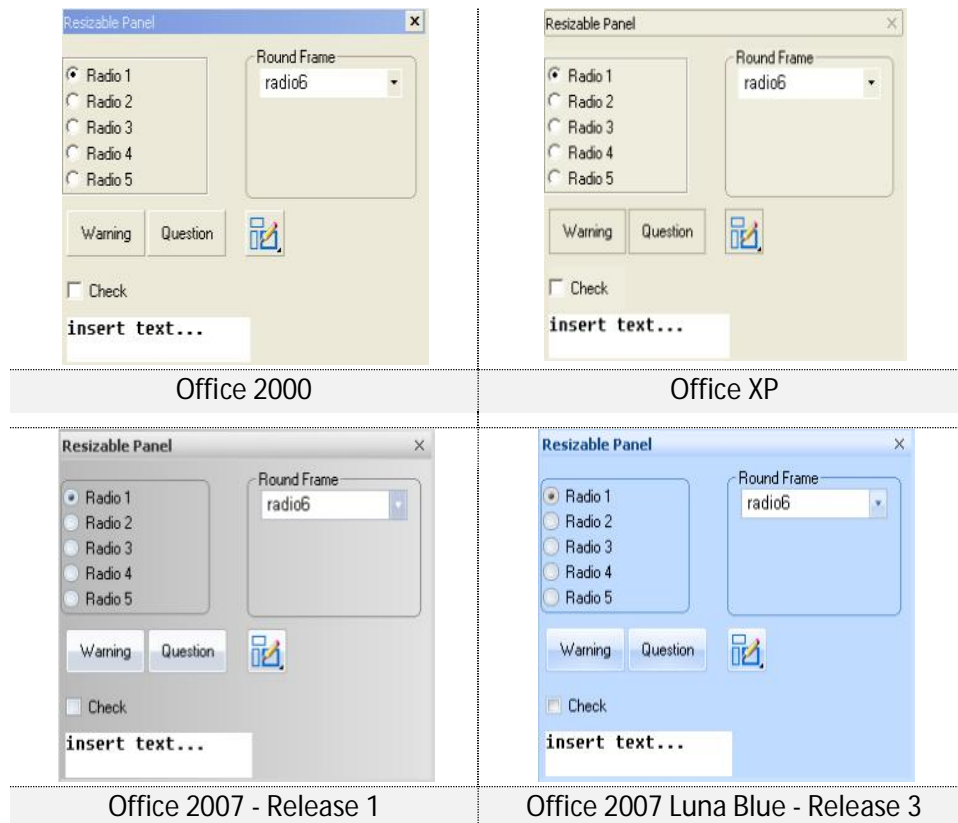
## 8.9 – Dialog Bar

A dialog bar is a kind of control bar that can contain any kind of control, just like a toolbar. Because it has the characteristics of a modeless dialog box, a “Dialog Bar” object provides a more powerful toolbar.

There are several key differences between a toolbar and “Dialog Bar” objects. A “Dialog Bar” object is created from a dialog-template resource, which can contain any kind of Windows control. The user can tab from control to control. You can specify an alignment style to align the dialog bar with any part of the parent frame window or even to leave it in place if the parent is resized.

In other respects, working with a “Dialog Bar” object is like working with a modeless dialog box. One of the virtues of dialog bars is that they can include controls other than buttons. These are objects used to complete an interface when the frame needs extra real estate to carry the desired controls of an application. One of the main justifications of control bars is that the user can display or hide them at will.

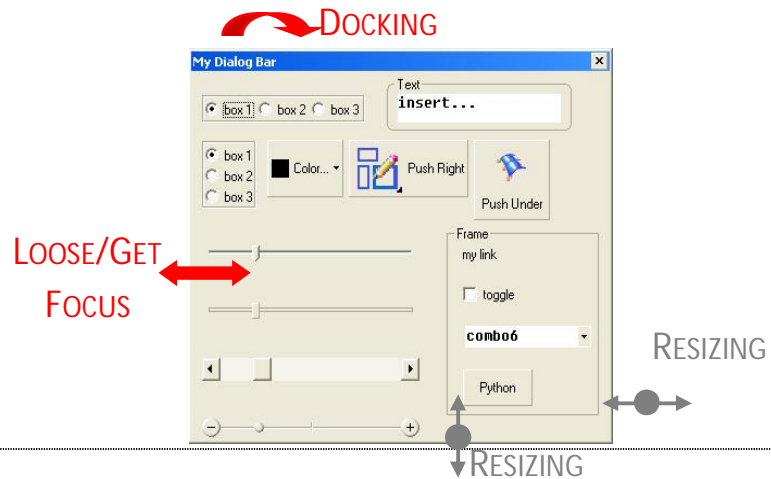
In Eagle v.14 “Dialog Bar” is an object of the family of “Control Bars”; the appearance is affected by the current selected theme:



and there are several styles : sizable, not sizable, docked, floating or modal. The next table shows the differences between each type:

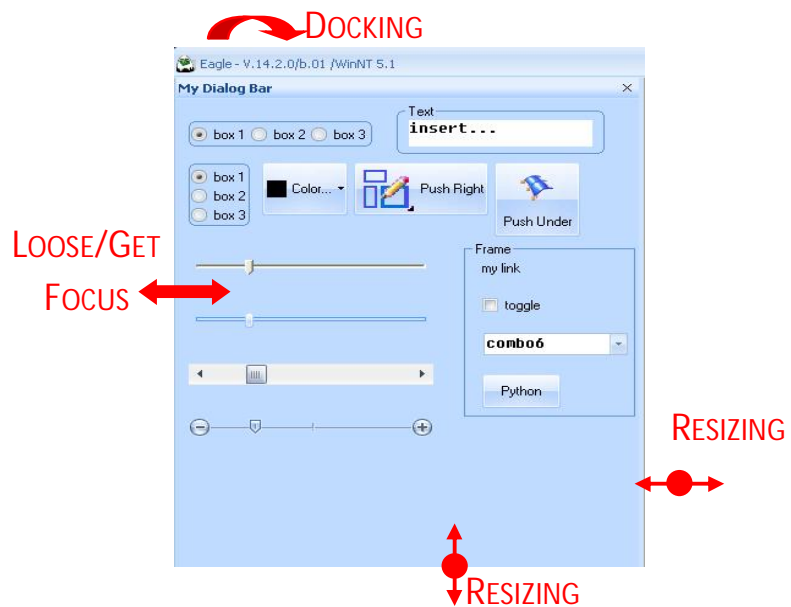
### Docked and Not sizable

PIN= -7, ATT= -1  
-8, -999  
-9,  
-10



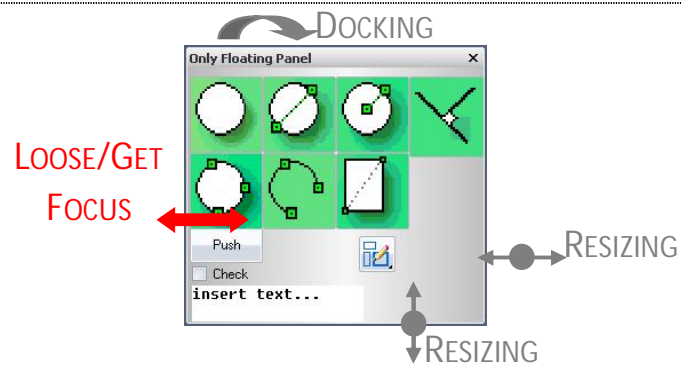
### Docked and Sizable

PIN= -17, ATT= -1  
-18, -999  
-19,  
-20



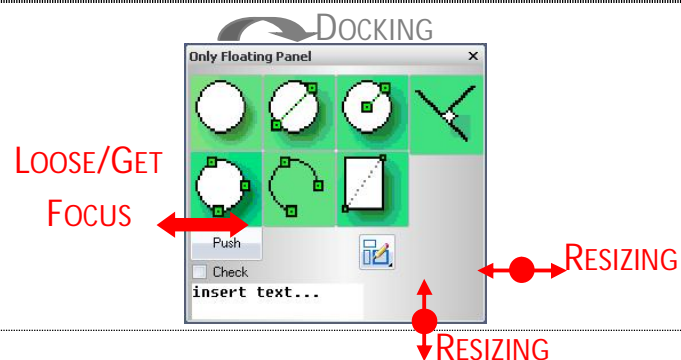
### Only Floating and Not sizable

PIN= -11 ATT= -1

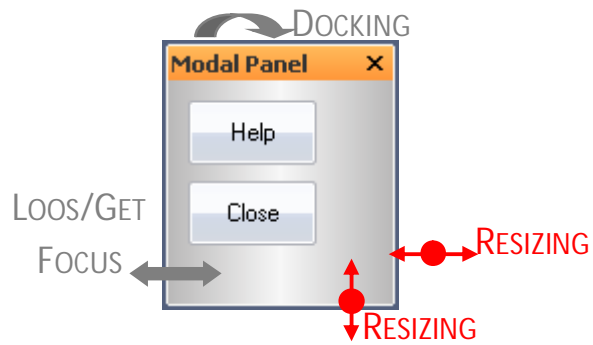


### Only Floating and Sizable

PIN= -21 ATT= -1



Modal	
PIN= -22	ATT= -1



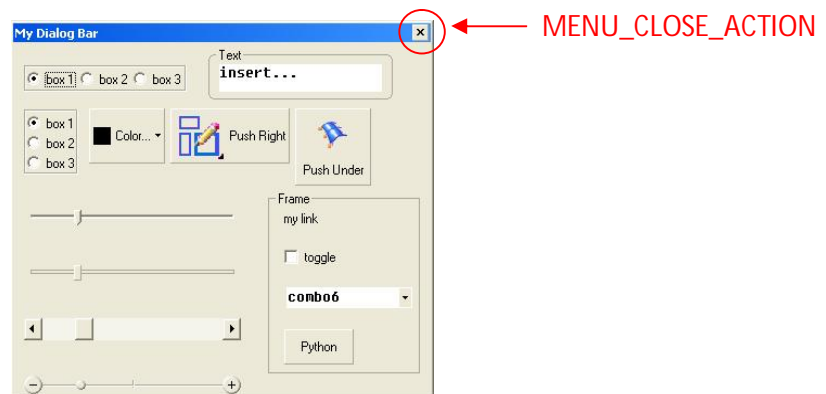
Note that in Eagle v.14 previous panel types (PIN=0 and PIN=1) are automatically converted into dialogBars as PIN=-21 (that is, floating-only resizable dialogbar). In this way, all button types are displayed with the same style and everything reflects the current active theme.

The modal dialog bar is a particular panel that maintains the focus until it is closed or destroyed. This kind of bar is very similar to dialog boxes typically used to send messages to the user (more details in the relevant chapter).

Also for “Dialog Bar”, there are two modes of creating a panel which is immediately available as floating, these are using:

- a normal bar (pin = -7, -8, -9, -10, -17, -18, -19 or -20) with the attach parameter -999 (att=-999)  
→ Dialog bar starts floating but it's dockable;
- the pin type -21  
→ Dialog bar is always floating, not dockable.

As we have seen in another section, the panel has a “close button” inside the caption . We can handle the closing event by using the “MENU\_CLOSE\_ACTION” entry in the INI file.



INI Sample :

```
MENU_CLOSE_ACTION = C:\MyFile\OnPanelClose.cmd
```

The command "panel" can be concentrated into:

Prototype :

```
panel {on|off}, p=<i>, u=<i>, r=<i>, c=<i>, vr=<i>, vc=<i>, t=<text>,
j=<n1>,<n2>, mod=<i>, pin=<i>, name=<i>, att=<i>, open, clos
```

where:

Parameter	Description
<b>on</b>	Initiate a new panel.
<b>off</b>	Remove a panel from the screen.
<b>p=&lt;i&gt;</b>	To specify the panel index (ID). A value between 1 and 96.
<b>u=&lt;i&gt;</b>	Is the unit in pixel of the menu grid ( default is 16).
<b>r=&lt;i&gt;</b>	Number of rows. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
<b>c=&lt;i&gt;</b>	Number of columns. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
<b>vr=&lt;i&gt;</b>	Number of visible rows. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
<b>vc=&lt;i&gt;</b>	Number of visible columns. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
<b>t=&lt;text&gt;</b>	Panel title.
<b>j=&lt;n1&gt;,&lt;n2&gt;</b>	<b>Floating Dialog Bar:</b> Position (x, y) on the screen in normalized units between 0 and 1 or in pixel unit. If j is not specified the default values are the center of the application frame.
<b>mod=&lt;i&gt;</b>	Is 0 if the panel widget is child of the non graphic widget hierarchy; this means that the panel widget is still present also after a DIALOG OFF command. It is 1 if the panel widget is child of the graphic widget hierarchy; this means that the panel widget is iconized when a DIALOG OFF command is executed and de-iconized as soon the graphic area is restored via a DIALOG ON command.
<b>pin=&lt;i&gt;</b>	To specify the type of panel, can be of one of the following types: <ul style="list-style-type: none"> <li>• dockable/floating dialogbar</li> <li>• floating (only) dialogbar</li> <li>• modal dialogbar</li> <li>• sizable/no sizable dialogbar</li> </ul> Permitted values are:-7,-8,-9,-10,-11,-17,-18,-19,-20,-21,-22.
<b>name=&lt;i&gt;</b>	If equal to 1 then the resource name is, <i>panel_&lt;i&gt;</i> , where "i" is the panel index, otherwise, equal to 0, the resource name is <i>panel</i> .
<b>att=&lt;i&gt;</b>	If set to -1 panel PIN option setting will define whether or not a panel is floating/dockable and to which extremity of the container windows it should be

	docked. Else if equal to -999 and the panel is a dialog bar, the bar will be created as floating from the start, but then dockable, you have to specify ATT=-999 and J=x,y (the starting position).
<b>open</b>	To SHOW an hidden panel.
<b>clos</b>	To HIDE the panel without destroying it. The panel is held in memory for successive use. (This corresponds to an unmanaged widget call). When using this primer no other primers are available.

Sample Code :	<p>Right Docked Not Resizable Panel →</p> <pre>panel p=3,on,u=1,r=400,c=400,vr=400,t='Panel',pin=-9,att=-1</pre> <p>Left Docked Resizable Panel →</p> <pre>panel p=4,on,u=1,r=400,c=400,vr=400,t='Panel',pin=-17,att=-1</pre> <p>Floating Not Resizable Panel →</p> <pre>panel p=5,on,u=1,r=400,c=400,vr=400,t='Panel',pin=-8,att=-999,j=200,200</pre> <p>Floating Resizable Panel →</p> <pre>panel p=6,on,u=1,r=400,c=400,vr=400,t='Panel',pin=-19,att=-999,j=200,200</pre> <p>Only Floating Not Resizable Panel →</p> <pre>panel p=7,on,u=1,r=400,c=400,vr=400,t='Panel',pin=-11,att=-999,j=200,200</pre> <p>Only Floating Not Resizable Panel →</p> <pre>panel p=8,on,u=1,r=800,c=400,vr=400,t='Panel',pin=-21,att=-999,j=200,200</pre> <p>Modal Panel →</p> <pre>panel p=9,on,u=1,r=800,c=400,vr=400,t='Modal Panel',pin=-22,att=-1,j=200,200</pre>
---------------	---

## 8.10 – FixedSize Bar

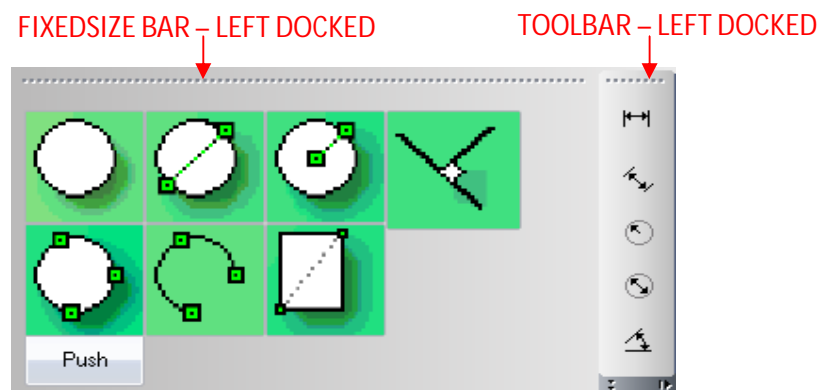
Sometimes there is a requirement to have panels which have the internal organization of a Dialog Bar, but use a container with the style and behavior of a toolbar.

The Eagle v.14 fixed-size control bar inherits some features from the toolbar:

- displaying its image when the control bar is dragged;
- a gripper instead a caption;
- docking and floating states.

This fixed-size window can be of any size and can be docked to any side of the frame window, but this control bar is not resizable by definition, so it cannot be minimized/maximized.

The use of this type of control bar is recommended when the application needs to have a docked toolbar more inward than the dialog (see the picture below). In this situation we are unable to use a standard dialog bar but instead we need to use a “FixedSize Bar”.



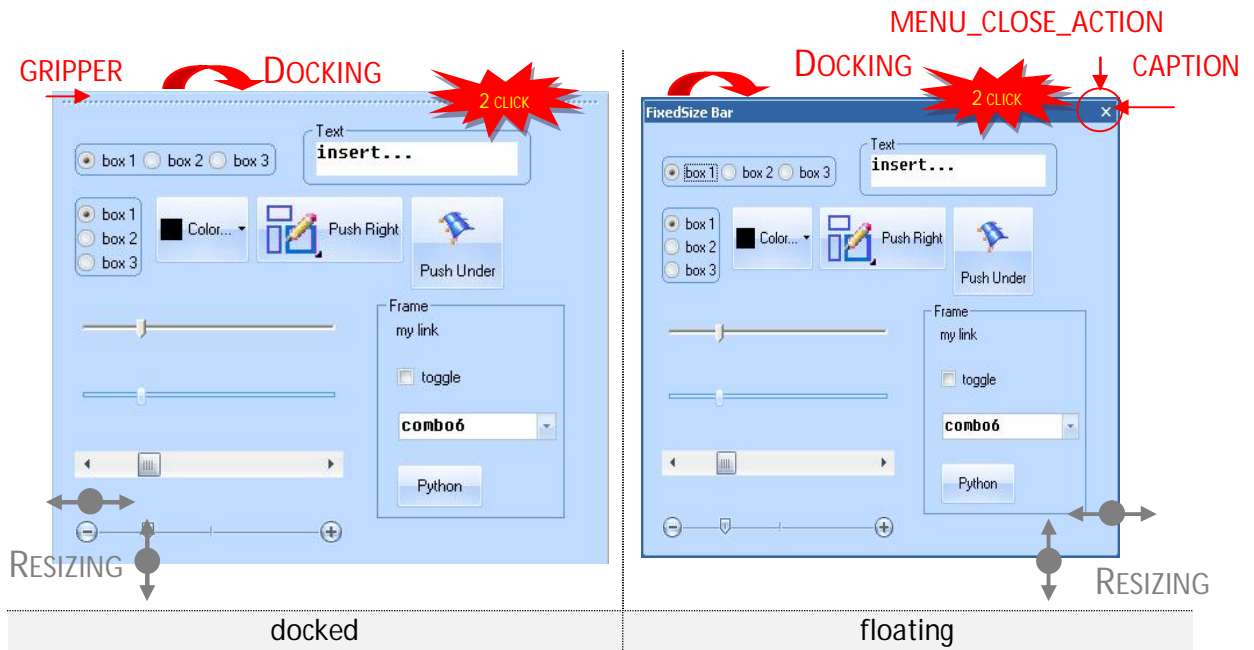
Unfortunately the dimension is fixed, so the developer has to organize both the layout of buttons and the correct size for the panel. Often a “FixedSize Bar” is intended to be docked vertically or horizontally, so if it’s docked to another side, it could result in an ill conceived appearance or be positioned over a large part of a GWINDOW. To prevent these effects, Eagle v.14 has introduced two solutions:

1. double click with the left mouse of the button to immediately move the panel to the floating state;
2. the “dock” attribute can be used inside the “panel” command to block docking in a particular orientation;
  - dock = 0 : no block → panel dockable everywhere;
  - dock = 1 : horizontal block → panel dockable vertically;
  - dock = 2 : vertical block → panel dockable horizontally;

When the panel is floating, it has the same look and behavior as a “Dialog Bar Not Sizable”. You can handle the “close button” event using the same INI entry (MENU\_CLOSE\_ACTION).

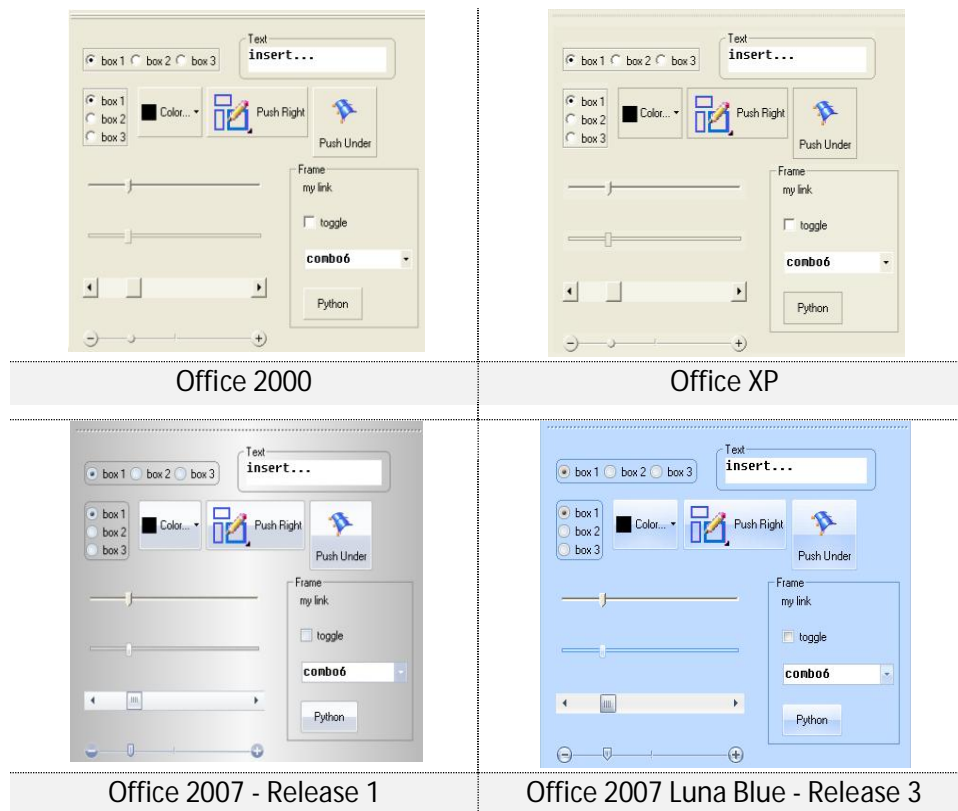
INI Sample :

```
MENU_CLOSE_ACTION = C:\MyFile\OnPanelClose.cmd
```



In the same way as other docked control bars, a “FixedSize Bar” can be started in a floating position using the parameter `att = -999`.

This type of panel also inherits its look from the current selected theme, the next table shows some illustrations:



The command “panel” can be reduced into:

Prototype :

```
panel {on|off}, p=<i>, u=<i>, r=<i>, c=<i>, vr=<i>, vc=<i>, t=<text>,
j=<n1>,<n2>, mod=<i>, pin=<i>, name=<i>, dock=<i>, att=<i>,
open, clos
```

where:

Parameter	Description
<b>on</b>	Initiate a new panel.
<b>off</b>	Remove a panel from the screen.
<b>p=&lt;i&gt;</b>	To specify the panel index (ID). A value between 1 and 96.
<b>u=&lt;i&gt;</b>	Is the unit in pixel of the menu grid (default is 16).
<b>r=&lt;i&gt;</b>	Number of rows. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
<b>c=&lt;i&gt;</b>	Number of columns. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
<b>vr=&lt;i&gt;</b>	Number of visible rows. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
<b>vc=&lt;i&gt;</b>	Number of visible columns. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
<b>t=&lt;text&gt;</b>	Panel title.
<b>j=&lt;n1&gt;,&lt;n2&gt;</b>	<b>Floating FixedSize Bar:</b> Position (x, y) on the screen in normalized units between 0 and 1 or in pixel unit. If j is not specified the default values are the center of the application frame.
<b>mod=&lt;i&gt;</b>	Is 0 if the panel widget is child of the non graphic widget hierarchy; this means that the panel widget is still present also after a DIALOG OFF command. It is 1 if the panel widget is child of the graphic widget hierarchy; this means that the panel widget is iconized when a DIALOG OFF command is executed and de-iconized as soon the graphic area is restored via a DIALOG ON command.
<b>pin=&lt;i&gt;</b>	To specify the type of panel, can be only a dockable/floating fixedsize bar. Permitted values are:-37,-38,-39,-40.
<b>name=&lt;i&gt;</b>	If equal to 1 then the resource name is, <i>panel_&lt;i&gt;</i> , where "i" is the panel index, otherwise, equal to 0, the resource name is <i>panel</i> .
<b>att=&lt;i&gt;</b>	If set to -1 panel PIN option setting will define whether or not a panel is floating/dockable and to which extremity of the container windows it should be docked. Else if equal to -999 and the panel is a bar, the bar will be created as floating from the start, but then dockable, you have to specify ATT=-999 and J=x,y (the starting position).



<code>dock = &lt;i&gt;</code>	<p>This parameter, if specified, allow to block the vertical or the horizontal docking when the docking mechanism is the same of in FixedSize Bar:</p> <p>0 = no block for docking, docking everywhere</p> <p>1 = block horizontal docking, only vertical docking permitted</p> <p>2 = block vertical docking, only horizontal docking permitted</p>
<code>open</code>	To SHOW a hidden panel.
<code>clos</code>	To HIDE the panel without destroying it. The panel is held in memory for successive use. (This corresponds to an unmanaged widget call). When using this primer no other primers are available.

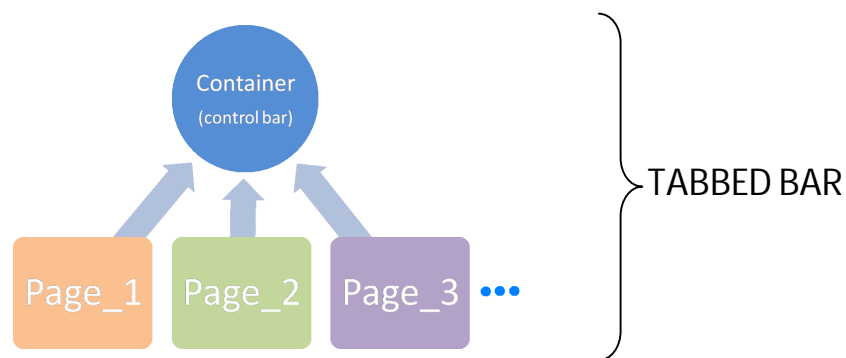
Sample Code :	<p>Right Docked FixedSize Panel →</p> <pre>panel p=3,on,u=1,r=400,c=400,vr=400,t='Panel',pin=-39,att=-1</pre> <p>Floating FixedSize Panel →</p> <pre>panel p=5,on,u=1,r=400,c=400,vr=400,t='Panel',pin=-38,att=-999,j=200,200</pre>
---------------	---

## 8.11 – Tabbed Bar

In the previous Eagle versions it was possible to define only one TabContainer which was defined as panel 96. In this panel it was possible to embed other panels as tabbed windows. There were other limitations, such as the tab title was text only and it was not possible to have buttons in the container common to all tabs.

The Eagle v14 release eliminates in one stroke all these limitations by introducing a new paradigm for the Tabbed Bar. Now there are two separated syntax constructs for the “panel” command that allow creation of a Tab-Container and Tab-Pages for a container. In this way, developers can create more than one tabbed container using the preferred style and the specific use of panel number 96 is no longer necessarily.

The container is a standard “Dialog Bar” that will be created using one of the available pin settings (-7, -8, -9, -10, -11, -17, -18, -19, -20, -21, -22). In fact a control bar can be a TabContainer, which means its internal layout is organized with TabPages, each holding a set of controls. In order to specify that a control bar is a container the panel command must define the “container” primer as “tab” i.e. container = tab.



For “Tab-Container” the “panel” instruction has the following prototype :

Prototype :

```
panel {on|off}, p=<i>, u=<i>, r=<i>, c=<i>, vr=<i>, vc=<i>, t=<text>,
j=<n1>,<n2>, mod=<i>, pin=<i>, name=<i>, att=<i>, open, clos,
container=tab
```

where:

Parameter	Description
on	Initiate a new panel.
off	Remove a panel from the screen.
p=<i>	To specify the panel index (ID). A value between 1 and 96.
u=<i>	Is the unit in pixel of the menu grid ( default is 16).
r=<i>	Number of rows. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
c=<i>	Number of columns. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).

<b>vr=&lt;i&gt;</b>	Number of visible rows. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
<b>vc=&lt;i&gt;</b>	Number of visible columns. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
<b>t=&lt;text&gt;</b>	Panel title.
<b>j=&lt;n1&gt;,&lt;n2&gt;</b>	<b>Floating Control Bar :</b> Position (x, y) on the screen in normalized units between 0 and 1 or in pixel unit. If j is not specified the default values are the center of the application frame.
<b>mod=&lt;i&gt;</b>	Specifies the type of container; the value between 1 and 16.
<b>pin=&lt;i&gt;</b>	To specify the type of panel, can be of one of the following types: <ul style="list-style-type: none"> <li>• floating dialog</li> <li>• fixed dialog</li> <li>• dockable/floating dialog</li> <li>• floating (only) dialog</li> </ul> Permitted values are: 0,1,-7,-8,-9,-10,-11,-17,-18,-19,-20,-21,-22, and the differences between each possibilities will be presented in the next sections.
<b>name=&lt;i&gt;</b>	If equal to 1 then the resource name is, <i>panel_&lt;i&gt;</i> , where "i" is the panel index, otherwise, equal to 0, the resource name is <i>panel</i> .
<b>att=&lt;i&gt;</b>	If set to -1 panel PIN option setting will define whether or not a panel is floating/dockable and to which extremity of the container windows it should be docked. Else if equal to -999 and the panel is a toolbar, the bar will be created as floating from the start, but then dockable, you have to specify ATT=-999 and J=x,y (the starting position).
<b>open</b>	To SHOW an hidden panel.
<b>clos</b>	To HIDE the panel without destroying it. The panel is held in memory for successive use. (This corresponds to an unmanaged widget call). When using this primer no other primers are available.
<b>container=tab</b>	To set a specific panel as "Tab Container", according with PIN value.

Sample Code :

```
panel on,p=10,u=1,r=400,c=400,t='Tb',pin=-19,att=1 ,mod=15,j=0,0, container =tab
```

Naturally, like a common "Dialog Bar", the "Tabbed Bar" has a facade corresponding to dialog panels.

Eagle v.14 also allows you to render a variety of styles for container. The design template you want to use can be achieved using the “mod” parameter which lets you choose the style and orientation for tabs. When the primer is not specified, the default value is “0”.

If MOD=2,6,10 or 14, then it is required to define the color to use for each tab. To specify the two colors that define the gradient to apply to the pages, the “panel” command has been extended and the syntax for Tab Pages also includes two primers to define two RGB colors:

Sample Code :

[Tab Page with colors →](#)

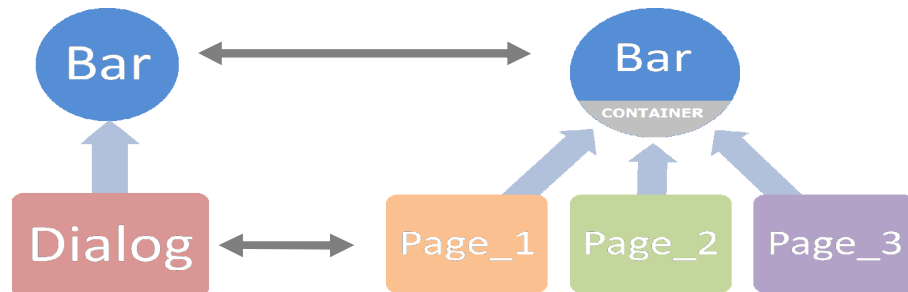
`panel p=3,on,..., pageof=10, col1=(100,100,0), col2=(100,0,0)`

Next table reports all available combinations:

Value	Position	Style	Value	Position	Style
0 (default)	Top		8	Bottom	
1	Top		9	Bottom	
2	Top		10	Bottom	
3	Top		11	Bottom	
4	Left		12	Right	
5	Left		13	Right	
6	Left		14	Right	
7	Left		15	Right	

Each tab in the container is linked to a “Tab Page”, where a page is a particular kind of panel, simply a bar this doesn’t need further information about pin, attach type, position, etc.

“Tab Container” is an extension of a “Dialog Bar” that stores one or more dialogs. These dialogs are usually called “pages” and they are still organized as an embedded panel, as before, but the graphical presentation and navigation of the Tabbed panel is different. The relationship between “Dialog Bar” and “Tabbed Bar” is symbolized in the diagram below (a simple bar with a single dialog vs. a bar tabbed with one or more pages):



Because Eagle v.14 has extended the number of “Tabbed Bars”, now it’s necessary to maintain the information about the container in which a page is contained. For that reason the “panel” command has been modified to create a Tab page with the information of corresponding Tab Container and without redundant parameters (pin, att, j, mod, etc).

So, in case of “Tab-Pages”, the command “panel” has a different prototype:

Prototype :

```

panel {on|off}, p=<i>, u=<i>, r=<i>, c=<i>, vr=<i>, vc=<i>, t=<text>,
pageof=<i>, open, tabbed

```

where:

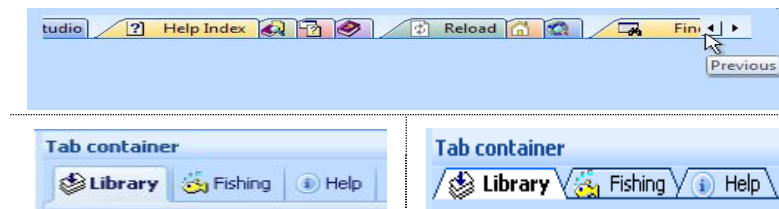
Parameter	Description
on	Initiate a new panel.
off	Remove a panel from the screen.
p=<i>	To specify the panel index (ID). A value between 1 and 96.
u=<i>	Is the unit in pixel of the menu grid ( default is 16).
r=<i>	Number of rows. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
c=<i>	Number of columns. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
vr=<i>	Number of visible rows. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
vc=<i>	Number of visible columns. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
t=<text>	Panel title and separated with a “#” the path name of an icon to insert in the tab; the complete syntax is : t=’title#path/icon.bmp’
pageof=<i>	Represents the id of the Tab Container in which the Tab Page will be added.

open	To SHOW a hidden panel.
tabbed	To set a specific panel as "Tab Page".
col1	Used when the Tab style is "OneNote"(mod=2,6,10,14),, to set the first color for the background of the Tab Page (more details below). In the form =(R, G, B), where R, G and B are three integer between 0 and 255.
col2	Used when the Tab style is "OneNote" (mod=2,6,10,14), to set the second color for the background of the Tab Page (more details below) . In the form =(R, G, B), where R, G and B are three integer between 0 and 255.

To create a page, the developer has to initialize the "pageof" parameter, which contains the related "Tab Container", and also specify the "tabbed" primer to identify the panel as a page.

As we have seen before, the last two parameters are applied only if the selected style for the container is like "OneNote" (mod = 2,6,10,14). They are two colors to create the back ground gradient from first to the second color for the "Tab Page". If the two colors are equal, or only the first is defined, the pages have a single color without gradient. These parameters are defined with the three components in the form col=(R,G,B) where 'R','G' and 'B' are integers in the range 0 – 255.

In addition, Eagle v.14 allows you to insert a bitmap near the page title. Obviously, the icon should have an appropriate size for the tab. The icon path name can be specified inside the text parameter after the special character '#', for instance: ..., t = 'title text#tabicon.bmp',...



Sample Code :

```
panel p=11,u=1,r=100,c=100,t='Tab-A#iconA.bmp',tabbed,pageof=10
panel p=12,u=1,r=100,c=100,t='Tab-B#
iconB.bmp',tabbed,pageof=10,col1=(10,10,99)
panel p=13,u=1,r=100,c=100,t='Tab-C',tabbed,pageof=10,
col1=(10,10,99),col2=(0,0,10)
panel p=14,u=1,r=100,c=100,t='Tab-D#',tabbed,pageof=10
panel p=15,u=1,r=100,c=100,t='Tab-E# iconE.bmp',tabbed,pageof=10
```

Eagle v.14 uses two entries in the configuration file to customize the tab presentation, these entries used for the TDI mode also:

- "TAB\_BOLD\_SELECTION" : this entry enables a "BOLD" style for the characters of the selected tab-page. If the parameter is not specified, the default value is "no";

INI Sample :

TAB\_BOLD\_SELECTION = no | yes

- "TAB\_ENABLE\_DRAGGING" : this entry enables the dragging of the tab pages. If the entry is to "YES", it's possible to drag a tab-page to after/before another tab-page. If set to yes you can drag a tab header and change its position:



If the parameter is not specified, the default value is "no";

INI Sample :

TAB\_ENABLE\_DRAGGING = no | yes

To complete this discussion about the Tab Container, the following example is explained:

Sample Code :

```
Container →
panel on, p=10, u=1, r=400, c=400, t='Tab container', pin=-19, att=-1, mod=2,
j=0,0, container=tab

Page A →
panel p=41, u=1, r=100, c=100, t='Tab-A#iconA.bmp', tabbed, pageof=10,
col1=(50,50,100), col2=(100,100,150)
option p=41, f=tabA.tab

Page B →
panel p=42, u=1, r=100, c=100, t='Tab-B#iconB.bmp', tabbed, pageof=10,
col1=(150,100,0), col2=(200,50,0)
option p=42, f=tabB.tab

Page C →
panel p=43, u=1, r=100, c=100, t='Tab-C#iconC.bmp', tabbed, pageof=10,
col1=(150,100,100), col2=(150,150,100)
```

option p=43, f=tabC.tab

Page D →

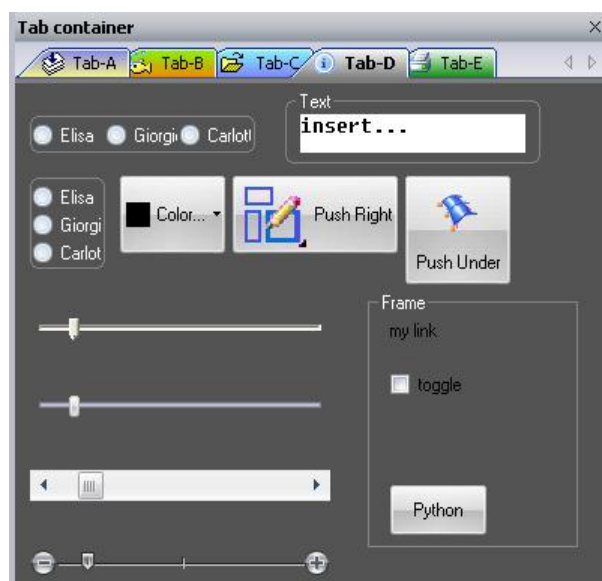
panel p=44, u=1, r=100, c=100, t='Tab-D#iconD.bmp', tabbed, pageof=10,  
col1=(200,200,200), col2=(250,250,250)

option p=44, f=tabD.tab

Page E →

panel p=45, u=1, r=100, c=100, t='Tab-E#iconE.bmp', tabbed, pageof=10,  
col1=(100,100,100), col2=(0,255,0)

GUI Result :



Examining the code above, you can understand that all pages have a link to an option file (file that contains the description of controls inside the bar, more details in next section), and nothing for the container. To have more details about the construction of containers, see the “Dialog Bar” section.

We have discussed at the beginning of the “Control Bar” chapter the freeze/unfreeze commands to freeze/unfreeze a control, a page or all the pages in a “Tab Container”.

There is another command to enable control over pages, “click” allows automatic changing of the currently selected page using an Eagle instruction.

Prototype :

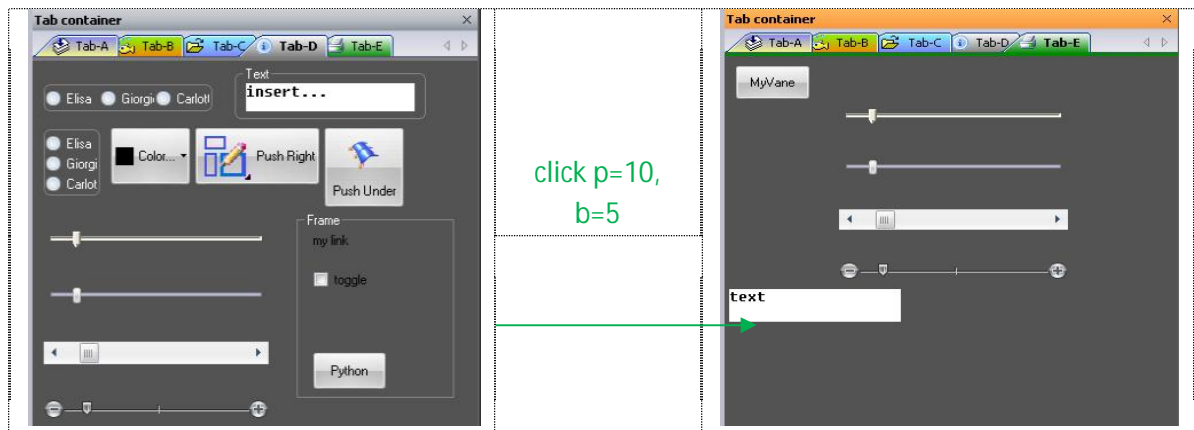
click p=<i>, b=<i>



where:

Parameter	Description
$p=<i>$	The tabbed bar identifier the range 1 to 96.
$b=<i>$	The index of pages, in the range 1 to N, where N is the number in the pages.

For instance :



## 8.12 – User Interface Persistence

Application interfaces in today's graphic driven application models are normally easy to use and flexible. This flexibility brings problems in ensuring that the user interface when changed can return to a default setting, or, when modified to reflect a user preference will retain these changes every time the application is restarted.

This topic is designed to show how to make an application GUI persistent, meaning the ability to reproduce the same GUI layout at startup time that was present at the last shutdown of the application. This means elements such as toolbars and dialogs and their appearance status, whether undocked or docked, their position, floating, whatever will be replicated exactly as required.

This task can be achieved by using the INFOGUI function when the application is closed to retrieve the order in which toolbars and dialogs are docked or floating and the INFOPANEL function to retrieve the information about the PIN-value, ATT-value, position, if open or closed or even "off-status" for each menu. The information gathered is then stored in a file for maintaining persistency. Note that this command can be initiated at any time and not only during start-up or shut-down processes.

