Specification for abstract user interface for Windows, GNOME and Aqua

Krasimir Andreev Angelov

2003-07-25

Contents

1 Introduction

This article describes the special features of the user interfaces under Aqua, GNOME and Windows. The aim of this exposition is to define platform independent programming interface which considers the specific features of these platforms. The applications developed with this interface must not only be portable, but also look natural in the targeted environment. In order to achieve this, the interface should offer sufficient high level of abstraction. All statements in the following pages are harmonized with Aqua[1] & GNOME[2] Human Interface Guidelines. The CGA abbreviation means Common GUI API.

[1] http://developer.apple.com/techpubs/macosx/Essentials/AquaHIGuidelines/AquaHIGuidelines.pdf
[2] http://developer.gnome.org/projects/gup/hig/1.0/hig-1.0.pdf

2 Document windows

The document windows are used for manipulation and visualization of a specific document. Usually the document is stored in a file or database. The peculiarity here is that the window should respond to events like New, Open, Save, Save As and others. The window title must correspond to the document name. Under Windows and GNOME the document name is the same as the file name without the path. Under Aqua the document name is also formed from the file name, but if the user has chosen to hide the file extension for this document type (the choice is made from Finder/Preferences) then the extension will also be hidden in the window title. The windows of the new documents, which are still not associated with files, are named ”untitled” under Aqua and ”Untitled” under Windows and GNOME. If there are a few new documents, then an index is added to the title. Under Aqua the first window will be named ”untitled” and the others: ”untitled2”, ”untitled3”, ”untitled4” ... Under Windows and GNOME the titles are ”Untitled1.txt”, ”Untitled2.txt”, ”Untitled3.txt” ... Depending on the way the documents are organized the applications are with Single (SDI) and Multiple (MDI) Document Interfaces.

3 SDI and MDI applications

The SDI application can handle only one document window. If the user wants to open a new document, then he/she should first close the old one, and then to load the new one. The MDI application can support many opened documents. The application should offer an easy way to switch between the documents. In HToolkit the application type is specified from the argument passed to the $start$ function:

$$\begin{aligned}
&start :: String && \text{— Application name}\\
&\quad \to String && \text{— Application version}\\
&\quad \to DocumentInterface && \text{— Document interface type for application}\\
&\quad \to [Prop\ Process] && \text{— Properties}\\
&\quad \to IO\ \alpha && \text{— Startup action}\\
&\quad \to IO\ ()
\end{aligned}$$

The $DocumentInterface$ type definition is:

$$\begin{aligned}
&\mathbf{data}\ \ DocumentInterface\\
&\quad = SDI && \text{— Single document interface}\\
&\quad \mid\ MDI && \text{— Multiple document interface}
\end{aligned}$$

The application type cannot be changed after the library is initialised.

Each platform has its own style of organizing SDI and MDI application:

Aqua - Under Aqua the difference between MDI and SDI is conditional. In both cases the application visualizes each document in a separate window. The only difference is that if the application is SDI and you try to open a new window, the GUI library will generate an exception. The aim of this restriction is to guarantee a better compatibility with GNOME and Windows. The Aqua guidelines specify that the SDI applications should have a Window menu while under GNOME and Windows this menu is required only for MDI application. In the Window menu for SDI applications in Aqua there should be only one element with the caption Minimize. Closing the application windows (for both MDI and SDI) will not close the application itself because the application menu bar remains active.

Windows - Under Windows both MDI and SDI applications can have only one main window. The main window contains the menu and the toolbars of the application and also the document window for the SDI applications. In MDI applications the document windows are placed (inside) within the MDIClient window and the MDIClient itself is placed inside the main window. The MDIClient window is able to display multiple child (document) windows and with its help the user can switch easily from one document to another. Closing the document windows does not mean closing the entire application because the main window, the menu and the toolbars remain active.

GNOME - The manipulation of MDI and SDI applications under GNOME is the same as under Windows. The difference is that instead of MDIClient GtkNotebook is used where each document window corresponds to a separate page in the GtkNotebook. The window title is displayed as a title of the notebook page.

4 Menus

One of the typical features of Aqua is that it supports only one menu for the whole application. The menu is displayed at the top of the screen and is not associated with a certain window, but with the whole application. Under Windows and GNOME each window can have its own menu, but in the scheme presented in the previous point, the menu is associated only with the main window. This cannot be considered as a restriction for the SDI application, as they can have only one document window. The menu of the MDI application may contain options that are typical only for certain windows. In this case the options can be forbidden if the active window does not support them, or they can just be removed from the menu. The forbidding/removing of the options is possible from handlers for ”activate” and ”deactivate” events for each window.

