Leggi

An ISO/ANSI text reader
Version 2.0

by Sebastiano Vigna

1 Introduction

`Leggi` is a rather flexible text reader which I developed some time ago with two main goals in my mind: first of all, supporting a simple subset of the IFF FTXT specification, enough to describe multi-font text files, and secondly being also a good Amiga application—ARexx support, font adaptivity and so on.

The former goal turned out to be a dead end: indeed, no one was interested in the FTXT support; the latter one was achieved *theoretically*, but a lot of bugs and some misfeature made `Leggi` 1.x much less widespread than I would have liked. My fault anyway. On the other hand, `Leggi`'s specification was a little bit ahead of its times—there was a real lack of system support for many things (for instance, a simple but transparent system tracking of the allocation sizes, or a system command line parser).

With Release 2, Commodore eventually produced a beautiful programming environment in which the routine part of the programmer's work is left to the operating system. This urged a revision of `Leggi`.

What really happened is that `Leggi` was completely recreated. I'm not only meaning rewritten—even the specification and the user interface was radically changed. Since most of `Leggi`'s functions were considered in the *Amiga User Interface Style Guide* (AUISG), Commodore staff's suggestions were taken as the primary source when designing new functions and while updating the old ones. At the same time, dropping the support for 1.2/1.3 brought the power of Release 2 (version 2.04) to the program (just to mention an example, `Leggi` had previously to emulate the whole `LayoutMenus()` function in gadtools.library). Finally, dropping the FTXT support (and consequently the handling of multi-font documents) reduced greatly the size of the code.

The result is a fully AUISG-compliant application, with lots of new Release 2 related features and no more bugs. I can resume in a couple of word what you will miss with `Leggi` 2.0: 1.2/1.3 and FTXT support. What you will get, instead, is:

- Faster text rendering/scrolling operations;
- Better keyboard/mouse handling (scrolling with cursor keys is no longer slower than scrolling with the mouse);
- Full AUISG menus and ARexx commands;
- Dozens of new ARexx commands that allow fine control of the window position and placement, displayed part of the document, etc.;

- Clipboard support—display the content of any clip in a `Leggi` window, or copy a displayed file to it;
- AppWindow support—drop a file icon in a `Leggi` window and the file will be loaded;
- Public screen support—open each window on a different public screen;
- Fast & residentable activator (à la `CED` and `TurboText`);
- Full configurability of the ARexx port (both global and local);
- Full configurability of the keyboard—any ARexx command can be assigned to any keystroke;
- Settings file which stores even the window position and size;
- Preferences editor which lets you directly edit settings files;
- Background mode—`Leggi` remains active even when the last window is closed, and will restart via the activator or via an ARexx command;
- Standard clock pointer for long operations—if you use an animation utility such as `ClockTick` you will get a spinning hand, too;
- Beautiful 3-D scrolling gadget;
- Faster load operations;
- Fully documented IFF settings file;
- Support for the backspace character;
- Release 3 new look menus support.

Before you ask: "leggi" in Italian means "read", and I am told it is pronounced "leeji" in English.

2 Usage

This chapter describes how to use `Leggi`. After a brief, idiotproof introduction a thorough explanation of the various interfaces follows.

2.1 First Steps

The usage of `Leggi` is extremely straightforward. Double click on its icon, and then open a file using the '`Open...`' item of the '`Project`' menu. You can even drop a file icon over `Leggi`'s window, and the file will be automatically loaded (that is, the window is an Application Window). If you prefer to use a Shell, you can type

`Leggi` *filename*

instead. Once loaded a file, you can move around using the scroll bar, the cursor keys or by clicking in the upper or lower region of the window.

The '`Settings`' menu allows you to customize the text display; in particular, you can try to change the display font using the '`Set Font...`' item, and to add some interline space using the '`Set Line Spacing...`' item.

2.2 The Workbench Interface

`Leggi` is clearly a GUI based application. It has full support for tooltypes, which can be specified both in the tool icon and in the project icons. This allows to control both global and local settings from the Workbench (see Section 3.5 [Settings], page 8).

The available tooltypes closely match the keywords available on the command line. They are described shortly here, and more detailed information can be found in Section 2.3 [The CLI Interface], page 4.

`PubScreen=`*public screen name*

lets you specify a public screen name.