Sample Code :

```

numeric v1\128
zero v1\100
v1=infogui(0)
numeric perpan[128],peractive[128],perpin[128],peratt[128]
point perpos[128]
numeric x,y,z
y=1
z=1
while vz <> 0
{
    v100=infopanel(vz)
    peractive[y]=v100
    perpin[y]=v101
    peratt[y]=v102
    perpos[y]=(v103,v104,0)
    perpan[y]=vz
    y=y+1
    z=z+1
}
output persistgui.dat
for z=1,y-1
    write perpan[z],peractive[z],perpin[z],peratt[z],perpos[z]
next z
output
  
```

Output : 1 1 -5 -1 2 42 0

```

3 1 -5 -1 -1 -1 0
30 1 -2 -1 2 0 0
88 1 -7 -1 37 113 0
12 1 -8 -1 3 580 0
31 1 -4 -1 961 0 0
  
```

At the next application startup, if a "persistent representation" of the GUI is present:

Sample Code :

```
iffile persistentgui.dat ...
```

instead of mounting the default application GUI, the file persistentgui.dat is read and toolbars, dialogs, etc. are created with their position and in their last saved appearance. For instance:

Sample Code :

```

numeric x,y,z,v1\10
numeric perpan[128],peractive[128],perpin[128],peratt[128]
point perpos[128]
input persistgui.dat
x=1
label read
  read perpan[x],peractive[x],perpin[x],peratt[x],perpos[x]
  x=x+1
  eof go finish
  goto read
label finish
  input
  do restoregui
  
```

where restoregui.cmd will create panel 1 using the values at position x where perpan[x]=1, and so on.

Sample Code :

```

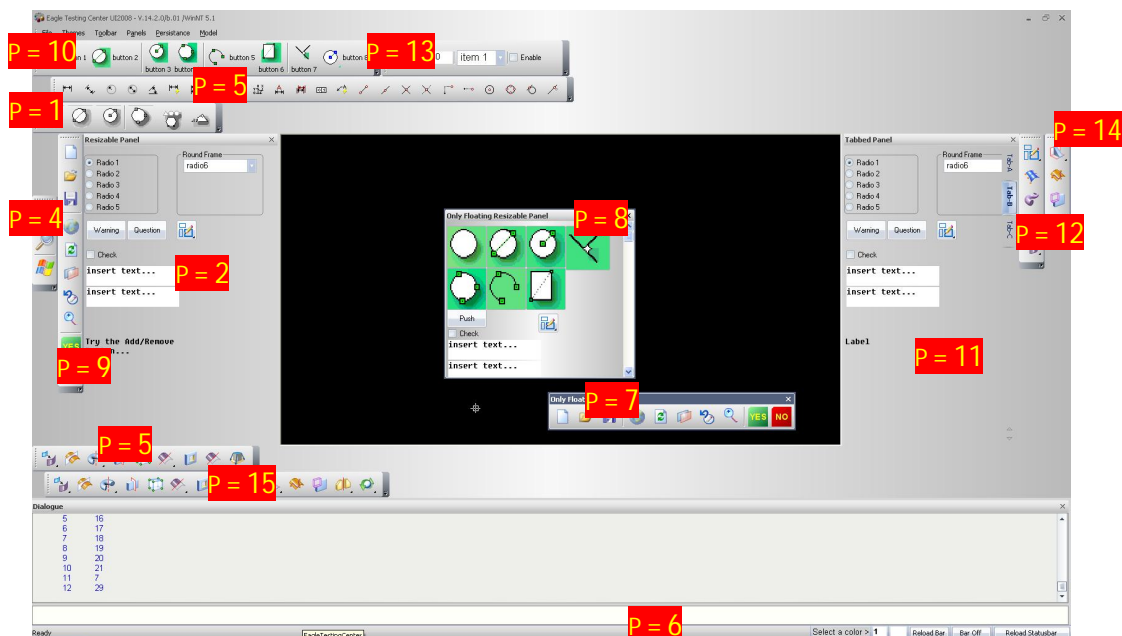
for z=1,6
  if perpan[z]=1 then
  {
    if peractive[z]<>0 then
    {
      panel p=1,on,u=1,r=21,c=28,pin=perpin[z],att=peratt[z]
      option p=1,f=menu1.tab
    }
  }
  if perpan[z]=3 then
  {
    ...
  }
  
```

next z

Of course, there are different and possibly better programming methods that could be utilized as alternatives to the one illustrated above, but the purpose of this developer snippet is to be clear rather than clever.

In Eagle v.14 the “infogui” and “infopanel” functions are callable at any time and also have been extended. Infogui, called with the parameter 0, returns the IDs of control bars (tool bars, dialog bars, fixed-sized bars and tabbed bars) which are active at a particular time in the Main Window. The returned list is ordered following this predefined strategy:

1. *Tool Bar on Top*
2. *Tool Bar on Left*
3. *Tool Bar on Bottom*
4. *Tool Bar on Right*
5. *Tool Bar Floating*
6. *Dialog Bar, FixedSize Bar and Tabbed Bar on Top*
7. *Dialog Bar, FixedSize Bar and Tabbed Bar on Left*
8. *Dialog Bar, FixedSize Bar and Tabbed Bar on Bottom*
9. *Dialog Bar, FixedSize Bar and Tabbed Bar on Right*
10. *Dialog Bar, FixedSize Bar and Tabbed Bar Floating*



Code Sample :	<code>numeric v1\100</code> <code>v1 = infogui(0)</code>
Output :	<code>10, 13, 5, 1, 2, 9, 15, 5, 14, 12, 7, 4, 11, 8</code>

Now if the parameter is -1 instead of 0 and the returned value is equal to 1, it means that the Menu Bar is present.

Additionally, Version 14 Eagle extends the “infogui” function to obtain information about the panels attached to a specific gwindow. Passing the as parameter the identifier of a specific gwindow to the “infogui” function, the command returns a list of panels attached to the child document at the moment. The list are ordered in the same way as the Main Window, so it contains toolbars (listed from the top to floating) sequenced before control bars (listed from the top to floating).

Prototype :	<code>infogui (param=&lt;i&gt;)</code>
-------------	--

where:

Parameter	Description
<code>param=&lt;i&gt;</code>	If param = 0 → returns the list of active panels
	If param = -1 → returns 1 if the Menu Bar is installed
	If param = id → returns the list of active panels associated with the gwindow with identifier “id”

The other function used to extract information from the GUI construction, is “infopanel”. This command retrieves the current attributes about a particular panel. The prototype requires the panel id to be passed to the function as the parameter in order to return data which corresponds to the following structure:

Variable Index	Description
1	0 if the panel does not exists 1 if the panel exists and is open 2 if the panel exists and is closed
2	the PIN= value
3	the actual ATT= value
4	the current x position in pixels
5	the current y position in pixels
6	1 if the panel or toolbar is locked or 0 if otherwise defined
7	the actual panel width
8	the actual panel height

Prototype :	<code>infopanel (p=&lt;i&gt;)</code>
-------------	--------------------------------------

where:

Parameter	Description
<code>p=&lt;i&gt;</code>	Panel Identifier

For example :

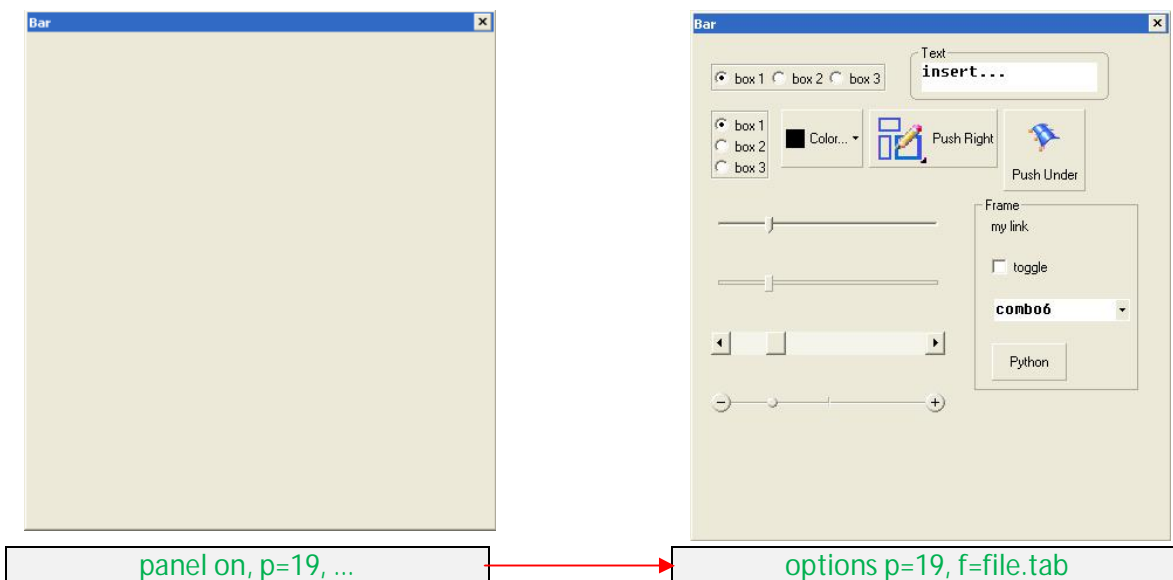


Code Sample :	<code>numeric v1\6</code> <code>v1 = infopanel(7)</code>
Output :	<code>1, -5, -1, 25, 0, 0</code>

### 8.13 – Options in a panel

In this section we'll explain how to insert controls inside a panel after its creation. The “options” command is used to populate the panel with different types of buttons available.

So, once you have created a panel, the button definitions are taken from a TAB file that contains all the information about the required objects to use, their size, position and color, if appropriate. The examples below illustrate the method of populating a panel:



Sample Code :

```
panel.cmd panel on, p=19, u=1, c=200, r=150, pin=-8, att=-1, t='Panel'
options p=19, f=options.tab
```

```
options.tab 1,10,15,150,45,0,0,39,'radios': tell '-----
2,180,5,170,47,0,0,120,'Text#R':;
3,190,20,150,25,0,0,6,'insert...': vane
4,10,55,50,120,0,0,29,'radios': tell '-----
5,70,60,70,50,0,0,500,'Color...': tell 'st'
6,145,60,110,50,0,0,4,'se1.bmp#Push Right':
7,260,60,70,70,0,0,4,'se2.bmp##Push Under':
8,10,150,200,20,0,0,10,'array_name': tell 'scroll'
9,10,200,200,20,0,0,210,'array_name': tell 'scroll'
10,10,250,200,20,0,0,410,'array_name': tell 'scroll'
11,10,300,200,20,0,0,610,'array_name': tell 'scroll'
12,240,150,50,20,0,0,501,'my link': tell '***'
13,250,170,100,50,0,0,8,'toggle':;
14,250,220,120,200,0,0,27,'radio2': tell '----- COMBO ---'
15,250,260,64,32,0,0,1,'Python':
16,235,135,140,165,0,0,120,'Frame#S':;
```

The “options” command can be applied to every type of panel except the “Tab Container” that doesn’t have controls inside it, the syntax of the instruction is :

Prototype :

```
options {on|off},p=<i>, f=<file>
```

where:

Parameter	Description
<b>on</b>	(default) Inserts the buttons in the panel.
<b>off</b>	Unload a panel without removing it from the screen.
<b>p=&lt;i&gt;</b>	Specify the panel index to fill up. The value must be in the range 1 to 96.
<b>f=&lt;file&gt;</b>	Specifies the menu file (extension TAB).

The format, very similar to the MEN file format in previous chapter, is explained below:

```
opt,x,y,w,h,f,b,t,<text>:<commands>
```

where :

<b>opt</b>	option number. If 0 then the option is skipped.
<b>x</b>	x co-ordinate of the button in the menu grid.
<b>y</b>	y co-ordinate of the button in the menu grid.
<b>w</b>	the width of the button in grid units. Width depends from the pixel unit (u parameter) specified in the “panel” command.
<b>h</b>	the height of the button in grid units. Height depends from the pixel unit (u parameter) specified in the “panel” command.
<b>f</b>	foreground color for buttons
<b>b</b>	background color for buttons

<b>t</b>	button type, more details in the section of "Controls"
<b>&lt;text&gt;</b>	This field generally contains the text displayed, but can also stores a name of a variable, an array, the path name for a file (i.e. bitmap) or specials character to select a specific layout
<b>&lt;command&gt;</b>	the actions to take when selected

The differences between each control will be explained in next section, but now it's interesting to introduce two features available for every buttons: one to fit the font and another to set a tooltip.

You are able to set the font for a control by adding an option with type 888 before the related button, for instance:

Sample Code :

```
0, 0, 0, 16, 0, 1, 0, 888, 'MS Sans Serif' : ;
5, 4, 0, 64, 32, 0, 0, 5, 'Name' : ;
```

Name	Name
0, 0, 0, 16, 0, 1, 0, 888, 'MS Sans Serif' : ;	0, 0, 0, 12, 0, 0, 0, 888, 'Arial' : ;
+ + +	
unused -+	
+--- 1=italic	
+--- 1=bold	

Using the syntax on the left, the font "MS San Serif", bold and dimension 16 will be applied to next button type "5". The one on the right defines font "Arial", dimension 12 and no other effects.

If no font has been set, the control receives the default for the application. Users can change the default font using the "winfont" command.

Prototype :

**winfont**

or

**winfont** name=<font\_name>, height=<character\_height>, bold, italic, underline

where:

<b>name</b>	The name of the font family between quotes.
<b>height</b>	The desired character height.
<b>bold</b>	Select the "bold" version of the font, if one exists.
<b>italic</b>	Select the "italic" version of the font, if one exists.



**underline** Select the "underline" version of the font, if one exists.

If no parameters are specified, the "Window Font" dialog, which enables interactive font selection, will automatically be displayed. Using this dialog the default font is applied directly instead of specifying attributes.



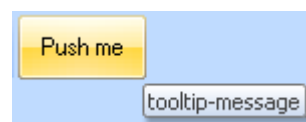
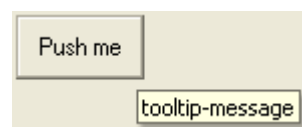
Sample Code :

```
winfont
winfont name='Arial', height=13
winfont name='Fixedsys',h=10,bold,italic
```

In the same way, tooltip messages are also available using type 999; a tooltip can be defined with a backward compatibility in the TAB file, that means that previous releases will run the TAB file without modifications. Before the option line which requires the tooltip insert a "zero-option" line like :

Sample Code :

```
0, 0, 0, 0, 0, 0, 0, 0, 999, 'tooltip-message' ;;
7, 32, 0, 64, 32, 0, 0, 1, 'Push me': ....
```

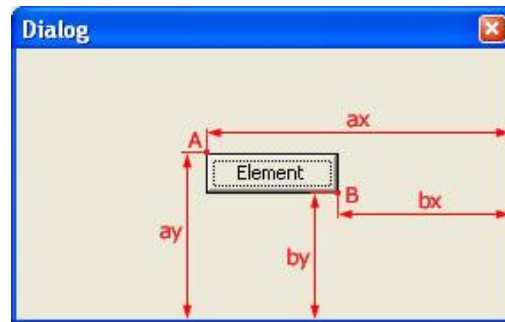


So, the message "tooltip-message" will be shown when the mouse pointer is over the button number 7.

## 8.14 – Anchoring and auto-resizable controls

Eagle V14 has extended the “option” command to include anchoring for resizable panels and consequently the auto-resizing of controls. Anchoring is a mechanism of implementing a window with controls which can dynamically change their sizes when the parent window changes its size.

The new Eagle version provides a complete, simple, and elegant solution, to this problem which is based on the use of “anchors”.

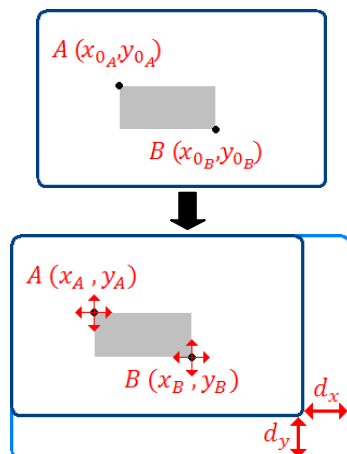


The anchor is a point on a control, which is used to bind the control's position relative to the parent window's edges. For example, the control on figure above has two anchors. If you anchor a control to a specific side or sides and then resize the parent window, the control's distance from the anchored side or sides will not change or will change by certain proportions.

Each control inside a panel can be described with two points: the top-left corner (point A) and the bottom-right corner (point B). Considering only these two points we are able to control how the size and/or position of the specific button changes in relation to the resizing of the panel that contains the button.

If we consider the point A and B in terms of coordinates x and y, the concept of anchoring can be thought in terms of how the coordinates change (in percentage from 0 to 100) considering the size of the panel. So, we need to define four percentage values for every button inside the panel that we want resized and/or moved when the panel is resized.

The algorithm that controls the size and position of a button can be defined as follows:



$$x_A = x_{0A} + (d_x * \Delta_{xA})$$

$$y_A = y_{0A} + (d_y * \Delta_{yA})$$

$$x_B = x_{0B} + (d_x * \Delta_{xB})$$

$$y_B = y_{0B} + (d_y * \Delta_{yB})$$

where :

- $x_A$  = x coordinate for point A;
- $y_A$  = y coordinate for point A;
- $x_B$  = x coordinate for point B;
- $y_B$  = y coordinate for point B;
- $x_{0A}$  = original x coordinate for point A;
- $y_{0A}$  = original y coordinate for point A;
- $x_{0B}$  = original x coordinate for point B;
- $y_{0B}$  = original y coordinate for point B;
- $d_x$  = increment or decrement of the panel's width;
- $d_y$  = increment or decrement of the panel's height ;
- $\Delta_{xA}$  = percentage variation for the x coordinate on point A;
- $\Delta_{yA}$  = percentage variation for the y coordinate on point A;
- $\Delta_{xB}$  = percentage variation for the x coordinate on point B;
- $\Delta_{yB}$  = percentage variation for the y coordinate on point B.

The percentage variations (  $\Delta_{xA}$  ,  $\Delta_{yA}$  ,  $\Delta_{xB}$  ,  $\Delta_{yB}$  ) are defined through a specific "tab" file.

Naturally, it is necessary to have a reference size in order to know if the dimension of the panel has been increased or reduced; for this reason the starting size (  $x_{0A}$  ,  $y_{0A}$  ,  $x_{0B}$  ,  $y_{0B}$  ) of the panel is considered as the reference. **IMPORTANT NOTE:** - When a panel starts in a docked state the real size could be greater than the number of rows and columns defined in the panel command.

Using this method it is possible to move and/or resize a button, if all the delta ( $\Delta$ ) values are different from zero the control will be moved alternatively it can be anchored to one or more corners; for instance, if the  $\Delta_{xA}$  and  $\Delta_{yA}$  are equal to 0, the point A remains will always locked in the same position, furthermore if  $\Delta_{xB}$  is equal to 50  $\Delta_{yB}$  = to 100, the point B will be moved for halfway (relative to the panel) along the x coordinate and entirely along the y coordinate.

In order to enable the anchoring technique a new parameter has been added to “option” command for sizable panels; when pin value is equal to -17, -18, -19, -20, -21 or -22; this parameter is called “anchor” and is fed by input from the specific additional “tab” file, so the “option” syntax now becomes:

Prototype :

<b>options</b> {on   off},p=<i>, f=<file>, layout=<file>
--

where:

Parameter	Description
<b>on</b>	(default) Inserts buttons in the panel.
<b>off</b>	Unload a panel without removing it from the screen.
<b>p=&lt;i&gt;</b>	Specify the panel index to populate. The value must be in the range 1 to 96.
<b>f=&lt;file&gt;</b>	Specifies the menu file (extension TAB).
<b>layout =&lt;file&gt;</b>	Specifies the anchoring configuration file (extension TAB).

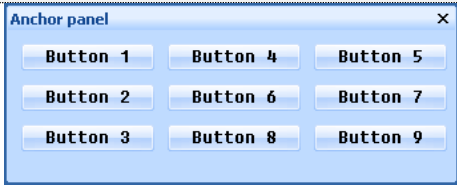
The layout file contains one row for every button to which we want apply configuration, and the syntax is as explained below:

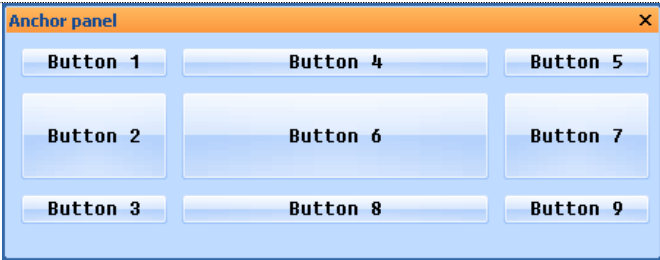
$$id, \Delta_{x_A}, \Delta_{y_A}, \Delta_{x_B}, \Delta_{y_B}$$

Where "id" is the Eagle's identifier for the control and the "delta" ( $\Delta$ ) parameters is an integer from 0 to 100 that define the percentage of variation for point A and point B.

When the anchoring file is not specified, the panel behavior remains the same as previously implemented, that is, without the auto-sizing of the controls. However, if the file is present but some controls do not have their layout defined, then they remain with both the same size and in the same position regardless (IMPORTANT NOTE - Having a panel with some controls having defined layout and some auto-sized could cause overlapping between different controls).

The examples below illustrate how the anchoring system works:

Sample Code :	
<b>panel.cmd</b>	<pre>panel p=3, on, u=1, r=120, c=340, t='Anchor panel', pin=-18, att=-999, j=200,200 option p=3, f=option.tab, lay=anchor.tab</pre>
<b>anchor.tab</b>	<pre>1, 0, 0, 0, 0 2, 0, 0, 0, 100 3, 0, 100, 0, 100 4, 0, 0, 100, 0 5, 100, 0, 100, 0 6, 0, 0, 100, 100 7, 100, 0, 100, 100 8, 0, 100, 100, 100 9, 100, 100, 100, 100</pre>
<b>Original</b>	

Width and Height Increased	
	Button 1 → Same Position and Same Size Button 2 → Same Position and Height Resized Button 3 → Moved along y axis and Same Size Button 4 → Same Position and Width Resized Button 5 → Moved along x axis and Same Size Button 6 → Same Position and Resized in both dimensions Button 7 → Moved along x axis and Height Resized Button 8 → Moved along y axis and Width Resized Button 9 → Moved in both axis and Same Size

#### 8.14.1 How to use the PBUTTON command on auto-resizable controls

When a panel is defined with auto-resizable controls with the LAYOUT= option of the OPTIONS command, as described in the previous paragraph, and one or more buttons are added with the PBUTTON command, it is necessary to define the anchorage of the new added controls. This can be achieved using the LAYOUT= option added to the PBUTTON command. This option requires four integer values that correspond to the corners defined in the "layout file"; the option number is not required.

Sample Code:

```
PBUTTON P=1,B=20,ADD='7, 32, 0, 64, 32, 0, 0, 1,'Push me':',LAYOUT=0,0,100,100
```

#### 8.14.2 An interesting example

If we want to have a list-view or a tree-view filling entirely a resizable panel then it would be nice if also the list-view or the tree-view were resizable.

The solution is to use the *list button*, type equal to 506, in the first case and the *tree-view button*, type equal to 300, in the second case and the option LAYOUT=layout.dat with the OPTION command. In the *layout.dat* file we simply define the four corners as 0, 0, 100, 100 to fill the entire panel area.

Sample Code :

```
panel p=1,on,u=1,c=400,r=400,t='Test List Button',pin=-11,j=100,100
option p=1,f=listbutt.tab, layout=listbutt.tab
```

```
# where the TAB file is
0,0,0,14,0,0,0,888,'Arial':;
```

```
1,0,0,400,200,0,0,506,'listsort1.dat':;
```

```
# and the layout file is  
1,0, 0, 100, 100
```

## 8.15 – Auto-hiding panels

Eagle 14 offers the auto-hide feature for all resizable control bars in the frame window in the same manner as Microsoft Visual Studio .NET. By default, all resizable control bars in the frame window have only one X button that allows closing of the bar; the auto hide feature adds the push-pin button which enables or disables auto-hiding.

This feature can be applied to dialog bars and tabbed bars, namely dockable panels that are created with pin numbers: -7, -8, -9, -10, -17, -18, -19 and -20. When the panel is floating no icon is displayed near the close button.



Standard dockable panel



Dockable panel with Auto-Hiding

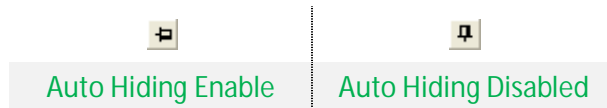
While the auto-hiding is switched on, the panel disappears if it loses current focus and at this point only the title of the panel page appears on the side of the frame where the bar is docked.



Without Focus

With Focus.

Alternatively, when auto-hiding is disabled the panel always remains visible. In order to enable/disable the auto-hiding feature the user has to click on the auto-hide button and the push-pin icon changing orientation from vertical (disabled) to horizontal (enabled) or vice versa.



The “panel” command has been extended to provide the autohide feature. In order to enable auto-hiding we need to append the keyword “autohide” to the panel command instruction. The relevant syntax of the panel command has been changed as follows:

The “panel” command can be condensed into :

Prototype :	<code>panel &lt;...&gt;, autohide</code>
-------------	--

where:

Parameter	Description
<code>&lt;...&gt;</code>	The same parameter combinations as defined in the section 8.9 or 8.11.
<code>autohide</code>	Enable the auto-hiding feature.

For example:

Sample Code :	<code>panel p=35, on, u=1, r=270, c=270, vr=270, t='Icon Panel', pin=-17, att=-1, autohide</code>
---------------	---

### 8.16 – Panels related to a gwindow

Eagle version 14 introduces a new feature which allows attaching of a control bar to a specific document window when running Eagle in multiple document mode. Using the same combination of the “panel” and “option” commands with which we are already familiar, we are able to add a control bar in a specific gwindow by using the “w” parameter. So in this case the prototype of the “panel” command should be changed as follows:

Prototype :	<code>panel {on off}, p=&lt;i&gt;, u=&lt;i&gt;, r=&lt;i&gt;, c=&lt;i&gt;, vr=&lt;i&gt;, vc=&lt;i&gt;, t=&lt;text&gt;, j=&lt;n1&gt;,&lt;n2&gt;, pin=&lt;i&gt;, name=&lt;i&gt;, w=&lt;i&gt;, dock=&lt;i&gt;, open, clos</code>
-------------	--



where:

Parameter	Description
on	Initiate a new panel.
off	Remove a panel from the screen.
p=<i>	To specify the panel index (ID). A value between 1 and 96.
u=<i>	Is the unit in pixel of the menu grid (default is 16).
r=<i>	Number of rows. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
c=<i>	Number of columns. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
vr=<i>	Number of visible rows. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
vc=<i>	Number of visible columns. (Note, The maximum width/height for panels extended to the dimensions of the Eagle frame).
t=<text>	Panel title.
j=<n1>,<n2>	<p><b>Floating Control Bar (Standard or Tab Container):</b>            Position (x, y) on the screen in normalized units between 0 and 1 or in pixel unit. If j is not specified the default values are the center of the application frame.</p> <p><b>Docked Tool Bar:</b>            Docking position (x, y) specified with J=&lt;x_value&gt;,&lt;y_value&gt;, defined in pixels. If j is not specified the default values are (-1, -1) = new row/column</p>
pin=<i>	<p>To specify the type of panel, can be of one of the following types:</p> <ul style="list-style-type: none"> <li>floating dialog</li> <li>fixed dialog</li> <li>dockable/floating dialog</li> <li>dockable/floating toolbar</li> <li>floating (only) toolbar</li> <li>floating (only) dialog</li> </ul> <p>Permitted values: -2, -3, -4, -5, -7, -8, -9, -10, -11, -14, -17, -18, -19, -20, -21, -37, -38, -39, -40</p>
name=<i>	If equal to 1 then the resource name is, <i>panel_&lt;i&gt;</i> , where "i" is the panel index, otherwise, equal to 0, the resource name is <i>panel</i> .
w=<i>	Represents the gwindow identifier to which the installed panel will be associated.
dock = <i>	<p>This parameter, if specified, enables blocking of vertical or horizontal docking when the docking mechanism is the same as the Tool Bar:</p> <p>0 = no block for docking, docking everywhere</p>

	1 = block horizontal docking, only vertical docking permitted 2 = block vertical docking, only horizontal docking permitted
	Usable only in Tool Bar
open	To SHOW a hidden panel.
close	To HIDE the panel without destroying it. The panel is held in memory for successive use. (This corresponds to an unmanaged widget call). When using this primer no other primers are available.

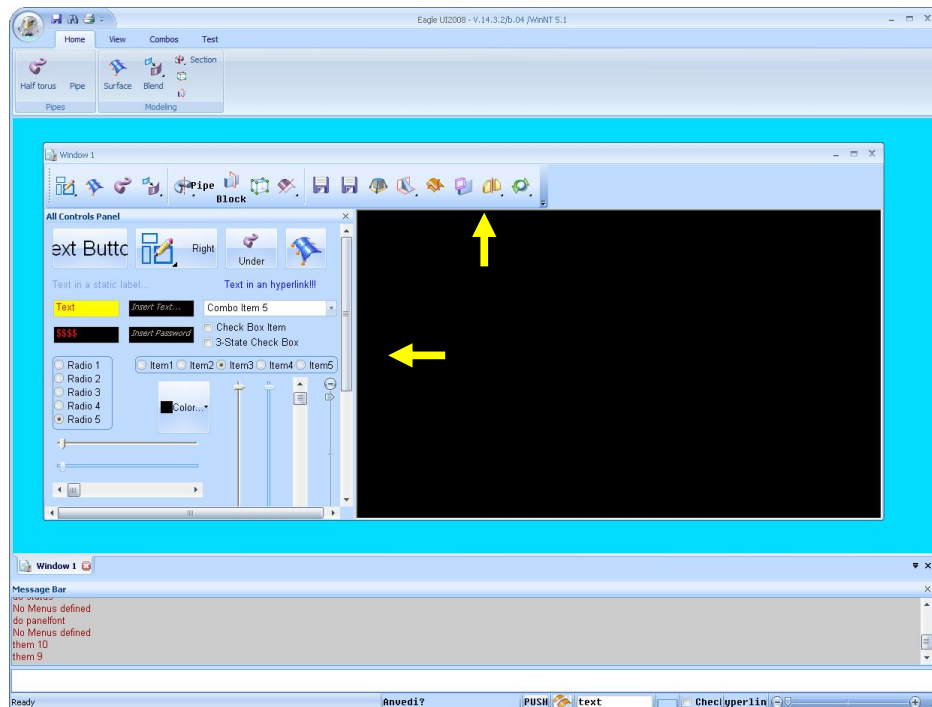
The window parameter (w) contains the identifier of the window in which the panel will be created, however, as shown in the table above, not all the types permitted for the Main Window can be added in a gwindow (i.e. modal panels and status bar) and not all the features can be enabled (for instance auto-hiding and pulldown toolbars).

The types of panels a child document may therefore host are listed as follows:

PIN	Type	Docking Status	Resizable	Dockable
-2	Tool Bar	Left	✓	✓
-3	Tool Bar	Bottom	✓	✓
-4	Tool Bar	Right	✓	✓
-5	Tool Bar	Top	✓	✓
-7	Dialog Bar Not Sizable	Left	✗	✓
-8	Dialog Bar Not Sizable	Bottom	✗	✓
-9	Dialog Bar Not Sizable	Right	✗	✓
-10	Dialog Bar Not Sizable	Top	✗	✓
-11	Floating Only Panel	Float Only	✗	✗
-14	Floating Only Tool Bar	Float Only	✓	✗
-17	Dialog Bar Sizable	Left	✓	✓
-18	Dialog Bar Sizable	Bottom	✓	✓
-19	Dialog Bar Sizable	Right	✓	✓
-20	Dialog Bar Sizable	Top	✓	✓
-21	Floating Only Dialog Bar	Float Only	✓	✗
-37	Fixed Size Bar	Left	✗	✓
-38	Fixed Size Bar	Bottom	✗	✓
-39	Fixed Size Bar	Right	✗	✓
-40	Fixed Size Bar	Top	✗	✓

Sample Code :

```
panel on,p=1,u=32,r=10,c=2,t='child menu', pin=-17, att=3
panel on,p=4,r=1,c=20, pin=-5, w=5, j=0,1
panel off,p=5
panel p=12,close
```



Naturally the commonly available features applicable to controls or control bars, such as freezing, locking, etc. remain valid also in child document panel objects.

Another interesting element related to panels inside a gwindow, is the "infogui" command which is extended to enable the obtaining of information concerning panels attached to a specific gwindow. This function is achieved by passing the gwindow identifier as parameter to the "infogui" function (refer to the section 8.12 for more details).

Note that the "att" parameter is not available when the panel has been created inside a gwindow.

When a panel related to a gwindow is in the floating state, in order to avoid conflict with bars of other windows, the panel will be hidden if the window loses focus. Note that if the panel has the focus, for example, the mouse pointer is over the dialog, and a change of focused gwindow is forced, for instance using "Ctrl+Tab", the panel remains visible until the focus is moved to another graphic object.

### 8.17 – Panel without caption

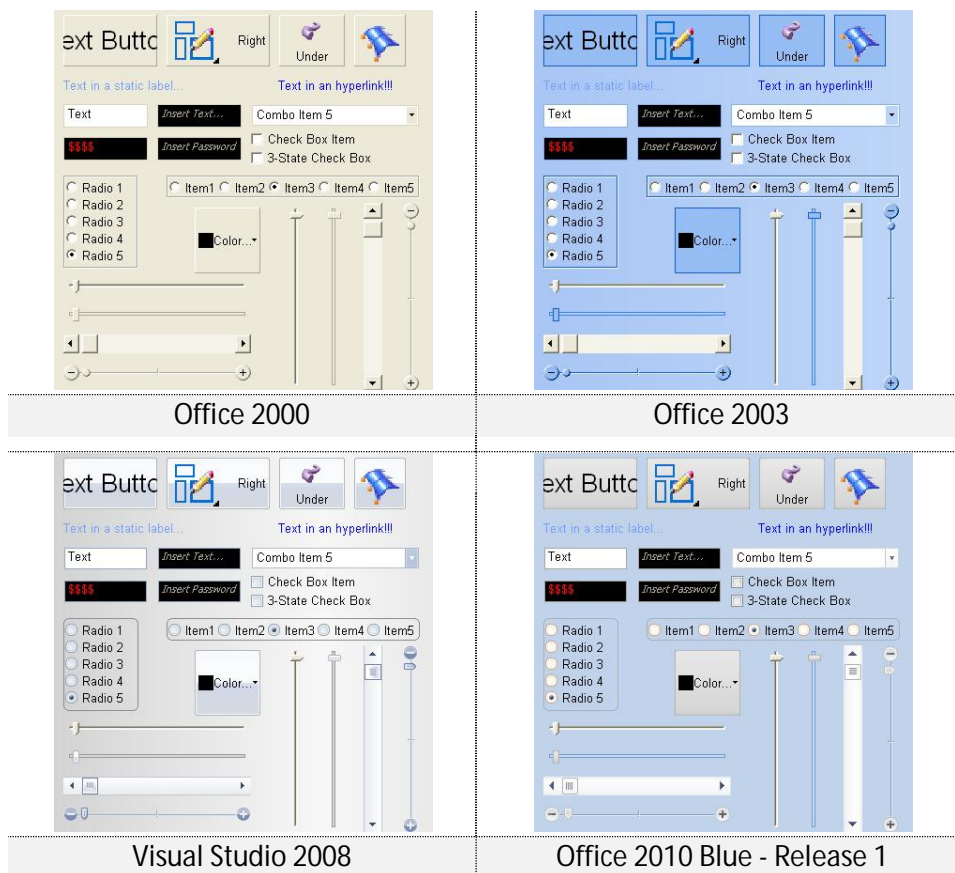
In order to maintain backward retro compatibility with the previous Eagle versions a new kind of panel has been implemented. This solution permits creation a dialog with the following features:

- without caption and frame borders,
- neither sizable nor movable,

- fixed position related to the main frame like previous Eagle versions, so when the main window is moved, the panel will also be shifted and maintain the same relative location to the top-left corner of the main frame.

In order to generate this kind of panel, there are two methods:

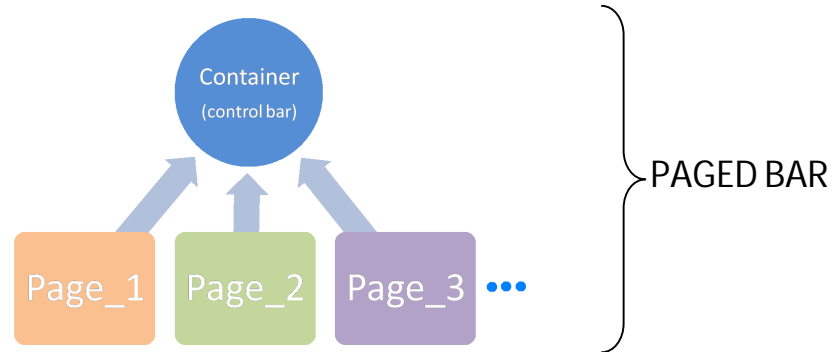
- define a panel using the new pin value -23 or use the legacy PIN=1;
- use the old panel with pin number 1 and not define the title in the "panel" command.
- Just like generic panels the appearance of these panels are painted with the current theme:



## 8.18 – Paged Bar

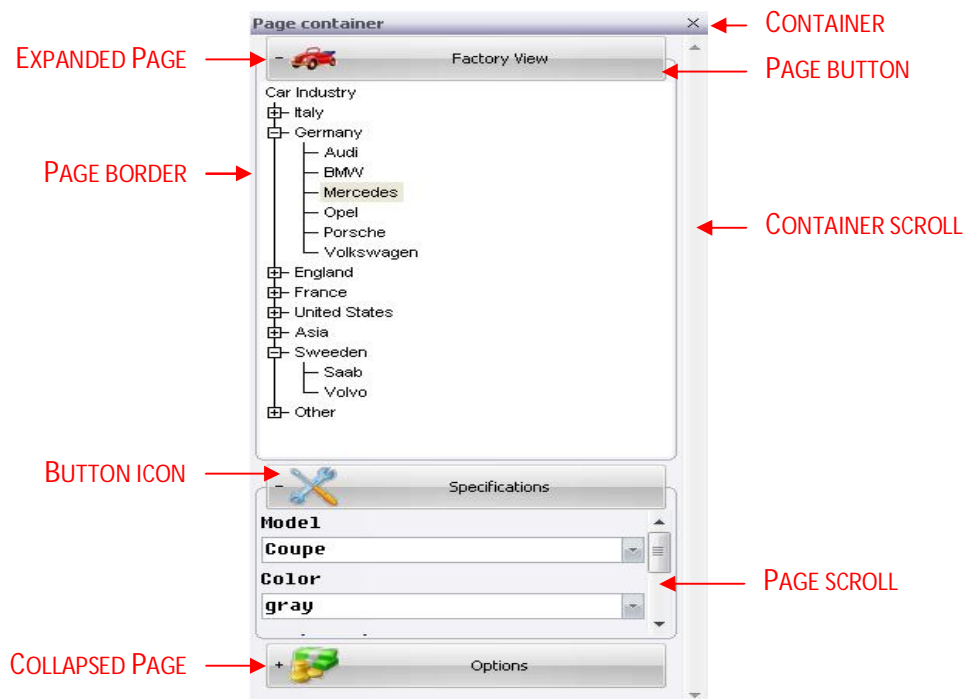
The Eagle paged bar implements the functionality of a page container window which also supports scrollable dialogs (similar to Microsoft's Outlook shortcut bar).

In this panel it's possible to embed other panels as page windows in a manner similar to Tab-Pages in a Tabbed-Bar and it is for this reason the construction steps are more or less the same.



The container is a standard “Dialog Bar” that is created using one of the available pin settings (-7, -8, -9, -10, -11, -17, -18, -19, -20, -21, -22). In this case the control bar contains a *PageContainer*, which means its internal layout is organized with Pages, each holding a set of controls. In order to specify that a control bar is a container the panel command must define the “container” primer as “page” i.e. container = page.

The behavior of a Paged Bar is completely different from that of a Tabbed Bar because the user doesn’t select a specific page from a tab one at a time, conversely all or some of the single pages could either be expanded (opened) or collapsed (closed). In this way the user is able to have more than one page opened at the same time. The operation to expand or collapse a page can be easily achieved by clicking over the title-button, a special icon in the top left corner of the button indicates the current status of the page: “+”, the page is expanded and “-”, the page is collapsed.



The first step in creating a “PageContainer” is to define the container to which all the pages will be attached. Eagle enables the developer to choose between various styles (mod=<i>parameter</i>) :

1. Vertical alignment and default behavior: Pages are displayed in a vertical layout pattern and the user is able to open more than one page at the same time or equally can to close all the pages.
2. Vertical alignment and only one page opened: Pages are organized like style 1, but in this case the user is only able to open one page at a time. So this time when a page button is pressed to open a new page, the previously expanded page is automatically collapsed.
3. Vertical alignment and at least one page opened: Pages are organized like style 1, but at least one page must be always expanded.
4. Horizontal alignment and default behavior: Pages are ordered with horizontal layout and has the default behavior (like that style 1) where multiple pages can be opened or closed.
5. Horizontal alignment and only one page opened: Pages are organized in the manner of style 2, but this time using horizontal layout.
6. Horizontal alignment and at least one page opened: Pages are organized with the behavioral style of style 2, but this time using horizontal layout.

The "panel" instruction has the following prototype for "Page-Container":

Prototype :

```
panel {on|off}, p=<i>, u=<i>, r=<i>, c=<i>, vr=<i>, vc=<i>, t=<text>,
j=<n1>,<n2>, mod=<i>, pin=<i>, name=<i>, att=<i>, open, clos,
container=page
```

where:

Parameter	Description
on	Initiate a new panel.
off	Remove a panel from the screen.
p=<i>	To specify the panel index (ID). A value between 1 and 96.
u=<i>	Is the unit in pixels of the menu grid ( default is 16).
r=<i>	Number of rows. (Note, the maximum width/height for panels extends to the dimensions of the Eagle frame).
c=<i>	Number of columns. (Note, the maximum width/height for panels extends to the dimensions of the Eagle frame).
vr=<i>	Number of visible rows. (Note, the maximum width/height for panels extends to the dimensions of the Eagle frame).
vc=<i>	Number of visible columns. (Note, The maximum width/height for panels extends to the dimensions of the Eagle frame).
t=<text>	Panel title.

<b>j=&lt;n1&gt;,&lt;n2&gt;</b>	Position (x, y) on the screen in normalized units between 0 and 1 or in pixel unit. If j is not specified the default values are the center of the application frame.
<b>mod=&lt;i&gt;</b>	Specifies the style of container; the value between 1 and 6: 1. Vertical alignment and default behavior. 2. Vertical alignment and only one page opened. 3. Vertical alignment and at least one page opened. 4. Horizontal alignment and default behavior. 5. Horizontal alignment and only one page opened. 6. Horizontal alignment and at least one page opened.
<b>pin=&lt;i&gt;</b>	To specify the type of panel, can be of one of the following types: <ul style="list-style-type: none"> <li>• floating dialog</li> <li>• fixed dialog</li> <li>• dockable/floating dialog</li> <li>• floating (only) dialog</li> </ul> Permitted values are: 0,1,-7,-8,-9,-10,-11,-17,-18,-19,-20,-21,-22, and the differences between each possibilities will be presented in the next sections.
<b>name=&lt;i&gt;</b>	If equal to 1 then the resource name is, <i>panel_&lt;i&gt;</i> , where "i" is the panel index, otherwise, equal to 0, the resource name is <i>panel</i> .
<b>att=&lt;i&gt;</b>	If set to -1 panel PIN option setting will define whether or not a panel is floating/dockable and to which extremity of the container windows it should be docked. Else if equal to -999 and the panel is a toolbar, the bar will be created as floating from the start, but then dockable, you have to specify ATT=-999 and J=x,y (the starting position).
<b>open</b>	Show a hidden panel.
<b>clos</b>	Hide the panel without destroying it. The panel is held in memory for subsequent use. (This corresponds to an unmanaged widget call). When using this primer no other primers are available.
<b>container=page</b>	To set a specific panel as "Page Container", according with PIN value.