Each platform has its own recommendations for the menu structure. Under Aqua the first sub menu is labelled with the Apple logo. It is a system specific and cannot be modified from the application. For the application working with CGA the existence of this menu must be completely transparent. The next menu is the application menu. Its title should be the same as the name of the application. This restriction is established in order to help the user to identify the application to which the current menu is associated. The title must be short and should not include any spaces. In HToolkit the first argument of the *start* function is the name of the application. Under GNOME the name and the version of the application are required to initialise the libgnome library (They are passed as arguments to the gnome_program_init function). The wxWindows library is based on GTK instead of GNOME and that is why this requirement is not necessary. Under GNOME and Windows the application name is used as a title of the main window and the same name can be used as a title of the application menu under Aqua. It is recommended to keep the title of the main window constant under GNOME and Windows. This will make the identification of the application easier. Under Aqua the same is recommended for the title of the application menu. The only exception for Windows is made when the document windows of the MDI applications are maximised. In order to ease the identification of the (active) current document it is accepted that the title of the main windows has the following form:

```
<AppName> - [<DocumentName>]
```

The elements of the application menu are:

- About `<AppName>` - The selection of this item will show the ”About” dialog. The item title should be formed from the word ”About” and the name of the application. For this purpose the argument passed to the *start* function can be used again. Under Windows and GNOME this option is placed in the ”Help” menu. In this case the graphical library, not the application code, should be responsible for the proper placing of the option. In HToolkit all global properties are accessible from the *pc* object of type *Process*. It is required to add the *appAbout* event to the *Process* type. The new event will be generated when the user selects the ”About” item.

  The proper view of the ”About” dialog is described in the Aqua recommendations. A similar dialog is included in libgnomeui under GNOME and it is expected that the GNOME application will use the standard dialog. In HToolkit the function *runAboutDialog* opens the standard ”About” dialog:

```
runAboutDialog :: String          — application name
               → String           — application version
               → String           — copyright
               → String           — comments
               → [String]         — authors
               → [String]         — documenters
               → String           — translator credits
               → Bitmap           — logo
               → Maybe Window     — The owner window
               → IO ()
```

The about dialog for Windows is implemented from HToolkit because it is missing in Win32 API. The information passed to *runAboutDialog* is enough to create the similar dialog for Aqua and this guarantee that the *runAboutDialog* function can be implemented for Aqua. The application name passed to the *runAboutDialog* is not necessary equal to the name passed to the *start* function but it is recomended to pass the same application version. The Aqua guidelines request a more descriptive name in the About dialog and a shorter one for the application menu.

The HToolkit API can be extended defining the following type:

```
dataAppAboutInfo =
    { appName               :: String     — application name
    , appShortName          :: String     — application name
    , appVersion            :: String     — application version
    , appCopyright          :: String     — copyright
    , appComments           :: String     — comments
    , appAuthors            :: [String]   — authors
    , appDocumenters        :: [String]   — documenters
    , appTranslatorCredits  :: String     — translator credits
    , appLogo               :: Bitmap     — logo
    }
```

The *start* and *runAboutDialog* should be redefined like that:

```
start :: AppAboutInfo → DocumentInterface → [Prop Process] → IO α → IO ()
```

```
runAboutDialog :: AppAboutInfo
               → Maybe Window
```

$\rightarrow\ \mathit{IO}\ ()$

In this way the library receives a complete description of the application during the initialisation and it allows the automatic installation of the default handler for the $\mathit{appAbout}$ event.

In HToolkit the events are presented by $\mathit{Event}\ \alpha$ type. The association and the removal of a handler to a given event is done with the following functions:

$$\begin{array}{l}\mathit{on}\ ::\ \mathit{Event}\ \omega\ \alpha\ \rightarrow\ \mathit{Attr}\ \omega\ \alpha \\ \mathit{off}\ ::\ \mathit{Event}\ \omega\ \alpha\ \rightarrow\ \mathit{Prop}\ \omega\end{array}$$

With the on function a new handler can be associated with a given event, while with the off function the handler is removed. For example:

$\mathit{set}\ \mathit{btn}1\ [\mathit{on}\ \mathit{command}\ =:\ \mathit{onClickBtn}1]$ — associate with onClickBtn1
$\mathit{set}\ \mathit{btn}1\ [\mathit{off}\ \mathit{command}]$ — remove association

In this way the library is always aware whether there is an associated handler to the given event. This can be used to show the ”About” item only when there is an associated handler with it and to hide it when the handler is removed.