`Settings=`*settings file name*

lets you override the default settings file name ('`PROGDIR:Leggi.prefs`').

`Background=On|Off`

when set to '`On`' tells `Leggi` to not exit definitively when the last window is closed or when asked to quit. Rather, it will wait for a command at its main ARexx port (usually this command will be sent by the activator). This allows you to keep `Leggi` always active (possibly by starting it in the startup-sequence or via the '`WBStartup`' drawer) and to activate one of its windows on your preferred public screen when needed. Since leggi uses less than 30K of memory, this can be a very pratical option for users with memory expansions.

`NoGUI=On|Off`

when set to '`On`' tells `Leggi` to not open the first window, and rather waiting in the background until a command arrives from the activator or from ARexx. It implies a '`Background=On`'.

`PortName=`*ARexx port name*

overrides the default port name, which is '`LEGGI`' for the main port, and '`LEGGI.`*n*', where *n* is the window number, for the window ports. You can override the global name (and thus all the subsequently derived window names) by putting this tooltype in the tool icon, or override the local window port name by putting it in the project icon.

The '`PubScreen`', '`PortName`' and '`Settings`' tooltypes are available for inclusion in project icons or in the activator tool icon (see Section 2.4 [The Activator], page 5), while the other tooltypes are only available in the `Leggi` main program tool icon. Of course, when included in the activator or project icons, they refer to local settings, and when included in the main program tool icon, they refer to global settings (see Section 3.5 [Settings], page 8).

2.3 The CLI Interface

While being a GUI oriented application, `Leggi` has a full-featured CLI interface. By typing

```
Leggi ?
```

you can get the template in standard AmigaDOS format, which is:

```
Files/M,PubScreen/K,PortName/K,Background/S,NoGUI/S,Settings/K
```

The template of the activator (see Section 2.4 [The Activator], page 5), instead, is

```
Files/M,PubScreen/K,PortName/K,Settings/K,Wait/S
```

'`Files`' is filled by an arbitrary number of file name specifications, possibly containing AmigaDOS wildcards. Each file will be loaded in a different window.

'`PortName`'

lets you specify an ARexx port name which will override the internal name generation of `Leggi`. When included in the main program command line, it sets the global port name, while if included in the activator command line, it sets the local port name.

Note that if you open several file via the activator using multiple filename/wildcards and you specify a '`PortName`' keyword, all opened windows will get the same port name, probably causing confusion when trying to control one of them via ARexx.

'`PubScreen`'

lets you specify a public screen name which `Leggi` will open its window on (by default, they are opened on the default public screen). The name is set globally or locally, depending on the calling program, just as for the '`PortName`' argument. An unexisting public screen name will keep `Leggi` from opening any window. Note that the name is case sensitive.

'`Settings`'

lets you specify a settings file name to open instead of reading the default file at startup or cloning in the standard way global or local settings. See Section 3.5 [Settings], page 8.

'`Background`'

tells `Leggi` to stay forever in the background even when the last window is closed or when asked to quit, waiting for a command from the activator or from ARexx.

'`NoGUI`' tells `Leggi` to not open the first window, and rather waiting in the background for a command. It implies a '`Background`' switch, even if not explicitly present.

'`Wait`' tells the activator to not return until the window has been closed (much like the '`Wait`' switch of `TurboText`).

2.4 The activator

In the same vein of other Amiga programs, `Leggi` has an activator, named `LG`. `LG` is a very short, residentable program which sends to the main program messages telling it to open certain windows, load certain files et cetera. `Leggi` is not residentable, but it doesn't need to be, since just one copy needs to be loaded (you can open different windows, with specific port names on different public screens using a single copy of `Leggi`).

Unless you have special needs (for instance, you could want to run `Leggi` in the background with the '`NoGUI`' option and some special settings file from your startup-sequence), you should always invoke `Leggi` through `LG`. `LG` checks if a copy of `Leggi` is present (it does it by checking for a port name '`LEGGI`'; if you started a copy of `Leggi` with another main port name, you can't use `LG` with it), and if so it sends it a series of messages by which `Leggi` learns what it has to do (opening windows, loading files, using certain settings files, etc.). If no copy of `Leggi` is active, it launches a `Run Leggi >NIL: <NIL: NoGUI` command, and then tries again for a number of seconds. If the '`LEGGI`' named port doesn't appear, it gives up.