Sample  
Code :

```
panel on,p=20,u=1,r=400,c=300,t='Pager',pin=-19,att=-1,mod=3,container=page
```

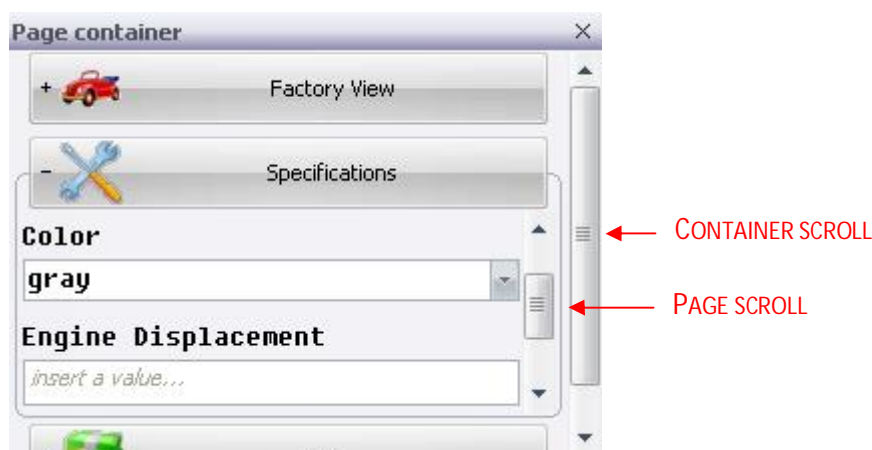
Each page attached to a container contains:

- a button to open or to collapse the panel;

- a title displayed inside the button;
- (optionally) an icon (\*.ico file) near the title;
- a border to delimitate the page;
- several controls defined through a classic "\*.tab" file.

Creating a page, the developer can define two different dimensions: the real size and a visible size. Naturally, when the visible size is not defined Eagle assumes that the entire panel is to be displayed. When the visible dimension is less than the real panel dimension the page shows the (vertical and/or horizontal) scroll bar. It is important to note that the page size also depends on the container dimensions, stretching the bar means the pages will also be resized.

It is possible to have more than one scroll bar at the same time because each page and the main container could conceivably have visible dimensions lower than their real size,



Consequently, in case of "Pages", the command "panel" has a different prototype:

Prototype :

**panel** {on|off}, p=<i>, u=<i>, r=<i>, c=<i>, vr=<i>, vc=<i>, t=<text>, pageof=<i>, open, clos, paged

where:

Parameter	Description
<b>on</b>	Initiate a new panel.
<b>off</b>	Remove a panel from the screen.
<b>p=&lt;i&gt;</b>	To specify the panel index (ID). A value between 1 and 96.
<b>u=&lt;i&gt;</b>	Is the unit in pixels of the menu grid ( default is 16).
<b>r=&lt;i&gt;</b>	Number of rows. (Note, the maximum width/height for panels extends to the dimensions of the Eagle frame).
<b>c=&lt;i&gt;</b>	Number of columns. (Note, the maximum width/height for panels extends to



	the dimensions of the Eagle frame).
<b>vr=&lt;i&gt;</b>	Number of visible rows. (Note, the maximum width/height for panels extends to the dimensions of the Eagle frame).
<b>vc=&lt;i&gt;</b>	Number of visible columns. (Note, the maximum width/height for panels extends to the dimensions of the Eagle frame).
<b>t=&lt;text&gt;</b>	Page title to display in the page button and separated by “#” to the path and name of an icon (ico file) to insert in the page button. The complete syntax is : t=‘title#path/icon.ico’
<b>pageof=&lt;i&gt;</b>	Represents the id of the Container into which the Page will be added.
<b>open</b>	Show a hidden panel.
<b>clos</b>	Hide the panel without destroying it. The panel is held in memory for subsequent use. (This corresponds to an unmanaged widget call). When using this primer no other primers are available.
<b>paged</b>	To set a specific panel as “Page”.

Sample Code :

```
panel p=51,u=1,r=300,c=100,vr=300,t='Factory View#car.ico',paged,pageof=20
```

A complete “Page Container” sample is illustrated below:

Sample Code :

```
#Container definition
panel on,p=20,u=1,r=400,c=300,t='Pager',pin=-19,att=-1,mod=1,container=page

#Page definition
panel p=51,u=1,r=300,c=100,vr=300,t='Factory View#car.ico',paged,pageof=20
panel p=52,u=1,r=200,c=100,vr=100,t='Specifications#sets.ico',paged,pageof=20
panel p=53,u=1,r=400,c=400,vr=100,t='Options#money.ico',paged,pageof=20

#Options for the pages
option p=51,f=pageView.tab
option p=52,f=pageSettings.tab
option p=53,f=pageOption.tab
```

The “click” command enables automatic expanding or collapsing a specific page,

Prototype :

```
click      p=<i>, b=<i>
```

where:

Parameter	Description
$p=<i>$	The tabbed bar identifier the range 1 to 96.
$b=<i>$	The index of pages, in the range 1 to N, where N is the number in the pages.

## 9 – Controls

The user interface of an application is made up of components that perform two simple functions, presenting the application information to the user and allowing the user to enter data for the application. Eagle v.14 offers several controls to obtain the desired type of action and presents different commands to work with buttons (not only confined to the “option” function discussed in the [previous section](#)).

At the highest level, components are organized into MainWindows, Menus and DialogBoxes and deciding which component group to use is one of the most important tasks for application designers.

The importance of screen menus and dialog boxes as well as areas where information about the state of the application is continuously updated cannot be underestimated, but a graphic application must not suffer in terms of graphic area size as a result of cluttering by screen menus. The approach to the design of the menu layout, window sizes and attributes, messages areas and, last but not least, the menu navigation model, must be considered carefully. All possible interaction scenarios and behavior should be organized under the control of the same formula if you are to reach consistency within the application interface.

The first thing to do when deciding on the correct components is to, define all the objects classes involved in the application, for example, text, graphics, windows and menus; Then define the subclasses, for the class Text define the subclasses of Help, Messages, Lists, etc.

Eagle v.14 introduces new version of components which inherit their look from the current theme and/or from the operating system, so common look and operability are automatically assured. In Eagle there is even more configurability such as assignment of fonts for individual panels where we absolutely want a specific look.

A principle guideline is to give to all the components a similar appearance in order to make them more familiar to the user. The same streamline in appearance design should be followed for each component group, as well as in behavior definition, to result in a common look and operability.

The controls available on Eagle v.14 are extended with new button types and new functionality is added. The following table presents a list of all the control features currently available.

Type	Index	Version Status
Push Button	1	Same in Eagle v.12
Button with Bitmap	4	Extended
Label	5	Same in Eagle v.12
Edit	6	Extended
Combo Box	27,127	Same in Eagle v.12
Multi-column Combo Box	37,47,57,67	New
Check Box	8	Same in Eagle v.12

Three State Check Box	18	New
Radio Group Box	29,39	Extended
Slider with arrow marker	10,110	Changed
Slider with indicator	210,310	New
Scroll Bar	410,510	New
Zoom Bar	610,710	New
Spin Button	11,111	New
Spin Edit Button	211,311, 411, 511	New
Frame Color	12	Extended
Frame Title	120	Extended
Static Label	220	Same in Eagle v.12
Tree View	300	Extended
Color Picker Button	500	New
Hyperlink	501	New
Progress Bar	502	New
Rich Edit	503	New
Grid View	none	New
List Button	505,506	New
List View	none	Extended

One of the most important benefits of using Eagle v.14, is the ability to add any type of control to every class of panel whether it is a tool bar, dialog bar, and so forth. This removes a limitation of earlier Eagle versions, where not every type of panel supported every kind of control.

IN the process of this GUI analysis we have seen various commands that we can apply to a control (i.e. freeze, click, etc.), so before introducing the specific button types, the other common issues for controls will be presented, for instance the icon format and the "pbutton" command are always the same.

## 9.1 – Icon Format





The icon format supported is Windows Bitmap (extension BMP), single icon, without limits on colors. Background transparency is generated by defining a practical background color and setting it in the "BITMAP\_TRANSPARENCY\_COLOR" variable of the configuration file.

The color is specified using a hash "#" followed with three hexadecimal values (one for red, one for green and one for blue), so the range of possible values is : #000000 to #FFFFFF;

INI Sample :

```
BITMAP_TRANSPARENCY_COLOR=#FF33DD
```

By default the value used for icon background is "red=255, green=0, blue=255" which Eagle transforms into a transparent background. So, if the BITMAP\_TRANSPARENCY\_COLOR is defined as #FF00FF, we accordingly will have the following presentation:

Bitmap Image	Button Resolution
Background Color = 255, 255, 0 	Colored Background  Push Under
Background Color = 255, 0, 255 	Transparent Background  Push Under

Developers can create a transparent background simply using any standard appropriate imaging software tool such as Axialis, Microsoft Paint, Corel or Photoshop.



Note sometimes, for instance in the main frame, in a gwindow caption, or in page containers, the icon format must be the classical "\*.ico" type. The cases where it applies have been documented in the relevant sections.



From V.14.5.2, Eagle supports "Vista style Icon multi-format" in ribbons, toolbars, etc.

## 9.2 – Adding and Removing controls

In the section about [panel and options](#) we have seen how to insert a collection of controls inside a panel, however Eagle also allows you to dynamically add or remove buttons from an existing panel using the "pbutton" command. When a control is added to the panel, the developer should specify the complete description in the same format defined for the option's (\*.tab) file.

Prototype :

pbutton p=<i>, b=<j>, add='button spec' | del

where:

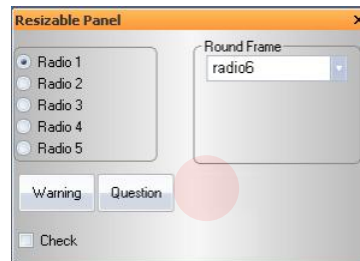
Parameter	Description
<b>p</b>	Indicates the number of the panel in which the button is present
<b>b</b>	Specify the index of the button to be added or removed. This parameter is also mandatory when using the "add" primer .
<b>add='spec'</b>	Specify the option-file information that determines the new button. The description is defined between two single quotes
<b>del</b>	Remove the button whose index is defined by means of the b= option.

Sample Code :

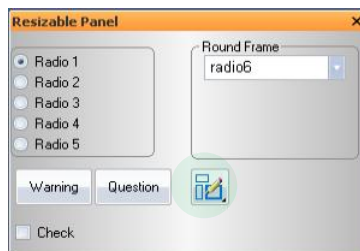
```
pbutton p=1,b=1,add='1,0,2,220,31,3,6,1,"New Button 1":;'
pbutton p=1,del,b=1
pbutton p=1,b=1,add='1,0,2,220,31,3,6,1,"Another Button 1":;'
```

For example :

```
pbutton p=10, b=9,
add='9,150,120,32,32,0,0,4,
"icon.bmp": tell "test"'
```



```
pbutton p=10,
b=9, del
```



The description of the newly defined control is contained inside single quotes (single inverted commas). If the syntax of the insert contains text (i.e. a command, a title, etc.), they should be delimited with double quotes in order to differentiate them from the single quotes item description.

### 9.3 – The “click” command

We have already introduced the click command during the presentation of tabbed bars, but the same instruction can be applied to execute also other control actions :

- Set the color to a “Color Frame”, “Edit field”, “Static label”, “Hyperlink”, “Color Picker”;
- Enable/Disable a “Check Box” or Enable/Disable/Gray a “Three State Check Box”;
- Change the selected radio in a “Radio Group Box”;
- Set an item in the “Combo Box”;
- Invert the background with the foreground;
- Select a row in a “List”;
- Open a branch in a “Tree View” (refer to section 9.20 for more details);
- Manage the items in a “Tree View” (refer to section 9.20 for more details).

Prototype :	
click	<code>p=&lt;i&gt;, b=&lt;i&gt; {,t=&lt;i&gt;} {,col=&lt;i&gt;} {,back=&lt;i&gt;} {,inv}</code> <code>{select='&lt;path&gt;'   '%&lt;uid&gt;,&lt;text_update_option&gt;'}</code>

where:

Parameter	Description
<b>p</b>	Indicates the number of the panel in which the button is present
<b>b</b>	The button number in the panel.
<b>t</b>	If the button is a "radio button" or a "combo box" then <toggle> identifies the option to activate starting from 0. The command reset also the current selected option.
<b>col</b>	Change the color in a "frame color button", the font color in an edit field, in a static label or in a hyperlink. The COL primer is defined using an Eagle color index.
<b>back</b>	Change the background color in an edit field. The COL primer is defined using Eagle color index.
<b>inv</b>	To invert the background and the foreground colors
<b>select='&lt;path&gt;'</b>	<p>Selection of a branch of a tree-view control from a macro. Example:</p> <p style="text-align: center;">CLICK P=1,B=1,SELECT='root/dir1/dir2/'</p> <p>Note: The value given represents the path from the root of the tree to the node (folder or leaf) to be selected. This value must terminate with a tree-view terminator character (e.g.: a slash), so the format is the same returned by the interactive selection of a node in the polling string.</p>
<b>select='%&lt;uid&gt;,&lt;update_option&gt;'</b>	<p>The CLICK command has been extended to allow the selection of a treeview node specifying its UID with the syntax:</p> <p style="text-align: center;">CLICK P=j,B=k,SELECT='%&lt;uid string&gt;'</p> <p>The % character is used to branch from a path-based select. The &lt;uid string&gt; has a maximum length of 16 characters; the select is further extended to support the UPDATE of a treeview node with the syntax:</p> <p style="text-align: center;">CLICK P=j,B=k,SELECT='%4567',&lt;update_option&gt;</p> <p>Where &lt;update_option&gt; is one or more of the options from amongst:</p> <p style="text-align: center;">       LABEL='&lt;node_label&gt;'        IMAGE=&lt;icon_filename&gt; (# Note no quotes)        MENU='&lt;context_menu_filename&gt;'        ACTION='&lt;action_filename&gt;'        DND='&lt;drang-n-drop_filename&gt;'        SILENT (No action for instance during a "loosefocus" action)     </p> <p>Alternatively, the \$ character can also be used to branch a specific node</p>

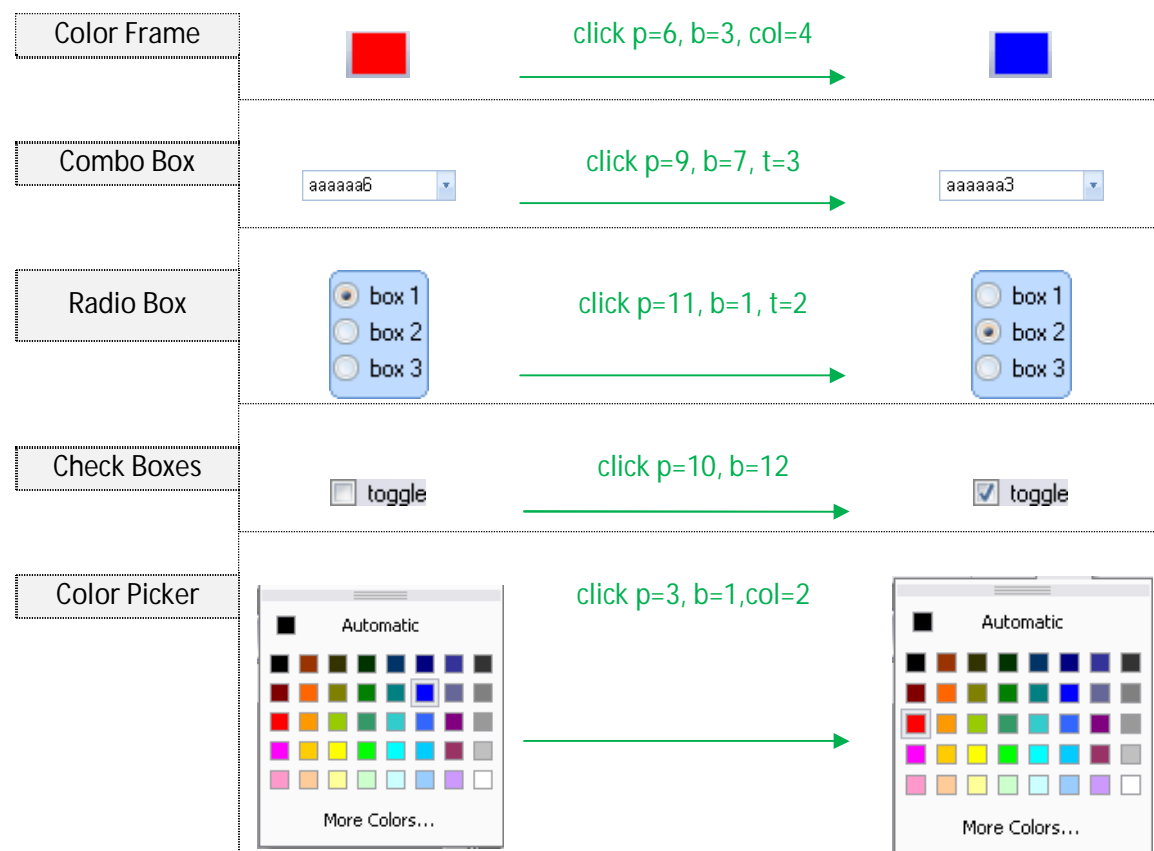
using the index. The <index> is the first value specified in the "dat" file to define a treeview node; the syntax is:

CLICK P=j, B=k, SELECT='\$3,<update\_option>

For example :

Sample Code :

```
click p=1, b=button
click p=panel, b=1, inv
click p=panel, b=4, t=2
click p=1, b=frame1, col=141
click p=1, b=colpick, col=2
click p=1, b=1, select='root/dir1/dir2/'
click p=j, b=k, select = '%4567', label='Inter Milan'
click p=j, b=k, select = '%4567', label='Inter Milan', imange=Scudetto
click p=j, b=k, select = '%4567', action='do @fitOnPage'
click p=j, b=k, select = '%4567', imange =lb2\books.bmp
click p=j, b=k, select = '$3'
```





### 9.3.1 – Pushed buttons

The “click” command applied to text buttons ([see 9.5](#)) and image buttons ([see 9.6](#)) permits changing of the “pushed” state of the control using the following syntax:

Prototype :

```
click    p=<i>, b=<i>
```

This function enables the user to able to switch the state of a button from an un-pressed state to pressed state and vice versa. This function is useful, if for instance, you need to show if an action related to a button has been executed or not.

Sample Code :

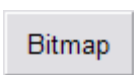
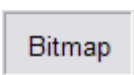
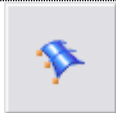





```
click p=33, b=1
click p=1 b=4
```

Because the style of the controls depends on the currently installed theme, the aspect of the pushed state is also different between the various themes.

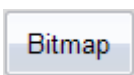

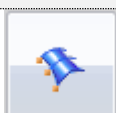





Furthermore, image buttons can have a different pushed color for dialog and toolbar, for instance with Office 2007 themes. This behavior is congruent to the theme and the OS characteristics, in fact in Microsoft Office 2007 there is also the same behavior which is presented in the Office 2007 Eagle’s themes.

The following table presents several cases in various themes:

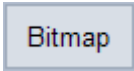
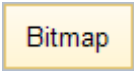
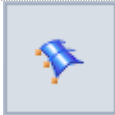





#### Office 2000

	Text Button Normal	Text Button Pushed	Image Button Normal	Image Button Pushed
Dialog Bar				
Tool Bar				

#### Office 2007 R1

	Text Button Normal	Text Button Pushed	Image Button Normal	Image Button Pushed
Dialog Bar				
Tool Bar				

#### Visual Studio 2010

	Text Button Normal	Text Button Pushed	Image Button Normal	Image Button Pushed
Dialog Bar				
Tool Bar				

## 9.4 – Set and Get a string

Eagle provides two different functions to control a “text” field; The “insert” command permits setting of the string of panel textual button:

Prototype :

```
insert      p=<i>, b=<i> ,t=<text>
```

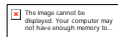
where:

Parameter	Description
<b>p</b>	Indicates the number of the panel in which the button is present
<b>b</b>	The button number in the panel, starting from 1.
<b>t</b>	Text to be inserted, between single quotes.

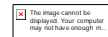
An example could be:

Sample Code :

```
insert p=1, b=2, t='Hello!'
```



```
Insert p=9, b=3,t='Hello!'
```



It's also possible to obtain the text string placed in a textual field, this can be achieved using the “fetch” command. The “fetch” returns a text string from a textual field of a menu in a panel:

Prototype :

```
fetch      p=<i>, b=<i> ,t=<svar>
```

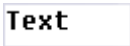
where:

Parameter	Description
<b>p</b>	Indicates the number of the panel in which the button is present
<b>b</b>	The button number in the panel, starting from 1.
<b>t</b>	Variable name where to store the text.

For example :

Sample Code :

```
fetch p=1, b=2, t=mystring
```

Graphic Object	Eagle Code	Output on Message Bar
	<pre>string mystr fetch p=9, b=5, t=mystr tell mystr</pre>	Text

## 9.5 – Text Button

A text button is one of the simplest controls in a GUI. In Eagle you can utilize this type of button in the interface by using the type "1" in the tab file :


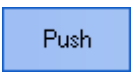
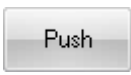
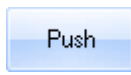




Tab Syntax :

```
opt, x, y, w, h, f, b, t, <text> : <action>
```

where:

Parameter	Description
<b>opt</b>	Index of the option.
<b>x</b>	X co-ordinate of the button in the menu grid.
<b>y</b>	Y co-ordinate of the button in the menu grid.
<b>w</b>	Width of the button in grid units.
<b>h</b>	Height of the button in grid units.
<b>f</b>	Foreground color for buttons.
<b>b</b>	Background color for buttons.
<b>t = 1</b>	Button type = 1
<b>&lt;text&gt;</b>	Message to display in the button.
<b>&lt;action&gt;</b>	Command to take when selected.

When the mouse pointer has moved over a button, it is highlighted and it's possible to push the button to perform the command added in the "action" field. The next table shows how a textual button is displayed in various themes (in normal status or highlighted status) :

	Office 2000	Office 2003	Office 2007 Standard - R1	Office 2007 Luna Blue - R2
Normal				
Highlighted				

When a button is pressed, through the polling loop it's possible to set these values :

- MN = panel number;
- BN = button number.

## 9.6 – Image Button

An evolution of the “textual” button is represented by the “image” button, which includes a “bitmap” image in one of the following configurations:

1. Only an image without text;
2. An image with a text on the right side;
3. An image over a text field;

The option’s type of this control is “4” and to choose the appropriate layout the you must define the text field on the” tab description” as shown below :

Tab Syntax :

```
opt, x, y, w, h, f, b, t, <text> : <action>
```

where:





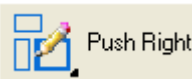
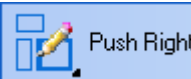






Parameter	Description
<b>opt</b>	Index of the option.
<b>x</b>	X co-ordinate of the button in the menu grid.
<b>y</b>	Y co-ordinate of the button in the menu grid.
<b>w</b>	Width of the button in grid units.
<b>h</b>	Height of the button in grid units.
<b>f</b>	Foreground color for buttons.
<b>b</b>	Background color for buttons.
<b>t = 4</b>	Button type = 4
<b>&lt;text&gt;</b>	<p>The text field allows to set the configuration to display (between single quotes) :</p> <ol style="list-style-type: none"> <li>1. ONLY IMAGE : specifying only file name                ' image.bmp' or ' image.ico'</li> <li>2. IMAGE AND TEXT ON THE RIGHT : specifying the file name before the text, separated by a special character #                ' image.bmp#text' or ' image.ico#text '</li> <li>3. IMAGE AND TEXT ON THE BOTTOM : specifying the file name before the text, separated by the special characters ##                ' image.bmp##text' or ' image.ico##text '</li> </ol>
<b>&lt;action&gt;</b>	Command to take when selected.

Similar to the “text” button, also the image button is affected by the theme and it can be highlighted when the mouse pointer goes over the control. For example:

Sample Code :

```
3,330,220,40,40,0,0,4, 'test.bmp': tell 'push'
6,202,150,100,40,0,0,4, 'test.bmp#Push Right': tell 'push'
9,240,220,70,70,0,0,4, 'test.bmp##Push Under': tell 'push'
```

The next table describes the three possible configurations for the “image” button:

	Office 2000	Office 2003	Office 2007 Standard R1	Office 2007 Luna Blue R2
Only Icon				
	3,330,220,40,40,0,0,4, 'test.bmp': tell 'push'			
Icon beside text				
	6,202,150,100,40,0,0,4, 'test.bmp#Push Right': tell 'push'			
Icon over text				
	9,240,220,70,70,0,0,4, 'test.bmp##Push Under': tell 'push'			

When a button is pressed, through the polling loop it's possible to set these values:

- MN = panel number;
- BN = button number.

If an “icon only” button contains a bitmap image without a transparent background that fills the entire area, the button click event (changing status) is not visible because the button effects are hidden by the defined image.

In this case, the user is able to define the “BUTTON\_IMAGE\_BORDER” variable in the configuration file to set a border size in pixel units to solve the problem. The button image border default value is 0.

INI Sample :

BUTTON\_IMAGE\_BORDER = 3

Furthermore, it is now also possible to use the icon format (\*.ico) as the image for the button. This means that it is necessary to set the file extension (\*.bmp or \*.ico) in the tab file to guide the correct format selection, but if no extension is specified the bitmap (.bmp) is the default format. Eagle will automatically select the appropriate icon format (i.e. 16x16, 32x32, etc.) depending on the displayed button size.

## 9.7 – Label

The label object, also called static, is a textual field which contains only a string without performing any action. The background of the control is painted according to the currently defined theme and the font type is selected by using the option we discussed previously.

The label text is always aligned to the left taking into consideration the width of the field and is centered taking into consideration the label height.

The option's type of this control is "5" and to choose settings the you have to define the "tab description" as shown below :

Tab Syntax :

```
opt, x, y, w, h, f, b, t, <text> :
```

where:

Parameter	Description
<b>Opt</b>	Index of the option.
<b>x</b>	X co-ordinate of the button in the menu grid.
<b>y</b>	Y co-ordinate of the button in the menu grid.
<b>w</b>	Width of the button in grid units.
<b>h</b>	Height of the button in grid units.
<b>f</b>	Foreground color for buttons.
<b>b</b>	Color for the font in the label
<b>t = 5</b>	Button type = 5
<b>&lt;text&gt;</b>	The text field to display (between single quotes)

For example :

Sample Code :

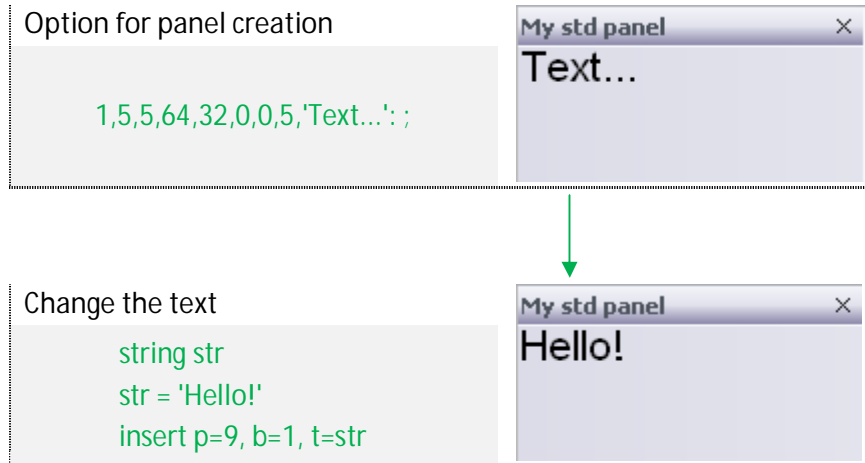
```
1,30,20,40,40,0,0,5, 'text': ;
7,20,10,100,40,0,0,5,'my text field: ;
```

The next table describes how labels are presented in different themes:

Office 2000	Office 2003	Office 2007 Standard R1	Office 2007 Luna Blue R2
<b>Text...</b>	<b>Text...</b>	<b>Text...</b>	<b>Text...</b>

```
3,136,120,64,32,0,0,5,'Text...': ;
```

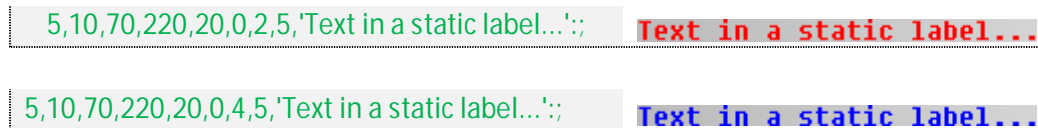
The example below explains how use the "insert" command to change the text inside a label:



Because a static doesn't support any actions, this item returns no values to the polling loop.

Another feature introduced with the version 14, is setting the color for a text field. It's now possible to change the font color either during creation of the control or also by dynamically changing the color of the font when it already exists (this option is also available for others controls).

The "f" parameter in the tab file permits changing of the color of the text displayed in the static control. It is possible to assign a color specifying the corresponding to the index of Eagle's color palette, for instance:





## 9.8 – Edit

The edit control is a rectangular control window, typically used in a dialog, which permit users to enter and edit text using keyboard or tablet tools. Single-line edit controls are useful for retrieving a single string from the user so the look of this field is always the same in all color schemes.

Eagle v.14 extends the edit object to provide a classical “password” control where entered text is obscured with the special “\*” character. In addition it is also possible to specify the maximum number of characters for an input field. The value is specified in the default value using a “#” character as separator (e.g. ...,6,'obrien#20' ...). When typing in the input field, if the maximum number of characters allowed is exceeded, a warning message is displayed in the message window.

The option’s type of this control is “6” and to choose settings the user must define the “tab description” as shown below :

Tab Syntax :

**opt, x, y, w, h, f, b, t, <text> : <action>**

where:

Parameter	Description
<b>opt</b>	Index of the option.
<b>x</b>	X co-ordinate of the button in the menu grid.
<b>y</b>	Y co-ordinate of the button in the menu grid.
<b>w</b>	Width of the button in grid units.
<b>h</b>	Height of the button in grid units.
<b>f</b>	Color for the edit field.
<b>b</b>	Color for the font in the edit
<b>t = 6</b>	Button type = 6
<b>&lt;text&gt;</b>	<p>‘?text@banner’</p> <p>Using the syntax above, this field allows to set:</p> <ul style="list-style-type: none"> <li>• a default text displayed (text)</li> <li>• the password style (?)</li> <li>• a cue banner (@banner)</li> </ul> <p>To handle “password” type the first character on the text field must be defined to “?”, where the “question mark” is the default to present the text obscured like a password; moreover developers are able to define a cue banner using the “@” character followed by the text to display.</p>
<b>&lt;action&gt;</b>	Command to take when the RETURN key has been pressed.

For example :

Sample Code :

```
4,4,160,150,32,0,0,6,'Text': vane;
5,4,194,150,32,0,0,6,'?pwd': do pwd
6,4,10,10,32,0,0,6,'?@insert text': do pwd
11, 4,100,100,32,0,0,6,'@insert value': do test;
```

A new feature for the edit is represented by the “password” attribute. Placing a “?” as the first character in the text field any entered input will be obscured.

It is also possible to define the “password character” displayed in the edit control by using the “PASSWORD\_CHAR” entry in the configuration file. If no value has been specified the default password character used is “\*”.

INI Sample :

```
PASSWORD_CHAR = $
```

In Eagle v.14 the edit control has the capability of displaying a “cue banner”, which is a line of 'grayed out' text which can be used as a prompt to the user offering information about what type of data is expected to be entered into that box. Naturally, the cue banner is only displayed when the input field is empty and awaiting user entry. The “cue banner” is also available with the password attribute. A developer can implement a banner by adding in “@” character in the text field followed by the relevant text to display. It is useful to remember that the field entry should have enough visible room to accommodate the “cue banner”.

The next pictures present samples to explain possible uses for the edit control :

Normal Edit	<div>text</div> <pre>1,10,10,150,32,0,0,6,'text': do mw.cmd;</pre>
Empty edit with a banner	<div>insert a value...</div> <pre>4,4,160,150,32,0,0,6,'@insert a value...': do edit</pre>
Edit with text and banner defined	<div>Is a text</div> <pre>6,4,228,150,32,0,0,6,'Is a text@Is a banner': vane</pre>
Password with default “*”	<div>*****</div> <pre>1,10,10,150,32,0,0,6,'?input: do pwd.cmd</pre>
Password with “\$” character defined	<div>\$\$\$\$\$\$\$\$</div> <pre>5,4,194,150,32,0,0,6,'?wordpass': do pwd.cmd</pre>

Password with a cue banner

*insert a password...*

5,4,194,150,32,0,0,6,'?@insert a password...': do  
go

Edit handles only the event associated with the RETURN key; When this key is pressed the relevant polling loop values modified are:

- MN = panel number;
- BN = button number;
- ST = string in the control.

The "f" and "b" parameters in the tab file are used to change the color of the text and the background to the edit control by specifying the corresponding Eagle color palette index, for example:

Default colors (white, black)

text

1,10,10,150,32,0,0,6,'text': do mw.cmd;

Modified colors (yellow, red)

Text

1,10,10,150,32,6,2,6,'text': do mw.cmd;

If no colors are specified, so "f" and "b" are equal to zero, the default colors will be applied (text is black and background is white). Note that this color can also be changed at runtime by using the click command.

The style of all the edit fields can be changed to display a border. The configuration setting entry "EDIT\_BORDER\_ENABLED" enables setting of the "border" style in the edit field. The default value for this setting is "no".

INI Sample :

EDIT\_BORDER\_ENABLED= yes | no

For example :

Text

EDIT\_BORDER\_ENABLED= no

Text

EDIT\_BORDER\_ENABLED= yes

### 9.8.1 – Focus-in and Focus-out actions

It is possible to add an action to be executed when the focus enters the edit control and another action to be executed when the focus exits, that is when the control loses the focus in favor of another control.

To achieve the functionality two extra lines are required in the TAB file both using the same option number as the control, as illustrated in the example below.

Sample Code :

```
4,4,160,150,32,0,0,6,'Text': vane;  
5,4,194,150,32,0,0,6,'?pwd': do pwd  
5,4,194,150,32,0,0,6,'': do pwd_focusin  
5,4,194,150,32,0,0,6,'': do pwd_focusout  
6,4,10,10,32,0,0,6,'?@insert text': do pwd  
11, 4,100,100,32,0,0,6,'@insert value': do test;
```

Every time the control gains the focus, the action associated with the second line (do pwd\_focusin, in the example) is executed; similarly, when the focus is lost, the action associated with the third line (do pwd\_focusout, in the example) is executed.

By default, the POLLING command does not exit when the focus in/out actions are executed, which is the effect of setting PATCH4084 in the INI file is “no”. If the user wants that the polling loop terminates then PATCH4084 should be set to “yes”.

### 9.8.2 – Set the Focus to a control

When the POLLING command starts, it is possible to set the focus to a specific control, such as an edit control or a combo-box using the FOCUS option as illustrated in the following example,

Sample Code :

```
POLLING FOC=30,12
```

where the focus goes to button 12 in panel 30, which presumably is an edit control.

## 9.9 – Combo Box

A combo box is a unique type of widget that combines much of the functionality of a list box and an edit control. Usually it is a combination of a drop-down list or list box and a single-line textbox, allowing the user to either type a value directly into the control or choose from a list of existing options. An example of this use is the address bar of web browsers.

Eagle v.14 provides not only the traditional combo box control, but also a combo box which integrates a multi column list and a combo that displays a grid with several columns (for instance with a check box, an image and text).

Option	Type	Description
27	Combo Box with Fixed List	classical combo box with a dropdown list that shows all the elements
127	Combo Box with Scrollable List	editable combo box with auto-complete, in which the dropdown list has an user defined height and a scrollbar to reveal the elements of the list which are not visible
37	Combo Box with Multi column List	multi column list, where each row can contain only a text string
47	Combo Box with Grid (3 Columns)	combo that displays a grid instead a list; the grid has three columns: the first with a check box, the second with an image and the last with text
57	Combo Box with Grid (2 Columns)	the grid has two columns: the first with an image and the second with text
67	Combo Box with Grid (2 Columns)	the grid has two columns: the first with a check box and the second with text
77	Combo Box with just an Image	the dropdown list shows only the icon and has no label

All the combo items have the same presentation as the currently defined theme, so the only difference is represented by the object inside the list (i.e. strings, images, etc.).

In Eagle v.14, a standard combo is defined using the option's type "27" and settings should be defined using the "tab description" as shown below :

Tab Syntax :

`opt, x, y, w, h, f, b, t, <array_name> : <action>`

where:

Parameter	Description
<code>opt</code>	Index of the option.
<code>x</code>	X co-ordinate of the button in the menu grid.
<code>y</code>	Y co-ordinate of the button in the menu grid.

<b>w</b>	Width of the button in grid units.
<b>h</b>	Height of the list in grid units. This item sets the height of the popup list; if the size is not enough to display all the rows, a vertical scroll bar will be activated, else if the value is over-dimensioned the list height will be auto-fitted to maximum size needed.
<b>f</b>	Foreground color for buttons.
<b>b</b>	Background color for buttons.
<b>t</b>	Button type = 27, 127, 37, 47, 57, 67, 77 depending on the combo required
<b>&lt;array_name&gt;</b>	The array field contains the name of an Eagle's array of strings, between single quotes. The size of the array define the number of elements in the combo list and the default element is identified using the "&" character as first.
<b>&lt;action&gt;</b>	Command to take when a new row is selected.

For example :

Sample Code :

**Environment variables :**

```
string combo[5]
combo [1]='item'
combo [2]='item_2'
combo [3]='&item_3'      # DEFAULT ITEM
combo [4]='item_4'
combo [5]='item_5'
```

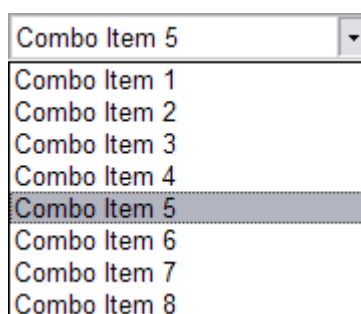
**Tab file :**

```
7,160,8,120,200,0,0,27,'combo': tell '--- COMBO ---';
```

Note: the maximum number of options for a combo box is 256.


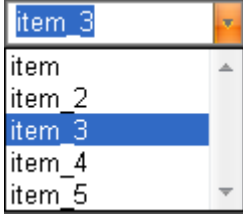
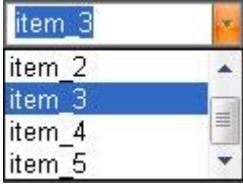
Type 27

The combo box 27 is the classic Eagle object in which the dropdown list displays all the elements available.



The type 27 list control is not very useful if the dropdown list contains a whole lot of items, for such cases it is better to use the combo 127 which also provides editing, auto-complete and a sizable dropdown list with scrollbar (if required).

In Eagle v.14, using the “height” value for type 127 combo box it is possible to set the height of the popup list, if the given value is not sufficient to display all the defined list rows, then a vertical scroll bar will be automatically activated allowing the user to scroll through all the available options. If a list given entry is greater value than the list height then it will auto-fit to required size, for instance:


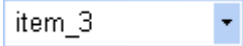


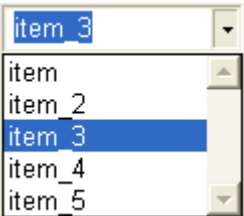
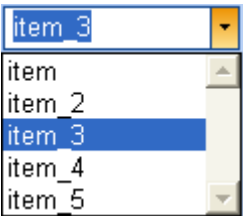
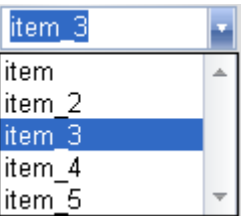
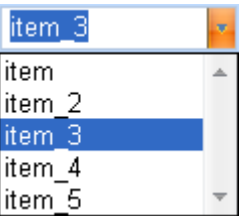
		
-	h = 200	h = 100
Combo Box	Auto-fitted Height	Scroll Bar Enabled

## Type 27

The type 127 combo boxes are editable, allow the use of the auto-complete function to quickly locate an option by typing part of the field entry and can be interacted with by :

- navigation using options : arrows, mouse wheel or mouse pointer (in the scroll bar);
- execution of a command : mouse single-click or return key;
- cancel operation : esc key or another click out the combo space.

The next table presents images to show how the combo box is painted in various theme styles :

Office 2000	Office 2003	Office 2007 Standard R1	Office 2007 Luna Blue R2
			
			

The Combo box handles events associated to the RETURN key and with the mouse single-click on an item in the list. When selected the relevant polling loop values modified are:

- MN = panel number;
- BN = button number;
- RN = index (1 based) of the item selected;
- ST = string of the item selected.

If a non-existing item has been inserted in the edit field when the RETURN key is pressed then the polling loop returns the contents of relevant non-existent “wrong” string in the ST attribute.

### Type 37

Type "37" enables creation of a multi column combo box which can contain only strings. These combo boxes are defined in the same way as standard combo but the array of items possesses different strings separated by a separator character.

The separator can be defined using the "MULTI\_COLUMN\_LIST\_SEPARATOR" configuration file setting. If this setting is not defined the default is ",". For example:

INI Sample :

```
MULTI_COLUMN_LIST_SEPARATOR = |
```

The number and the size of columns is indicated using another string array, where the size of the array represents the number of columns and the values present in the strings are the size in pixels for each of the columns.

Tab Syntax :

```
opt, x, y, w, h, f, b, t, <array_row#array_column> : <action>
```

where:

Parameter	Description
<b>opt</b>	Index of the option.
<b>x</b>	X co-ordinate of the button in the menu grid.
<b>y</b>	Y co-ordinate of the button in the menu grid.
<b>w</b>	Width of the button in grid units.
<b>h</b>	Height of the list in grid units. This item sets the height of the popup list; if the size is not enough to display all the rows, a vertical scroll bar will be activated, else if the value is over-dimensioned the list height will be auto-fitted to maximum size needed.
<b>f</b>	Foreground color for buttons.
<b>b</b>	Background color for buttons.
<b>t = 37</b>	Button type = 37 multi column combo box which can contain only strings
<b>&lt;row#column&gt;</b>	<p>The field contains the name of two Eagle's array of strings, between single quotes and separated with a "#" :</p> <p style="text-align: center;">'array_row_name#array_column_name'</p> <p>where :</p> <ul style="list-style-type: none"> <li>➤ array_row_name is a set of strings which define each row.</li> <li>➤ array_column_name is a set of string with the pixels for each column, the dimension of the array define the number of columns.</li> </ul> <p>The default element is identified using the "&amp;" character as the first character in the array_row_name.</p>



**<action>** Command to take when a new row is selected.

The next example shows how to create a multi column combo box which has 3 columns of 40, 50 and 60 pixels in size:

Sample Code :

**Environment variables :**

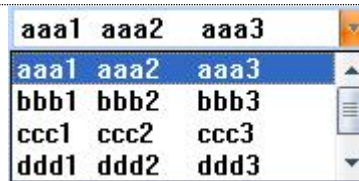
```
string columns[3]
columns[1]='40'
columns[2]='50'
columns[3]='60'

string multicb[8]
multicb[1]='aaa1|aaa2|aaa3'
multicb[2]='bbb1|bbb2|bbb3'
multicb[3]='ccc1|ccc2|ccc3'
multicb[4]='&ddd1|ddd2|ddd3'
multicb[5]='eee1|eee2|eee3'
```

**Tab file :**

```
7,160,8,120,200,0,0,37,'multicb#columns': tell '--- COMBO ---';
```

**Combo Box :**



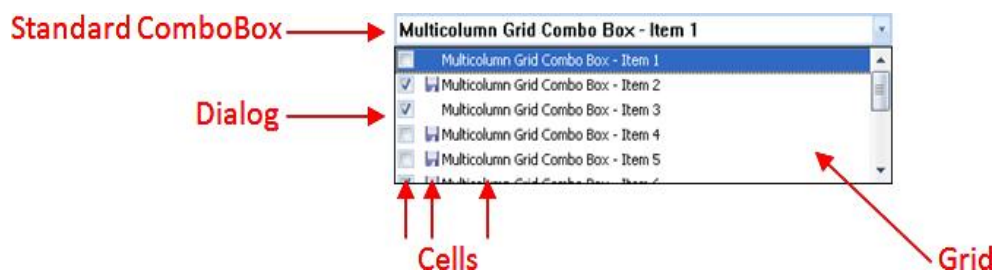
Whilst in the “polling loop” the type 37 combo box returns the same values as a normal combo, with the exception of the ST, which in this case, contains every row string divided by a separator. The separator character is the same one as defined in the “MULTI\_COLUMN\_LIST\_SEPARATOR” field.