- -Separator-
- Preferences - The selection of this item will show the dialog for the application setup. The setup should refer to the whole application and not to a specific document. There aren’t exact requirements for the setup dialog design because the type of the setup is different in each application. For that reason for the Process type the $\mathit{appPreferences}$ event should be defined. The event will occur when the user selects the ”Preferences” option. The option will not be shown if there isn’t a defined function for $\mathit{appPreferences}$.
- -Separator-
- Services - The Services option is a submenu which should contain options oriented to inter application communication. The menu isn’t obligatory and its contents are not strongly specified. It can be omitted if the application cannot communicate with other applications. In the first CGA version this specification may not be supported.

- Hide `<AppName>` - Hides all windows in the application. The option can be handled from the low-level library and in that way it remains invisible for the application working with CGA. The standard accelerator key is Command+H. The Option title is formed from the word ”Hide” and from the application name. (the appShortName field from AppAboutInfo).
- Hide Others - The usage of the option is not specified in the ”Aqua Human Interface Guideline”.
- Show All - Unlike the Hide option, the Show All option shows all opened windows in the application.
- -Separator-
- Quit `<AppName>` - Finishes the execution of the application. The accelerator key is Command+Q. Under GNOME the Quit option is situated in the File menu. Under Windows the option is also situated in the File menu but the option name is ”Exit”.

The next menu in Aqua is the File menu, while under GNOME and Windows it always comes at first place. Under Aqua in this menu are situated only functions for file management, while under GNOME and Windows the Quit/Exit option is added at the end of this menu. If the application doesn't work with files, then under Aqua it would not have a File menu, while under GNOME and Windows the menu will have only the Quit/Exit option.

The contents of the menu under Aqua are:

- New - Creates a new document. Accelerator key Command+N.
- Open - Opens a document. The file Open dialog is shown on the screen and if the user selects a file then the file will be opened in a new window.
- Open recent - This is a submenu which contains the titles of the recently opened files. When the user selects an item in the submenu, the application should open the corresponding file.
- -Separator-
- Close - Closes the active window. If the application supports more than one window (view) for one and the same document, then this option should be replaced with these two options: Close Window and Close File `<FileTitle>` where `<FileTitle>` is the title of the document. If the application is MDI, then when the Option button is hold pressed then the Close option will be replaced with Close All. The Close All option under GNOME is situated in the Window menu. The associated accelerator key is Command+W.

- Save - Saves the document in the active window. If the document is a new one, the file Save dialog will show on the screen. The accelerator key is Command+S.
- Save As... - Saves the document under a new name. The program asks the user to give the file a new name with the SaveAs dialog and then saves the file. The accelerator key is Shift+Command+S.
- Save All - Saves all changed files. Under GNOME this item is situated in the Window menu. If the application is SDI, this item should not appear.
- Revert to saved - Reloads the document from its copy in the file.
- -Separator-
- Page Setup - Sets up the page properties. The accelerator key is Shift+Command+P.
- Print - Prints the active document. The accelerator key is Command+P.

The contents of the menu under GNOME are:

- New - Creates a new document. The accelerator key is Ctrl+N.
- Open - Opens a document. The command will show the file open dialog and if the user select a file then the file will be opened in a new window.
- -Separator-
- Save - Saves the document in the active window. If the document is a new one, the file Save dialog will show on the screen. The accelerator key is Ctrl+S.
- Save As... - Saves the document under a new name. The program asks the user to give the file a new name with the SaveAs dialog and then saves the file. The accelerator key is Shift+Ctrl+S.
- Save a copy - Saves a copy of the document in the active window.
- Revert - Reloads the document from its copy in the file.
- -Separator-
- Page Setup - Sets up the page properties. The accelerator key is Shift+Command+P.
- Print Preview - Displays a print preview for the active document. The accelerator key is - Shift+Ctrl+P.