Note that since under Release 2 a program can get the directory it belongs to, you can store a `Leggi.prefs` file in the same directory as of the `Leggi` main program, and it will be loaded even if you don't specify the '`Settings`' argument. This is also where settings are saved when you select the '`Save Settings`' menu item.

Note also that by using `LG`'s '`Wait`' switch, you can use `Leggi` as a text-reader invoked by other programs (such as directory utilities, mail readers et cetera).

2.5 Keyboard & Mouse

Navigation through a document is accomplished through the keyboard and the mouse. If you click on the upper or lower part of a window, or while approximately in the center in the leftmost or rightmost part, you will see the contents of the window scrolling in the opposite direction. This behaviour is fixed and cannot be modified; however, you can disable selectively the left or right mouse scrolling via the ARexx commands `MouseLeft` and `MouseRight` (see Chapter 4 [The ARexx Interface], page 11), or via `LeggiPrefs` (see Section 7.1 [The Settings Editor], page 31).

The default keyboard configuration is that cursor keys move by one line/column, `SHIFT`ed cursor keys by a page (either horizontally or vertically) and `ALT`ed cursor key move to the top/bottom (or they center the horizontal position). `CTRL`ed vertical cursor keys move to the next page key (see Section 3.5 [Settings], page 8; practically, they do a search for the page key forwards or backwards).

The function keys move to the corresponding bookmark, while the `SHIFT`ed functions keys set the bookmark. The `B` and `BACKSPACE` keys are equivalent to `SHIFT-UP`, the space bar is equivalent to `SHIFT-DOWN`, while `ESC` closes a window (just like `AMIGA-K`).

All the keyboard definitions can be overriden by associating a new ARexx command to a specific key (of course, keystrokes associated to menus such as `AMIGA-O` can't be redefined since they're directly handled by Intuition). See Section 7.1 [The Settings Editor], page 31.

3 Menus

`Leggi`'s menus are extremely straightforward. They follow closely Commodore's guidelines, and in the near future you should find more and more applications using a similar, coherent menu scheme.

3.1 Project

The Project menu contains standard and rather obvious items.

'`New`' creates a new window, cloning the current settings.

'`Open...`' brings up the ASL file requester and lets you choose a file to open. Note that if you have powerpacker.library, you can load a PowerPacked file, and it will be automatically decrunched (however, if you have powerpacker.library `Leggi` sometimes won't be able to display the very last character in a file).

'`Close`' closes the current window. If the current window is the last window open and `Leggi` wasn't requested to stay in the background (see Section 2.2 [The Workbench Interface], page 3, and see Section 2.3 [The CLI Interface], page 4), it exits.

'`About...`'
displays some information about the program.

'`Quit`' closes all opened windows. If `Leggi` wasn't requested to stay in the background (see Section 2.2 [The Workbench Interface], page 3, and see Section 2.3 [The CLI Interface], page 4), it exits.

3.2 Edit

This menu is named "Edit" just because it contains the '`Cut`', etc. items which are always under that name. You can't really edit anything, which is obvious because `Leggi` is a text reader, not a text editor.

'`Cut`' clears the current window, putting its contents in the Clipboard.

'`Copy`' copies the content of the current window in the Clipboard.

'`Paste`' erases the content of the current window, and then copies in it the content of the Clipboard.

'`Erase`' erases the content of the current window.

3.3 Search

This menu controls the search system. You can ask `Leggi` to search for any string; the string can also include any AmigaDOS wildcard (including the '`*`' for '`#?`' if you activated it via `StarBurst` or a similar utility). If no wildcard is specified, a simplified version of the Boyer-Moore algorithm

is used, which guarantees the highest possible performance. The search speed depends on many factors, including the number of characters of the search string (in optimal conditions it can scan about one Megabyte per second). If wildcards are present, the scan is done using the system functions, which are many times slower (nonetheless they are extremely fast if you consider they handle wildcards).

The search starts from the second displayed line. When a match is found, the line containing the match becomes the top line. If no match is found, a screen flash is generated. You can iterate the search both forwards and backwards.

'`Find...`' brings up a requester that lets you specify the search string, and then it starts a forward search.