### Type 47

Using the type “47”, developers are able to initiate a combo box that displays a grid component in the popup window. This element holds three columns:

1. check box;
2. image;
3. text.

The “Multi column Grid Combo box” has composed by different parts, a standard combo, a dialog instead the list window, a grid and cells :

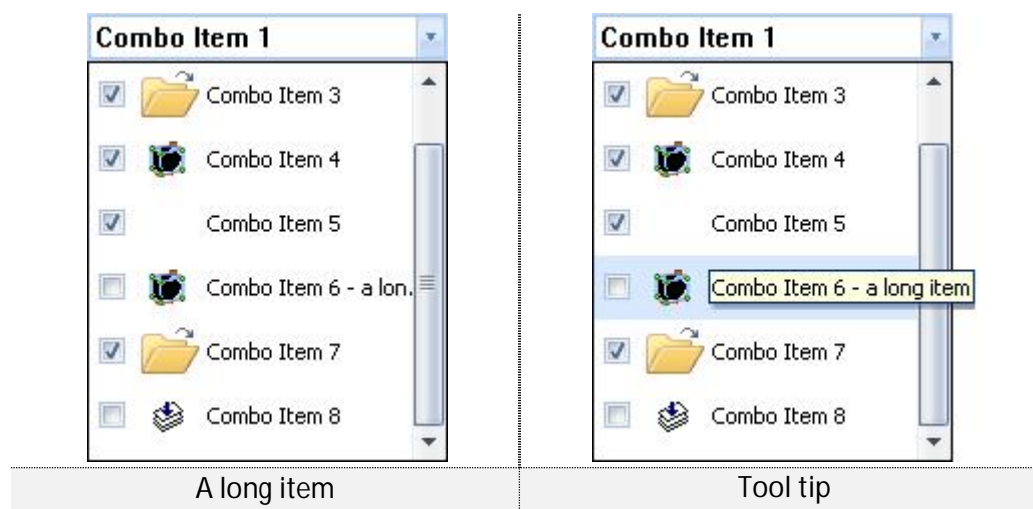


By means of a standard combo box as foundation for this control, the style of grid combo is the same of a normal combo box and when an item is selected the correspondent text will be displayed in the combo's edit field.

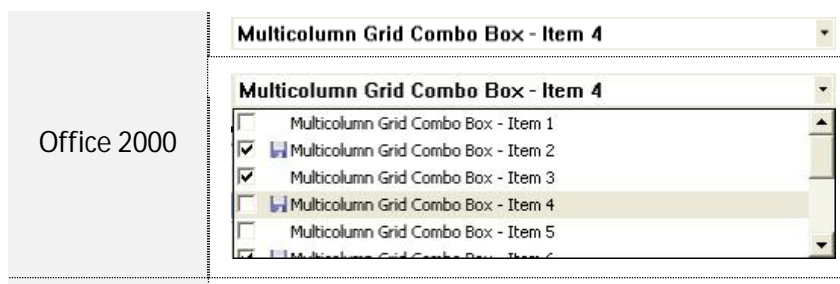
The height of the rows is calculated considering the maximum value in pixels from amongst all the icons present and the font used in the combo box. The width of the check box column is always the same, whilst the icons column depends on the width of the largest bitmap and finally the width of the text field is obtained using remaining difference.

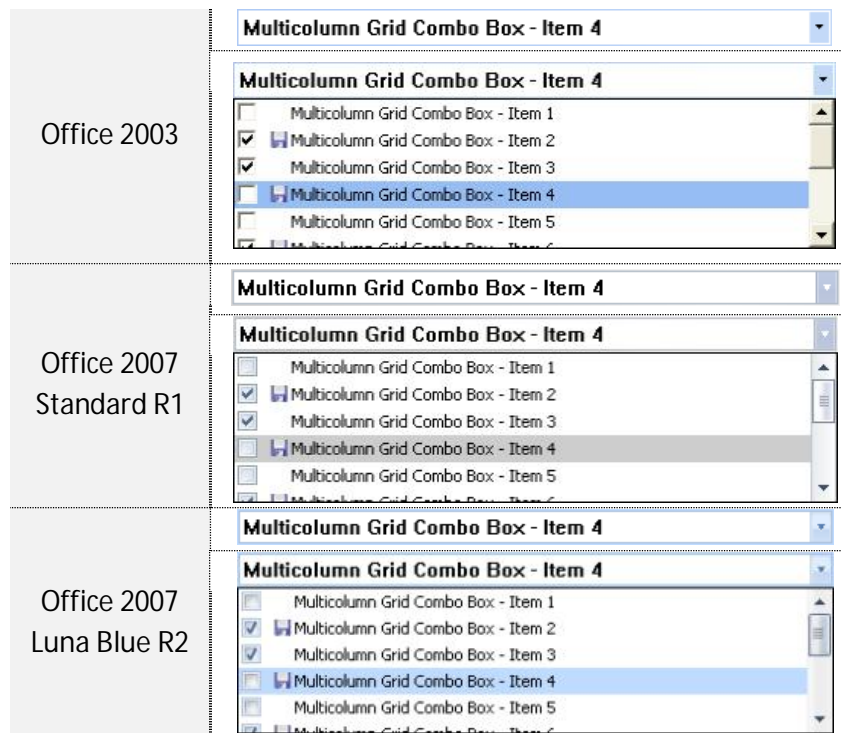
Check boxes and icons are always aligned to the center of the cell, whereas the text is left aligned and center in the height of the cell.

When a string is too long to fit in the available text field, dotted points will be added to the end of the line and in this case when the mouse pointer travels over the text a tooltip showing the entire string will be displayed.



The next table shows how the combo box is painted in various theme styles:





Furthermore in this case, the “h” parameter may be used to define the height of the list but now the array field reports two different variables :

1. Array of Rows
2. Array of Icons

The icon’s array contains a list of filename paths of icon resources (“\*.bmp” or “\*.ico”) whereas the array of rows defines the layout of each lines in the grid window :

&@Text#Icon\_index

where :

- & (optional) = default item
- @ (optional) = check box checked, if not present the box is unchecked
- Text = string in the text cell
- Icon\_index = 1-based index in the array of icon.

Tab Syntax :

opt, x, y, w, h, 0, 0, 47, 'array\_rows#array\_icons': <action>

where:

Parameter	Description
opt	Index of the option.
x	X co-ordinate of the button in the menu grid.
y	Y co-ordinate of the button in the menu grid.
w	Width of the button in grid units.

<b>h</b>	Height of the list in grid units. This item sets the height of the popup list; if the size is not enough to display all the rows, a vertical scroll bar will be activated, else if the value is over-dimensioned the list height will be auto-fitted to maximum size needed.
<b>f</b>	Foreground color for buttons.
<b>b</b>	Background color for buttons.
<b>t = 47</b>	Button type = 47
<b>&lt;array_rows#array_icons&gt;</b>	<p>The field contains the name of an Eagle's array of strings and Icon's array, between single quotes and separated by a "#" character.</p> <p>The icon's array defines a list of the complete paths of image resources "*.bmp" or "*.ico".</p> <p>The size of the array of strings define the number of elements in the combo list and the default element is identified using the "&amp;" character as first; the syntax for these strings is :</p> <p style="text-align: center;">&amp;@Text#Icon_index</p> <p>Where :</p> <ul style="list-style-type: none"> <li>&amp; (optional) = default item</li> <li>@ (optional) = check box checked</li> <li>Text = string in the text cell</li> <li>Icon_index = 1-based index in the array of icon</li> </ul>
<b>&lt;action&gt;</b>	Command to take when a new row is selected.

For example:

Sample Code :

**Environment variables :**

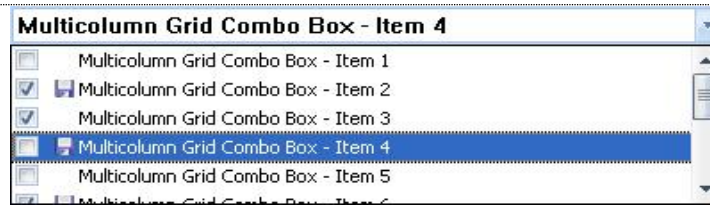
```
string combo[6]
combo [1]='Multicolum Grid Combo Box – Item 1'
combo [2]='@Multicolum Grid Combo Box – Item 2#1'
combo [3]='@Multicolum Grid Combo Box – Item 3'
combo [4]='&Multicolum Grid Combo Box – Item 4#1' # DEFAULT ITEM
combo [5]='Multicolum Grid Combo Box – Item 5'
combo [6]='@Multicolum Grid Combo Box – Item 6#'

string icons[2]
icons[1] = 'C:\MyRes\disk.bmp'
icons[2] = 'C:\Documents\flag.ico'
```

**Tab file :**

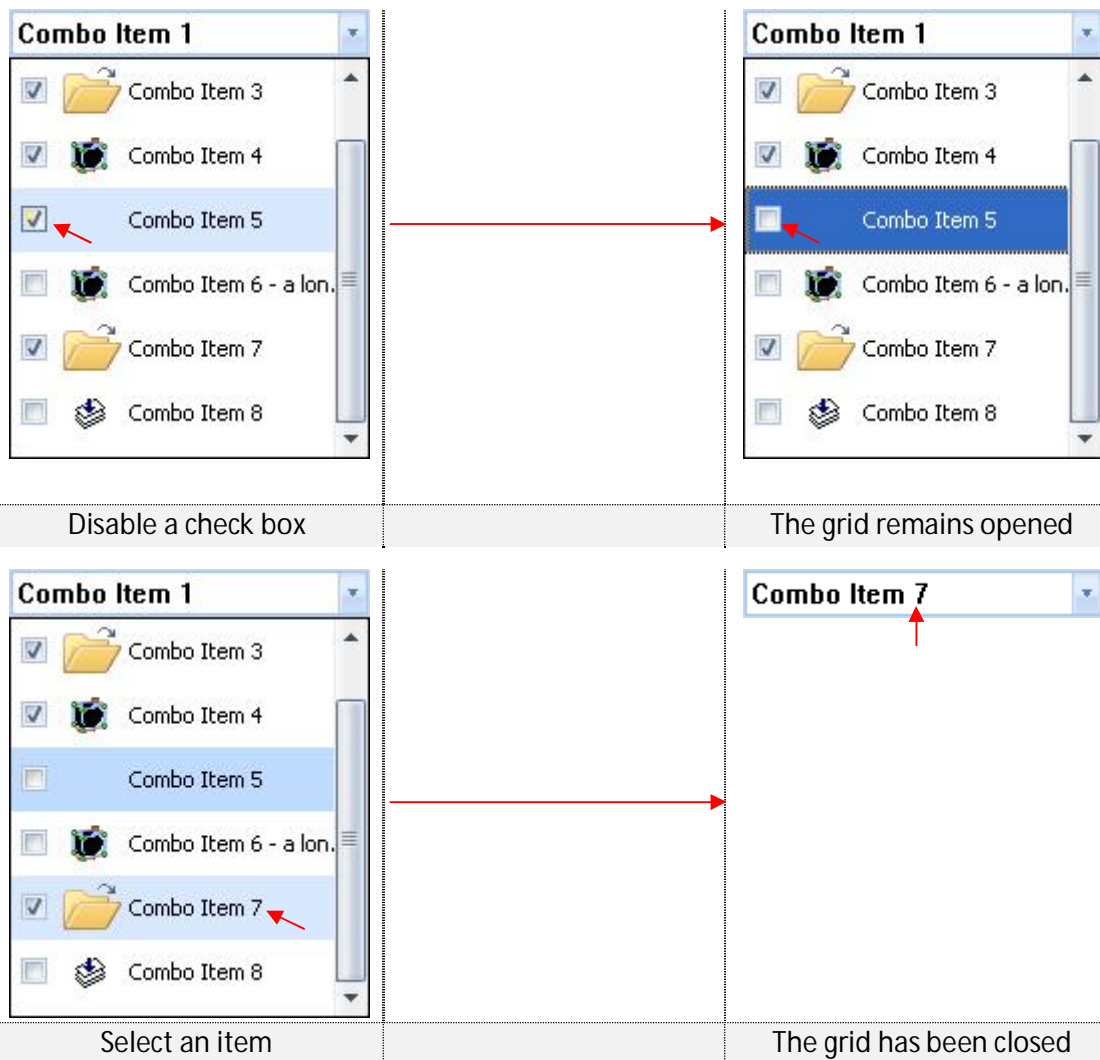
```
7,160,8,120,200,0,0,47,'combo#icons': tell '--- COMBO ---';
```

**Multicolumn Grid Combo Box :**



When the user selects an item in the grid the multi column grid combo box exhibits two different behavior patterns:

1. Clicking on a check box results in exiting from the "polling loop" but the grid remains visible;
- 2.
3. Clicking on text or an image results in exiting from the polling loop and the grid is hidden.



When a check box state is changed, the grid stays open, the polling returns the following values:

- MN = panel number;
- BN = button number;
- RN = index (1 based) of the item selected;
- LN = 0;
- ST = 1 if the check box is checked, otherwise -1;

Instead, if an image or a text is selected, the grid switches off and polling returns these values:

- MN = panel number;
- BN = button number;
- RN = index (1 based) of the item selected;
- LN = 1 if the check box is checked, otherwise -1;
- ST = string of the item selected.

### Type 57

This kind of multi column grid combo box contains two columns, the first with an image and another with text, The height and the width of the cells is calculated by considering the icons size and font height in the same manner as Type 47. In this grid combo the array of rows has the following syntax:

&Text#Icon\_index

where:

& (optional) = default item  
 Text = string in the text cell  
 Icon\_index = 1-based index in the array of icon.

Tab Syntax :

```
opt, x, y, w, h, 0, 0, 57, 'array_rows#array_icons': <action>
```

where:

Parameter	Description
opt	Index of the option.
x	X co-ordinate of the button in the menu grid.
y	Y co-ordinate of the button in the menu grid.
w	Width of the button in grid units.
h	Height of the list in grid units. This item sets the height of the popup list; if the size is not enough to display all the rows, a vertical scroll bar will be activated, else if the value is over-dimensioned the list height will be auto-fitted to maximum size needed.
f	Foreground color for buttons.

<b>b</b>	Background color for buttons.
<b>t = 57</b>	Button type = 57
<b>&lt;array_rows#array_icons&gt;</b>	<p>The field contains the name of an Eagle's array of strings and Icon's array, between single quotes and separated by a "#" character.</p> <p>The icon's array defines a list of the complete paths of image resources "*.bmp" or "*.ico".</p> <p>The size of the array of strings define the number of elements in the combo list and the default element is identified using the "&amp;" character as first; the syntax for these strings is :</p> <p style="text-align: center;">&amp;Text#Icon_index</p> <p>Where :</p> <p style="margin-left: 40px;">&amp; (optional) = default item</p> <p style="margin-left: 40px;">Text = string in the text cell</p> <p style="margin-left: 40px;">Icon_index = 1-based index in the array of icon</p>
<b>&lt;action&gt;</b>	Command to take when a new row is selected.

For example :

Sample Code :

**Environment variables :**

```
string multib2[8]
multib2[1]='Combo Item 1#1'
multib2[2]='Combo Item 2#2'
multib2[3]='Combo Item 3#3'
multib2[4]='Combo Item 4#1'
multib2[5]='Combo Item 5'
multib2[6]='Combo Item 6#1'
multib2[7]='&Combo Item 7#3'
multib2[8]='Combo Item 8#2'
```

```
string icons2[3]
icons2[1]='xse7.bmp'
icons2[2]='books.bmp'
icons2[3]='fm_open.bmp'
```

**Tab file :**

```
31,10,505,180,200,0,0,57,'multib2#icons2': tell '--- COMBO ---';
```

**Multicolumn Grid Combo Box :**



When an image or a text is selected, the grid switch off and polling returns these values :

- MN = panel number;
- BN = button number;
- RN = index (1 based) of the item selected;
- LN = 0;
- ST = string of the item selected.

#### Type 67

The multi column grid combo box type 67 contains two columns, the first with a check box and another with text> The height of rows depends only on the font size. The width of the text fields can be considered fixed because the check box column has a standard dimension.

In this case, the array of icons is not used and the array of rows has the following syntax :

&@Text

where :

- & (optional) = default item
- @ (optional) = check box checked, if not present the box is unchecked
- Text = string in the text cell.

Tab Syntax :

opt, x, y, w, h, 0 ,0, 67,'array\_rows': <action>

where:

Parameter	Description
opt	Index of the option.
x	X co-ordinate of the button in the menu grid.
y	Y co-ordinate of the button in the menu grid.
w	Width of the button in grid units.



<b>h</b>	Height of the list in grid units. This item sets the height of the popup list; if the size is not enough to display all the rows, a vertical scroll bar will be activated, else if the value is over-dimensioned the list height will be auto-fitted to maximum size needed.
<b>f</b>	Foreground color for buttons.
<b>b</b>	Background color for buttons.
<b>t = 67</b>	Button type = 67
<b>&lt;array_rows&gt;</b>	<p>The field contains the name of an Eagle's array of strings, between single quotes.</p> <p>The size of the array of strings define the number of elements in the combo list and the default element is identified using the "&amp;" character as first; the syntax for these strings is :</p> <p style="text-align: right;">&amp;@Text</p> <p>Where :</p> <ul style="list-style-type: none"> <li>&amp; (optional) = default item</li> <li>@ (optional) = check box checked</li> <li>Text = string in the text cell</li> </ul>
<b>&lt;action&gt;</b>	Command to take when a new row is selected.

For example :

Sample Code :

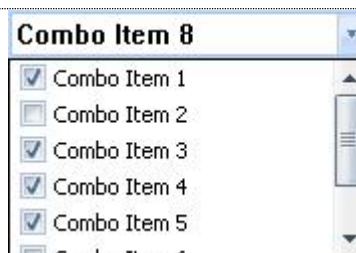
**Environment variables :**

```
string multib3[8]
multib3[1]='@Combo Item 1'
multib3[2]='Combo Item 2'
multib3[3]='@Combo Item 3'
multib3[4]='&@Combo Item 4'
multib3[5]='@Combo Item 5'
multib3[6]='Combo Item 6'
multib3[7]='@Combo Item 7'
multib3[8]='&Combo Item 8'
```

**Tab file :**

```
32,200,505,180,100,0,0,67,'multib3';;
```

**Multicolumn Grid Combo Box :**



When a check box state is changed, the grid stays open, the polling returns the following values :

- MN = panel number;
- BN = button number;
- RN = index (1 based) of the item selected;
- LN = 0;
- ST = 1 if the check box is checked, otherwise -1;

Instead, if a text cell is selected, the grid switch off and the polling returns these values :

- MN = panel number;
- BN = button number;
- RN = index (1 based) of the item selected;
- LN = 1 if the check box is checked, otherwise -1;
- ST = string of the item selected.

#### Type 127

The "insert" and "fetch" commands can be used to set and retrieve the value of an editable combo box type 127.

## 9.10 – Check Box and Three State Check Box

A check box (also called toggle box) is a graphical user interface widget that enables the user to emulate the (true / false) binary logic. Normally, check boxes are shown on screen as a square boxes that contain white space (for false) or a tick mark (for true), as pictured below. A caption describing the meaning of the check box is normally shown adjacent to the check box. Inverting the current state of a check box is achieved by clicking the mouse on the box, or the caption, or by using a keyboard shortcut, such as the space bar.

Just like previous Eagle versions, in V14 it is also possible to place check boxes in a panel using the button option defined as "8":

Tab Syntax :

```
opt, x, y, w, h, f, b, t, <text> : <action>
```

where:


Parameter	Description
<b>opt</b>	Index of the option.
<b>x</b>	X co-ordinate of the button in the menu grid.
<b>y</b>	Y co-ordinate of the button in the menu grid.
<b>w</b>	Width of the button in grid units.
<b>h</b>	Height of the button in grid units.
<b>f</b>	Foreground color for buttons.
<b>b</b>	Background color for buttons.
<b>t = 8</b>	Button type = 8
<b>&lt;text&gt;</b>	The text field to display near the box.
<b>&lt;action&gt;</b>	Command to take when the status has been changed.

For example:

Sample Code :

```
12,100,336,200,96,0,0,8,'toggle':;  
20,10,30,300,100,0,0,8,'Enable polling loop': do ploop;
```

The following table illustrates how check boxes are displayed in a selection of different themes:

Office 2000	Office 2003	Office 2007 Standard R1	Office 2007 Luna Blue R2
 toggle	 toggle	 toggle	 toggle
<pre>12,100,336,200,96,0,0,8,'toggle':;</pre>			

This component handles the event associated with the change of the status, from False to True (box is unchecked and becomes checked) and vice versa (box is checked and becomes

unchecked). When the status of this control is changed the following polling loop values are modified:

- MN = panel number;
- BN = button number;
- ST = 1 if the box is checked, -1 unchecked.

The “three state” check box is like a standard check box in appearance but it permits not only the checked and unchecked states but also allows for an “indeterminate” (or grayed out) condition:

	Unchecked	Checked	indeterminate
Office 2000	<input type="checkbox"/> 3-State Check Box	<input checked="" type="checkbox"/> 3-State Check Box	<input type="checkbox"/> 3-State Check Box
Office 2007 Luna Blue R2	<input type="checkbox"/> 3-State Check Box	<input checked="" type="checkbox"/> 3-State Check Box	<input type="checkbox"/> 3-State Check Box

To create this kind of control in a panel we need to use the option type numbered “18”, otherwise using the same syntax as a standard check box :

Tab Syntax :

```
opt, x, y, w, h, f, b, t, <text> : <action>
```

where:

Parameter	Description
<b>opt</b>	Index of the option.
<b>x</b>	X co-ordinate of the button in the menu grid.
<b>y</b>	Y co-ordinate of the button in the menu grid.
<b>w</b>	Width of the button in grid units.
<b>h</b>	Height of the button in grid units.
<b>f</b>	Foreground color for buttons.
<b>b</b>	Background color for buttons.
<b>t = 18</b>	Button type = 18
<b>&lt;text&gt;</b>	The text field to display near the box.
<b>&lt;action&gt;</b>	Command to take when the status has been changed.

For example:

Sample Code :

```
33,10,540,150,25,0,2,18,'3-State Check Box':;
```

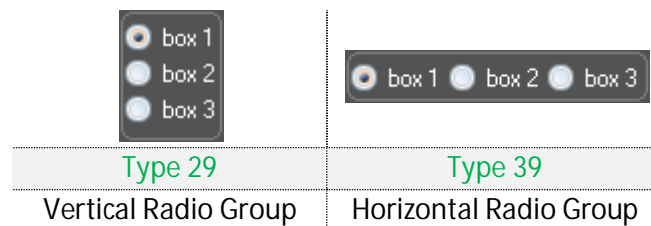
*The three state checkbox control has the same behavior as a check box, but instead when the status is changed the following polling loop values are modified:*

- MN = panel number;
- BN = button number;
- ST = 1 if the box is checked, 0 if grayed and -1 if unchecked.

## 9.11 – Radio Group

A “radio group” is a component with a behavior similar to the combo box. In this case the items are not contained in a list but are displayed using a specific graphic element called the “radio box”.

Eagle v.14 extends the previously available radio group component to allow creation of vertical (type “29”) and horizontal (type “39”) groups. This element is composed of a frame combined with a collection of selectable radio box items.



The prototype is quite similar to the combo box :

Tab Syntax :

`opt, x, y, w, h, f, b, t, <array_name> : <action>`

where:

Parameter	Description
<code>opt</code>	Index of the option.
<code>x</code>	X co-ordinate of the element in the menu grid.
<code>y</code>	Y co-ordinate of the element in the menu grid.
<code>w</code>	Width of the element in grid units.
<code>h</code>	Height of the element in grid units.
<code>f</code>	Foreground color for buttons.
<code>b</code>	Background color for buttons.
<code>t = 29, 39</code>	Button type = 29 for a vertical radio group, = 39 for an horizontal radio group
<code>&lt;array_name&gt;</code>	The array field contains the name of an Eagle's array of strings, between single quotes. The size of the array define the number of elements in the radio group and the default element is identified using the “&” character as first.
<code>&lt;action&gt;</code>	Command to take when a new box is selected.

For example :

Sample Code :

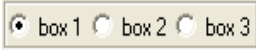
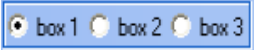
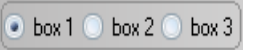
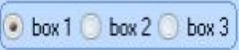
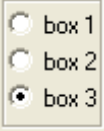
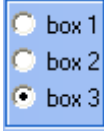
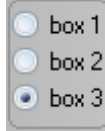
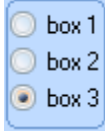
**Environment variables :**

```
string radio[3]
radio [1]='&box 1'      # DEFAULT ITEM
radio [2]='box 2'
radio [3]='box 3'
```

**Tab file :**

```
1,10,10,150,40,0,0,39,'radio': tell '--- RADIO H ---'
2,10,50,50,120,0,0,29,'radio': tell '--- RADIO V ---'
```

Note that for a radio group it's very important to correctly define the size and the position on the panel (using the x,y,w and h parameters). It is the developer's responsibility to adjust these settings to arrive at the proper correct appearance for the specific arrangement of elements. The next table presents some images to show how the combo boxes are painted in various themes:

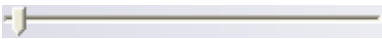
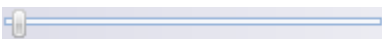
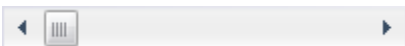
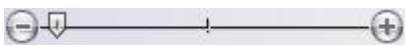
Office 2000	Office 2003	Office 2007 Standard R1	Office 2007 Luna Blue R2
			
			

Radio group handles events associated with changes to the currently selected item in the group and the relevant polling loop the values returned will be:

- MN = panel number;
- BN = button number;
- ST = index of the item selected.

## 9.12 – Slider, Scroll and Zoom Bar

This section describes the “bar” elements that are graphical objects created in a GUI to represent the movement between two directions or to set a value by moving an indicator, usually in a horizontal fashion. The user is also able to click on a point on the bar to change the setting. For this class of widget Eagle v.14 provides the following options:

Slider with arrow marker	
	type 10 – horizontal type 110 – vertical
Slider with indicator	
	type 210 – horizontal type 310 – vertical
Scroll Bar	
	type 410 – horizontal type 510 – vertical
Zoom Bar	
	type 610 – horizontal type 710 – vertical

All these components can be positioned either horizontally or vertically. Depending on the specific need, developers can choose the right element or elements for the application; For instance, if you need to control the size (scale) of a particular view the appropriate widget of this could be the zoom bar.

The syntax model is the same for each item requiring a string array to define the configuration. The array must be structured as follows:

```
array_name[1] = start value
array_name[2] = end value
array_name[3] = default value
array_name[4] = step for the increment(*)
```

(\*) the “step” is used only for Scroll Bar and Zoom Bar when the arrow and the plus/minus buttons are pressed. The increment for the thumb button is always 1.

The prototype for the option file is:

Tab Syntax :

```
opt, x, y, w, h, f, b, t, <array_name> : <action>
```

where:



Parameter	Description
<b>opt</b>	Index of the option.
<b>x</b>	X co-ordinate of the element in the menu grid.
<b>y</b>	Y co-ordinate of the element in the menu grid.
<b>w</b>	Width of the element in grid units.
<b>h</b>	Height of the element in grid units.
<b>f</b>	Foreground color for buttons.
<b>b</b>	Background color for buttons.
<b>t = 10, 110, 210, 310, 410, 510, 610, 710</b>	Button type
	= 10      Horizontal slider with arrow marker
	= 110     Vertical slider with arrow marker
	= 210     Horizontal slider with indicator
	= 310     Vertical slider with indicator
	= 410     Horizontal scroll bar
	= 510     Vertical slider scroll bar
	= 610     Horizontal zoom bar
	= 710     Vertical zoom bar
<b>&lt;array_name&gt;</b>	The array field contains the name of an Eagle array of strings, between single quotes.
	The array define the configuration of the bar, especially the start value, the end value and the default value.
<b>&lt;action&gt;</b>	Command to take when the marker's place has updated.

For example :

Sample Code :

**Environment variables :**


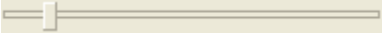
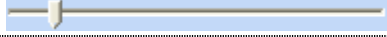





```
string config[4]
```

```
config[1] = 1
config [2] = 100
config [3] = 3
config [4] = 10
```

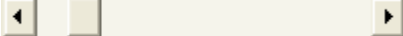

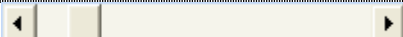

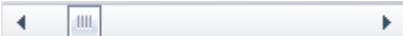

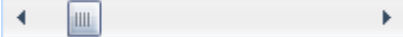

**Tab file :**

```
10,100,100,200,20,0,0,10,'config': tell 'A'
20,100,100,200,20,0,0,210, 'config': tell 'B'
30,100,150,200,20,0,0,410, 'config': tell 'C'
40,100,200,200,20,0,0,610, 'config': tell 'D'
```

Eagle v.14 offers two slider's types for which the only difference is the style of marker and bar. The "arrow marker" has the classical arrow as a pointer and a sunken bar shaped like Windows, whereas the new release has a rounded indicator and a "soft" bar.

	Slider with arrow marker	Slider with indicator
Office 2000		
Office 2003		
Office 2007 Standard R1		
Office 2007 Luna Blue R2		

In the same way, the "scroll bar" and "zoom bar" have a similar behavior, moving the locator between two extreme points. The table below shows how these bars are presented in different themes :

	Scroll Bar	Zoom Bar
Office 2000		
Office 2003		
Office 2007 Standard R1		
Office 2007 Luna Blue R2		

The "insert" and "fetch" commands can be used to set and retrieve the value of the current slider position.

The events associated with the movement of the marker for these controls means that when the position of the marker is changed the following values in the polling loop are modified:

- MN = panel number;
- BN = button number;
- ST = value related to the new position of the marker.

The scroll button action calls on every change of the cursor position, including during a continuous shift, but the polling loop will be never interrupted.



Next you can find a complete sample to implement a "Zoom Bar" :

Sample Code :

Create a zoom bar



```
string config[4]
config[1] = 1
config [2] = 100
```










	<pre>config [3] = 3 config [4] = 10 --- 40,100,200,200,20,0,0,610, 'config': tell 'D'</pre>
Press the "+" button to move the marker	
	
Verify the variable st with the new position	
	tell st
Output	
	//C/Program Files/myfile.cmd

### 9.13 – Spin Button and Spin Edit Button

A spin button is a control equipped with two buttons and each button displays an arrow. The spin button allows the user to navigate through a range of values using the arrow buttons to increase or decrease the value held by the control.

In Eagle V14 there are two categories of spin control, one composed only of arrows, and another with arrows and an edit field that displays the current value. In order to select the correct type we need to set the “t” parameter (control type) to one of the following values:

- 11 : Only Vertical Arrows;
- 111 : Only Horizontal Arrows;
- 211 : Edit and Vertical Arrow in the Right Side of the edit field;
- 311 : Edit and Horizontal Arrow in the Right Side of the edit field;

	Spin Vertical	Spin Horizontal	Spin Edit Vertical	Spin Edit Horizontal
Office 2000				
Office 2003				
Office 2007 Standard R1				
Office 2007 Luna Blue R2				

The developer is able to define the current value and range of permitted values (from minimum to maximum); The textual section in the “tab” option is used to set the parameters of the spin button with the following syntax:

`'current_value#minumum_value#maximum_value'`

The prototype for the option file is as follows:

Tab Syntax :

`opt, x, y, w, h, f, b, t, 'settings' : <action>`

where:

Parameter	Description
<b>opt</b>	Index of the option.
<b>x</b>	X co-ordinate of the element in the menu grid.
<b>y</b>	Y co-ordinate of the element in the menu grid.
<b>w</b>	Width of the element in grid units.
<b>h</b>	Height of the element in grid units.
<b>f</b>	Foreground color for buttons.
<b>b</b>	Background color for buttons.
<b>t = 11, 111,</b>	Button type

211, 311, 411, 511	= 11 : Only Vertical Arrows; = 111 : Only Horizontal Arrows; = 211 : Edit and Vertical Arrow in the Right Side of the edit field; = 311 : Edit and Horizontal Arrow in the Right Side of the edit field; = 411 : Edit and Vertical Arrow in the Left Side of the edit field; = 511 : Edit and Horizontal Arrow in the Left Side of the edit field.
'settings'	The setting field is a textual which contains the configuration for the control, expressed with the syntax: 'current_value#minumum_value#maximum_value' between single quotes.
<action>	Command to take when the marker's place has updated.

Note: accepted values are positive integers only.

For example:

Sample Code :

```
34,10,540,40,40,0,2,11,'1#0#10';;
35,60,540,40,40,0,2,111,'50#0#100';;
36,110,540,60,25,0,2,211,'30#10#50';;
37,170,540,60,25,0,2,311,'10#1#20';;
```

The "insert" and "fetch" commands can be used to set and retrieve the current value of the spin button.

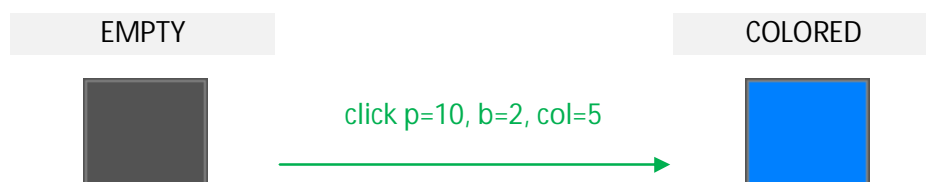
The events associated with the increasing or decreasing of the current value (i.e. pushing an arrow or writing a number in the edit field) results in the following values in the polling loop:

- MN = panel number;
- BN = button number;
- ST = value related to the current value of the spin control.
- RN = 1 if the arrow pressed is UP or RIGHT, -1 if DOWN or LEFT

## 9.14 – Frame Color

A “Frame Color” is an empty box that is only able to display a colored frame. This button doesn’t produce any event but it stores the information about a selected color shade inside the background of the frame.

In Eagle v.14 this type of button is identified with the option number “12”. When created the frame is empty and then the background color can be changed using the click command.



The prototype for the option file is:

Tab Syntax :

```
opt, x, y, w, h, f, b, t, "": ;
```

where:









Parameter	Description
<b>opt</b>	Index of the option.
<b>x</b>	X co-ordinate of the element in the menu grid.
<b>y</b>	Y co-ordinate of the element in the menu grid.
<b>w</b>	Width of the element in grid units.
<b>h</b>	Height of the element in grid units.
<b>f</b>	Foreground color for buttons.
<b>b</b>	Background color for buttons.
<b>t = 12</b>	Button type 12
<b>“”</b>	An empty string, required to maintain symmetry in the syntax of options function

For example :

Sample Code :




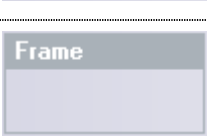
```
3,0,0,24,24,0,0,12,"";
```

There are no significant differences between the various themes, as we can see in the following table:

Office 2000	Office 2003	Office 2007 Standard R1	Office 2007 Luna Blue R2
			
			

## 9.15 – Frame Title

The “Frame Title” is a static border which can also have a title. Normally this element has used to group a collection of controls. In Eagle v.14 to create a frame with a title, the developer uses the option type “120” from which it is possible to chose a number of different styles:

Squared		#S (default)
Rounded		#R
Etched		#E
Caption		#C

The style of a frame is applied by extending the string for the title with one of the keyword triggers listed in the table above, namely #S, #R, #E or #C. If no keyword is specified, by default the style chosen is the squared one.

Tab Syntax :

*opt, x, y, w, h, f, b, t, <text>;*

where:

Parameter	Description
<i>opt</i>	Index of the option.
<i>x</i>	X co-ordinate of the element in the menu grid.
<i>y</i>	Y co-ordinate of the element in the menu grid.
<i>w</i>	Width of the element in grid units.
<i>h</i>	Height of the element in grid units.
<i>f</i>	Foreground color for buttons.
<i>b</i>	Background color for buttons.
<i>t = 120</i>	Button type 120
<i>text</i>	<p>This field defines the text displayed in the title and the frame’s style; the text must be specified between single quotes and it could have one of the following aspects :</p> <ul style="list-style-type: none"> <li>➤ ‘text’ = specified only the title, by default the style is the Squared</li> <li>➤ ‘text#S’ = specified the title and the Squared style</li> <li>➤ ‘text#R’ = specified the title and the Rounded style</li> <li>➤ ‘text#E’ = specified the title and the Etched style</li> </ul>





- 'text#C' = specified the title and the Caption style

For example :

Sample Code :

```
19,10,350,100,50,0,0,120,'Txxt;;
9,235,135,140,165,0,0,120,'Box#S';
10,20,20,100,100,0,0,120,'Options#R';
20,400,30,50,50,0,0,120,'Test#E';
11,10,100,200,200,0,0,120,'Frame#C';
```

Note that it is important to create the Frame Title after the creation of the controls that will reside inside the frame. If you create a Frame before the controls, then all the element inside the frame will be hidden behind the Frame, for instance :

Tab file	<pre>5,240,150,50,20,0,0,501,'my link': tell *** 12,250,170,100,50,0,0,8,'toggle'; 11,250,220,120,200,0,0,27,'array': tell '-C-' 21,250,260,64,32,0,0,1,'Python': 9,235,135,140,165,0,0,120,'Frame#S';</pre>	<pre>9,235,135,140,165,0,0,120,'Frame#S'; 5,240,150,50,20,0,0,501,'my link': tell *** 12,250,170,100,50,0,0,8,'toggle'; 11,250,220,120,200,0,0,27,'array': tell '-C-' 21,250,260,64,32,0,0,1,'Python':</pre>
Result		

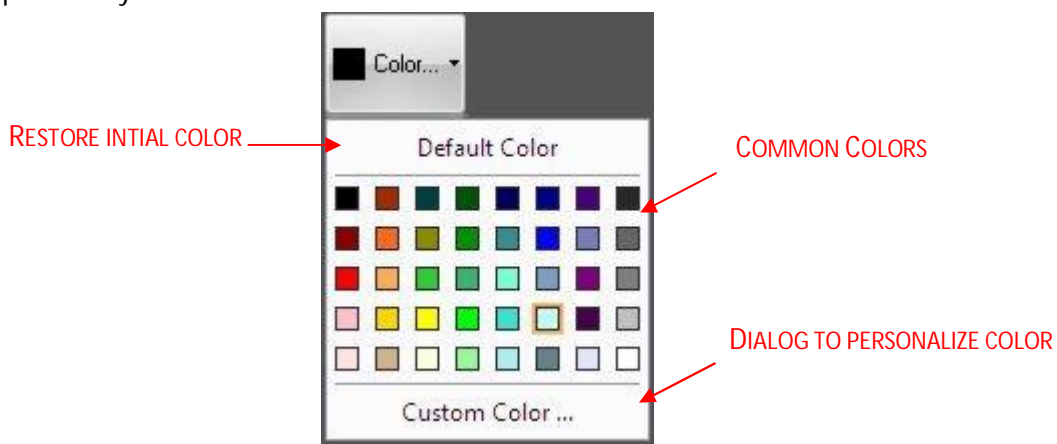
Close to the "Frame Color", also in this case there are no significant differences between the various themes.

## 9.16 – Color Picker

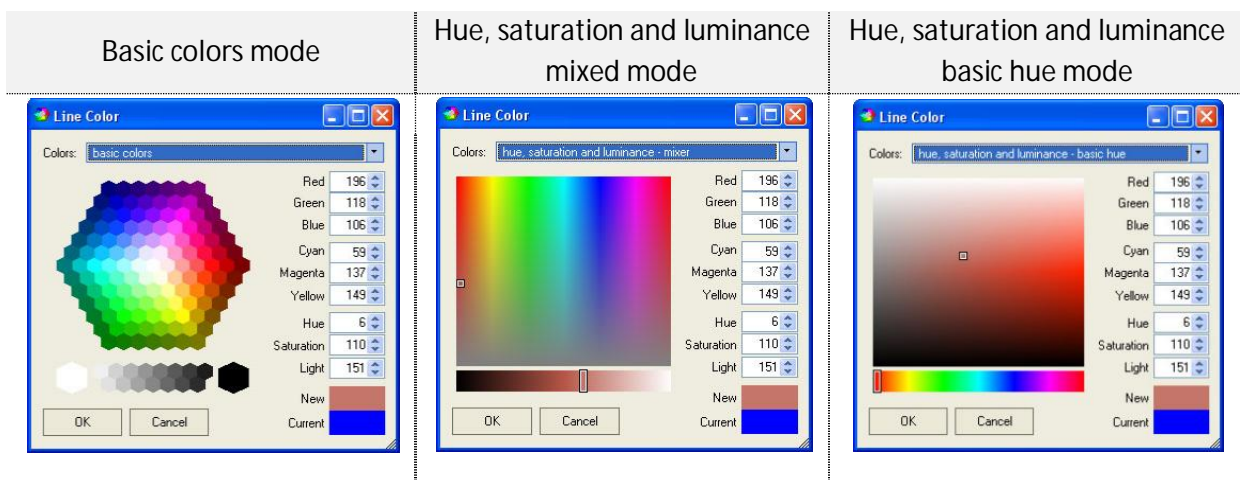
Eagle v.14 introduces the “Color Picker” button as part of the general program of extension of controls and improvements of how they are rendered. The color picker is a multi-option control that allows the user to select a color from a color palette or to define a specific RGB pattern.