- Print - Prints the active document. The accelerator key is - Ctrl+P.
- Send To - Sends the document to another destination. The accelerator key is - Ctrl+M.
- -Separator-
- Properties - Correspond to the ”Preferences” option in the application menu under Aqua.
- -Separator-
- The list of last few opened files.
- -Separator-
- Close - Accelerator key - Ctrl+W
- Quit - Accelerator key - Ctrl+Q

The menu can be the same under Windows and GNOME except for the Quit option which is renamed to Exit.

As you see, the organization of the File menu under Aqua, GNOME and Windows is different and in order to guarantee its native look it should be built from the graphical library, not from the application code. To make this in HToolkit one should introduce an abstraction for documents and for document templates. The document template is defined like this:

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathit{data\ DocumentTemplate}\ \alpha \ = \ \mathit{DocumentTemplate}} \\
\quad \{ \ \mathit{dtMimeType} & :: & \mathit{String} \\
\quad , \ \mathit{dtOrder} & :: & \mathit{Int} \\
\quad , \ \mathit{dtDescription} & :: & \mathit{String} \\
\quad , \ \mathit{dtExtensions} & :: & [\mathit{String}] \\
\quad , \ \mathit{dtNewDocument} & :: & \mathit{IO}\ \alpha \\
\quad , \ \mathit{dtOpenDocument} & :: & \mathit{FilePath} \rightarrow \mathit{IO}\ \alpha \\
\quad , \ \mathit{dtSaveDocument} & :: & \mathit{FilePath} \rightarrow \alpha \rightarrow \mathit{IO}\ () \\
\quad , \ \mathit{dtPrintDocument} & :: & \alpha \rightarrow \mathit{IO}\ () \\
\quad , \ \mathit{dtOpenWindow} & :: & \mathit{Document}\ \alpha \rightarrow \mathit{IO}\ \mathit{WindowHandle} \\
\quad , \ \mathit{dtCompatibleTemplates} & :: & [\mathit{String}] \\
\quad \} & &
\end{array}
$$

The application should be able to define the document templates with the following function:

$$\mathit{registerDocumentTemplate} :: \mathit{DocumentTemplate}\ \alpha \rightarrow \mathit{IO}\ ()$$

The document itself is defined as $Document\ \alpha$:

$$
\begin{array}{lll}
data\ Document\ \alpha\ =\ Document & & \\
\quad (IORef\ (Bool, \alpha)) & \text{— reference to document value} \\
 & \text{— and flag for modification} \\
\quad (IORef\ FilePath) & \text{— file path} \\
\quad (IORef\ [Window]) & \text{— list of windows} \\
\quad (DocumentTemplate\ \alpha) & \text{— template}
\end{array}
$$

The data in the document is accessible with the following functions:

$$
\begin{array}{lll}
readDoc & :: & Document\ \alpha \rightarrow IO\ \alpha \\
writeDoc & :: & Document\ \alpha \rightarrow \alpha \rightarrow IO\ ()
\end{array}
$$

The $writeDoc$ function automatically sets the flag for modification to $True$. Under Aqua it is recommended to display a small black point over the close button of the windows which contain modified documents. Under Windows and GNOME to the title of the windows with modified document is added the ”*” symbol. This indication can be handled automatically with the flag for modification by the $writeDoc$ function.

Other useful functions are:

$$
\begin{array}{lll}
getDocIsModified & :: & Document\ \alpha \rightarrow IO\ Bool \\
getDocFilePath & :: & Document\ \alpha \rightarrow IO\ (Maybe\ FilePath) \\
 & & \\
newDoc & :: & DocumentTemplate\ \alpha \rightarrow IO\ (Document\ \alpha) \\
openDoc & :: & FilePath \rightarrow DocumentTemplate\ \alpha \rightarrow IO\ (Document\ \alpha) \\
revertDoc & :: & Document\ \alpha \rightarrow IO\ () \\
saveDoc & :: & FilePath \rightarrow Document\ \alpha \rightarrow IO\ () \\
openDocWindow & :: & Document\ \alpha \rightarrow IO\ Window \\
printDoc & :: & Document\ \alpha \rightarrow IO\ ()
\end{array}
$$

As the $Document$ and $DocumentTemplate$ types are parameterised with the type of the underlying data we need an additional type to build a list of documents and templates.

$$\mathbf{data}\ Holder\ \omega\ =\ \mathbf{forall}\ \alpha\ .\ Holder\ (\omega\ \alpha)$$

With the $Holder$ type the list of the documents can be defined as $[Holder\ Document]$ and the list of templates as $[Holder\ DocumentTempalate]$.