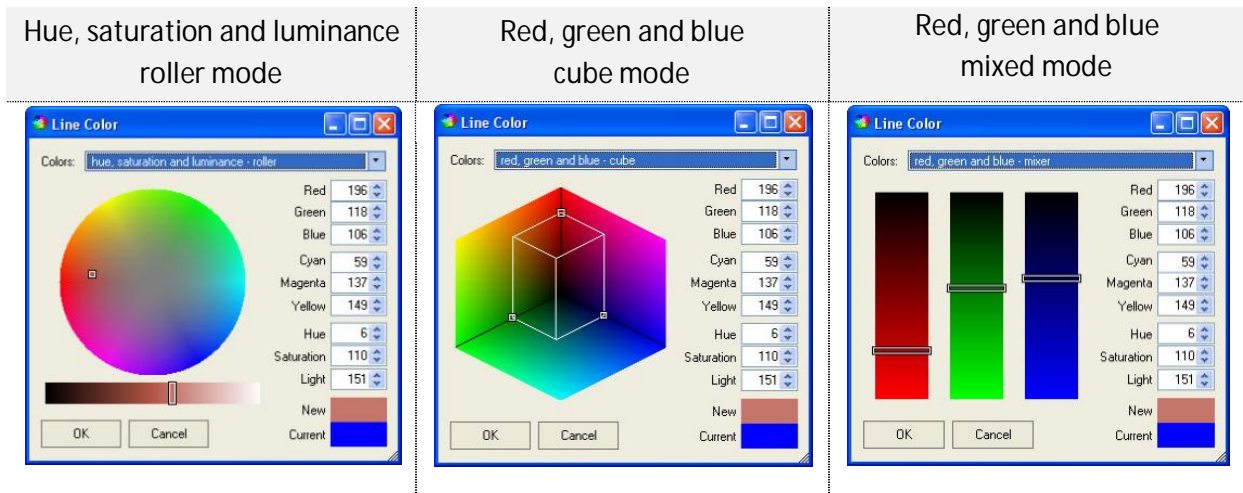


As shown in the figure above the “Color Picker” has a square which contains the current selected color and optionally a string. Every time the button is pressed a popup window appears as illustrated by the image below. The popup includes a button option to restore the previous color, a palette with the commonly used colors and a button to open a dialog to personalize the specific pattern layout.

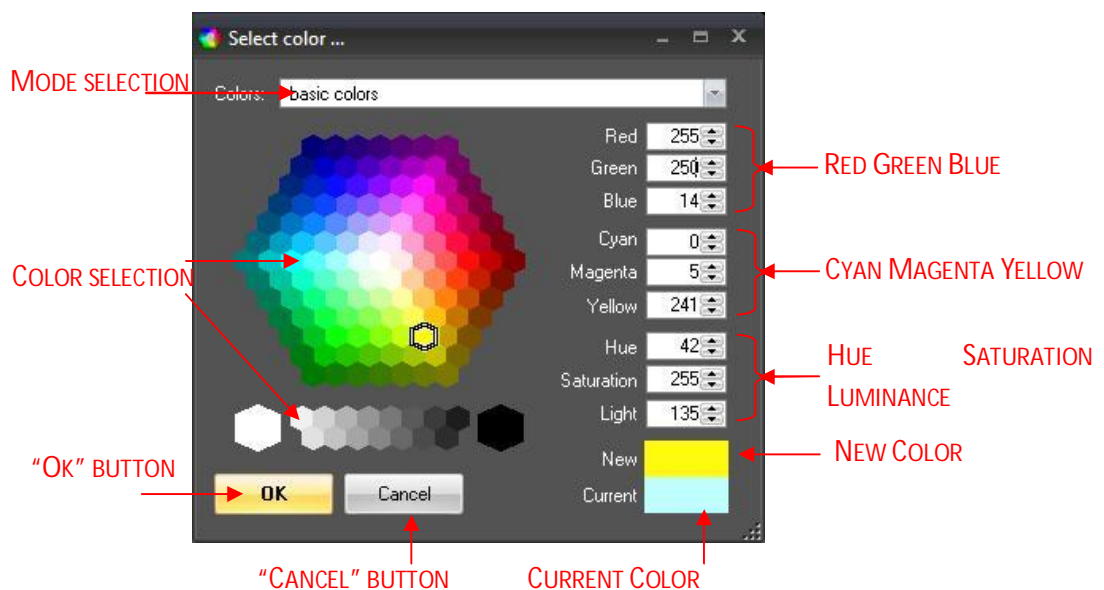


The text inside the popup menu or in the dialog depends from the language selected in the operating system. Eagle v.14 offers various modes from which it is possible to personalize a color pattern, each mode corresponds to a particular color scheme representation, for instance :





In addition the extended solutions shown above, namely the “personalization dialog” also allows direct specification of the red, green and blue components, the cyan, magenta and yellow parts or the hue, saturation and brightness values, as shown in the next image:



Tab Syntax :

opt, x, y, w, h, f, b, t, <text> : <action>

where:

Parameter	Description
opt	Index of the option.
x	X co-ordinate of the element in the menu grid.
y	Y co-ordinate of the element in the menu grid.
w	Width of the element in grid units.
h	Height of the element in grid units.

<b>f</b>	Foreground color for buttons.
<b>b</b>	Background color for buttons.
<b>t = 500</b>	Button type 500
<b>text</b>	(optional) Text displayed near the square with current color
<b>&lt;action&gt;</b>	Action performed when a new color is selected; if the color is not changed, no action will be executed.

For example :

Sample Code :	<code>6,70,60,70,50,0,0,500,'Color...': tell st</code>
---------------	--

These components handle the events associated with changing the selected color, so when the tint is modified the following values are set in the polling loop:

- MN panel number;
- BN button number;
- ST returns a string that contains the RGB components in Eagle coordinates and the position of the color in the Eagle palette, in the form : 'R,G,B,index'

Note, Eagle describes colors using the RGB normalized to 100, so when the user chooses a color, the RGB values will be transformed (from the range [0, 255] to the range [0,100]). In addition Eagle controls a color palette with a maximum size of 256 items (zero based), so if a picked color is not present in the list, the new tint will be added starting at the end of the list (from the position 255 descending), in other words the color at position number 255 will be substituted with the new RGB values and so on.


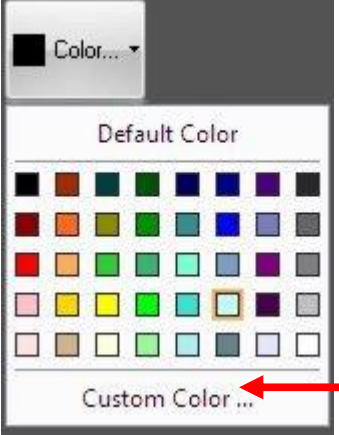
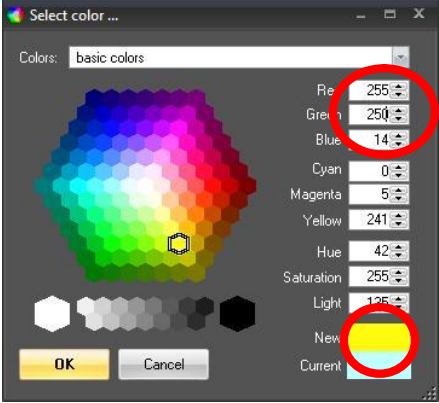

Therefore if a user selects another color which is not present in the palette, it will be then inserted at position 254, and so on for any further unassigned selections, for instance:

Color picked (normalized RGB)	Present in the palette	Position
34, 40, 50	No	255
60, 60, 10	Yes	84
23, 11, 80	No	254

Only the basic colors (lower index) cannot be updated and when all the possible changeable colors have been modified the procedure restarts from the last position again. This process ensures the most used colors will always be available in the Eagle palette.

Below there is a complete sample to illustrate how the "Color Picker" works:

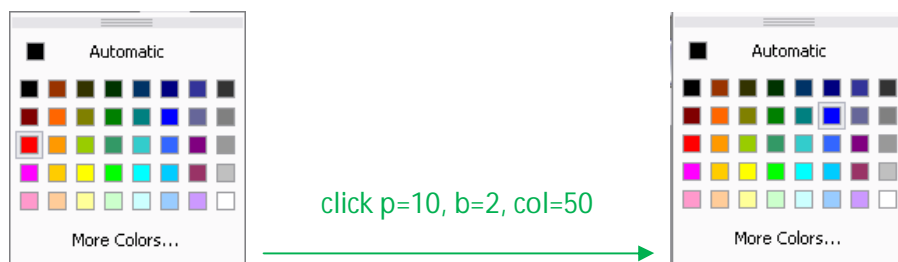
Sample Code:	<code>Create a color picker in a panel</code>
--------------	---

	<pre>6,70,60,70,50,0,0,500,'Color...': tell 'st'</pre>
Press the button and open the "Personalization" dialog	
	
Input a color with Red=150, Green=200, Blue=50 and click "Ok"	
	
Verify the variable st	
	<pre>tell st</pre>
Output	
	<pre>100,98,5,255</pre> <p>where :</p> <ul style="list-style-type: none"> <li>➤ 100 = red normalized coordinate</li> <li>➤ 98 = green normalized coordinate</li> <li>➤ 5 = blue normalized coordinate</li> <li>➤ 255 = index in the Eagle's color palette</li> </ul>

In Eagle v.14 the color can be changed using the click command.

EMPTY

COLORED



## 9.17 – Hyperlink

A hyperlink, usually shortened to the name link, is a label which is able to execute an action directly from selection just like a standard button. Eagle v.14 introduces this element with the option number "501".

The text is always aligned to the left aligned considering the width of the field and to center of the total height.

In the graphic interface the hyperlink appears like a standard label, but when the mouse pointer goes over the text, it is highlighted and underlined, so the user is aware that the label is selectable to execute a particular event.

my link

Mouse over link

my link

Tab Syntax :

`opt, x, y, w, h, f, b, t, <text> : <action>`

where:

Parameter	Description
<b>opt</b>	Index of the option.
<b>x</b>	X co-ordinate of the button in the menu grid.
<b>y</b>	Y co-ordinate of the button in the menu grid.
<b>w</b>	Width of the button in grid units.
<b>h</b>	Height of the button in grid units.
<b>f</b>	Foreground color for buttons.
<b>b</b>	Color for the font in the label
<b>t = 501</b>	Button type = 501
<b>&lt;text&gt;</b>	Message to display in the link.
<b>&lt;action&gt;</b>	Command to take when selected.

The next table shows how a hyperlink button is displayed in various themes (in normal status or highlighted status):

	Office 2000	Office 2003	Office 2007 Standard - R1	Office 2007 Luna Blue - R2
Normal	Tutorial...	Tutorial...	Tutorial...	Tutorial...
Highlighted	Tutorial...	Tutorial...	Tutorial...	Tutorial...

When a link is pressed, the polling loop can set the following values:

- MN = panel number;
- BN = button number.

The "f" parameter in the tab file is used to change the color of the text displayed in the hyperlink control by specifying the corresponding Eagle color palette index, for example:

```
6,190,70,220,20,0,2,501,'Text in an hyperlink!!!';
```

Text in an hyperlink!!!

```
6,190,70,220,20,0,4,501,'Text in an hyperlink!!!';
```

Text in an hyperlink!!!



## *9.18 – Progress Bar*

Future Data

## 9.19 – Rich Edit

A rich edit control is an object that resembles an edit box but can handle text that is formatted. This means that it is capable of displaying text with a variety of character formats and paragraphs with different alignments. A rich edit control can also allow a user to interactively make changes to both the formatting of characters and alignment of paragraphs.

Eagle V14 introduces the rich edit type as:

- Multi line rich edit → TYPE = 16

A rich edit component can be created as “read only” by simply specifying the hash “#” as first character of the text feature in the option declaration. Using “read only” settings it is possible to perform the “Cut” action in the component but cannot use the “Paste” action or insert characters in the text area.

The user can also load an RTF file to display a formatted document. Eagle automatically recognizes the presence of the RTF file parsing the “text” parameter in the option declaration.

V. 14.

```
*****
*** Eagle V14: Rich Edit control
*****

**** RTF FILE
****

-----
Date   File           Folder
-----
```

Multi Line Rich Edit with RTF file

The option types for this control is “16” (multi line) with the “tab description” as shown below to define settings:

Tab Syntax :

`opt, x, y, w, h, f, b, t, <text> : <action>`

where:

Parameter	Description
opt	Index of the option.
x	X co-ordinate of the button in the menu grid.

<b>y</b>	Y co-ordinate of the button in the menu grid.
<b>w</b>	Width of the button in grid units.
<b>h</b>	Height of the button in grid units.
<b>f</b>	Color for the edit field.
<b>b</b>	Color for the font in the edit
<b>t = 16</b>	Button type = 16 for a Multi Line rich edit
<b>&lt;text&gt;</b>	'#file.rtf' : this field setting is used as follows: <ul style="list-style-type: none"> <li>• # (optional) for a read only rich edit</li> <li>• RTF (Rich Text File) to display</li> </ul>
<b>&lt;action&gt;</b>	Command to initiate when the RETURN key is pressed.

For example :

Sample Code :

```
3,0,100,400,360,0,0,16,'mods.rtf': ;
```

The purpose of the rich edit control is to show text or a simple message to the users without having to fire off actions like key presses or mouse clicks, .for this reason the polling loop does not exit nor return any values.

Eagle V14 has extended the "insert" command to work also with the RTF files and the rich edit controls. New primers have been created to append or to substitute with an RTF file the content into a rich edit object:

Prototype :

```
insert p=<i>, b=<i>, rtf=<file>,append
```

where:

Parameter	Description
<b>p</b>	Indicates the number of the panel in which the button is present
<b>b</b>	The button number in the panel, starting from 1.
<b>rtf</b>	Text to be inserted, between single quotes.
<b>append</b>	Specify to append the file into the rich edit

## 9.20 – Tree View

A tree view is a graphical user interface element that presents a hierarchical view of information. Each item, often called a branch or a node, can have a number of sub-items visualized by indentation in a structured list which can be browsed, selected, drag and dropped, and customized. An item can be expanded to reveal sub-items, if any exist, and collapsed to hide sub-items.

Tree views are frequently seen in file manager applications, where they allow the user to navigate file system directories, or alternatively are used to present hierarchical data (also called a dataset).

Eagle enables creation of this type of widget control by using button type “300” in the tab file :

Tab Syntax :

```
opt, x, y, w, h, f, b, t, <text> ::
```

where:

Parameter	Description
<b>opt</b>	Index of the option.
<b>x</b>	X co-ordinate of the button in the menu grid.
<b>y</b>	Y co-ordinate of the button in the menu grid.
<b>w</b>	Width of the button in grid units.
<b>h</b>	Height of the button in grid units.
<b>f</b>	Foreground color for buttons.
<b>b</b>	Background color for buttons.
<b>t = 300</b>	Button type = 300
<b>&lt;text&gt;</b>	“dat” file which defines the tree’s node (i.e. tree.dat)

A file with the “dat” extension must be used in the text field of the option to define the structure of the tree view. The dat file which contains raw data for each tree item presented in the control (root, node or leaf) is defined with the following format:

Dat Syntax :

```
Node id , Parent node id , Text , Action, Menu, Icon, Drag & Drop, UID
```

where:

Parameter	Description
<b>node id</b>	Identifier of the node
<b>parent node id</b>	Identifier of the parent to which the node is connected. If the node is a root, the value is equal to 0.
<b>text</b>	Standard node - Text displayed on the right of the node icon. Label

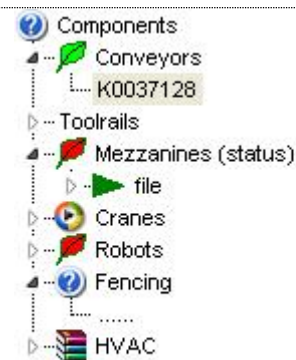
	<p>Node with radio - Radio group to which the node belongs and text            GroupName#Label</p> <p>Node with extra info - One of the previous syntaxes plus the extra info            Label{ExtraInfo}            GroupName#Label{ExtraInfo}</p>
<b>action</b>	An Eagle command which is to be executed when the particular node is selected. The relevant selection method which triggers this action is defined by the the rule set by the TREEVIEW_BEHAVIOUR variable in the Registry/INI.
<b>menu</b>	This field represents the context menu for the node shown when the mouse right click is initiated, using which the user can define a popup menu file (extension ".men") or run a macro (extension ".cmd"). This option is enabled by setting the TREEVIEW_RIGHTCLICK variable to yes in the Registry/INI file.
<b>icon</b>	<p>Represents a bitmap file to display an icon for the node. The the bitmap supports the alpha color, so it is possible to have a transparent bitmap using the color 255, 0, 255.</p> <p>It is also possible to define two separate images, one representing the collapsed state and another for expanded state. Where two files are required the files are separated by a hash #:            collapse.bmp#expanded.bmp</p>
<b>Drag &amp; Drop</b>	An Eagle command action which will be executed when a node is dragged and dropped on the Eagle graphic window.
<b>UID</b>	<p>Unique Identifier for the node. You need to trigger the use of this identifier by setting the environment variable TREEVIEW_ST_AS_UID=yes</p> <p>The UID code is a string of maximum 16 characters length, can be used to "select" a node using the CLICK command. The default is "no" meaning the extra field in the input file is not required.</p>

For example:

<b>Sample Code :</b>	
	<b>TAB file</b>
	<pre>0,0,0,14,0,0,0,888,'Arial'; 1,0,0,200,400,0,0,300,'components.dat';</pre>
	<b>DAT file</b>
	<pre>1,0,Components,-,-,-icon/help.bmp,vane,U4001 2,1,Conveyors,do checkpoll,cap1.cmd,-icon/green_leaf.bmp,-,U4002 3,2,K0037128,-,-,-icon/loadcomp('K0037128'),U4003 4,1,Toolrails,-,-,-,U4004 5,4,.....,-,-,-,U4005 6,1,Mezzanines (status),-,-,-icon/red_leaf.bmp,-,U4006</pre>

```
7,6,file,-,blue.cmd,-icon/file.bmp,components.dat,U4007
8,7,Strucure steel,vane,-,-icon/fish.bmp,-,U4008
9,8,.....,-,-,-,U4009
10,9,Catwalk_Overhead walkway,-,-,-icon/people.bmp,-,U4010
11,10,.....,-,-,-,U4011
12,1,Cranes,-,-,-icon/mmplayer.bmp,-,U4012
13,12,Circle,-,-,-icon/notes.bmp,-,U4013
14,1,Robots,-,-,-icon/red_leaf.bmp,-,U4014
15,14,.....,-,-,-,U4015
16,1,Fencing,-,-,-icon/help.bmp,-,U4016
17,16,.....,-,-,-,U4017
18,1,HVAC,-,-,-icon/books.bmp,-,U4018
```

### Tree View



The new UI Framework version of the Tree View differs in a positive way from earlier Eagle version implementations because it overcomes a number of limitations and is also extended with several new features, for instance:

1. There can be more than one tree view control per panel;
2. The button number for a tree view control can be different from 1 (one);
3. An unlimited number of nodes per tree-view control is possible;
4. It is possible to have different types of child nodes (label, radio and check node);
5. The icon placement is now in the real node (near the text) and not in the tree structure;
6. Users can delete or insert nodes and sub-nodes;
7. The tree view can have more than one root;
8. It is possible to insert a sub-tree view inside an existing tree view;
9. The widget is painted using the currently selected theme;
10. The node selection in the click command can be performed through UID, Node ID and Node Path;

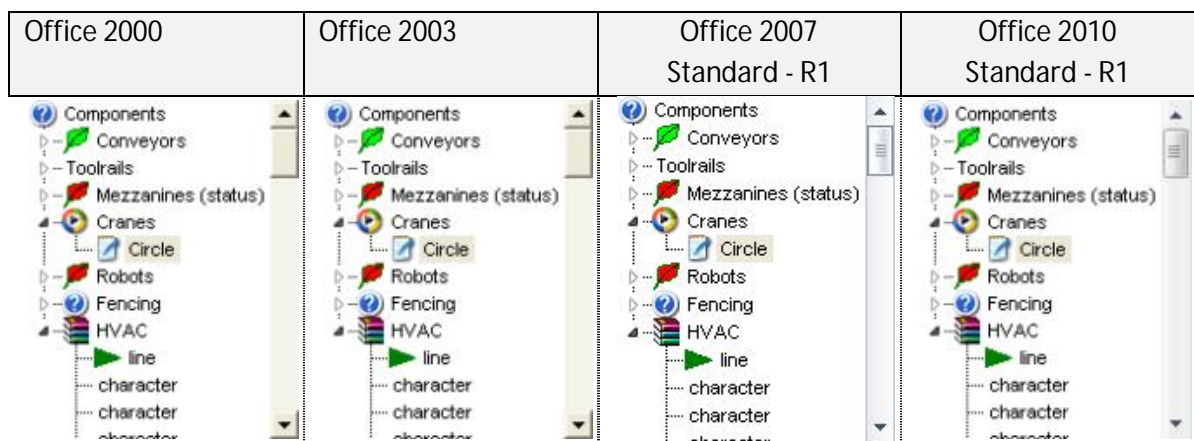
11. The style of tree view lines and boxes can be customized.

Eagle Version 14 tree view accepts several types of item:

- root node;
- folder node (with or without icon);
- leaf node standard (only label);
- leaf node with icon;
- leaf node with check box;
- leaf node with check box three state;
- leaf node with radio box that belongs to a specific radio group

---ADD IMAGE TABLE---

The tree view control maintains UI consistency of appearance in that like other buttons in Eagle V14 its appearance is painted using currently selected theme, for example:



When defining the tree object it's possible to specify both background and foreground colors using the indices of the Eagle color map.

The default values are, white for the background and black for the foreground. If colors have been defined with the same color index the background assumes the default color (white).

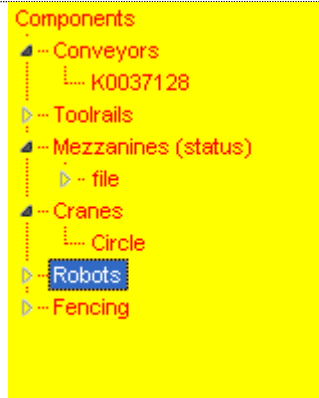
Now in this version the assigned foreground color also influences the tree structure, meaning lines, boxes and fonts in the tree will have the same color as is illustrated in the next example and image:

Sample Code :

TAB file

```
0,0,0,14,0,0,0,888,'Arial';  
1,0,0,100,200,6,2,300,'string.dat';
```

Tree View with Yellow background and Red foreground



### 9.20.1 Treeview Appearance Configuration

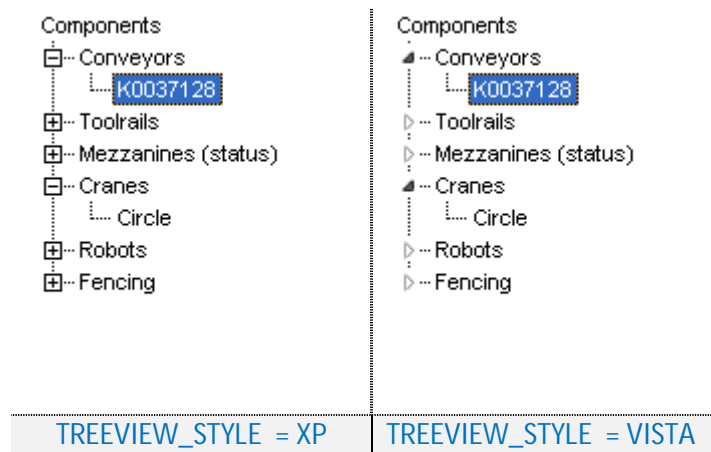
The tree view characteristics can be customized in several ways using configuration variables. The first of which is the "TREEVIEW\_STYLE" variable which regulates the skin style for expanded/collapsed buttons.

INI Sample :

```
TREEVIEW_STYLE =Vista
```

The TREEVIEW\_STYLE option provided the choice of presenting the Treeview with the "Windows Xp" style (classical boxes near the folder) or "Windows Vista" style (new style box images); the default style is Windows Xp.





Depending on which TREEVIEW\_STYLE is selected it is also possible to further customize the boxes used in the tree view by using the TREEVIEW\_BOX configuration entry. Using this entry it is possible to select one of three different types of box appearance:

- ROUND : (only for XP skin) a round box displayed before the folders;
- SQUARE : (only for XP skin) a square box displayed before the folders (default);
- BITMAP : the images used for boxes are user defined using the following additional environment variables:
  - TREEVIEW\_BOX\_IMAGE\_COLLAPSED : image file that specifies the picture to display for a collapsed node;
  - TREEVIEW\_BOX\_IMAGE\_COLLAPSED\_HOVER : image file that specifies the picture to display when the mouse pointer is hovered over a collapsed node;
  - TREEVIEW\_BOX\_IMAGE\_EXPANDED : image file that specifies the picture to display for an expanded node;
  - TREEVIEW\_BOX\_IMAGE\_EXPANDED\_HOVER : image file that specifies the picture to display when the mouse pointer is hovered over an expanded node.

When the skin TREEVIEW\_STYLE is "Vista", the default forced box style is the "triangle" box, so the "ROUND" and "SQUARE" modes have no effect.

INI Sample :

```
TREEVIEW_BOX = Round
TREEVIEW_BOX_IMAGE_COLLAPSED = c:\images\cfolder.bmp
TREEVIEW_BOX_IMAGE_COLLAPSED_HOVER = c:\images\chfolder.bmp
TREEVIEW_BOX_IMAGE_EXPANDED = c:\images\efolder.bmp
TREEVIEW_BOX_IMAGE_EXPANDED_HOVER = c:\images\ehfolder.bmp
```

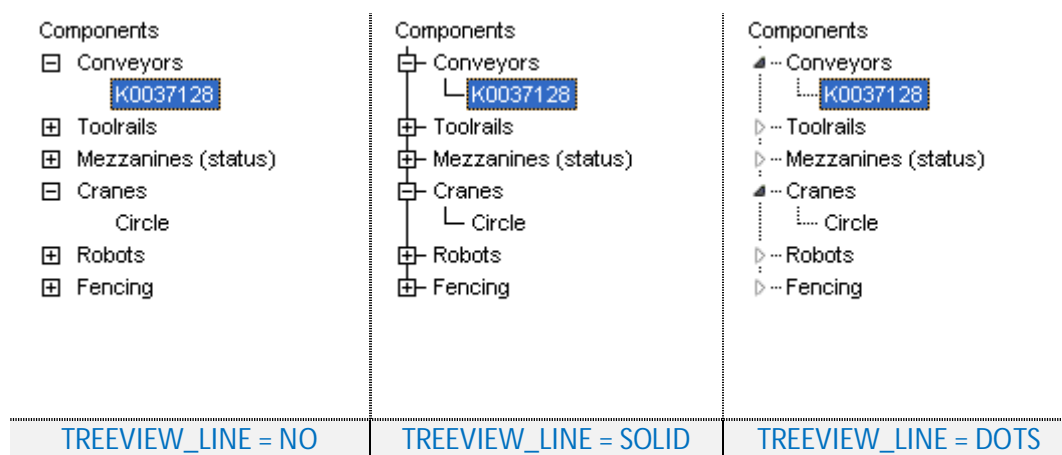
The TREEVIEW\_LINE setting in the configuration file enables changing of the shape of lines that connect the tree view nodes. Using this variable it is possible to choose from three different line styles

- NO : no line is drawn between nodes;
- SOLID : a solid line is drawn between nodes;
- DOTS : a dotted line is drawn between nodes (default).

INI Sample :

TREEVIEW\_LINE = No

For example:



## 9.20.2 Treeview Polling Information and Behaviour

Just like previous versions, the tree view permits different node selection behavior types. The choice of node selection is driven by an environment variable called TREEVIEW\_BEHAVIOUR. The behavior attributed to the value of this variable represents one of the three scenarios described below:

=1

single-click on a folder expands the sub-tree but does not exit from the polling loop; no action is executed when a folder is selected;  
double-click on a folder behaves like a single-click;  
single-click on a leaf execute the associated action, if any and exits from the polling loop;  
double-click on a leaf behaves like a single-click; exiting from the polling loop, the name of the leaf is stored in the system variable ST and the full path to the leaf is stored in the polling output string;

=2	single-click on a folder does not expand the sub-tree, but exits from the polling loop as a single-click on a leaf, this means setting the system variable ST to the name of the folder and the polling output string to the full path to the folder with a separator at the end of the string (e.g. doctor/evil/); single-click on a leaf behaves like in scenario 1, where the polling output string contains the full path to the leaf without a separator at the end of the string (e.g. doctor/evil/alanparsonproject); double-click on a folder expand the sub-tree and exit from the polling loop; double-click on a leaf behaves like in scenario 1;
=3	same as 2 but the action is executed also when a folder is selected;

The default value is TREEVIEW\_BEHAVIOUR=1.

INI Sample :

```
TREEVIEW_BEHAVIOUR=1
```

When a node is pressed with a single or double click, the polling loop returns the following values :

- WT = type of the control (a panel);
- ST = the label (with extra info, see the variable TREEVIEW\_EXTRAINFO\_SEPARATOR) associated to the selected leaf node or the user definable node id UID if the variable TREEVIEW\_ST\_AS\_UID is set to 'yes';
- OS = the full path from the root;
- MN = the panel number;
- BN = the button number;
- LN = the node ID;
- RN = 0 if no children, 1 otherwise.

---POLLING FOR CHECK, RADIO AND 3STATE---

Usually, during the polling loop the ST value returns the text displayed in the node's label, however two INI settings permit modification of the ST parameter returned value. When the environment variable called TREEVIEW\_ST\_AS\_UID is defined to "yes" (default is "no"), the ST contains the node UID instead the label of the selected node.

INI Sample :

```
TREEVIEW_ST_AS_UID=yes
```

The entry TREEVIEW\_EXTRINFO\_SEPARATOR gives the possibility to return from the polling loop ST parameter extra information that is not displayed in the label of the node. This environment variable is defined using two characters which delimit the extra information provided (i.e. `TREEVIEW_EXTRINFO_SEPARATOR = { }`). So, if the label field in the "DAT" file contains these characters, the text inside the characters represents the extra information (i.e. `18,1,HVAC{my info},-,-,icon/books.bmp,-,U4018`). In this case Eagle does not display the included extra info in the node but returns everything in the ST parameter when the node has been selected (i.e. `HVAC` in the node and `HVAC{my info}` in the ST value).

INI Sample :

```
TREEVIEW_EXTRINFO_SEPARATOR = { }
```

The default values for TREEVIEW\_EXTRINFO\_SEPARATOR are the curly brackets. Naturally and obviously, if the TREEVIEW\_ST\_AS\_UID is set to "yes" then it is not possible to see the extra information in the ST parameter.

The environment variable called TREEVIEW\_PATH\_SEPARATOR defines the separator character for the path string returned in the POLLING output string when clicking on a leaf of a tree-view control. The default value is the "/" (slash) character.

INI Sample :

```
TREEVIEW_PATH_SEPARATOR = |
```

### 9.20.3 Treeview Drag & Drop

Treeview Drag & Drop behavior is the action that occurs when a file (node) is selected and dragged from the tree onto the Eagle graphics window. IN order to enable this option we have to set the TREEVIEW\_DRAGNDROP variable in the Registry/INI to yes (default setting is no).

INI Sample :

```
TREEVIEW_DRAGNDROP = yes
```

The called macro can then use the following environment variables to recover information about the dropped object and dropped position in the graphics environment:

- D&D\_FILENAME – contains the content of the Drag & Drop field.
- D&D\_3DPOINT – contains the 3D coordinates of the point where the drop operation has been made.

- D&D\_2DPOINT - same as 3DPOINT the difference being that this is the 2D coordinates where the drop has been made. This is useful when the graphic view is orthogonal.

Note that now if the Drag & Drop is enabled but the action is not defined in the dat file, the Drag & Drop cursor is not displayed. This acts as a notification to the user that the action has not been defined for the current item.

## 9.20.4 Treeview Node Right Click

As we have seen previously, the "DAT" file may also include the option to display right click context popup menu related to a specific node. This option is enabled by defining the environment variable TREEVIEW\_RIGHTCLICK to "yes" in the configuration INI settings.

INI Sample :

```
TREEVIEW_RIGHTCLICK = yes
```

*Note: The other environment variables in the INI file, used to set the tree view in the previous Eagle's versions are now obsolete.*

## 9.20.5 Treeview CLICK Behaviour

In the new UI Framework tree view, the "click" command functionalities have been improved to extend the selection approach. Now we can select an item just to open/close a branch or to modify the attributes of a node in three ways:

1. Passing the complete PATH: the selection is made utilizing the complete path, that is, from the root to the desired item (folder or leaf). This value must terminate with a tree-view terminator character (e.g.: a slash), so the format is the same as that returned by the interactive selection of a node in the polling string.

The "select" option in the click command has the following syntax:

Dat Syntax :

```
select = 'root/path1/...
```

2. Using the NODE ID: the selection is made using directly the node identifier, that is, the first value used to define the item in the dat file.

The string in the "select" option needs a \$ character before the Node Id:

Dat Syntax :

```
select = '$id'
```

3. Specifying the UID: the selection is made using the UID if it exists, that is, the last optional value defined for the item in the dat file.

The string in the "select" option needs a % character before the UID:

Dat Syntax :

```
select = '%UID'
```

For example:

Sample Code :

```
click p=88, b=1, select='Components/Conveyors/'
click p=88, b=1, select='$6'
click p=88, b=1, select='%U007'
```

The actions that may use the mechanisms are those listed below:

1. Selection of a node that performs the action;
2. Silent selection that not performs the action;
3. Change of label;
4. Image substitution;
5. Replacement of a popup menu;
6. Change of an action;
7. Redefinition of a Drag & Drop action.

Consequently, the syntax of the "click" command in the case of a tree view is as follows:

Prototype :

```
click p=<i>, b=<i>, select='<path>'|'%<uid>'|'$<id>, {silent},
      {label='text'}, {image=text}, {menu='text'}, {action='text'},
      {dnd='text'},
```

where:

Parameter	Description
<b>p</b>	Number of the panel in which the tree view is located.
<b>b</b>	The tree view identifier in the panel.
<b>select='&lt;path&gt;'</b>	Selection of a branch of a tree-view control from a macro.
<b>select='%UID'</b>	The string provided represents:

<code>select='\$id'</code>	<ul style="list-style-type: none"> <li>➤ path from the root to the node value terminated with a tree-view terminator character;</li> <li>➤ the UID preceded by a %;</li> <li>➤ the Node Id preceded by a \$.</li> </ul>
<code>silent</code>	Perform a node selection without executing the action related to the selected node.
<code>label='text'</code>	String that contains the new label for a node
<code>image=file1.bmp#file2.bmp</code> <code>image=file.bmp</code>	(# Note no quotes) <ul style="list-style-type: none"> <li>➤ One file : a single bitmap as image of the node</li> <li>➤ Two files separated by a # : the first file is the bitmap for the collapsed state and the second is the one for the expanded state.</li> </ul>
<code>menu='file.men'</code> <code>menu='file.cmd'</code>	Popup menu that is displayed using the mouse right click over a node; it is possible to insert a menu file (.men) or a command file (.cmd).
<code>action='file.cmd'</code>	Action to execute when the node is selected.
<code>dnd='file.cmd'</code>	Command executed when the item is dragged onto the graphics area.

For example:

Sample Code :	<pre> click p=1, b=1, select='root/dir1/dir2/' click p=j, b=k, select = '%4567', label='David' click p=j, b=k, select = '\$27', label='Juventus', image=Scudetto.bmp click p=j, b=k, select = '%4567', action='do @fitOnPage' click p=j, b=k, select = '%4567', image =lb2\books.bmp </pre>
---------------	---

### 9.20.6 Treeview Insert

Using the "insert" command in V14 it is possible to dynamically add or delete items at run-time without having to create an entirely new tree. The syntax of the insert command is as follows:

Prototype :	<pre> insert    p=&lt;i&gt;,    b=&lt;i&gt;,    delnode=&lt;id&gt;         addnode='&lt;item&gt;'                addtree=file.dat </pre>
-------------	--

where:

Parameter	Description
<code>p</code>	Number of the panel in which the tree view is located.

<b>b</b>	The tree view identifier in the panel.
<b>delnode=&lt;id&gt;</b>	Node id of the item to delete. If the node has sub-items, also the sub-node will be removed.
<b>addnode='&lt;item&gt;'</b>	String that defines the new node to append in the tree view; the syntax is the same seen in the ".dat" file.
<b>addtree=file.dat</b>	Filename with extension ".dat" to append a new tree view at the end of the current tree.

### 9.20.7 Treeview Freezing and Unfreezing

Sometimes we need to temporarily enable or disable a node or a branch. The "freeze" and "unfreeze" commands enable us to perform these functions. Naturally if an item has sub-nodes, then freezing or unfreezing a node means that every element in the branch will be enabled or disabled.

The syntax is the same for both the instructions:

Prototype :	<b>freeze</b> <b>p=&lt;i&gt;, b=&lt;i&gt;, node=&lt;id&gt;</b>
-------------	--

and

Prototype :	<b>unfreeze</b> <b>p=&lt;i&gt;, b=&lt;i&gt;, node=&lt;id&gt;</b>
-------------	--

where:

Parameter	Description
<b>p</b>	Number of the panel in which the tree view is located.
<b>b</b>	The tree view identifier in the panel.
<b>node=&lt;id&gt;</b>	Node id of the item to freeze or unfreeze. If the node has sub-items, also the sub-node will be enabled or disabled.



## 9.21 – List View

Lists are a collection of items, usually strings, which can have one or more columns. Eagle permits creation a list view control to visualize lists with vertical layout. This organization works considering data sets as rows (or lines) and not columns.

In the previous Eagle versions was possible to create a list as an independent control (that is, in a floating dialog) or as a child of a panel; now, with the new user interface framework, the first approach has been depreciated so that now only the second option remains valid.

Note that if the panel is deleted, then any lists owned by the panel will also be deleted.

Lists differ from other controls in that, the instruction used to insert a list object into a panel is the "list" command and not the more frequently used "tab" file. The relevant syntax is defines as follows:

Prototype :	
	list {on off}, w=<i>, vc=<i>, vr=<i> {f=<file>} {pos=<i>}, {add='<text>'} del} {pan=<i>, bx=<i>, by=<i>} {typ=<i>}
	or
	list w=<i>, l=<file>

where:





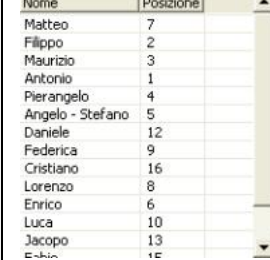
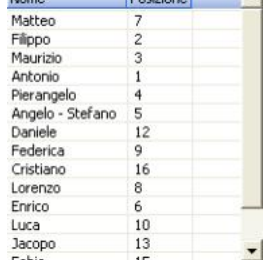
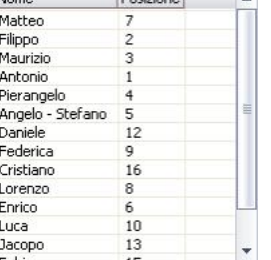

Parameter	Description
on	(default) Create the list
off	Destroy the list
w	List index, in the range 1 to 10
vc	The number of visible columns
vr	The number of visible rows (maximum 128)
f	Data file name, in quotes
l	Output file name, in quotes, to store data after a multiple selection. There is no default for the file name extension
pos	Position (line index, 1-based) in which to add (ADD) a new line, delete (DEL) or select (SEL) an existing line; note that: <ul style="list-style-type: none"> <li>• Add a line with pos = 0, the new line will appended at the bottom;</li> <li>• Delete with pos = 0, all lines will be removed</li> <li>• Add a line with pos = 1, the new line will be inserted at the top</li> <li>• Delete with pos = 1, the first line will be removed</li> </ul>
add='text'	Text for a new line
del	Delete a line
pan	The panel number (id) to which the list belongs
bx	The x position of the list in the panel
by	The y position of the list in the panel
typ	Display the content of a list box with different columns. Set typ=0 for single column or typ=1 for multi-column

For example :

Sample Code :

```
list on,w=1,vc=46,vr=24,f='dir.dat',t='directory ^1', bx=0,by=0,tpy=0
list w=2,off list w=1,pos=4,add='a new line'
list w=1,pos=4,del
list w=1,f='newdir.dat'
list w=1,vc=32,vr=12,f='data.dat',pan=1,bx=0,by=12,tpy=1
list w=3,on,vr=10,vc=10,f='report.dat',ok
list w=3,l='select.dat'
list w=1,pos =<line>
```

The appearance of the new list, like other UI Framework controls, is influenced by the current theme. The control has a number of visible rows, visible columns and horizontal and vertical scroll bars; If the list size (VC x VR) is smaller than that required, scrollbars will be automatically added.

	Office 2000	Office 2003	Office 2007 Standard R1	Office 2007 Luna Blue R2
Single Column				
Multiple Column				

Two different types of line selection are available, single line selection using a left button click and multiple line selection using a left click combined with the shift key to select or the ctrl button to unselect an entry or entries. In the first case the line is stored in a '<text>' string declared in the POLLING command and in the second case, data lines are stored in an output file specified by the l= primer in a separate LIST command, so the l=<file> primer cannot be used in the statement defining the LIST widget. To use this primer, another instance of the LIST command must be initiated.

When the left button of the mouse is clicked the following values are set in the polling loop:

- MN panel number;
- LN (and BN) list identifier;
- RN index of the row selected;
- OS (and ST) returns a string that contains the selected line.

It is also possible to update the contents of a list by specifying the existing list and the new file using the f= primer.

### 9.21.1 – Single column

The single column list represents the simplest case of a list as it is composed only of a text field and scroll bars if required.

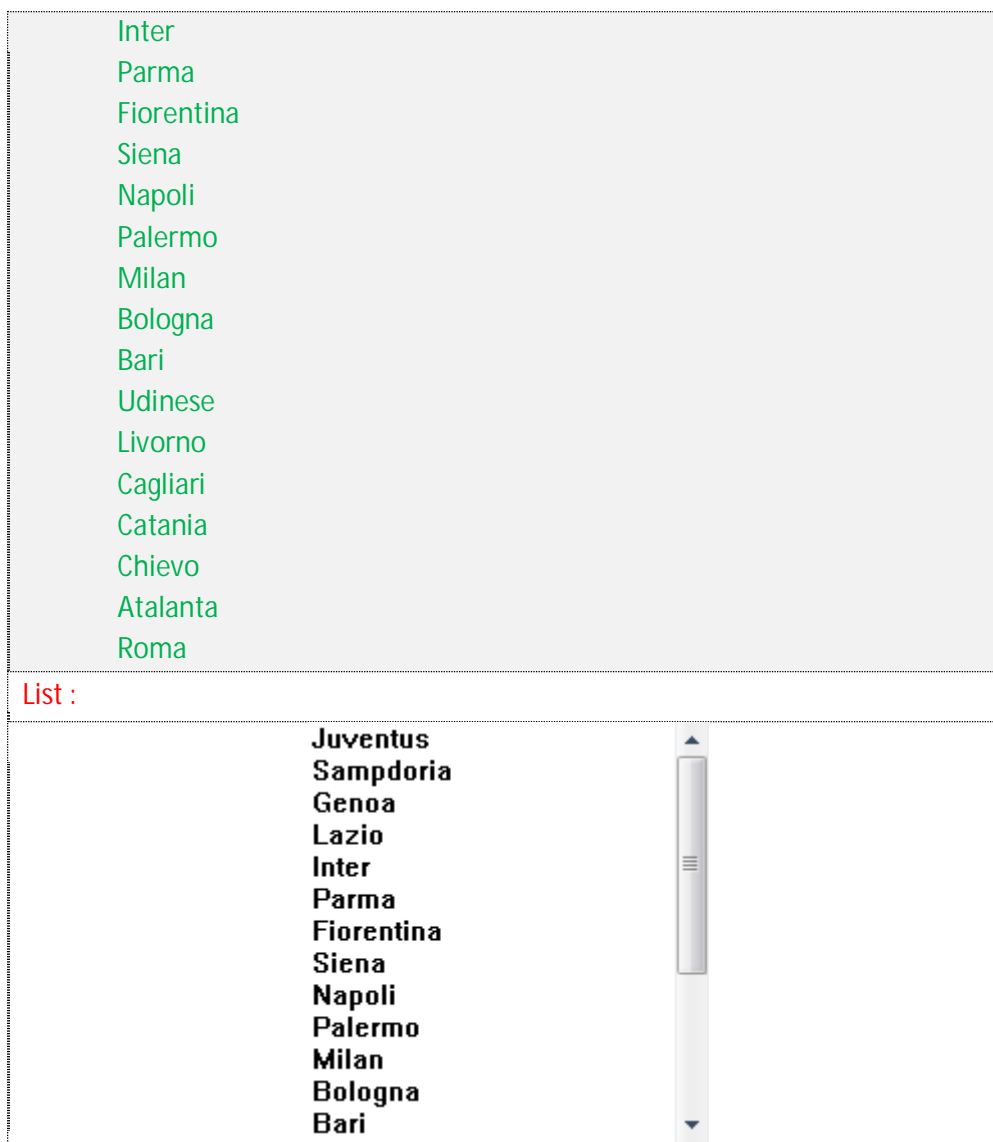


The data file (.dat) only contains the sequential lines of the list containing the following data structure :

*Line\_1*  
*Line\_2*  
*Line\_3*  
 ...  
*Line\_N*

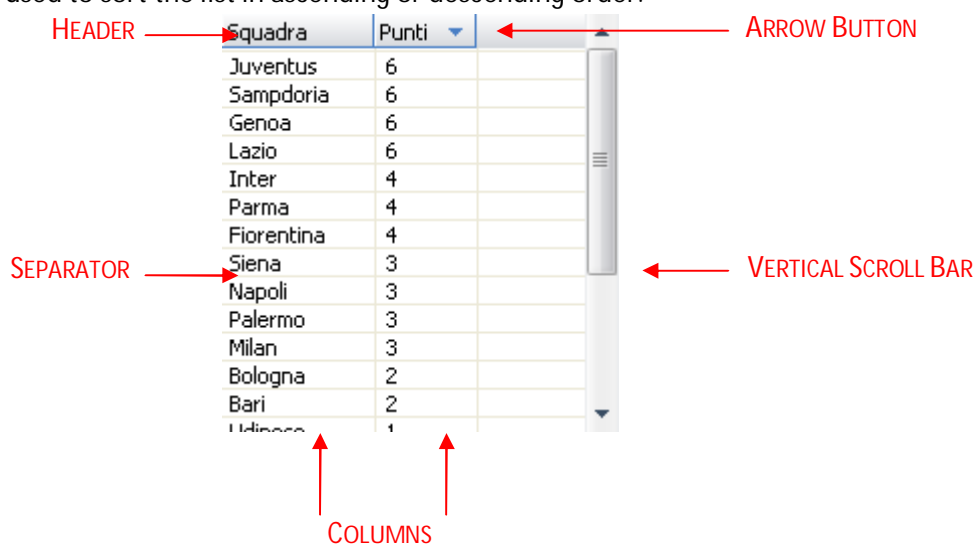
As shown previously, the correct syntax to create a single column list needs the "typ" parameter to be set to 0, for example:

Sample Code :	
	<b>List Creation :</b>
	<code>list w=2,on,vr=14,vc=20,f='foot.dat',typ=0,pan=11,bx=0,by=0</code>
	<b>Dat file :</b>
	Juventus Sampdoria Genoa Lazio



### 9.21.2 – Multiple column with header

The multiple column list contains a variety of features: a text field for each column, scroll bars if required, cells separators and a header which contains a title for every column together with an arrow button used to sort the list in ascending or descending order.



The environment variable called MULTI\_COLUMN\_LIST\_SEPARATOR enables the setting of a user defined column separator for multi-column lists. Example:

INI Sample :

```
MULTI_COLUMN_LIST_SEPARATOR = |
```

The default value for this variable is the comma (",").

The ".dat" file contains the header titles and the string for each cell. Headers and lines have the same structure, to define a line the user inputs a set of strings each divided by the list separator as defined by MULTI\_COLUMN\_LIST\_SEPARATOR or its default. The header line is distinguished by having a sharp "#" (or hash) as the first character of the line and usually this line will be placed as first line of the dat file. For instance, assuming the current separator is the comma ",", the file will follow a format similar to the following :

```
#Header_1, Header_2,... Header_M
Line1_1, Line1_2,..., Line1_M
Line2_1, Line2_2,..., Line2_M
...
LineN_1, LineN_2,..., LineN_M
```

The relevant syntax to create a multi column list needs the "typ" parameter to be set to 1, for example :

Sample Code :

List Creation :

```
list w=3,on,vr=14,vc=20,f='foot2.dat',typ=1,pan=11,bx=0,by=0
```

Dat file :

```
#Team|Points
Juventus|6
Sampdoria|6
Genoa|6
Lazio|6
Inter|4
Parma|4
Fiorentina|4
Siena|3
Napoli|3
Palermo|3
```

Milan|3  
Bologna|2  
Bari|2  
Udinese|1  
Livorno|1  
Cagliari|1  
Catania|0  
Chievo|0  
Atalanta|0  
Roma|0

List :

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

Clicking on a header title it is possible to reorder the items in an ascending (arrow down) or descending (arrow up) mode thereby producing an entire list sort relative to the selected column header, for example:

Squadra	Punti		Squadra	Punti		Squadra	Punti	
Juventus	6		Roma	0		Juventus	6	
Sampdoria	6		Atalanta	0		Sampdoria	6	
Genoa	6		Chievo	0		Genoa	6	
Lazio	6		Catania	0		Lazio	6	
Inter	4		Cagliari	1		Inter	4	
Parma	4		Livorno	1		Parma	4	
Fiorentina	4		Udinese	1		Fiorentina	4	
Siena	3		Bari	2		Siena	3	
Napoli	3		Bologna	2		Napoli	3	
Palermo	3		Palermo	3		Palermo	3	
Milan	3		Milan	3		Milan	3	
Bologna	2		Napoli	3		Bologna	2	
Bari	2		Siena	3		Bari	2	
Udinese	1		Fiorentina	4		Udinese	1	
Creation			Sort Descending			Sort Ascending		

When a multicolumn list is created the size of each column defined by fitting to the size of the header length, afterwards users can resize columns to obtain the best possible display.

### 9.21.3 – Selection and multiple selection

There are two types of selection: single or multiple selections.

Nome	Posizione		
Matteo	7		Matteo
Filippo	2		Filippo
Maurizio	3		Maurizio
Antonio	1		Antonio
Pierangelo	4		Pierangelo
Angelo - Stefano	5		Angelo - Stefano
Daniele	12		Daniele
Federica	9		Federica
Cristiano	16		Cristiano
Lorenzo	8		Lorenzo
Enrico	6		Enrico
Luca	10		Luca
Jacopo	13		Jacopo
Enrico	15		Jacopo

Single Selection

Multiple Selection

A single line of a list can be selected either by using the left button of the mouse or by using the "list" command with the "pos" parameter specified with a value from 1 to the number of lines :

Sample Code :

```
list w=<id>,pos =<line>
```

Multiple selections require the use of the left button in combination with the "shift" key to add items or the "ctrl" key to remove items.

Additionally, it is possible to save the result of a multiple selection action to a file; to allow this mechanism we need to invoke a new instance of the "list" instruction specifying the list id and the name of the output file in the "l" parameter, for example:

Sample Code :

**Creation :**

```
list w=2, on, vr=14, vc=20, f='list.dat', typ=1, pan=11, bx=0, by=0
delete buffer.dat
list w=2,l='buffer.dat'
```

**Selection :**

First	Second	Third	
item 1,1	item 1,2	item 1,3	
item 2,1	item 2,2	item 2,3	
item 3,1	item 3,2	item 3,3	
item 4,1	item 4,2	item 4,3	
item 5,1	item 5,2	item 5,3	

**Output File :**

```
item 2,1|item 2,2|item 2,3
item 3,1|item 3,2|item 3,3
item 4,1|item 4,2|item 4,3
```

### 9.21.4 – Add a new line

The “list” command also permits insertion of a new line using the “add” parameter. The “pos” entry is used to choose where the new item will be added:

- pos = 1 → the item will be added as the first element;
- pos = 2,...,N → the item will be inserted in a specific position;
- pos = 0 → the item will be appended at the bottom of the list.

For example:

Sample Code :

```
list w=<id>,pos =<line>,add='string'
```

First	Second	Third	First	Second	Third	First	Second	Third
→ insert 1	insert 2	insert 3	item 1,1	item 1,2	item 1,3	item 1,1	item 1,2	item 1,3
item 1,1	item 1,2	item 1,3	item 2,1	item 2,2	item 2,3	item 2,1	item 2,2	item 2,3
item 2,1	item 2,2	item 2,3	→ insert 1	insert 2	insert 3	item 3,1	item 3,2	item 3,3
item 3,1	item 3,2	item 3,3	item 3,1	item 3,2	item 3,3	item 4,1	item 4,2	item 4,3
item 4,1	item 4,2	item 4,3	item 4,1	item 4,2	item 4,3	item 5,1	item 5,2	item 5,3
item 5,1	item 5,2	item 5,3	item 5,1	item 5,2	item 5,3	→ insert 1	insert 2	insert 3
pos = 1			pos = 3			pos = 0		

### 9.21.5 – Remove a line

In a similar way to insertion above, it is conversely possible to remove an item or all items from a list using the “del” parameter. In this case the “pos” entry can have the following possible values:

- pos = 1 → the first item will be removed;
- pos = 2,...,N → the item in a specific position will be deleted;
- pos = 0 → all the items will be erased.

For example:

Sample Code :

```
list w=<id>,pos =<line>,del
```

First	Second	Third	First	Second	Third	First	Second	Third
→ item 2,1	item 2,2	item 2,3	item 1,1	item 1,2	item 1,3			
item 3,1	item 3,2	item 3,3	item 2,1	item 2,2	item 2,3			
item 4,1	item 4,2	item 4,3	→ item 4,1	item 4,2	item 4,3			
item 5,1	item 5,2	item 5,3	item 5,1	item 5,2	item 5,3			
pos = 1			pos = 3			pos = 0		



### 9.21.6 – Standalone list

Eagle V14 enables creation of a list in a standalone mode, meaning the developer can create a floating panel that contains a list view object with the same features present in the above sections. The list inside the panel completely fills the dialog space and is resized when the panel is stretched.

The “list” command to create the standalone list is structured a little differently:

Prototype :	
	<code>list on, w=&lt;i&gt;, vc=&lt;i&gt;, vr=&lt;i&gt; ,f=&lt;file&gt;, bx=&lt;i&gt;, by=&lt;i&gt;, typ=&lt;i&gt;, j=&lt;n1&gt;,&lt;n2&gt;, t=&lt;text&gt;</code>

where:

Parameter	Description
<code>on</code>	Create the list
<code>W</code>	List index, in the range 1 to 10
<code>vc</code>	The number of visible columns
<code>vr</code>	The number of visible rows (maximum 128)
<code>F</code>	Data file name, in quotes
<code>bx</code>	The x position of the list in the panel
<code>by</code>	The y position of the list in the panel
<code>typ</code>	Display the content of a list box with different columns. Set typ=0 for single column or typ=1 for multi-column
<code>j=&lt;n1&gt;,&lt;n2&gt;</code>	Floating position (x, y) specified with J=<x_value>,<y_value>, defined in pixels. If j is not specified the default values are (-1, -1) = new row/column
<code>t&lt;text&gt;</code>	Panel title.

For example :

Sample Code :	
	<code>list on,w=1,vc=46,vr=24,f='dir.dat',t='directory ^1', bx=0,by=0,typ=0</code>

Like other GUI elements the standalone list panel is also affected by the current theme presentation:

Office 2000																																																																																																																							
<div> <div>BARCLAYS SCOTTISH OPEN, 8 TO 11 JULY 2010</div> <table> <tr> <th>POS</th><th>START</th><th>NAME</th><th>SURNAME</th><th>SCORE</th><th>1</th><th>2</th><th>3</th><th>4</th><th></th></tr> <tr><td>1</td><td>1</td><td>Edoardo</td><td>MOLINARI</td><td>-12</td><td>66</td><td>69</td><td>63</td><td>74</td><td></td></tr> <tr><td>2</td><td>2</td><td>Darren</td><td>CLARKE</td><td>-9</td><td>65</td><td>67</td><td>67</td><td>76</td><td></td></tr> <tr><td>3</td><td>7</td><td>Raphaël</td><td>JACQUELIN</td><td>-8</td><td>71</td><td>68</td><td>69</td><td>68</td><td></td></tr> <tr><td>4</td><td>9</td><td>Stephen</td><td>GALLACHER</td><td>-7</td><td>67</td><td>73</td><td>69</td><td>68</td><td></td></tr> <tr><td>4</td><td>3</td><td>Francesco</td><td>MOLINARI</td><td>-7</td><td>68</td><td>69</td><td>68</td><td>72</td><td></td></tr> <tr><td>4</td><td>3</td><td>Peter</td><td>HEDBLUM</td><td>-7</td><td>67</td><td>69</td><td>69</td><td>72</td><td></td></tr> <tr><td>7</td><td>5</td><td>Shane</td><td>LOWRY</td><td>-6</td><td>68</td><td>73</td><td>66</td><td>71</td><td></td></tr> <tr><td>8</td><td>1</td><td>Johan</td><td>EDFORS</td><td>-5</td><td>67</td><td>76</td><td>68</td><td>68</td><td></td></tr> <tr><td>8</td><td>9</td><td>Ross</td><td>FISHER</td><td>-5</td><td>71</td><td>73</td><td>65</td><td>70</td><td></td></tr> <tr><td>8</td><td>7</td><td>Rory</td><td>SABBATINI</td><td>-5</td><td>70</td><td>69</td><td>69</td><td>71</td><td></td></tr> </table> </div>										POS	START	NAME	SURNAME	SCORE	1	2	3	4		1	1	Edoardo	MOLINARI	-12	66	69	63	74		2	2	Darren	CLARKE	-9	65	67	67	76		3	7	Raphaël	JACQUELIN	-8	71	68	69	68		4	9	Stephen	GALLACHER	-7	67	73	69	68		4	3	Francesco	MOLINARI	-7	68	69	68	72		4	3	Peter	HEDBLUM	-7	67	69	69	72		7	5	Shane	LOWRY	-6	68	73	66	71		8	1	Johan	EDFORS	-5	67	76	68	68		8	9	Ross	FISHER	-5	71	73	65	70		8	7	Rory	SABBATINI	-5	70	69	69	71	
POS	START	NAME	SURNAME	SCORE	1	2	3	4																																																																																																															
1	1	Edoardo	MOLINARI	-12	66	69	63	74																																																																																																															
2	2	Darren	CLARKE	-9	65	67	67	76																																																																																																															
3	7	Raphaël	JACQUELIN	-8	71	68	69	68																																																																																																															
4	9	Stephen	GALLACHER	-7	67	73	69	68																																																																																																															
4	3	Francesco	MOLINARI	-7	68	69	68	72																																																																																																															
4	3	Peter	HEDBLUM	-7	67	69	69	72																																																																																																															
7	5	Shane	LOWRY	-6	68	73	66	71																																																																																																															
8	1	Johan	EDFORS	-5	67	76	68	68																																																																																																															
8	9	Ross	FISHER	-5	71	73	65	70																																																																																																															
8	7	Rory	SABBATINI	-5	70	69	69	71																																																																																																															
Office 2003																																																																																																																							
<div> <div>BARCLAYS SCOTTISH OPEN, 8 TO 11 JULY 2010</div> <table> <tr> <th>POS</th><th>START</th><th>NAME</th><th>SURNAME</th><th>SCORE</th><th>1</th><th>2</th><th>3</th><th>4</th><th></th></tr> <tr><td>1</td><td>1</td><td>Edoardo</td><td>MOLINARI</td><td>-12</td><td>66</td><td>69</td><td>63</td><td>74</td><td></td></tr> <tr><td>2</td><td>2</td><td>Darren</td><td>CLARKE</td><td>-9</td><td>65</td><td>67</td><td>67</td><td>76</td><td></td></tr> <tr><td>3</td><td>7</td><td>Raphaël</td><td>JACQUELIN</td><td>-8</td><td>71</td><td>68</td><td>69</td><td>68</td><td></td></tr> <tr><td>4</td><td>9</td><td>Stephen</td><td>GALLACHER</td><td>-7</td><td>67</td><td>73</td><td>69</td><td>68</td><td></td></tr> <tr><td>4</td><td>3</td><td>Francesco</td><td>MOLINARI</td><td>-7</td><td>68</td><td>69</td><td>68</td><td>72</td><td></td></tr> <tr><td>4</td><td>3</td><td>Peter</td><td>HEDBLUM</td><td>-7</td><td>67</td><td>69</td><td>69</td><td>72</td><td></td></tr> <tr><td>7</td><td>5</td><td>Shane</td><td>LOWRY</td><td>-6</td><td>68</td><td>73</td><td>66</td><td>71</td><td></td></tr> <tr><td>8</td><td>1</td><td>Johan</td><td>EDFORS</td><td>-5</td><td>67</td><td>76</td><td>68</td><td>68</td><td></td></tr> <tr><td>8</td><td>9</td><td>Ross</td><td>FISHER</td><td>-5</td><td>71</td><td>73</td><td>65</td><td>70</td><td></td></tr> <tr><td>8</td><td>7</td><td>Rory</td><td>SABBATINI</td><td>-5</td><td>70</td><td>69</td><td>69</td><td>71</td><td></td></tr> </table> </div>										POS	START	NAME	SURNAME	SCORE	1	2	3	4		1	1	Edoardo	MOLINARI	-12	66	69	63	74		2	2	Darren	CLARKE	-9	65	67	67	76		3	7	Raphaël	JACQUELIN	-8	71	68	69	68		4	9	Stephen	GALLACHER	-7	67	73	69	68		4	3	Francesco	MOLINARI	-7	68	69	68	72		4	3	Peter	HEDBLUM	-7	67	69	69	72		7	5	Shane	LOWRY	-6	68	73	66	71		8	1	Johan	EDFORS	-5	67	76	68	68		8	9	Ross	FISHER	-5	71	73	65	70		8	7	Rory	SABBATINI	-5	70	69	69	71	
POS	START	NAME	SURNAME	SCORE	1	2	3	4																																																																																																															
1	1	Edoardo	MOLINARI	-12	66	69	63	74																																																																																																															
2	2	Darren	CLARKE	-9	65	67	67	76																																																																																																															
3	7	Raphaël	JACQUELIN	-8	71	68	69	68																																																																																																															
4	9	Stephen	GALLACHER	-7	67	73	69	68																																																																																																															
4	3	Francesco	MOLINARI	-7	68	69	68	72																																																																																																															
4	3	Peter	HEDBLUM	-7	67	69	69	72																																																																																																															
7	5	Shane	LOWRY	-6	68	73	66	71																																																																																																															
8	1	Johan	EDFORS	-5	67	76	68	68																																																																																																															
8	9	Ross	FISHER	-5	71	73	65	70																																																																																																															
8	7	Rory	SABBATINI	-5	70	69	69	71																																																																																																															
Office 2007 R1																																																																																																																							
<div> <div>BARCLAYS SCOTTISH OPEN, 8 TO 11 JULY 2010</div> <table> <tr> <th>POS</th><th>START</th><th>NAME</th><th>SURNAME</th><th>SCORE</th><th>1</th><th>2</th><th>3</th><th>4</th><th></th></tr> <tr><td>1</td><td>1</td><td>Edoardo</td><td>MOLINARI</td><td>-12</td><td>66</td><td>69</td><td>63</td><td>74</td><td></td></tr> <tr><td>2</td><td>2</td><td>Darren</td><td>CLARKE</td><td>-9</td><td>65</td><td>67</td><td>67</td><td>76</td><td></td></tr> <tr><td>3</td><td>7</td><td>Raphaël</td><td>JACQUELIN</td><td>-8</td><td>71</td><td>68</td><td>69</td><td>68</td><td></td></tr> <tr><td>4</td><td>9</td><td>Stephen</td><td>GALLACHER</td><td>-7</td><td>67</td><td>73</td><td>69</td><td>68</td><td></td></tr> <tr><td>4</td><td>3</td><td>Francesco</td><td>MOLINARI</td><td>-7</td><td>68</td><td>69</td><td>68</td><td>72</td><td></td></tr> <tr><td>4</td><td>3</td><td>Peter</td><td>HEDBLUM</td><td>-7</td><td>67</td><td>69</td><td>69</td><td>72</td><td></td></tr> <tr><td>7</td><td>5</td><td>Shane</td><td>LOWRY</td><td>-6</td><td>68</td><td>73</td><td>66</td><td>71</td><td></td></tr> <tr><td>8</td><td>1</td><td>Johan</td><td>EDFORS</td><td>-5</td><td>67</td><td>76</td><td>68</td><td>68</td><td></td></tr> <tr><td>8</td><td>9</td><td>Ross</td><td>FISHER</td><td>-5</td><td>71</td><td>73</td><td>65</td><td>70</td><td></td></tr> <tr><td>8</td><td>7</td><td>Rory</td><td>SABBATINI</td><td>-5</td><td>70</td><td>69</td><td>69</td><td>71</td><td></td></tr> </table> </div>										POS	START	NAME	SURNAME	SCORE	1	2	3	4		1	1	Edoardo	MOLINARI	-12	66	69	63	74		2	2	Darren	CLARKE	-9	65	67	67	76		3	7	Raphaël	JACQUELIN	-8	71	68	69	68		4	9	Stephen	GALLACHER	-7	67	73	69	68		4	3	Francesco	MOLINARI	-7	68	69	68	72		4	3	Peter	HEDBLUM	-7	67	69	69	72		7	5	Shane	LOWRY	-6	68	73	66	71		8	1	Johan	EDFORS	-5	67	76	68	68		8	9	Ross	FISHER	-5	71	73	65	70		8	7	Rory	SABBATINI	-5	70	69	69	71	
POS	START	NAME	SURNAME	SCORE	1	2	3	4																																																																																																															
1	1	Edoardo	MOLINARI	-12	66	69	63	74																																																																																																															
2	2	Darren	CLARKE	-9	65	67	67	76																																																																																																															
3	7	Raphaël	JACQUELIN	-8	71	68	69	68																																																																																																															
4	9	Stephen	GALLACHER	-7	67	73	69	68																																																																																																															
4	3	Francesco	MOLINARI	-7	68	69	68	72																																																																																																															
4	3	Peter	HEDBLUM	-7	67	69	69	72																																																																																																															
7	5	Shane	LOWRY	-6	68	73	66	71																																																																																																															
8	1	Johan	EDFORS	-5	67	76	68	68																																																																																																															
8	9	Ross	FISHER	-5	71	73	65	70																																																																																																															
8	7	Rory	SABBATINI	-5	70	69	69	71																																																																																																															
Office 2010 R1																																																																																																																							
<div> <div>BARCLAYS SCOTTISH OPEN, 8 TO 11 JULY 2010</div> <table> <tr> <th>POS</th><th>START</th><th>NAME</th><th>SURNAME</th><th>SCORE</th><th>1</th><th>2</th><th>3</th><th>4</th><th></th></tr> <tr><td>1</td><td>1</td><td>Edoardo</td><td>MOLINARI</td><td>-12</td><td>66</td><td>69</td><td>63</td><td>74</td><td></td></tr> <tr><td>2</td><td>2</td><td>Darren</td><td>CLARKE</td><td>-9</td><td>65</td><td>67</td><td>67</td><td>76</td><td></td></tr> <tr><td>3</td><td>7</td><td>Raphaël</td><td>JACQUELIN</td><td>-8</td><td>71</td><td>68</td><td>69</td><td>68</td><td></td></tr> <tr><td>4</td><td>9</td><td>Stephen</td><td>GALLACHER</td><td>-7</td><td>67</td><td>73</td><td>69</td><td>68</td><td></td></tr> <tr><td>4</td><td>3</td><td>Francesco</td><td>MOLINARI</td><td>-7</td><td>68</td><td>69</td><td>68</td><td>72</td><td></td></tr> <tr><td>4</td><td>3</td><td>Peter</td><td>HEDBLUM</td><td>-7</td><td>67</td><td>69</td><td>69</td><td>72</td><td></td></tr> <tr><td>7</td><td>5</td><td>Shane</td><td>LOWRY</td><td>-6</td><td>68</td><td>73</td><td>66</td><td>71</td><td></td></tr> <tr><td>8</td><td>1</td><td>Johan</td><td>EDFORS</td><td>-5</td><td>67</td><td>76</td><td>68</td><td>68</td><td></td></tr> <tr><td>8</td><td>9</td><td>Ross</td><td>FISHER</td><td>-5</td><td>71</td><td>73</td><td>65</td><td>70</td><td></td></tr> <tr><td>8</td><td>7</td><td>Rory</td><td>SABBATINI</td><td>-5</td><td>70</td><td>69</td><td>69</td><td>71</td><td></td></tr> </table> </div>										POS	START	NAME	SURNAME	SCORE	1	2	3	4		1	1	Edoardo	MOLINARI	-12	66	69	63	74		2	2	Darren	CLARKE	-9	65	67	67	76		3	7	Raphaël	JACQUELIN	-8	71	68	69	68		4	9	Stephen	GALLACHER	-7	67	73	69	68		4	3	Francesco	MOLINARI	-7	68	69	68	72		4	3	Peter	HEDBLUM	-7	67	69	69	72		7	5	Shane	LOWRY	-6	68	73	66	71		8	1	Johan	EDFORS	-5	67	76	68	68		8	9	Ross	FISHER	-5	71	73	65	70		8	7	Rory	SABBATINI	-5	70	69	69	71	
POS	START	NAME	SURNAME	SCORE	1	2	3	4																																																																																																															
1	1	Edoardo	MOLINARI	-12	66	69	63	74																																																																																																															
2	2	Darren	CLARKE	-9	65	67	67	76																																																																																																															
3	7	Raphaël	JACQUELIN	-8	71	68	69	68																																																																																																															
4	9	Stephen	GALLACHER	-7	67	73	69	68																																																																																																															
4	3	Francesco	MOLINARI	-7	68	69	68	72																																																																																																															
4	3	Peter	HEDBLUM	-7	67	69	69	72																																																																																																															
7	5	Shane	LOWRY	-6	68	73	66	71																																																																																																															
8	1	Johan	EDFORS	-5	67	76	68	68																																																																																																															
8	9	Ross	FISHER	-5	71	73	65	70																																																																																																															
8	7	Rory	SABBATINI	-5	70	69	69	71																																																																																																															

INI Sample :

LIST\_CLOSE\_ACTION = closelist.cmd

### 9.21.7 – List as button in tab file

Eagle V14 introduces a new way to attach a list to a panel, now the developer can create a list object directly in a tab file using common syntax. This approach enables a list to be consider just like a standard control, so all the features related to the option command or the tab file, for instance the anchor mechanism ([see 8.14](#)) and the font settings ([see 8.13](#)), are also available for lists.

As mentioned in the sub-chapter above, in Eagle there are two types of lists, the single column and multi-column list. In order to the desired list mode the button type must be defined as one of the following:

- 505 : single column list;
- 506 : multi-column list.

The complete syntax for this object is:

Tab Syntax :

opt, x, y, w, h, 0, 0, t, &lt;file.dat&gt; ;

where:

Parameter	Description
<b>opt</b>	Index of the option.
<b>x</b>	X co-ordinate of the button in the menu grid.
<b>y</b>	Y co-ordinate of the button in the menu grid.
<b>w</b>	Width of the list in grid units (pixels).
<b>h</b>	Height of the list in grid units (pixels).
<b>t = 505</b> <b>t = 506</b>	Button type, 505 for the single column list and 506 for multicolumn list.
<b>&lt; file.dat &gt;</b>	Data file name that contains the list content, in quotes

The “dat” file contents and the list separator have the same meaning as the old style list, so developers don’t have to make any changes. Furthermore the behavior of single selection, multiple selection (with or without output file) and polling are also the same as the previous list type, refer the sub-chapter above for more details.

For example:

Sample Code :

```
3,0,0,200,200,0,0,505,'list.dat';
2,0,0,200,200,0,0,506,'listmulti.dat';
2,0,100,200,200,0,0,506,'listsort1.dat';
```

In order to carry out common actions on the list, for example adding, removing or selecting a row, the ‘list’ command has been modified to introduce two parameters which identify the list with the panel id and the button id, a behavior similar to the click command. As a result, in this case, the syntax for the ‘list’ command is the following:

Prototype :

```
list      on, off, p=<i>, b=<i>, ,f=<file> ,pos=<i>, add='<text>'}, del
          l=<file>
```

where:

Parameter	Description
<b>on</b>	(default) Create the list
<b>off</b>	Destroy the list
<b>p</b>	Panel identifier
<b>b</b>	Button (list) identifier
<b>f</b>	Data file name, in quotes
<b>l</b>	Output file name, in quotes, to store data after a multiple selection. There is no default for the file name extension
<b>pos</b>	Position (line index, 1-based) in which to add (ADD) a new line, delete (DEL)

or select (SEL) an existing line; note that:

- Add a line with pos = 0, the new line will appended at the bottom;
- Delete with pos = 0, all lines will be removed
- Add a line with pos = 1, the new line will be inserted at the top
- Delete with pos = 1, the first line will be removed

<b>add='text'</b>	Text for a new line
<b>del</b>	Delete a line

The next table shows how to use the "list" command to perform all the actions on a list object when it has been created as a button:

Action	Command
Delete list	list p=<i>, b=<i>, off LIST P=1, B=2, OFF
Add row	list p=<i>, b=<i>, pos=<index>, add='text' LIST P=1, B=2, POS=3, ADD='new line'
Remove row	list p=<i>, b=<i>, pos=<index>, del LIST P=1, B=2, POS=3, DEL
Select row	list p=<i>, b=<i>, pos=<index> LIST P=1, B=2, POS=3
Output file	list p=<i>, b=<i>, l='file.out' LIST P=1, B=2, L='list.out'
Replace list	list p=<i>, b=<i>, f='file.dat' LIST P=1, B=2, F='list.dat'

Being a button in all respects, if the layout option is used when creating the panel to define the resizable layout ([see 8.14](#)), then the list also follows these settings and consequently it is possible to automatically resize a list when resizing a panel (or dialog, docked or floating).

Obviously, because this kind of list works like a standard control, it is also possible to use the "pbutton" command to add or delete the object in a panel ([see 9.2 for more details](#)).

Sample Code :

```
pbutton p=11, b=2, add='2,0,0,200,200,0,0,506,"listmulti.dat";'  
pbutton p=11, b=2, del
```

To define a font for list button developers have to insert the “font” settings before the list entry in the tab file, as shown in [section 8.13](#).

## 9.22 – Grid View

A grid view is a graphical user interface element that presents a tabular (grid) view of data. A typical grid displays the values of a data source in a table where each column represents a field and each row represents a record.

The grid control supports the following features:

- Varied selection systems (by cell, row or column);
- Sort order by clicking a column header of the grid
- Alignment of cell contents, guided by the column;
- Resizable columns and rows;
- Different types for cells (text, image, check box, etc.);
- In-place editing of the viewed data;
- Read-only cells;
- Row headers with user defined text or incremental index;
- Separate polling exits for selection and editing.

Working with a grid control utilizes the new “gridview” command introduced to provide an experience similar to that of a list control.

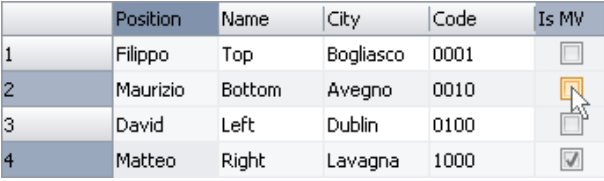
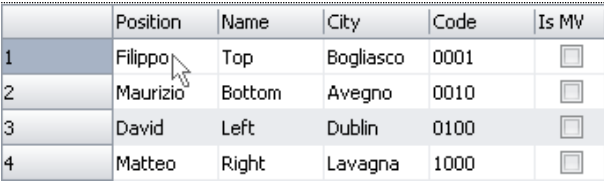
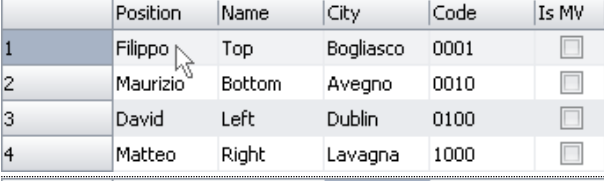
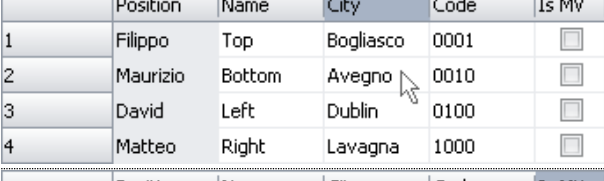
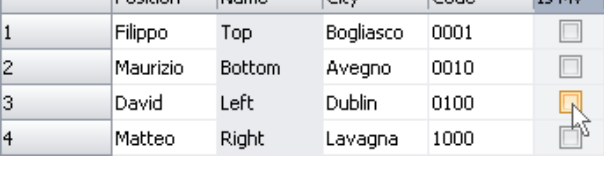
Like other UI Framework controls the appearance of the grid is influenced by the currently defined theme.

### 9.22.1 – Styles and behavior

The user can apply different styles and behavior to the grid control. A number of these features must be selected at creation time and the others may be defined by manipulating the control (i.e. adding a new column).

The most important option is the “style” of the grid as this choice has a significant influence on the grid behavior. Eagle V14 has six different grid types:

Type	Selection	Highlight	Sample					
1	Cell	Selected cell;		Position	Name	City	Code	Is MV
		Pointed cell;	1	Filippo	Top	Bogliasco	0001	<input type="checkbox"/>
		Pointed row header;	2	Maurizio	Bottom	Avegno	0010	<input type="checkbox"/>
		Pointed column header;	3	David	Left	Dublin	0100	<input type="checkbox"/>
			4	Matteo	Right	Lavagna	1000	<input type="checkbox"/>

2	Cell	Selected cell; Row of selected cell; Column of selected cell; Pointed cell; Pointed row header; Pointed column header;	
3	Row	Selected row; Pointed row header;	
4	Row	Selected row; Pointed row; Pointed row header;	
5	Column	Selected column; Pointed column header;	
6	Column	Selected column; Pointed column; Pointed column header;	

The style defined in the command creation also effects the values returned by the polling loop, this issue will be explained in an appropriate section below.

Another characteristic definable at creation time is the grid "mode". Using this attribute it's possible to set the cell's editing mode. The permitted values are presented in the table below:

Type	Cell editing
1	<i>F2 key</i>
2	Double click of the left mouse button
3	<i>F2 key + double click of the left mouse button</i>

In Eagle, the return key is not handled for the grid edit mode because in the most commonly used grid view the return event is used to select the next cell.

Creating a grid, the user can also select a sort behavior. In Eagle it is possible to choose between four "sort" classes:

Type	Sort behavior	Column Header	Row Header
none	no sort (default)		
row	rows sort	-	Ascending 2

				Descending	3
column	columns sort	Ascending	Name ▲	-	
		Descending	Name ▼		
all	columns and rows sort	Ascending	Name ▲	Ascending	2
		Descending	Name ▼	Descending	3

In addition to these styles, there are two further features that can be defined when inserting a new column in the grid control. The first of these enables creation of a “read-only” column, so the end-user is unable to change the cell contents using the editing mechanism (i.e. for a text cell) or by using mouse events (i.e. a check box cell).

Finally, it is possible to indicate the “alignment” for column items where the possible values are displayed below:

	Left	Center	Right
Type	1	2	3
Sample	Top	Filippo	Bogliasco
	Bottom	Maurizio	Avegno
	Left	David	Dublin
	Right	Matteo	Lavagna

### 9.22.2 – The gridview command

A new “gridview” command has been developed to work with grid controls. The relationship between this new command and grids can be likened to the relationship between the “list” command and list controls. Using the gridview command it is able to:

- create and destroy grids;
- add and delete columns;
- add and delete rows;
- set the cell content;
- select a row, a column or a cell;
- fetch the content of a row, a column, a cell or an entire multi-selection area.

Grids differ from other controls in that the instruction used to insert a grid object into a panel is the “gridview” command and not the more frequently used “tab” file. The relevant syntax of the command is defines as follows:

Prototype :	
gridview	{on   off   add   delete   set   fetch=<s>   select}, id=<i>, panel=<i>, x=<i>, y=<i>, w=<i>, h=<i>, style=<i>,



```
mode=<i>, sort=<type>, align=<i>, row=<i>, column=<i>,
size=<i>, title=<i>, header, type=<i>, text=<s>, options=<string
array>
```

where:

Parameter	Description
on	(default) Create the grid
off	Destroy a grid
add	Add row or column
delete	Delete row or column
set	Set the content of a cell
fetch	Get the content of a cell, row, column or multi-selection area. This field requires an Eagle variable to save the appropriately formatted resulting string.
select	Select a cell, a row, a column or the entire grid
id	Grid identifier
panel	(only with the "on" primer) Panel id in which the grid will be created
x	X co-ordinate of the button in the menu
y	Y co-ordinate of the button in the menu
w	Width of the button. A value of "-1" indicates auto-resize.
h	Height of the button. A value of "-1" indicates auto-resize.
style	(only with the "on" primer) Define the grid behavior: <ul style="list-style-type: none"> <li>• style=1 → (default) cell selection with header highlight on cursor tracking;</li> <li>• style=2 → cell selection with complete highlight on cursor tracking;</li> <li>• style=3 → row selection with header highlight on cursor tracking;</li> <li>• style=4 → row selection with complete highlight on cursor tracking;</li> <li>• style=5 → column selection with header highlight on cursor tracking;</li> <li>• style=6 → column selection with complete highlight on cursor tracking;</li> </ul>
mode	1. Creating a grid ("on" primer) – Enables the editing mode selection: <ul style="list-style-type: none"> <li>• mode = 1 → (default) F2</li> <li>• mode = 2 → Double click with left button</li> <li>• mode = 3 → F2 + double click with left button</li> </ul>
sort	(only with the "on" primer) String that defines the sorting behavior: <ul style="list-style-type: none"> <li>• none : (default) → sorting disabled</li> <li>• column : enable sort only for columns</li> <li>• row : enable sort only for rows</li> <li>• all : enable all sort types</li> </ul>
align	(only with the "add" primer for column) Setting the content alignment in the

	<p>column's cells:</p> <ul style="list-style-type: none"> <li>• align=1 → left</li> <li>• align=2 → center</li> <li>• align=3 → right</li> </ul>
column	<p>1-based value that indicates the column index (1 based) or determinates the position in which the column will be added or removed;</p> <p>A special behavior is related to "column = -1":</p> <ul style="list-style-type: none"> <li>• adding a column, the new column will be appended to the end;</li> <li>• removing columns, each columns will be deleted;</li> </ul>
row	<p>1-based value that indicates the row index or determines the position in which a row will be added or removed;</p> <p>A unique behavior is related to "row = -1":</p> <ul style="list-style-type: none"> <li>• adding a row, the new row will be appended to the end;</li> <li>• removing rows, each row will be deleted;</li> </ul>
size	<p>(only with the "add" primer) The dimension of the new column (width) or a new row (height)</p>
title	<p>(only with the "add" primer) Contains the text to insert in the header of the row or column. Note in the case of a row the title will be displayed if the "header" parameter is applied</p>
header	<p>(only with the "add" primer for row) When applied, indicates that the text inside the header is defined using the "title" parameters.</p> <p>If the "header" is not included in the row definition, the header will display the index of the row, like Microsoft Excel</p>
type	<p>(only with the "add" primer for column) Define the type of the cells in new columns. Developers can choose between various types:</p> <ul style="list-style-type: none"> <li>➤ type=1 : edit field (default)</li> <li>➤ type=2 : check box</li> <li>➤ type=3 : three-state check box</li> <li>➤ type=4 : combo box</li> <li>➤ type=5 : image cell (bitmap file with transparent background)</li> </ul>
text	<p>(only with the "set" primer) String to be assigned a specific cell. The content depends on the cell type:</p> <ul style="list-style-type: none"> <li>➤ type=1 : string with text for edit field</li> <li>➤ type=2 : "1" for check and "-1" for uncheck in checkbox</li> <li>➤ type=3 : "1" for check, "0" for indeterminate and "-1" for uncheck in three-state checkbox</li> <li>➤ type=4 : <u>not used</u>; the combo box needs an options array</li> <li>➤ type=5 : a string that contains the picture display mode (optional) and the bitmap path separated by a "#" character: ' image.bmp ' or ' type#image.bmp '</li> </ul>

	The display mode can assume the following value: <ul style="list-style-type: none"> <li>• a : (default) picture adopts column alignment</li> <li>• t : picture is tiled</li> <li>• s : picture is stretched</li> </ul>
<b>options</b>	(only with the "set" primer in a combo box cell) String array containing the combo box options

For example:

Sample Code :	<pre> gridview on, id=1, pan=3, x=10, y=10, w=400, h=150, style=2, mode=2 gridview on, id=2, pan=3, x=10, y=210, w=400, h=150, style=6, mode=2, sort=all  gridview id=1, add, col=3, title='Address',type=3,size=60 gridview id=1, add, col=-1, title='City',type=4,size=60,align=2  gridview id=2, add, row=3, title='Three',size=20 gridview id=1, add, row=-1, title='One',size=20,header  gridview id=1, del, col=4 gridview id=1, del, row=1  string cells[100] gridview id=2,row=1, col=1,fetch=cells[1]</pre>
---------------	--

The gridview command will be discussed in more detail in the following subsections.

### 9.22.3 – Create and destroy a grid

When we need to activate a grid component, the first step is to attach the control in a specific panel.

In this situation the "gridview" command can be simplified in this way:

Prototype :	<pre> gridview    on, id=&lt;i&gt;, panel=&lt;i&gt;, x=&lt;i&gt;, y=&lt;i&gt;, w=&lt;i&gt;, h=&lt;i&gt;, style=&lt;i&gt;, mode=&lt;i&gt;, sort=&lt;type&gt;</pre>
-------------	---

where:

Parameter	Description
<b>on</b>	(default) Create the grid
<b>id</b>	Grid identifier
<b>panel</b>	Panel id in which the grid will be created

<b>x</b>	X co-ordinate of the grid in the menu
<b>y</b>	Y co-ordinate of the grid in the menu
<b>w</b>	Width of the grid
<b>h</b>	Height of the grid
<b>style</b>	Define the grid behavior: <ul style="list-style-type: none"> <li>• style=1→ (default) cell selection with header highlight on cursor tracking;</li> <li>• style=2→ cell selection with complete highlight on cursor tracking;</li> <li>• style=3→ row selection with header highlight on cursor tracking;</li> <li>• style=4→ row selection with complete highlight on cursor tracking;</li> <li>• style=5→ column selection with header highlight on cursor tracking;</li> <li>• style=6→ column selection with complete highlight on cursor tracking;</li> </ul>
<b>mode</b>	Allows to select the editing mode: <ul style="list-style-type: none"> <li>• mode = 1 → (default) F2</li> <li>• mode = 2 → Double click with left button</li> <li>• mode = 3 → F2 + double click with left button</li> </ul>
<b>sort</b>	String that defines the sorting behavior: <ul style="list-style-type: none"> <li>• none : (default) → sorting disabled</li> <li>• column : enable sort only for columns</li> <li>• row : enable sort only for rows</li> <li>• all : enable all types of sort</li> </ul>

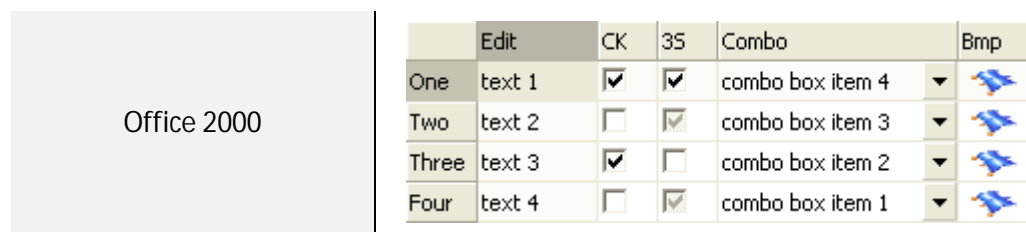
For example :













Sample Code :

```
gridview on, id=1, pan=3, x=10, y=10, w=400, h=150, style=2, mode=2
gridview on, id=2, pan=3, x=10, y=210, w=400, h=150, style=6, mode=2,
sort=all
```

The meaning of “style”, “mode” and “sort” parameters has already been introduced in the previous subsection (9.22.1).