Using the information for the templates the library can build and handle the File menu. If there aren’t defined templates in the application, then under Aqua the application will not have a File menu, while under GNOME and Windows the menu will have only the Close and Quit/Exit options. The commands associated with items in the File menu can be defined as:

- New - If only one template is defined in the application, then from this template the *dtNewDocument* function is called, and after that from the result of its execution a new document (a value of type $Document\ \alpha$) is created. The document is passed to the *dtOpenWindow* function. If the templates are more than one, then under GNOME it is recommended to replace the New option with submenu. Each item in the menu should be of the form New `<DocumentType>`. The text corresponding to `<DocumentType>` can be taken from the field *dtDescription* in the *DocumentTemplate*. The items in the menu should be ordered by the frequency of usage. For that reason the field *dtOrder* is added to the document template. The accelerator Ctrl+N should be associated with the most frequently used type of document, i.e. the template - $dtOrder = 0$. Under Aqua and Windows if the application has multiple templates, then the New command opens a dialog from which the user can select the type of the new document. The types are sorted by the *dtOrder* field. The new document will be registered in the list of active documents. The captions of the new windows are formed according to the rules specified in point 2.

- Open - The action opens the standard ”Open” dialog in the platform. In HToolkit this dialog is opened by *runInputFileDialog* function. The function receives as an argument a filter for the files which can be chosen from the dialog. The filter is an argument of the $[(String, [String])]$ type. In fact it is a list of pairs of the the filter name and a list the file extensions which enters in the filter. The list can be received from the list of the registered templates and the field *dtDescription* is taken for a name of the filter. The path to the chosen file is given to the *dtOpenDocument* function from the document template. The result of the function is used for creating a document which after that is given to the *dtOpenWindow* function.

- Open recent - The list of the last opened files is stored in Windows Registry under Windows, in GConf database under GNOME and in the appropriate place under Aqua. The path to the file can be recovered from the storage and after that the process follows the same steps as in the Open option.

- Save - If the flag for modification of the active document is set to *True*, then the library calls the *dtSaveDocument* function from the template and after that the flag is set to *False*.

- Save As... - The library asks the user for the new file name with the ”Save” dialog and after that the library calls the *dtSaveDocument* function with the new file path. The flag for modification is set to *False*. In HToolkit the *runOutputFileDialog* function opens the standard ”Save” dialog.

- Save All - The processing is the same as for the ”Save” option but the operation is performed for each opened document.
- Revert to saved - The loading of the file copy is done by the $dtOpenDocument$ function from the document template. The loaded data is saved in the same document (in the $Document\ \alpha$ structure). The flag of the modified document is set to $False$ after reloading.

An additional advantage of this scheme is that the graphic library will be automatically able to identify the type of the file and to load it if the user drags a file from the file manager (Windows Explorer, Nautilus) and drops it over the main window. In the same way the users will be automatically able to open files from Finder in the application under MacOS.

The next two menus have to be Edit and View. The Edit menu contains the following options: Undo, Redo, Cut, Copy, Paste, Delete, Find and so on. The View menu has options for setting up the view of the application. As the structure of these two menus depends on the type of the application, their managing should be organised by it. The Aqua and GNOME guidelines concerning the Edit menu are very similar, while there are no recommendations for the View menu under Aqua. The View menu described for GNOME is typical mainly for the file manager Nautilus.

All other menus typical for the application are situated after the View and Edit menus.

The last but one menu is Window. Under Windows and GNOME the SDI applications do not have a Window menu, while under Aqua the Window menu has only one option - Minimize. The contents of the menu under Aqua are:

- Minimize - The accelerator key is Command+M. The option minimizes the active window.
- Zoom - The option restores the real size of the minimized window.
- -Separator-
- Bring All to Front - Brings all application windows over the others.
- -Separator-
- A list of the open windows. Choosing any option moves the corresponding window over the windows of the other applications. The dialogs and the help windows are not included in the list. The help windows under Aqua correspond to the floating toolbars under Windows and GNOME.

The type of all menu items is such that the corresponding commands can be fulfilled directly, without any additional help from the application.

Under GNOME the menu contains the following options:

- List of the open windows.
- -Separator-
- Save All - Saves all changed documents.
- Close All - Closes all active windows.

The Save All and Close All options are the same as the ones described in the File menu under Aqua.

Under Windows the menu contains the following options:

- Arrange Icons
- Tile Vertically
- Tile Horizontally
- Cascade
- -Separator-
- A list of the open windows.