The appearance of the new grid control, like other UI Framework controls, is influenced by the currently applied theme.



Office 2003		Edit	CK	3S	Combo	Bmp
	One	text 1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	combo box item 4	
	Two	text 2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	combo box item 3	
	Three	text 3	<input checked="" type="checkbox"/>	<input type="checkbox"/>	combo box item 2	
	Four	text 4	<input type="checkbox"/>	<input checked="" type="checkbox"/>	combo box item 1	
Office 2007 – Release 1		Edit	CK	3S	Combo	Bmp
	One	text 1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	combo box item 4	
	Two	text 2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	combo box item 3	
	Three	text 3	<input checked="" type="checkbox"/>	<input type="checkbox"/>	combo box item 2	
	Four	text 4	<input type="checkbox"/>	<input checked="" type="checkbox"/>	combo box item 1	
Office 2010 – Release 1		Edit	CK	3S	Combo	Bmp
	One	text 1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	combo box item 4	
	Two	text 2	<input type="checkbox"/>	<input checked="" type="checkbox"/>	combo box item 3	
	Three	text 3	<input checked="" type="checkbox"/>	<input type="checkbox"/>	combo box item 2	
	Four	text 4	<input type="checkbox"/>	<input checked="" type="checkbox"/>	combo box item 1	

In the same way it is possible to destroy a grid control using the “off” option with the following syntax:

Prototype :	<code>gridview off, id=&lt;i&gt;</code>
-------------	---

where:

Parameter	Description
<code>off</code>	Destroy a grid
<code>id</code>	Grid identifier

### 9.22.3 – Add columns

Columns could be considered one of the most important concepts in a grid owing to the fact that the columns structure defines how the user is able to interact with the grid components; In other words columns guide grid behavior.

As shown in the previous sections, the “gridview” syntax allows adding a column by specifying the cell type, the position, if the cells are read-only, the initial width, the content alignment and the title displayed in the column header.

Prototype :	<code>gridview add, id=&lt;i&gt;, mode=&lt;i&gt;, align=&lt;i&gt;, column=&lt;i&gt;, size=&lt;i&gt;, title=&lt;i&gt;, type=&lt;i&gt;, readonly</code>
-------------	---

where:

Parameter	Description
<b>add</b>	Specify to add a new column
<b>id</b>	Grid identifier
<b>mode</b>	Enable r/w status of columns enabling definition "read only" columns: <ul style="list-style-type: none"> <li>• mode = 0 → (default) cells readable and writable</li> <li>• mode = 1 → read-only cell</li> </ul>
<b>align</b>	(only with the "add" primer for column) Set the content alignment for column cells: <ul style="list-style-type: none"> <li>• align=1 → left</li> <li>• align=2 → center</li> <li>• align=3 → right</li> </ul>
<b>column</b>	1-based value that determines the position in which the column will be added. Use "column = -1" to append the new column at the end.
<b>size</b>	The width of the new column.
<b>title</b>	Contains the text to insert in the header of the column
<b>type</b>	Define the type of cells for new columns. Developers can choose between various types: <ul style="list-style-type: none"> <li>➤ type=1 : edit field (default)</li> <li>➤ type=2 : check box</li> <li>➤ type=3 : three-state check box</li> <li>➤ type=4 : combo box</li> <li>➤ type=5 : image cell (bitmap file with transparent background)</li> </ul>
<b>readonly</b>	When adding a column ("add" for column) it enables the creation of a column as "read only": <ul style="list-style-type: none"> <li>• (default) → column readable and writable</li> <li>• readonly → read-only column</li> </ul>

For example :

Sample Code :

```
gridview id=1, add, col=1, title='Edit',type=1,size=60, align=1
gridview id=1, add, col=2, title='CK',type=2,size=30, align=1
gridview id=1, add, col=3, title='3S',type=3,size=30, align=1
gridview id=1, add, col=4, title='Combo',type=4,size=120, align=1
gridview id=1, add, col=5, title='Bmp',type=5,size=40, align=1
```

#### 9.22.4 – Insert a row

Without rows users are unable to interact with the grid and consequently perform different actions such as selections or inputs.

In this case we can rethink the "gridview" prototype in this way:

Prototype :	
	<code>gridview    add, id=&lt;i&gt;, row=&lt;i&gt;, size=&lt;i&gt;, title=&lt;i&gt;, header</code>



where:

Parameter	Description
<code>add</code>	Add row or column
<code>id</code>	Grid identifier
<code>row</code>	1-based value that determinates the position in which the row will be added. Use "row = -1" to append the new row at the end.
<code>size</code>	The height of the new row
<code>title</code>	Contains the text to insert in the header of the row. Note that the title will be displayed if the "header" parameter is applied.
<code>header</code>	When applied, indicates that the text inside the header is defined by the user through the "title" parameters. If the "header" is not included in the row definition, the header will have a named index row, like Microsoft Excel

For example:

Sample Code :	
	<code>gridview id=1, add, row=3, title='Three',size=20,header</code> <code>gridview id=2, add, row=3, size=100</code>

The grid rows can be created in two header mode, with a progressive integer or with a user defined text; both the mode can coexist in the same grid.

	Sample						Syntax
Index	3	David	Left	Dublin	0100		-
Custom Header	Zero	Filippo	Top	Bogliasco	0001		title='Zero', header

### 9.22.5 – Set content of a cell

The "gridview" command enables setting of cell content for every cell type; therefore the parameters have a different meaning depending on the defined type. Basically the common syntax can be explained as illustrated below:

Prototype :	
	<code>gridview    set, id=&lt;i&gt;, row=&lt;i&gt;, column=&lt;i&gt;,               { text=&lt;s&gt;   options=&lt;string array&gt; }</code>

where:

Parameter	Description
<b>set</b>	Set the content of a cell
<b>id</b>	Grid identifier
<b>column</b>	1-based value that indicates the column index
<b>row</b>	1-based value that indicates the row index
<b>text</b>	<p>String to be assigned a specific cell; the content depends on the cell types, in fact:</p> <ul style="list-style-type: none"> <li>➤ type=1 : string with the text input cell</li> <li>➤ type=2 : "1" for check and "-1" for uncheck checkbox</li> <li>➤ type=3 : "1" for check, "0" for indeterminate and "-1" for uncheck triple state checkbox</li> <li>➤ type=4 : <u>not used</u>; the combo box needs an options array</li> <li>➤ type=5 : a string that contains the picture display mode (optional) and the bitmap path separated by a "#" character:              ' image.bmp ' or ' type#image.bmp '</li> </ul> <p>The display mode can assume the following value:</p> <ul style="list-style-type: none"> <li>• a : (default) picture obtains column alignment</li> <li>• t : picture is tiled</li> <li>• s : picture is stretched</li> </ul>
<b>options</b>	(only for combo box cell) String array containing the combo box options

For example :

Sample Code :

```
gridview id=1, set, row=4, col=1, text='text 4'
gridview id=1, set, row=2, col=2, text='-1'
gridview id=1, set, row=2, col=3, text='0'
gridview id=1, set, row=4, col=4, options=combo[1]
gridview id=1, set, row=1, col=5, text='se2.bmp'
```

### Type 1 – edit cell

The set command on the "edit" cells can be used to define text inside cells.

For example:

Sample Code :

**original**

**command**

```
gridview id=1, set, row=1, col=1, text='text 1'
```

**result**





Naturally, an edit field can be edited using the “F2” function key or with a double click of the left button, depending on the “mode” utilized during the grid creation. Note that if an edit column is read only (mode = 1 adding the column) the user will be able to “cut” but not modify the content using mouse or keyboard inputs, consequently only the “set” instruction is able to change text.

#### Type 2 – check box cell

Check fields declare a “true/false” behavior, so the user can check or uncheck the box. In this case the string of the “text” parameter must be defined as follows



- text='1' → CHECK
- text='-1' → UNCHECK

Sample Code :	
original	
command	<code>gridview id=1, set, row=1, col=2, text='1'</code>
result	

#### Type 3 – three state check box cell


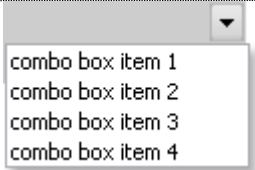
The three state check field is similar to the standard check box but it also allows the indeterminate state, consequently admissible values are as follows:

- text='1' → CHECK
- text='0' → INDETERMINATE
- text='-1' → UNCHECK
- text='-1' → UNCHECK

Sample Code :	
original	
command	<code>gridview id=1, set, row=2, col=3, text='0'</code>
result	

#### Type 4 – combo box cell

Assigning the option to a combo box requires creation of a string array with a string for each option and the first instance of the array must be established in the “options” parameters. The following example explains “how to” insert the options in a combo:

Sample Code :	
original	
array	<pre>string combo[4] combo[1]='combo box item 1' combo[2]='combo box item 2' combo[3]='combo box item 3' combo[4]='combo box item 4'</pre>
command	<pre>gridview id=1, set, row=4, col=4, options=combo[1]</pre>
result	

#### Type 5 - image cell



An image cell contains a bitmap picture that must be produced using a transparent background. See the section 9.1 for details of transparent bitmaps.

The "text" parameter has a particular syntax that permits specification of both the image layout and the image file, thus the admitted attributes are:



1. text="image.bmp" : define the bmp file and use the default layout
2. text="type#imagebmp" : define a specific layout and the bmp file

Type	Meaning
a	(default)the image have the column alignment
t	the image is tiled in the cell area
s	the image is stretched in the cell area

For example:

Sample Code :	
original	
command	<pre>gridview id=1, set, row=2, col=5, text='a#se2.bmp'</pre>
result	

This type of cell has particular behavior when the mouse cursor is over the cell; the image will be highlighted in a tooltip as show in the table below:

Normal	Mouse Over
	

### 9.22.6 – Fetch the content

Like the list control or edit button, in the grid view it's also possible to fetch the contents of one or more cells. To perform this action the "gridview" command must be used in the following manner:

Prototype :	<code>gridview fetch=&lt;s&gt;, id=&lt;i&gt;, row=&lt;i&gt;, column=&lt;i&gt;</code>
-------------	--

where:

Parameter	Description
<code>fetch</code>	Get the content of a cell, row, column or multi-selection area. This field requires an Eagle variable in which to save the appropriately formatted resulting string
<code>id</code>	Grid identifier
<code>column</code>	1-based value that indicates the column index;
<code>row</code>	1-based value that indicates the row index;

For example :

Sample Code :	<pre>string cells[100] gridview id=2,row=1,col=1,fetch=cells[1] gridview id=2,row=1,fetch=cells[1] gridview id=2,col=3,fetch=cells[1]</pre>
---------------	---

The fetch instruction can be executed in three different modes:

1. Get the content of a specific cell: both "row" and "column" parameters must be defined;

Sample Code :	<pre>string cells[100] gridview id=2,row=1,col=1,fetch=cells[1]</pre>
---------------	---

2. Include the content of an entire row: only the "row" parameter must be used;

Sample Code :	<pre>string cells[100]</pre>
---------------	------------------------------

```
gridview id=2,row=1,fetch=cells[1]
```

3. Obtain the content of an entire column: only the "column" parameter must be used;

Sample Code :

```
string cells[100]
gridview id=2,col=3,fetch=cells[1]
```

In the second and third case, the values of each cell are separated by a specific character defined in the "GRIDVIEW\_CELL\_SEPARATOR" entry of the configuration file. If this variable is not defined, the default value is "," comma.

INI Sample :

```
GRIDVIEW_CELL_SEPARATOR = |
```

The values returned in the string depend from the cell types:

- Edit : returns the current text;
- Check box : returns the string "1" if it's checked and "-1" if it's unchecked;
- Three state : returns the string "1" for checked, "0" for indeterminate and "-1" for unchecked;
- Combo box : returns the text of the current selected option;
- Image : no return.

The following sample gives details about a fetch on a grid row that contains all cell types:

Sample Code :

row



command

```
string cells[100]
gridview id=1,row=1,fetch=cells[1]
```

returned string

```
text 1 | 1 | 0 | combo box item 3 |
```

### 9.22.7 – Cell selection

Eagle V14 permits selection of a cell, row, column or multi selection where one or more cells can be selected using:

- the gridview command with the select option;
- the click of the *left button*;
- the click of the *left button* plus the *Control* key to select multi cells in arbitrary positions;
- the click of the *left button* plus the *Shift* key to select multi cells in consecutive positions;

- the *Return* key to select the next cell of the current selection.

It's important to note that selection also depends on the grid style, because for instance using styles 1 and 2 we can perform "single" cell selection, with styles 3 and 4 we can have entire row selections and finally with styles 5 and 6 only column selections are permitted.

The syntax of the gridview instruction to produce a selection can be modified as follows:

Prototype :	
	<code>gridview select, id=&lt;i&gt;, row=&lt;i&gt;, column=&lt;i&gt;</code>

where:

Parameter	Description
<code>select</code>	Select a cell, a row, a column or the entire grid
<code>id</code>	Grid identifier
<code>column</code>	1-based value that indicates the column index or -1 to obtain all the columns
<code>row</code>	1-based value that indicates the row index or -1 to obtain all the rows

For example :

Sample Code :	
	<pre> gridview select, id=2,row=5, col=3 gridview select, id=2,row=4, col=-1 gridview select, id=2,row=-1, col=5 gridview select, id=2,row=-1, col=-1 </pre>

Observing the select instruction it's possible to understand that we have various selection types:

STYLE	ROW	COLUM	SELECTION
1, 2	index	index	Single Cell
	-1	index	Column
	index	-1	Row
	-1	-1	Entire Grid
3, 4	index	index	Row
	-1	index	Entire Grid
	index	-1	Row
	-1	-1	Entire Grid
5, 6	index	index	Column
	-1	index	Column
	index	-1	Entire Grid
	-1	-1	Entire Grid

### 9.22.8 – Remove columns and rows

Finally, the ultimate action to analyze relates to the “delete” option. Using this parameter the developer can remove rows and columns. In this instance, the prototype now for the “gridview” command will be:

Prototype :

```
gridview delete, id=<i>, row=<i>, column=<i>
```

where:

Parameter	Description
Delete	Delete row or column
Id	Grid identifier
Column	1-based value that determinates the position of the column to remove. Use “column = -1” to delete all columns.
Row	1-based value that determines the position of the row to remove. Use “row = -1” to delete all rows.

For example:

Sample Code :

```
gridview delete, id=2,row=5
gridview delete, id=2, col=-1
gridview delete, id=2,row=-1, col=-1
```

If both the “column” and “row” parameters have -1 as the set value, the grid is completely erased, each rows and columns will be removed. Other than this previous instance, it is not otherwise permitted to set both parameters (with a value greater than or equal to 0) using the command in this way.

### 9.22.9 – Grid polling behavior

A grid handles many events but the most important are “selection” and “editing”. Consequently in Eagle 14 we have decided to separate the two events and generate two polling exits with different WT values:

- WT = 14 → indicates a polling exit for SELECTION
- WT = 15 → indicates a polling exit for EDITING. In this situation we are able to recognize if the text in an edit field has been modified, if the status of a check box or the selected combo item has been changed and so on.

Additionally, it is necessary to manage two separated events because sometimes selection and editing events are overlapped. For example, consider a check box cell in the UNCHECKED status, if the cell is not selected and the user click over the cell the event sequence is:

1. SELECTION → the cell keeps focus (status is UNCHECKED)
2. EDITING → the cell changes status to CHECKED.

If we only handle the selection event we cannot understand the real current status of the check box because the selection is executed before the editing event and consequently selection returns "UNCHECKED" in polling loop.

### 9.22.9.1 – During selection

The selection event returns different polling arguments depending on the grid style. Basically we can consider two macro classes: single selection and multiple selections.

The single selection is related to a SINGLE CELL selection for the grid styles 1 and 2, SINGLE ROW or SINGLE COLUMN in the other styles. Event returns are illustrated as follows :

- SINGLE CELL:
  - VB = 1
  - MN = panel identifier
  - BN = grid-view identifier
  - WT = 14
  - LN = row index
  - RN = column index
  - ST = cell content as shown in the section 9.22.6
- SINGLE ROW:
  - VB = 1
  - MN = panel identifier
  - BN = grid-view identifier
  - WT = 14
  - LN = row index
  - RN = 0
  - ST = content of each cell in the row, separated by GRIDVIEW\_CELL\_SEPARATOR.
- SINGLE COLUMN:
  - VB = 1
  - MN = panel identifier
  - BN = grid-view identifier
  - WT = 14
  - LN = 0
  - RN = column index

- ST = content of each cell in column, separated by GRIDVIEW\_CELL\_SEPARATOR.

Because SINGLE ROW and SINGLE COLUMN selection return in ST more than one argument it is possible to separate each returned value using the character specified in the "GRIDVIEW\_CELL\_SEPARATOR" entry of the configuration/INI file. If the variable is not defined, the default is "," the comma.

INI Sample :

```
GRIDVIEW_CELL_SEPARATOR = |
```

For example :

Sample Code :

```
ST = aaa|bbb|ccc|ddd|eee
```

Multiple selections are made by using the cell selection combined with the SHIFT or CONTROL keys. This type of selection always returns the same parameters except for the ST value that is structure depending on the grid style; the common parameters for a multiple selection are:

- VB = 1
- MN = panel identifier
- BN = grid-view identifier
- WT = 14
- LN = 0
- RN = 0
- ST = depending on grid style

The structure of ST for a multi selection in a grid with a style such as 1 or 2 is a collection of arguments separated by the "GRIDVIEW\_CELL\_SEPARATOR":

```
arg_1 | arg_2 | arg_3 | ... | arg_N
```

Other selection styles will have a separator to discriminate between cells that belong to different rows or columns. This distinction is operated by the introduction of another entry in the configuration file, called "GRIDVIEW\_LINE\_SEPARATOR". If the variable is not defined, the default value is ";" (semi-colon).

INI Sample :

```
GRIDVIEW_LINE_SEPARATOR = ;
```

In the instance of the grid where selection "by row" using (style 3, 4) is made, the ST parameter contains single collections for each row separated with the "GRIDVIEW\_LINE\_SEPARATOR":



```
arg_1_row_1|...|arg_N_row_1;arg_1_row_2|...|arg_N_row_2;...;
arg_1_row_M|...|arg_N_row_M
```

An analog structure has been defined for the grid with selections “by column” (style 5, 6):

```
arg_1_col_1|...|arg_N_col_1;arg_1_col_2|...|arg_N_col_2;...; arg_1_col_M|...|arg_N_col
_M
```

#### 9.22.9.2 – Cell editing complete

When the user modifies the contents of a cell, Eagle returns the event on the polling loop with the following parameters:

- VB = 1
- MN = panel identifier
- BN = grid-view identifier
- WT = 15
- LN = row index
- RN = column index
- ST = new cell content as shown in the section 9.22.6

As has been said previously, the difference between the editing event and the single cell selection event is the WT value (15 instead 14).

## 10 – Dialogs and Message Boxes

In graphical user interfaces, a dialog (or dialogue) box is a special window, used in user interfaces to display specific information to the user, or to get a specific response if needed. They are so-called because they form a dialog between the computer and the user—either informing the user of something, or requesting input from the user, or both. These controls are generally standard in interface design allowing you to specify how to carry out an action.

In Eagle v14, different types of dialog boxes are used for particular classes of user interaction :

- notify messages
- prompt dialog
- font selection dialog
- file browser dialog
- color picker dialog

As you would expect, dialog boxes are painted according to the currently selected theme.

### 10.1 – Notify

We have already seen the “Color Picker” dialog when we have introduced the “Color Picker” button, but the simplest type of dialog is the “notify”, which displays a message and only requires a response to a message, usually clicking a button. Messages are used to provide simple confirmation of an action or to include program termination notices or confirmation information due to malfunction or intentional closing of an operation by the user.

Prototype :

```
notify      t=<text, title=<text>, default=1|2|3
            data block(s);
```

where:

Parameter	Description
<b>t</b>	Text denoting the type of the notify box. It can be: <ul style="list-style-type: none"> <li>= err      error message</li> <li>= inf      information message</li> <li>= mes      simple message</li> <li>= war      warning message</li> <li>= que      question message</li> <li>= xque     extended to three state question message</li> <li>= xwar     extended to three state warning message</li> </ul> <p>A variable cannot be used for this field.</p>
<b>title=&lt;text&gt;</b>	Title displayed in the caption
<b>default</b>	Default button : <ul style="list-style-type: none"> <li>default = 1 → “Ok” or “Yes”</li> <li>default = 2 → “No”</li> <li>default = 3 → “Cancel”</li> </ul>

**data block(s)** The text of the message. Each line should be in the form of <message>, with a semi-colon used to terminate the last line.

For example :

Sample Code :

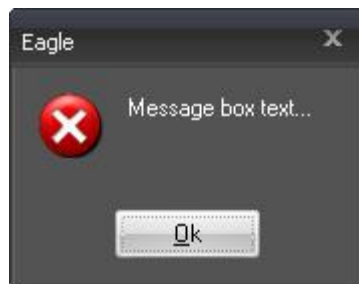
```
notify t=war
'hey ! are you sure about this ?'
'what you are doing could take a long time ...';

notify t=err,f
'you missed the target. try again !';

notify t=que,h=save.hlp
'do you want to rewrite this file ?';
```

Developers can display a message on the screen creating one of the next type of boxes :

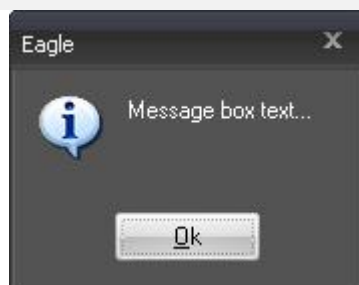
#### ERR - ERROR MESSAGE



Notify usually is used to report an error occurrence during execution. It contains the message, the error icon and the 'OK' button only.

```
notify t=err; 'Message box text...';
```

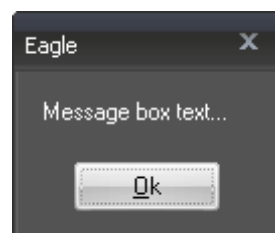
#### INF - INFORMATION MESSAGE



Used to inform the user about actions or events. Similar to the previous type, the control has a text, an 'OK' button and the information icon.

```
notify t=inf; 'Message box text...';
```

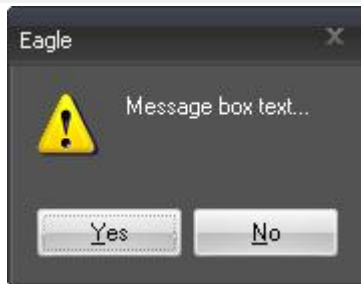
#### MES - SIMPLE MESSAGE



Simple generic message which only displays text and an 'OK' button.

```
notify t=mes; 'Message box text...';
```

#### WAR - WARNING MESSAGE



Message to show critical information. A dialog contains two buttons, 'Yes' and 'No', a message and a warning icon.

`notify t=war; 'Message box text...';`

#### QUE - QUESTION MESSAGE

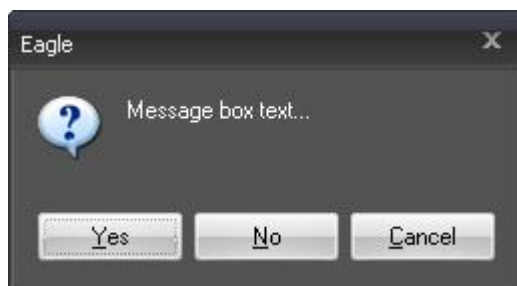


Dialog that is used to confirm or abort an operation.

It's similar to the warning message but the icon is the classical question mark.

`notify t=que; 'Message box text...';`

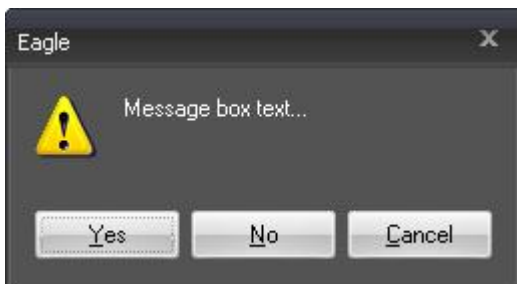
#### XQUE - THREE STATE QUESTION MESSAGE



Dialog which is extension of the question message to permit, abort or cancel a process. To provide the three-state functionality, the dialog has a 'Yes', 'No' and 'Cancel' button. Like the others boxes this dialog has text and a question icon

`notify t=xque; 'Message box text...';`

#### XWAR - THREE STATE QUESTION MESSAGE



Dialog which is extension of the warning message to permit, abort or cancel a process. To provide the three-state functionality, the dialog has a 'Yes', 'No' and 'Cancel' button. Like the others boxes this dialog has text and a question icon

`notify t=xwar; 'Message box text...';`

When a message box is displayed the user can response to the notification by just pushing a button (For example "OK" to validate). These actions set a flag (ifyes, ifno, iferr or vb = 9), so developers could understand the action performed after the message has been closed:

#### ERR - ERROR MESSAGE

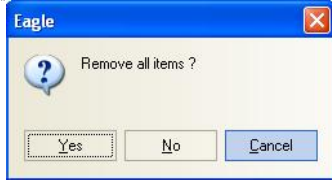
**Ok Button** Set the variable ifyes and ifok to true, vb = 1

**Escape** Set the variable ifyes and ifok to true, vb = 1

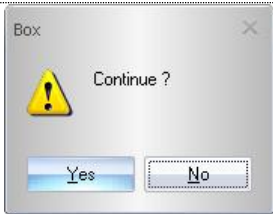
#### INF - INFORMATION MESSAGE

Ok Button	Set the variable <b>ifyes</b> and <b>ifok</b> to true, <b>vb</b> = 1
Escape	Set the variable <b>ifyes</b> and <b>ifok</b> to true, <b>vb</b> = 1
MES - SIMPLE MESSAGE	
Ok Button	Set the variable <b>ifyes</b> and <b>ifok</b> to true, <b>vb</b> = 1
Escape	Set the variable <b>ifyes</b> and <b>ifok</b> to true, <b>vb</b> = 1
WAR - WARNING MESSAGE	
Yes Button	Set the variable <b>ifyes</b> and <b>ifok</b> to true, <b>vb</b> = 1
No Button	Set the variable <b>ifno</b> to true, <b>vb</b> = 1
QUE - QUESTION MESSAGE	
Yes Button	Set the variable <b>ifyes</b> and <b>ifok</b> to true, <b>vb</b> = 1
No Button	Set the variable <b>ifno</b> to true, <b>vb</b> = 1
XQUE - THREE STATE QUESTION MESSAGE	
Yes Button	Set the variable <b>ifyes</b> and <b>ifok</b> to true, <b>vb</b> = 1
No Button	Set the variable <b>ifno</b> to true, <b>vb</b> = 1
Cancel Button	Set the variable <b>vb</b> to 9
Escape	Set the variable <b>vb</b> to 9
XWAR - THREE STATE WARNING MESSAGE	
Yes Button	Set the variable <b>ifyes</b> and <b>ifok</b> to true, <b>vb</b> = 1
No Button	Set the variable <b>ifno</b> to true, <b>vb</b> = 1
Cancel Button	Set the variable <b>vb</b> to 9
Escape	Set the variable <b>vb</b> to 9

The following examples explain two case studies for notify and how the returned values could be possibly be used :

Sample Code :	
	Create a notify "ext"
	<code>notify t=ext; 'Remove all items ?';;</code>
	Push the "Cancel" button
	
	Verify the vb variable
	<code>tell vb</code>
Output	
	9

Sample Code :	
	Create a notify "war"

notify t=war, title='Box', default=2; 'Continue ?';;
Push the "Yes" button

Verify the ifyes variable
ifyes tell 'yes'
Output
yes

The position of the dialogue box is relative to the center of the main frame and the style depends on the current defined theme:

Office 2000	Office 2003	Office 2007 Standard R1	Office 2007 Luna Blue R2
			

## 10.2 – Prompt command

Using the "prompt" command, Eagle can generate a command line interface ready to accept typed data. An input panel for this purpose based on Windows widgets can be activated to request information from the user. Using the keyboard, the user then has the possibility to enter data and confirm entry with the <return> key. If the "cancel" button is chosen then set the "IFNO" indicator. A default string is placed in the input area and it can be confirmed, edited or erased using the keyboard.

Prototype :

**prompt**      d=<svar>, p=<message>:<varlist>

where:

Parameter	Description
<b>d</b>	The default value.
<b>p</b>	The text of a question to be answered by the keyboard operator.
<b>&lt;varlist&gt;</b>	A list of arithmetic, points or string variables into which user answers are




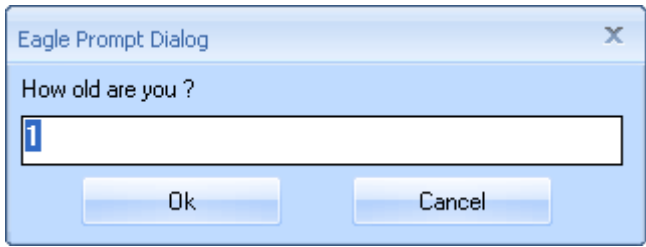
recorded.

For example :

Sample Code :

```
prompt p='Give the pizza type': margherita
prompt p='What comes after a in the alphabet': b
prompt d='170558', p='Enter your birthdate': bdate
prompt d='white',p='Wall color': wall_color
prompt d='^color',p='Wall colour': wall_color
```

The prompt dialog is presented rendered to the currently defined theme some of which are illustrated in the following table of cases:

Office 2000	
Office 2003	
Office 2007 Standard R1	
Office 2007 Luna Blue R2	

When the prompt dialog has created, Eagle handles different events:

- Press the “Enter” key or push the “Ok” button
- Press the “Escape” key or push the “Cancel” button

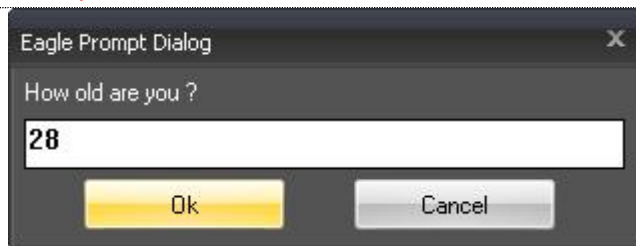
The example below describes how this type of dialog could possibly operate:

Sample Code :

Create a prompt

```
numeric years
years = 1
prompt d=years, p='How old are you ?':years
```

Insert '28' and push the “Ok” button



Verify the variable

```
tell years
```

Output

```
28
```



### 10.3 – Font Dialog

Another common dialog is the “Font dialog” or font selection dialog, which enables a dialog to facilitate user selection of particular font type and style. Eagle provides the “winfont” command for the purpose of selecting the current Windows font :

Prototype :

<b>winfont</b>	none
----------------	------

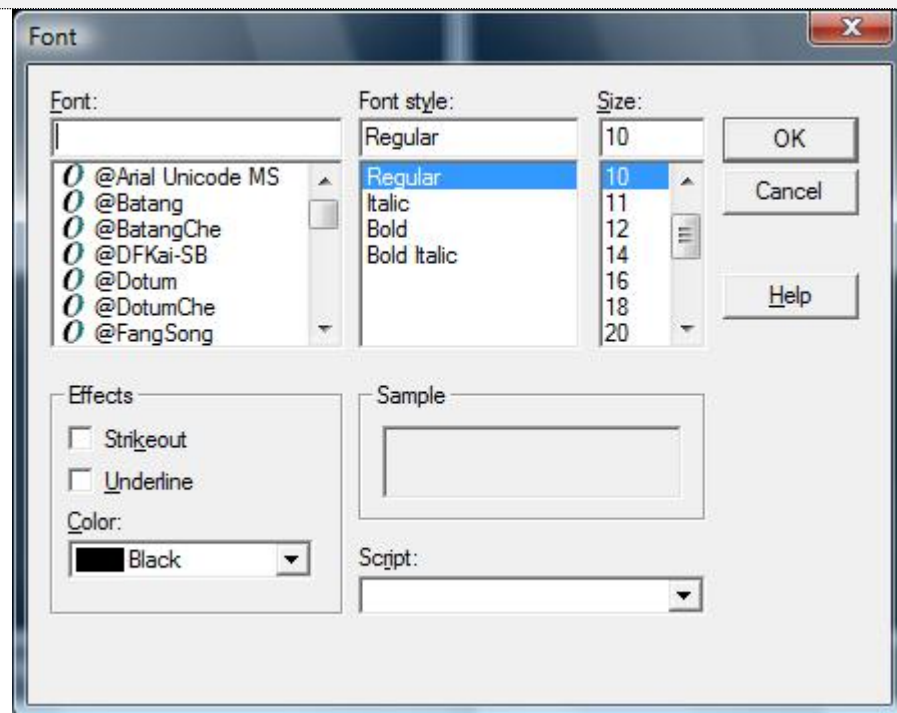
where:

Parameter	Description
none	Activate the font selection common dialog for interactive font selection from those installed on the system. The character height returned by the font selection dialog box is managed by the system matching the best font setting and can be different from the selected size in the dialog.

For example:

Sample Code :

<b>winfont</b>
----------------



## 10.4 – File dialog

Usually a Windows application provides a standard dialog from which a file can be opened or saved. Eagle V14 extends this standard functionality by enabling the display of two different types of file dialog:

1. Operating System based file dialog;
2. Eagle file dialog.

### 10.4.1 – Operating System file dialog

Eagle provides the “file” function together with the “os” parameter for management of the “Open” and “Save As” dialog boxes; the appearance of these dialogues are based on the template as defined by the Operating System. This function command opens a panel which enables the user to browse the file system for required files:

Prototype :

```
file      <svar>, os
          {,cwd='<directory>' {,flt='<wildspec>' {,full=yes|no }},{saveas}
```

where:

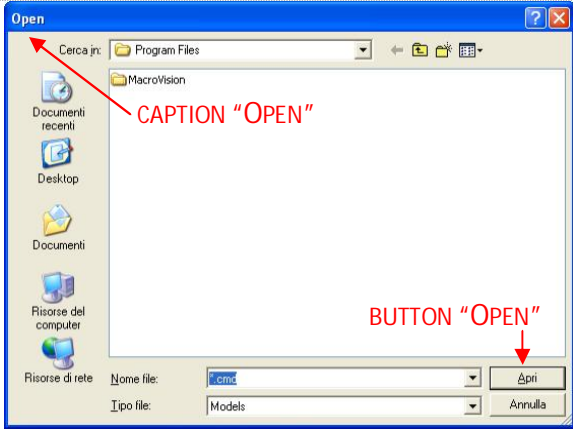
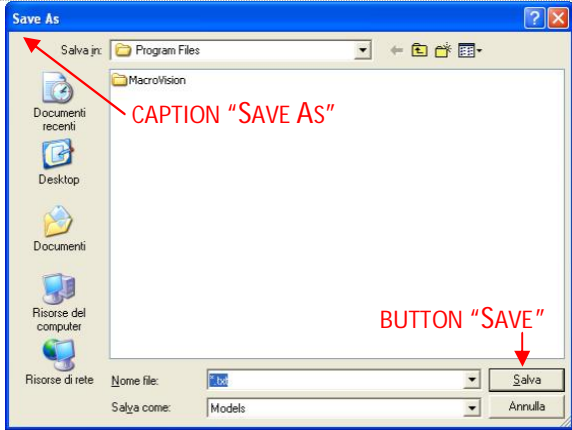
Parameter	Description
<svar>	The string variable where the filename is stored.
os	Required. Enable the Operating System file dialog
cwd	The starting directory ('<directory>' in quotes) name from which the files list is to be created. The default value for the is ('-')the current directory.
flt	A string that specifies a filter to select file list names. The default value is '*.*'.
full	Yes returns the full pathname of files found. No returns the filename only. The default is no.
saveas	This primer calls the Save As version of the panel option.

For example :

Sample Code :

```
file s1, os
file f_name, os, cwd='.', flt='*.cmd', full=y
```

The differences between "Open" and "SaveAs" dialogs are presented in the next table:

Open	Save As
	
file os, cwd='C:\Program Files', flt='*.cmd'	file os, cwd='C:\Program', flt='*.txt', saveas

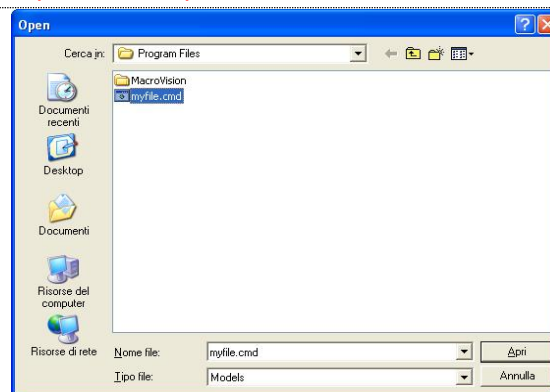
The next example describes how to obtain a string that contains the full path of an opened file :

Sample Code :

Create a prompt

```
string name
file name, os, cwd='C:\Program Files', flt='*.cmd', full=yes
```

Select a file and press the "Open" button



Verify the variable

```
tell name
```

Output

```
//C:/Program Files/myfile.cmd
```

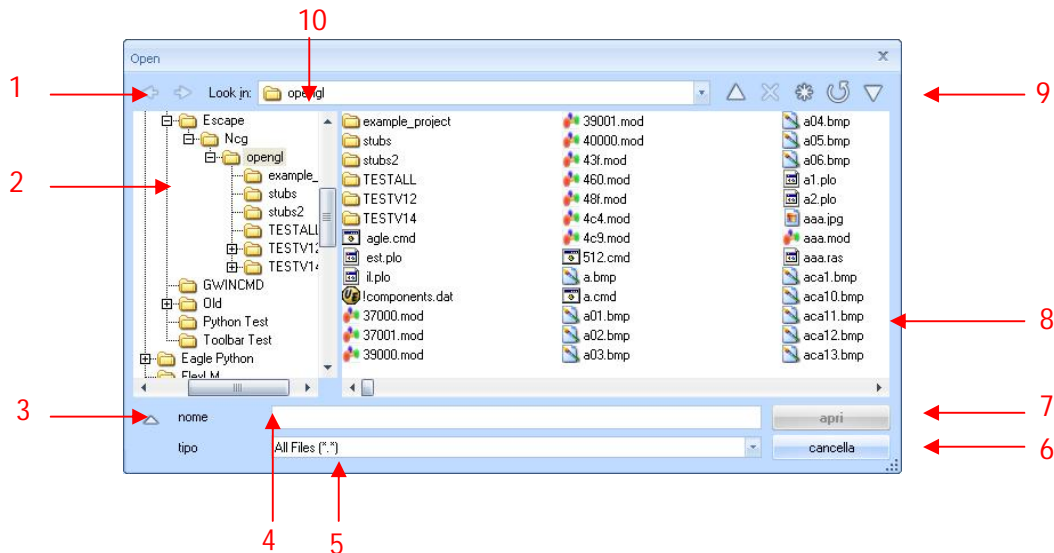
The limitation of this solution is that the presented dialog doesn't consider the current defined theme and therefore will be always be displayed using the OS template.

## 10.4.2 – Eagle file dialog

If we want to have a file dialog skinned with the current theme, then we instead have to use the Eagle file dialogues. The same “file” command is again used to create these types of dialog. IN the case of an Eagle dialog however it is possible to have an extended set of dialog types:

1. Open a single file (default);
2. Open multiple files;
3. Save As

The resulting dialog looks similar to the Windows Explorer, with a shell tree view on the left and the shell list view on the right. The persistence state is supported for the following elements and values: the window and splitter positions, selected folder in the tree view, splitter position, list view mode, column widths and sorting rules. The new file dialog is composed by the following illustrated elements:



1. Back and Forward Folder buttons;
2. Tree view of the file system with a splitter to resize the tree dimension;
3. Collapsing button to switch to the collapsed view of the dialog;
4. File name edit;
5. Filter combo box;
6. Cancel button;
7. Open or Save button;
8. File browser;
9. Control buttons (Up One Level, Delete, Create New Folder, Refresh and View Menu);
10. Current folder combo box.

The alternative collapsed view (triggered by option 3 above) is represented with following layout:



1. Expanding button to switch back to the full dialog;
2. File name edit;
3. Filter combo box;
4. Cancel button;
5. Open or Save button.

In this case the syntax of the “file” function will have the following syntax:

Prototype :	
file	<pre>           &lt;svar&gt;           ,title='title'           ,cwd='&lt;directory&gt;' ,flt='&lt;wildspec&gt;' ,full=yes no           ,{openmulti,saveas}           ,tree ,collapsed         </pre>

where:

Parameter	Description
<svar>	The string variable where the filename is stored.
title	Title of the dialog
cwd	The starting directory ('<directory>' in quotes) name from which the files list is to be created. The default value for the is ('./') the current directory.
flt	A string that specifies a filter to select file list names. The default value is '*.*'.
full	Yes returns the full pathname of files found. No returns the filename only. The default is no.
openmulti	This primer calls the Open Multiple File version of the panel option.
saveas	This primer calls the Save As version of the panel option.
tree	Insert to display a tree in the left side to browse the file system.
collapsed	Initially show the dialog in the collapsed state.

For example:

Sample Code :	
	<pre> file s1 file name, collapsed file f_name, cwd='.', flt='*.cmd', tree         </pre>

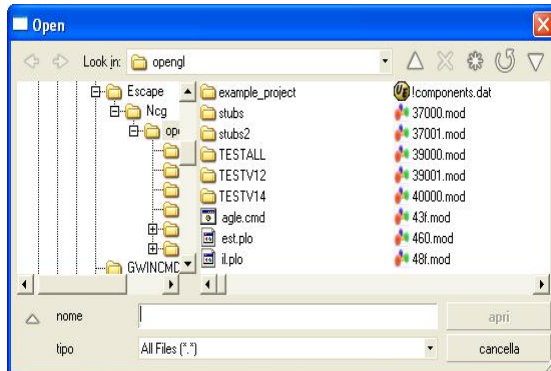
Eagle’s file dialog has a variety of customization possibilities which are accessed directly using the file function for specific dialog features or by settings in the INI configuration file which can be used when the relevant setting is designed to be general for all the file dialogs present in the application:

- Tree view

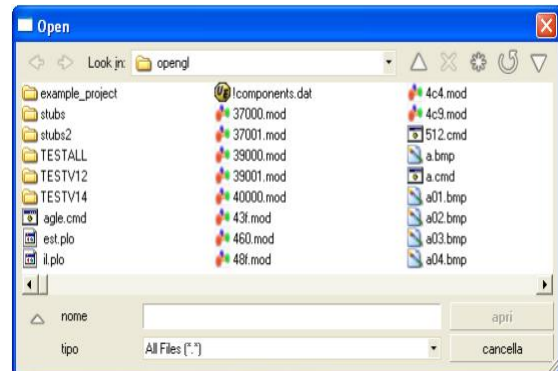
the tree entry in the “file” syntax will show a tree view used for browsing within the file system;

Sample Code :

```
file ..., tree
```



WITH TREE



WITHOUT TREE

- Starting collapsed

the “collapsed” parameter in the “file” instruction enables initial creation of the dialog in the collapsed status;

Sample Code :

```
file ..., collapsed
```

- Open button

the “FILE\_DIALOG\_OPEN\_BUTTON” entry in the configuration file can be used to define the text of the “Open” button when type selected is “open single file” or “open multiple files”. If no text is defined, the default will be applied.