The menu is already supported in HToolkit and does not request an additional support from the application.

The last menu is the Help menu. Under Aqua there are no specific requirements for this menu, and under Windows and GNOME the only restriction is that the last option should be ”About”.

When creating each menu in HToolkit one should point the parent menu where the new item should be situated. If it has to be situated in the menu bar, then the $mainMenu$ constant is used as a parent menu. In order to handle the Edit, View and Help menus, three more constants have to be defined: editMenu, viewMenu and helpMenu. The Edit and View menus are automatically added before adding the first item to them. The Help menu is created at the very beginning (for GNOME and Windows) and contains the About item. All elements in Help are situated before the About item.

5 Dialogs

Dialogs are used to fill in help or managing information. There are two types of dialogs: modal and modeles. Under Aqua the modal dialogs are divided into two subtypes: modal to the document and modal to the application.

In HToolkit under Windows and GNOME all document windows are situated in one unifying main window, and each application can have only one main window. Unlike them the dialogs are not situated in the main window - they float over it. When a dialog is created, its window owner should be specified. The created dialog is situated always over its owner. When the owner is minimized or closed, its subordinate dialogs are also minimized or closed. On the other side, each subordinate dialog can be the owner of other dialogs. This creates a chain whose basis is always the main window of the application. Consequently the minimizing or the closing of the main window minimizes or closes all other dialogs. The owner is set by an argument from type $\mathit{Maybe\ Window}$. If the argument is with a value $\mathit{Nothing}$ or $\mathit{Just}\ \omega$, where ω is a document window, the main window of the application becomes an owner. If the argument is $\mathit{Just}\ \omega$ and ω is a dialog, ω becomes an owner of the new dialog.

Under Aqua the support of dialogs is the same. The only difference is that the dialogs are supported modal to the document. If $\mathit{Nothing}$ is the owner of the dialog, then the dialog is modal to the whole application. If $\mathit{Just}\ \omega$ is the owner of the dialog, then the owner of the dialog is ω regardless whether it is a dialog or a document window.

The created dialog can become modal by using the function:

$$\mathit{runDialog} :: \mathit{Window} \rightarrow \mathit{IO}\ ()$$

The modal dialog is situated always over its owners, and the user is not allowed to select either of the owners while the modal window is open. The execution of the $\mathit{runDialog}$ function finishes after closing the modal window.

Under Aqua and GNOME there are special requirements about the location of the buttons in the dialogs. The buttons should be located at the bottom of the dialog. The button that is set by default (OK) should be located on the right side. The Cancel button should be located on the left side of the OK button. If there are other buttons in the dialog which can lead to the closing of the window, then they should be located on the left side of the Cancel button. If the action associated with these buttons may lead to loss of information (for instance the ”Don’t Save” button), they should be separated at least 12 pixels from the Cancel button. The Help information button should be situated at the bottom left corner, and all the rest managing buttons should be located on its right side.

Under Windows the Help button should be located at the bottom right hand corner. The OK and Cancel buttons are located to the left. All other buttons are located at the bottom left hand corner.

Under Windows and GNOME the OK button is surrounded by a thicker frame to show that this is the button set by default. Under Aqua the button is displayed in light blue for the same purpose. For the three platforms pressing the Enter key is associated with confirming the default action, and pressing the Esc key - with Cancel.

The automatic location of the buttons, in the most appropriate for the platform way, should be done by the graphical library. For this purpose we define the following function:

$$addButton :: Button \rightarrow ButtonType \rightarrow Window \rightarrow IO\,()$$

The $ButtonType$ type is defined as:

$$\begin{aligned}&\mathbf{data} Button\\&\quad\quad = DefaultButton\\&\quad\quad\mid CancelButton\\&\quad\quad\mid DismissButton\\&\quad\quad\mid DismissAndLostButton\\&\quad\quad\mid HelpButton\\&\quad\quad\mid SimpleButton\end{aligned}$$

Unlike other controls, the buttons do not have to be included in the expression associated with the $layout$ property of the window. They have to be added by the $addButton$ function. This allows the graphical library to choose the location of the buttons on base of the $ButtonType$ parameter.

6 Summary

Some of the described ideas are already realized in HToolkit, but still there is a lot to do. The specification is a sort of a plan for the future development of the project. I hope some of the ideas to be accepted by the CGA specification. This specification is not final and is about to be developed.