INI Sample :

```
FILE_DIALOG_OPEN_BUTTON = Open File
```

- Save button

when a “Save” or “Save As” dialog has been created, it is possible to set the “FILE\_DIALOG\_SAVE\_BUTTON” entry in the configuration file to define the text of the “Save” button. If no text is defined, the default will be applied.

INI Sample :

```
FILE_DIALOG_SAVE_BUTTON = Save!
```

- Cancel button

the configuration file "FILE\_DIALOG\_CANCEL\_BUTTON" entry defines the "Cancel" text for each type of file dialog. If no text is defined, the default will be applied.

INI Sample :

```
FILE_DIALOG_CANCEL_BUTTON = Exit
```

- File name label

using the "FILE\_DIALOG\_FILENAME\_LABEL" entry in the INI file we can set the label before the edit field that contains the file name. If no text is defined, the default will be applied.

INI Sample :

```
FILE_DIALOG_FILENAME_LABEL = File select...
```

- Title

the newly introduced "title" parameter for the "file" command can be used to define a title for the dialog. if no title is added then the default is used (for instance Open for open dialog).

Sample Code :

```
file ..., title='File dialog...'
```

- File type label

the "FILE\_DIALOG\_FILETYPE\_LABEL" entry in the IN file enables setting of the label before the combo box field that contains the permitted file extensions. If no text is defined, the default will be applied.

INI Sample :

```
FILE_DIALOG_FILETYPE_LABEL = Filters
```

- Separator character

when the file dialog permits selection of more than one item, the string returned to the variable will contain all the selected files separated by a predefined character. It is possible to define a custom separator by using the "FILE\_DIALOG\_SEPARATOR" entry in the configuration file. If no text is defined, the default comma (,) separator will be applied.

INI Sample :

```
FILE_DIALOG_SEPARATOR = @
```

- File Extension Filter

the 'FILEDIALOG\_FILETYPES' entry in the configuration file is used to set the file extensions to be displayed in the combo box filter. The relevant syntax to define the filter string is :

```
Type_1 | Type_2 | ..... | Type_N |
```

where each "Type" must be defined in the following way:

```
Description (*.ex1, *.ext2, ... , *.exN) | *.ex1; *.ext2; ... ; *.ex
```

When no filter is defined, the default " All Files (\*.\*) / \*.\*" will be applied.

INI Sample :

```
FILEDIALOG_FILETYPES = Portable Network Graphics  
(*.png;*.mng) | *.png; *.mng | GIF Files (*.gif) | *.gif | All Files  
(*.*) | *.* |
```

Furthermore setting the 'FILEDIALOG\_FILETYPES\_DEFAULT' option in the INI file, specifies the default index type from FILEDIALOG\_FILETYPES to use when the dialog is opened.

INI Sample :

```
FILEDIALOG_FILETYPES_DEFAULT = 3
```



## 10.5 – Balloon Message

The Balloon represents another way of displaying a message to users. Actually a balloon message could be considered as being quite similar to a message box, but with more features such as the shape, definable position and a timer set before auto-hiding the message.

Balloons are one way of displaying messages in a non-obtrusive way. Balloons are taking precedence over message boxes for the display of small messages as they avoid the user from having to click an “OK” button whilst still getting the message across. These kinds of message balloons are becoming a part of standard user interface with MSN explorer and Windows XP where for instance the tray icons may ask the system tray to display a balloon message to the user when a particular event has occurred. Unlike the Window's tray system, Eagle V14 allows presentation of more than one balloon message at the same time.

The Eagle “balloon” command is used to create balloon messages. The command has the following syntax:

Prototype :

```
balloon icon=<text>, caption=<text>, message=<text>, timer=<num>,  
x=<num>, y=<num>
```

where:

Parameter	Description
<b>icon</b>	Text denoting the icon type in the message. It can be: = err error icon = inf information icon = war warning icon A variable cannot be used for this field.
<b>caption</b>	Title displayed in the caption
<b>message</b>	The text of the message. Each line should be in the form of <message>, with a semi-colon used to terminate the last line.
<b>timer</b>	A decimal value that correspond to the time in second, or fraction of second.
<b>x,y</b>	Coordinates where the balloon will be displayed, if no coordinates are defined the message will be shown on the center of the application frame.

For example :

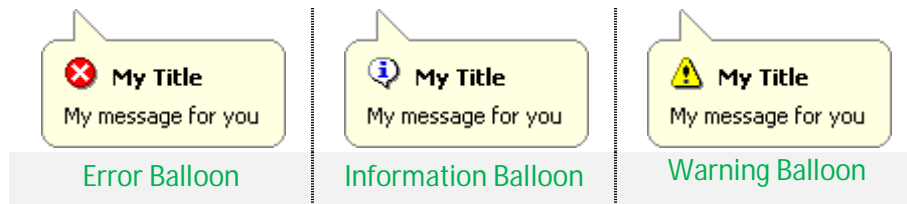
Sample Code :

```
balloon icon=err, caption='Title', message='text message', timer=3  
balloon i=err, c='MyTitle', m='my personal text for you', t=0.5, x=10, y=10
```

Another difference with the message box is that balloons are thread independent and consequently they do not return any value to polling loop.

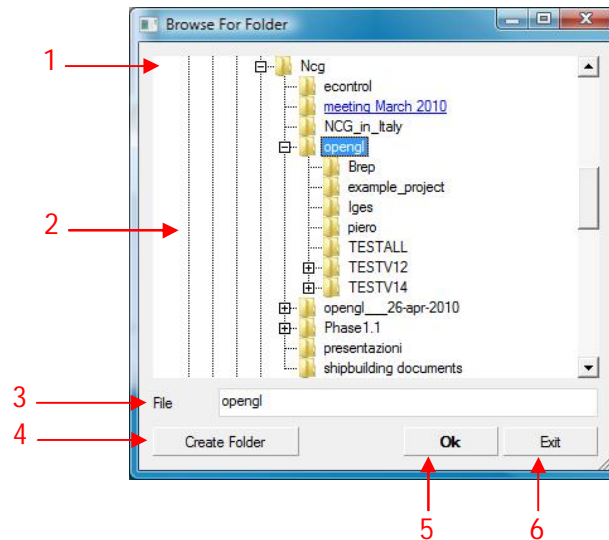
When the message is displayed the user has two possibilities to remove it, either wait until the set timer expires or click on the balloon.

As shown In the syntax, it's possible to chose between three icon's types :



## 10.6 – Folder Dialog

Eagle V14 provides a dialog box specially provided to allow a user select a specific folder if the programmer judges it necessary that the application needs this type of selection feedback. This dialog box enables the user to select a Shell folder and it appears as follows:



1. Text Message to help the user;
2. Tree view of the file system;
3. Folder main edit:
4. New folder button;
5. Ok button;
6. Cancel button.

When this dialog box appears, it displays the current selected folder, the root being the Desktop directory and all other available folders can be located from this one. In order to use it, the user clicks one of the folders or drives and then clicks the OK button. If the desired folder is initially unseen, but is available, the user can then expand the folders and drives listing, click the desired folder and then click OK. If the required folder is not existent, the user can in this instance first select an existing folder or drive, click the New Folder button, type a name for the new folder, and finally click OK to create the new location.

The Eagle “folder” function is used to display the browse folders dialog, the specifics of the syntax of this command are expanded below:

Prototype :

<code>folder</code>	<code>&lt;svar&gt;</code> <code>,title='title' ,cwd='&lt;directory&gt;', description = ' message',change</code>
---------------------	--

where:



Parameter	Description
<svar>	The string variable where the folder path name is stored.
title	Title of the dialog. If no title has been defined, the default will be applied.
cwd	The starting directory ('<directory>' in quotes) name from which the files list is to be created. The default value for the is ('-') the current directory.
description	The text of the message (in single quotes) to help the user. If no message has been defined, the note string remains empty.
change	A flag to change the current directory when the dialog is closed through the Ok button.

For example :

Sample Code :

```
string pathname
folder pathname, title='My Browser', desc='Search the Image Directory'
folder pathname, change
```

The persistence state is supported for the window position and selected folder. All the controls of this dialog can be customized as follows:

- New Folder button

it is possible to define the string for the "New Folder" button using the "FOLDER\_DIALOG\_NEW\_BUTTON" entry in the configuration file. If no text is defined, the button will be removed from the dialog.

INI Sample :

```
FOLDER_DIALOG_NEW_BUTTON = Create Folder
```

- Ok button

the "FOLDER\_DIALOG\_OK\_BUTTON" entry in the configuration file can be used to define the text of the "OK" button. If no text is defined, the default will be applied.

INI Sample :

```
FOLDER_DIALOG_OK_BUTTON = Select
```

- Cancel button

the "FOLDER\_DIALOG\_CANCEL\_BUTTON" entry in the configuration file is used to set the text of "Cancel" button. If no text is defined, the default will be applied.

INI Sample :

```
FOLDER_DIALOG_CANCEL_BUTTON = Exit
```

- Folder name label

using the "FOLDER\_DIALOG\_EDIT\_LABEL" entry in the INI file we can set the label before the edit field that contains the selected path name. If no text is defined, the label and the edit field will be removed from the dialog

INI Sample :

```
FOLDER_DIALOG_EDIT_LABEL = File select...
```

- Title

another parameter for the new "folder" command can be used to define a title for the dialog. If no title is defined the default one is used.

Sample Code :

```
folder ..., title=Folder dialog...'
```

- Description

the "description" parameter permits insertion of text to describe the meaning of the folder dialog that has been opened. The text can contain two lines of text which must be separated by the "MULTI\_COLUMN\_LIST\_SEPARATOR". If none is defined the default one is used.

INI Sample :

```
MULTI_COLUMN_LIST_SEPARATOR = |
```

Sample Code :

```
folder ..., desc='line 1 on folder dialog|line 2...'
```

## 11 – ActiveX

Before reading this section it is advisable to have at least a basic knowledge of Microsoft ActiveX technology. ActiveX is a framework for defining reusable software components (known as controls) that perform a particular function or a set of functions in a way that is independent of the programming language used to implement them. You can find more details about ActiveX in the relevant section of MSDN - the "Microsoft Developer Network".

Many Microsoft Windows applications; including many of those from Microsoft itself, such as Internet Explorer, Microsoft Office, Microsoft Visual Studio, and Windows Media Player; use ActiveX controls to build their feature-set and also encapsulate their own functionality as ActiveX controls which can then be embedded into other applications. Internet Explorer also allows embedding ActiveX controls onto web pages.

ActiveX controls could be considered like small program building blocks which can serve creation of distributed applications, examples include customized applications for gathering data, viewing certain kinds of files, and displaying animation.

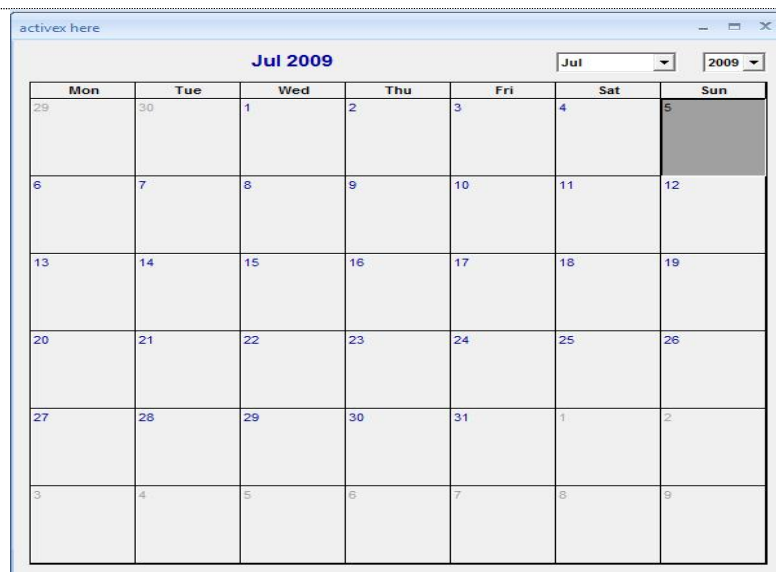
Eagle version 14 allows installation of ActiveX controls in a document window (gwindow) or as an option in a panel (usually a dialog bar).

First of all the ActiveX has an identifier (ProgID) that distinguishes registered controls and the developer uses this identifier to install the control in a document window:

Sample Code :

Create a gwindow that display the "Calendar" ActiveX

```
gwin 2, pos = 100,100, w = 600, h = 600 ,t = 'activex here', progid = '
MSCAL.Calendar.7'
```



or alternatively in a button:

Sample Code :

Tab file for a panel that contains the "Calendar" ActiveX

```
55,2,8,400,280,0,0,320,'MSCAL.Calendar.7';;
```



Secondly an ActiveX object is organized with interfaces that provide a set of functions with which the control can be managed. Using an external Eagle utility tool (ActiveX2Eagle.exe) create series of command files (\*.cmd) is created, also called an "Eagle Stubs Library", which contains the properties and functions exposed by the selected ActiveX needed to control it from within Eagle..

In the "Command Prompt" we call the ActiveX2Eagle executable specifying both the folder in which save the files and the ActiveX identifier using the following syntax:

Prototype :

```
ActiveX2Eagle [destin.directory] [ActiveX ProgID]
```

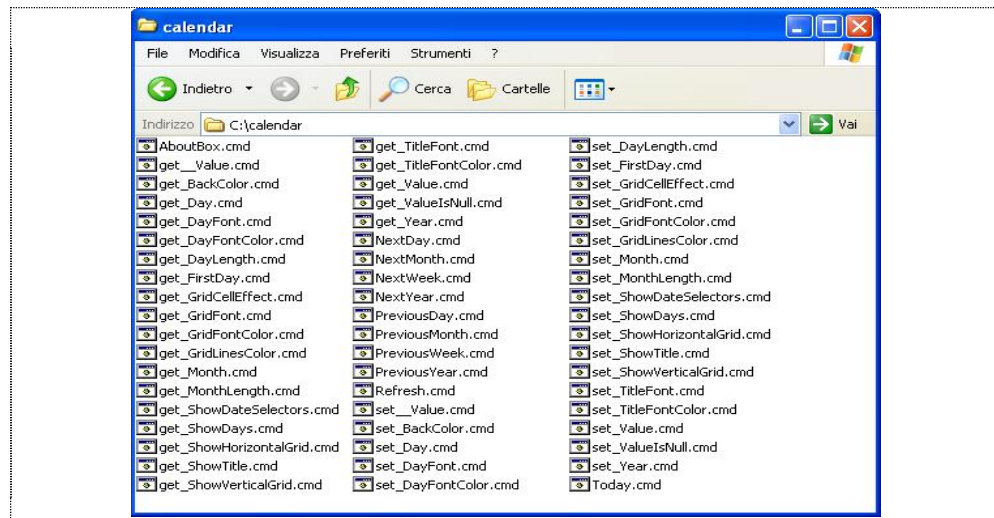
For example:

Sample Code :

Create Stubs files for the "Calendar" ActiveX

```
ActiveX2Eagle c:\calendar\ MSCAL.Calendar.7
```





The created command files can be called like any standard cmd file using the relevant parameters.

The first parameter is always the index of the ActiveX object, so in the case of a control inside a gwindow the index is the Eagle window Id, instead of the option in a tab file the index is calculated using the following formula:

$$(\text{panel\_id} * 2^{**}8) + \text{button\_id}$$

The following table presents two examples to clarify how to use stubs functions :



string ret  
numeric id  
id = 3  
do  
PreviousDay(id,@ret)



string ret  
numeric id  
id = (7 \* 2^{\*\*}8) + 55  
do NextDay(id,@ret)



## 12 – Eagle window inside a panel

Future Data

## 13 – How to customize controls via DLL

Future Data

## Appendix A – Configuration file settings

This section includes and A to Z catalog of GUI entries for the Eagle INI Configuration file:

	<b>BAR_CLOSE_ACTION</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	In menu bar and toolbar floating, defines the action to call when the "X" button in the caption is pressed
Default	no action
Sections	7.1, 8.6, 8.8
Example	BAR_CLOSE_ACTION = C:\MyFolder\closebar.cmd
	<b>BAR_DOCKING_ENABLED</b>
Values	no   yes
Description	To enable or disable the undocking mechanism of the menu bar
Default	yes
Sections	7.1
Example	BAR_DOCKING_ENABLED = no
	<b>BAR_EXPANDABLE</b>
Values	no   yes
Description	If "YES" new features for the menu are enabled, then all menus and popups in the application will be displayed with hidden items, the chevron button and exhibit the display behavior that consists of a short delay before displaying the least used options. If "NO" every menu will be displayed using the standard presentation without any of the new elements.
Default	yes
Sections	7.1, 7.2
Example	BAR_EXPANDABLE = no
	<b>BITMAP_TRANSPARENCY_COLOR</b>
Values	the range of possible values is : #000000 to #FFFFFF
Description	Color to define the transparent background for the bitmap images
Default	#FF00FF
Sections	9.1
Example	BITMAP_TRANSPARENCY_COLOR = #335500
	<b>BUTTON_IMAGE_BORDER</b>
Values	the size in pixel
Description	Size in pixel for the border in a "only icon" image button when the bitmap doesn't have a transparency
Default	0
Sections	9.6
Example	BUTTON_IMAGE_BORDER = 2

	<b>COMMAND_BACKGROUND_COLOR</b>
Values	the range of possible values is : #000000 to #FFFFFF
Description	Change the color of the background (#RGB) in the command bar; the color will be specified using a "#" followed with three hexadecimal values (one for red, one for green and one for blue).
Default	#000000
Sections	5.1
Example	COMMAND_BACKGROUND_COLOR = #335500

	<b>COMMAND_TEXT_COLOR</b>
Values	the range of possible values is : #000000 to #FFFFFF
Description	Change the color of the text (#RGB) in the command bar; the color will be specified using a "#" followed with three hexadecimal values (one for red, one for green and one for blue).
Default	#FFFFFF
Sections	5.1
Example	COMMAND_TEXT_COLOR = #000099

	<b>COMMAND_TITLE</b>
Values	a string
Description	Title displayed on the caption of the command bar
Default	no title
Sections	5.1
Example	COMMAND_TITLE = "Command"

	<b>COMMAND_WINDOW_FONT_NAME</b>
Values	the name of a font
Description	Set font used in the command window
Default	Courier
Sections	5.1
Example	COMMAND_WINDOW_FONT_NAME = "Calibri"

	<b>COMMAND_WINDOW_FONT_SIZE</b>
Values	the height in pixel for a font
Description	Sets the font size used in the command window
Default	10
Sections	5.1
Example	COMMAND_WINDOW_FONT_SIZE = 12

	<b>COMMAND_WINDOW_TITLE</b>
Values	no   yes
Description	Defines whether or not to use the Eagle command window caption or title bar
Default	no
Sections	5.1
Example	COMMAND_WINDOW_TITLE = yes

	<b>DEFAULT_GWINDOW</b>
Values	no   yes
Description	Defines if the first gwindow has created at start up
Default	no
Sections	4.2
Example	DEFAULT_GWINDOW = yes

	<b>DOCKING_MARKERS_TYPE</b>
Values	BYTHEME, STUDIO2003, STUDIO2005, STUDIO2008XP, STUDIO2008VISTA
Description	Set the docking markers style and behavior
Default	BYTHEME
Sections	3.2
Example	DOCKING_MARKERS_TYPE = STUDIO2005

	<b>EAGLE_DOCUMENT_ICON</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	Permits setting of the icon for GWindows
Default	no icon
Sections	4.2
Example	EAGLE_DOCUMENT_ICON = C:\Mylcon\gwindoc.ico

	<b>EAHEIGHT</b>
Values	a number
Description	Represents the window height at start up
Default	300
Sections	3.3
Example	EAHEIGHT=700

	<b>EAWIDTH</b>
Values	a number
Description	The start up width of the main window
Default	600
Sections	3.3
Example	EAWIDTH=900

	<b>EDIT_BORDER_ENABLED</b>
Values	a character
Description	Enables setting of the “border” style in the edit field
Default	no
Sections	9.8
Example	EDIT_BORDER_ENABLED= yes

	<b>FILE_DIALOG_CANCEL_BUTTON</b>
Values	a string
Description	Define the string for the “Open” button in the File Dialog
Default	“Open”
Sections	10.4.2
Example	FILE_DIALOG_OPEN_BUTTON = Open File

	<b>FILE_DIALOG_FILENAME_LABEL</b>
Values	a string
Description	Define the label before the edit field that contains the file name in the File Dialog
Default	“File name:”
Sections	10.4.2
Example	FILE_DIALOG_FILENAME_LABEL = File select...

	<b>FILE_DIALOG_FILETYPE_LABEL</b>
Values	a string
Description	Define the label before the combo box that contains the file types in the File Dialog
Default	“File of type:”
Sections	10.4.2
Example	FILE_DIALOG_FILETYPE_LABEL = Filters

	<b>FILE_DIALOG_OPEN_BUTTON</b>
Values	a string
Description	Define the string for the “Open” button in the File Dialog
Default	“Open”
Sections	10.4.2
Example	FILE_DIALOG_OPEN_BUTTON = Open File

	<b>FILE_DIALOG_SAVE_BUTTON</b>
Values	a string
Description	Define the string for the “Cancel” button in the File Dialog
Default	“Cancel”
Sections	10.4.2
Example	FILE_DIALOG_CANCEL_BUTTON = Exit

	FILE_DIALOG_SEPARATOR
Values	a string
Description	Define the character to separate the file name returned when the multiple selections are enabled in the File Dialog,
Default	,
Sections	10.4.2
Example	FILE_DIALOG_SEPARATOR = @

	FILEDIALOG_FILETYPES
Values	a string
Description	Set the file extensions displayed in the combo box filter for the File Dialog.
Default	All Files (*.*)  *.*
Sections	10.4.2
Example	FILEDIALOG_FILETYPES = Portable Network Graphics (*.png;*.mng)  *.png;*.mng  GIF Files (*.gif)  *.gif  All Files (*.*)  *.*

	FILEDIALOG_FILETYPES_DEFAULT
Values	a number
Description	Specify the default index type from FILEDIALOG_FILETYPES to use when the dialog is opened.
Default	1
Sections	10.4.2
Example	FILEDIALOG_FILETYPES_DEFAULT = 3

	FOLDER_DIALOG_CANCEL_BUTTON
Values	a string
Description	Define the string for the "Cancel" button in the Folder Dialog
Default	"Cancel"
Sections	10.6
Example	FOLDER_DIALOG_CANCEL_BUTTON = Exit

	FOLDER_DIALOG_EDIT_LABEL
Values	a string
Description	Define the string for the "New Folder" button in the Folder Dialog
Default	Empty, edit and label removed from the Folder Dialog
Sections	10.6
Example	FOLDER_DIALOG_EDIT_LABEL = File select...

	FOLDER_DIALOG_NEW_BUTTON
Values	a string
Description	Define the string for the "New Folder" button in the Folder Dialog
Default	Empty, no button added to the Folder Dialog
Sections	10.6
Example	FOLDER_DIALOG_NEW_BUTTON = Create Folder

	<b>FOLDER_DIALOG_OK_BUTTON</b>
Values	a string
Description	Define the string for the “Ok” button in the Folder Dialog
Default	“Ok”
Sections	10.6
Example	FOLDER_DIALOG_OK_BUTTON = Select

	<b>GWINDOW_CLOSE_MACRO</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	The action called when a gwindow has been closed
Default	no action
Sections	4.2
Example	GWINDOW_CLOSE_MACRO=C:\Actions\closeWnd.cmd

	<b>GWINDOW_GET_FOCUS</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	The action called when a gwindow acquires focus
Default	no action
Sections	4.2
Example	GWINDOW_GET_FOCUS=C:\Actions\getfocus.cmd

	<b>GWINDOW_ICONIZED_MACRO</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	The action called when a gwindow has been minimized
Default	no action
Sections	4.2
Example	GWINDOW_ICONIZED_MACRO=C:\Actions\ minimize.cmd

	<b>GWINDOW_INITIAL_MAXIMIZED</b>
Values	no   yes
Description	When defined as “yes”, at start up the first gwindow is created maximized
Default	no
Sections	4.2
Example	GWINDOW_INITIAL_MAXIMIZED = yes

	<b>GWINDOW_LOOSE_FOCUS</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	The action called when focus is lost from a gwindow
Default	no action
Sections	4.2
Example	GWINDOW_LOOSE_FOCUS=C:\Actions\loosefocus.cmd



	<b>GWINDOW_MAXIMIZED_MACRO</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	The action called when a gwindow has been maximized
Default	no action
Sections	4.2
Example	GWINDOW_MAXIMIZED_MACRO=C:\Actions\ maximize.cmd
	<b>GWINDOW_RESTORED_MACRO</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	The action called when a gwindow has been restored
Default	no action
Sections	4.2
Example	GWINDOW_RESTORED_MACRO=C:\Actions\ restore.cmd
	<b>GRIDVIEW_CELL_SEPARATOR</b>
Values	a character
Description	Separator for values of different single cells in a grid view
Default	,
Sections	9.22.7 and 9.22.9
Example	GRIDVIEW_CELL_SEPARATOR =
	<b>GRIDVIEW_LINE_SEPARATOR</b>
Values	a character
Description	Separator discriminating between groups of cells belonging to different rows or columns in a grid view
Default	;
Sections	9.22.9
Example	GRIDVIEW_LINE_SEPARATOR = -
	<b>HISTORY_ASCENDING</b>
Values	no   yes
Description	Defines if the last command issued will be placed at the top of the list with the scroll bar remaining at the top (ascending) or the last command will be placed at the bottom of the list and the scroll bar stays on bottom (descending)"
Default	no
Sections	5.1
Example	HISTORY_ASCENDING = yes

	<b>HISTORY_BUTTON_ICON</b>
Values	".ico" file name (with the complete path if the file is not stored in the current directory)
Description	Defines the icon file to your for the Eagle command window history button
Default	no icon
Sections	5.1
Example	HISTORY_BUTTON_ICON = C:\MyIcon\story1.ico

	<b>HISTORY_DUPLICATES</b>
Values	no   yes
Description	Enables duplication of commands within the history dialog list
Default	no
Sections	5.1
Example	HISTORY_DUPLICATES = yes

	<b>HISTORY_TITLE</b>
Values	"<Text>"
Description	Define a tile for the history dialog list
Default	History
Sections	5.1
Example	HISTORY_TITLE="Storia"

	<b>LIST_CLOSE_ACTION</b>
Values	file name
Description	Action executed when a standalone list is closed through the "close button"
Default	no action
Sections	9.21.6
Example	LIST_CLOSE_ACTION = closelist.cmd

	<b>MENU_CLOSE_ACTION</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	Defines the action to call when the "X" button in the caption is pressed for dialog bar and fixed-size dialog bar
Default	no action
Sections	8.6, 8.9, 8.10, 8.11
Example	MENU_CLOSE_ACTION = C:\MyFolder\closebar.cmd

	<b>MESSAGE_BACKGROUND_COLOR</b>
Values	the range of possible values is : #000000 to #FFFFFF
Description	Change the color of the background (#RGB) in the message bar; the color will be specified using a "#" followed with three hexadecimal values (one for red, one for green and one for blue).
Default	#000000
Sections	5.2
Example	MESSAGE_BACKGROUND_COLOR = #335500

	<b>MESSAGE_CONTEXT_MENU</b>
Values	*.men file
Description	Customize the popup menu of the message area, using a standard Eagle's menu, defined in a *.men file.
Default	default popup
Sections	5.2
Example	MESSAGE_CONTEXT_MENU = myfile.men

	<b>MESSAGE_LINES</b>
Values	a number
Description	Set the number of lines in the message bar at start up
Default	10
Sections	5.2
Example	MESSAGE_LINES = 8

	<b>MESSAGE_READONLY</b>
Values	no   yes
Description	Define whether user written entries are permitted or not in the message window
Default	no
Sections	5.2
Example	MESSAGE_READONLY = yes

	<b>MESSAGE_TEXT_COLOR</b>
Values	the range of possible values is : #000000 to #FFFFFF
Description	Change the color of the text (#RGB) in the message bar. The color will be specified using a "#" followed with three hexadecimal values (one for red, one for green and one for blue).
Default	#FFFFFF
Sections	5.2
Example	MESSAGE_TEXT_COLOR = #000099

	<b>MESSAGE_TITLE</b>
Values	a string
Description	Title displayed on the caption of the message bar
Default	no title
Sections	5.2
Example	MESSAGE_TITLE = MyMessageTitle

	<b>MESSAGE_WINDOW_FIXED</b>
Values	no   yes
Description	To enable/disable the docking/undocking of the message bar
Default	no
Sections	5.2
Example	MESSAGE_WINDOW_FIXED = yes

	<b>MESSAGE_WINDOW_FONT_NAME</b>
Values	the name of a font
Description	Set font used in the message window
Default	Courier
Sections	5.2
Example	MESSAGE_WINDOW_FONT_NAME = "Calibri"

	<b>MESSAGE_WINDOW_FONT_SIZE</b>
Values	the height in pixel for a font
Description	Sets the font size used in the message window
Default	10
Sections	5.2
Example	MESSAGE_WINDOW_FONT_SIZE = 12

	<b>MESSAGE_WINDOW_TITLE</b>
Values	no   yes
Description	Defines whether or not to use the Eagle message window caption or title bar
Default	no
Sections	5.2
Example	MESSAGE_WINDOW_TITLE = yes

	<b>MESSAGE_WINDOW_TYPE</b>
Values	NOSIZE, FIXED, LEFT, TOP
Description	select one of the four types of the message bar
Default	NOSIZE
Sections	5.2
Example	MESSAGE_WINDOW_TYPE = top

	<b>MULTI_COLUMN_LIST_SEPARATOR</b>
Values	a character
Description	Defines the separator for the multi column list used in list and combo box type 37
Default	,
Sections	9.9, 9.21
Example	MULTI_COLUMN_LIST_SEPARATOR =
	<b>MULTI_COLUMN_LIST_SORT</b>
Values	yes   no
Description	Defines whether created multi-column lists can be sorted by clicking on the column header
Default	yes
Sections	9.9
Example	MULTI_COLUMN_LIST_SEPARATOR =
	<b>PASSWORD_CHAR</b>
Values	no   yes
Description	Define the custom "password character" displayed in an edit control
Default	*
Sections	9.8
Example	PASSWORD_CHAR = \$
	<b>POLLING_HINT_IN_STATUSBAR</b>
Values	no   yes
Description	To enable the display of polling "hint" information
Default	no
Sections	8.7
Example	POLLING_HINT_IN_STATUSBAR = yes
	<b>POPUP_IGNORE_TITLE</b>
Values	no   yes
Description	To allow or not the title in popup menus
Default	no
Sections	7.2
Example	POPUP_IGNORE_TITLE = yes
	<b>RIBBON_HELP_ACTION</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	The action called when the help button in the ribbon bar is pressed
Default	no action
Sections	6.5
Example	RIBBON_HELP_ACTION = C:\Actions\help.cmd

	<b>RIBBON_HELP_ICON</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	Set the icon to enable the help button in the ribbon bar
Default	no icon and button disabled
Sections	6.5
Example	RIBBON_HELP_ICON = C:\Mylcon\help.ico

	<b>RIBBON_HELP_TEXT</b>
Values	a string
Description	Text displayed as tooltip for the help button in the ribbon bar
Default	no text
Sections	6.5
Example	RIBBON_HELP_TEXT = Support

	<b>ROOT_BACKGROUND_COLOR</b>
Values	the range of possible values is : #000000 to #FFFFFF
Description	Change the color of the background on the main window; the color will be specified using a "#" followed with three hexadecimal values (one for red, one for green and one for blue). Change triggered dynamically when THEME is changed
Default	#000000
Sections	3.3
Example	ROOT_BACKGROUND_COLOR = #777710

	<b>ROOT_CAPTION_ICON</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	Set the icon for the main window
Default	Eagle icon
Sections	3.3
Example	ROOT_CAPTION_ICON = C:\Mylcon\Eagle.ico

	<b>ROOT_CLOSE_ACTION</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	The action called before closing the application
Default	no action
Sections	3.3
Example	ROOT_CLOSE_ACTION = C:\Actions\closeroot.cmd

	<b>ROOT_CLOSE_ENABLED</b>
Values	no   yes
Description	Enable/disable the close button on the main window
Default	no
Sections	3.3
Example	ROOT_CLOSE_ENABLED = yes

	<b>ROOT_MAXIMIZEBOX_ENABLED</b>
Values	no   yes
Description	Enable/disable the maximize button on the main window
Default	no
Sections	3.3
Example	ROOT_MAXIMIZEBOX_ENABLED = yes
	<b>ROOT_MINIMIZEBOX_ENABLED</b>
Values	no   yes
Description	Enable/disable the minimize button on the main window
Default	no
Sections	3.3
Example	ROOT_MINIMIZEBOX_ENABLED = yes
	<b>ROOT_RESIZE_ENABLED</b>
Values	no   yes
Description	Enable/disable the resizing for the main window
Default	no
Sections	3.3
Example	ROOT_RESIZE_ENABLED = yes
	<b>ROOT_SIZE_ACTION</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	The action performed when the main window has been resized
Default	no action
Sections	3.3
Example	ROOT_SIZE_ACTION = C:\Actions\sizeroot.cmd
	<b>ROOT_START_MAXIMIZED</b>
Values	no   yes
Description	Allow to start the main window maximized
Default	no
Sections	3.3
Example	ROOT_START_MAXIMIZED = yes
	<b>ROOT_THEME</b>
Values	OFFICE2000, OFFICEXP, OFFICE2003, OFFICE2003NOTHEMES, STUDIO2005, STUDIO2008, NATIVEXP, OFFICE2007_R1, OFFICE2007_R2_LUNABLU, OFFICE2007_R2_OBSIDIAN, OFFICE2007_R2_SILVER, OFFICE2007_R3_LUNABLU, OFFICE2007_R3_OBSIDIAN, OFFICE2007_R3_SILVER
Description	Select the default theme at startup
Default	OFFICE2007_R2_LUNABLU
Sections	3.1
Example	ROOT_THEME = NATIVEXP

	<b>SPLASH_WINDOW_BITMAP</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	The image to use as the background of the Splash window
Default	Eagle default bitmap
Sections	3.6
Example	SPLASH_WINDOW_BITMAP = C:\res\splash.bmp

	<b>SPLASH_WINDOW_ENABLE</b>
Values	no   yes
Description	Enable the splash screen
Default	no
Sections	3.6
Example	SPLASH_WINDOW_ENABLE = yes

	<b>SPLASH_WINDOW_FONT_BOLD</b>
Values	no   yes
Description	Apply the "bold" style to the text displayed
Default	no
Sections	3.6
Example	SPLASH_WINDOW_FONT_BOLD = yes

	<b>SPLASH_WINDOW_FONT_ITALIC</b>
Values	no   yes
Description	Apply the "italic" style to the text displayed
Default	no
Sections	3.6
Example	SPLASH_WINDOW_FONT_ITALIC = yes

	<b>SPLASH_WINDOW_FONT_NAME</b>
Values	the name of a font
Description	Set the font name of the text displayed
Default	Default font for the current application
Sections	3.6
Example	SPLASH_WINDOW_FONT_NAME = "Calibri"

	<b>SPLASH_WINDOW_FONT_SIZE</b>
Values	the height in pixel for a font
Description	Set the size of the text displayed
Default	Default font size for the current application
Sections	3.6
Example	SPLASH_WINDOW_FONT_SIZE = 12



	<b>SPLASH_WINDOW_FONT_UNDERLINED</b>
Values	no   yes
Description	Apply the "underlined" style to the text displayed
Default	no
Sections	3.6
Example	SPLASH_WINDOW_FONT_UNDERLINED = yes
	<b>SPLASH_WINDOW_SLEEP</b>
Values	a number
Description	The minimum time (in milliseconds) to show a message
Default	0
Sections	3.6
Example	SPLASH_WINDOW_SLEEP = 10
	<b>SPLASH_WINDOW_TEXT_COLOR</b>
Values	the range of possible values is : #000000 to #FFFFFF
Description	Change the color of the text displayed in the splash window. The color will be specified using a "#" followed with three hexadecimal values (one for red, one for green and one for blue).
Default	#000000
Sections	3.6
Example	SPLASH_WINDOW_TEXT_COLOR = #777710
	<b>STATUSBAR_COORDINATES</b>
Values	no   yes
Description	Enable coordinates in a statusbar text field
Default	no
Sections	8.5
Example	STATUSBAR_COORDINATES = yes
	<b>TAB_BOLD_SELECTION</b>
Values	no   yes
Description	Enables "bold" style for the characters of the tab-page (gwindow or dialog) when selected
Default	no
Sections	4.4, 4.5, 8.11
Example	TAB_BOLD_SELECTION = yes
	<b>TAB_CLOSE_ON_PAGE</b>
Values	NO, ACTIVE, ALL
Description	Allows insertion of a "close button" inside a tab-page
Default	no
Sections	4.4, 4.5
Example	TAB_CLOSE_ON_PAGE = active

	<b>TAB_ENABLE_DRAGGING</b>
Values	no   yes
Description	Enables the dragging of the tab-page (gwindow or dialog); if the entry is to "YES", it's possible to drag a tab-page /before/after another tab-page
Default	no
Sections	4.4, 4.5, 8.11
Example	TAB_ENABLE_DRAGGING = yes

	<b>TAB_ENABLE_GWINDOW_CLOSE</b>
Values	no   yes
Description	Enables closing a tab-page by clicking on the "black x" near the list of tab-pages
Default	no
Sections	4.4, 4.5
Example	TAB_ENABLE_GWINDOW_CLOSE = yes

	<b>TAB_ENABLE_SCROLL</b>
Values	no   yes
Description	Enables an extended version of the "scroll" buttons when there are a lot of tab-page present and some page are not visible
Default	no
Sections	4.4, 4.5
Example	TAB_ENABLE_SCROLL = yes

	<b>TAB_HOVER_FOCUS</b>
Values	no   yes
Description	Allows changing of the focused page (gwindow) without having to click on the tab-page but only by moving the mouse pointer over the tab-page
Default	no
Sections	4.4, 4.5
Example	TAB_HOVER_FOCUS = yes

	<b>TAB_ORIENTATION</b>
Values	TOP, LEFT, RIGHT, BOTTOM
Description	Set the orientation of tabs in TDI or TMDI mode
Default	BOTTOM
Sections	4.4, 4.5
Example	TAB_ORIENTATION=TOP

	<b>TITLE</b>
Values	a string
Description	Add a title to the Eagle based application
Default	Eagle
Sections	3.3
Example	TITLE = MyApplication

	<b>TOOLBAR_CONTEXT_ACTION</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	Allows you to attach a "context" action that will be executed after a click of the right button inside the toolbar area
Default	no action
Sections	8.8
Example	TOOLBAR_CONTEXT_ACTION = C:\MyFile\ContexToolbarAction.cmd

	<b>TMDI_CONTEXT_MENU</b>
Values	1   2   3
Description	Define the behavior of node and leaf in the tree view for the selection with single and double click
Default	1
Sections	9.20
Example	TREEVIEW_BEHAVIOUR = 2

	<b>TREEVIEW_BOX</b>
Values	ROUND   SQUARE   BITMAP
Description	Style of the boxes in the tree view
Default	SQUARE
Sections	9.20
Example	TREEVIEW_BOX = BITMAP

	<b>TREEVIEW_BOX_IMAGE_COLLAPSED</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	Image in the box of the tree view when TREEVIEW_BOX = BITMAP and the node is collapsed
Default	standard box
Sections	9.20
Example	TREEVIEW_BOX_IMAGE_COLLAPSED = C:\image\cfolder.bmp

	<b>TREEVIEW_BOX_IMAGE_COLLAPSED_HOVER</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	Image in the box of the tree view when TREEVIEW_BOX = BITMAP, the node is collapsed and the mouse is hover the box
Default	standard box
Sections	9.20
Example	TREEVIEW_BOX_IMAGE_COLLAPSED_HOVER = C:\image\chfolder.bmp

	<b>TREEVIEW_BOX_IMAGE_EXPANDED</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	Image in the box of the tree view when TREEVIEW_BOX = BITMAP and the node is expanded
Default	standard box
Sections	9.20
Example	TREEVIEW_BOX_IMAGE_EXPANDED = C:\image\efolder.bmp

	<b>TREEVIEW_BOX_IMAGE_EXPANDED_HOVER</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	Image in the box of the tree view when TREEVIEW_BOX = BITMAP, the node is expanded and the mouse is hover the box
Default	standard box
Sections	9.20
Example	TREEVIEW_BOX_IMAGE_EXPANDED_HOVER = C:\image\ehfolder.bmp

	<b>TREEVIEW_DRAGNDROP</b>
Values	no   yes
Description	Enable or disable the drag & drop function for the nodes in a tree view
Default	no
Sections	9.20
Example	TREEVIEW_DRAGNDROP = yes

	<b>TREEVIEW_EXTRINFO_SEPARATOR</b>
Values	two characters
Description	Characters used to define the extra information inside the label of a node.
Default	{ }
Sections	9.20
Example	TREEVIEW_EXTRINFO_SEPARATOR =

	<b>TREEVIEW_LINE</b>
Values	NO   SOLID   DOTS
Description	Style of the lines in the structure of a tree view (no line, solid line or dots line)
Default	DOTS
Sections	9.20
Example	TREEVIEW_LINE = solid

	<b>TREEVIEW_PATH_SEPARATOR</b>
Values	a character
Description	Character that separates the items in a tree view path
Default	\
Sections	9.20
Example	TREEVIEW_PATH_SEPARATOR =
	<b>TREEVIEW_RIGHTCLICK</b>
Values	no   yes
Description	Enable or disable the popup menu displayed with the right click hover a tree node
Default	no
Sections	9.20
Example	TREEVIEW_RIGHTCLICK = yes
	<b>TREEVIEW_ST_AS_UID</b>
Values	no   yes
Description	Enable or disable the return of the UID in the ST string during the polling loop
Default	no
Sections	9.20
Example	TREEVIEW_ST_AS_UID = yes
	<b>TREEVIEW_STYLE</b>
Values	XP   VISTA
Description	Define the style for the tree views
Default	XP
Sections	9.20
Example	TREEVIEW_STYLE = vista
	<b>USERHELP</b>
Values	file name (with the complete path if the file is not stored in the current directory)
Description	To call a custom application help file;
Default	Eagle Help file
Sections	3.5
Example	USERHELP=C:/Program Files/MyApp/MyApp.chm