'`Find Next`'

'`Find Previous`'

iterate forward or backward, respectively, the search with the current string.

3.4 Extras

This menu contains a couple of special items.

'`Jump To Line...`'

issues a requester asking for a line number and jumps to it.

'`Jump To Column...`'

issues a requester asking for a column number and jumps to it (negative numbers can be used to move to the left).

'`Refresh Window`'

causes a complete refresh of the window, just in case it became corrupted for any reason.

'`Exec ARexx Macro...`'

brings up the ASL file requester and lets you choose an ARexx macro which will be executed; it will have as default address port the window port.

3.5 Settings

In order to give you the maximum flexibility and customizability, `Leggi` features a full settings load and save system. *Settings* here denotes all the options which are contained in the '`Settings`'

menu, the window size and position and the keyboard macro assignments. Moreover, a particular role is given to the public screen name (it is cloned or inherited as any other setting, but it's not loaded or saved). There are also some "hidden" settings flags (`Background`, `MouseLeft` and `MouseRight`). Since they are option which are seldom changed, they can be set only via the ARexx interface or by using `LeggiPrefs` (see Chapter 4 [The ARexx Interface], page 11, and see Section 7.1 [The Settings Editor], page 31).

First of all, we have to distinguish between two kind of settings: global and local. Global settings are defined at startup time as the default settings (no flags on, TAB size 8, public screen Workbench, and system default font) or as the content of the settings file, if present (the default name of the settings file is '`PROGDIR:Leggi.prefs`', but it can be overriden by the keyword or by the tooltype '`Settings`'). Global settings are cloned by the first window opened by `Leggi`, and by any new window which is created by a `New` or an `Open` command arriving at `Leggi`'s main port (i.e., the one named '`LEGGI`'). Global settings can be modified at run-time by specific ARexx command sent to the main port (`LoadSettings`, `Font`, and so on). Note that while the public screen name is not included in the settings file, it can be specified using the '`PubScreen`' keyword or tooltype.

Each window has additionally an associated set of local settings, which can be modified by using the '`Settings`' menu or by sending an ARexx message to the window specific ARexx port (say, '`LEGGI.4`'). In particular, you can save the current settings of a window in the currently used settings file using '`Save Settings`', while '`Save Settings As...`' allows you to specify a different file name. '`Load settings...`' sets your window settings to the contents of a previously saved settings file. When you create a new window with the '`New`' menu item or by sending a `New` command at the window ARexx port, your current local settings (including the public screen and the current window size) are cloned.

The '`Settings`' menu includes a series of on/off menu items (which are checkmarked if activated) and a series of "valued" menu items (whose current value appears in the requester they generate when activated); finally, the load/save commands allow to load/store the current settings.

'`Smart Refresh`'

> The Amiga lets you manage the refresh of your windows in several different ways. `Leggi` defaults to `SIMPLE_REFRESH` windows, which have to manually redraw obscured parts which are made visible. This process tends to slow down the system a little bit, but a `SIMPLE_REFRESH` window doens't pratically need any memory in order to work. `Leggi` has a reasonably fast text display, so this is the default refresh mode. If, however, you checkmark this menu item, `Leggi` will use `SMART_REFRESH` windows, in which the operating system will automatically save obscured regions in Chip RAM in order to refresh the display when the regions are made visible, without prompting the application. The depth arrangement and the size reduction are completely transparent

to the application—`Leggi` needs to redraw your window only if the user enlarges it. The disadvantage is that each window requires memory for its obscured parts, and if you have many windows this can become a major issue: for instance, on a 640x512 4-color screen a full-size fully obscured window requires 80K. If this memory is not available, the system will usually refuse to drag, depth arrange or size other windows. Unless you use a lot of non full-size windows with frequent switches from one to another, I would suggest always using `SIMPLE_REFRESH` windows.

Note that since this option has to be specified when the window is opened, changing the menu item will have no effect on the current window—the change will be effective from the next window opened (and will be recorded if you save your settings). If you want the first window being `SMART_REFRESH`, you have to override `Leggi`'s global defaults by writing a settings file.

'`Line Numbers`'

When this menu item is checkmarked, the total number of lines of the current file and the line number of the current top line will be displayed in the window title. Updating the top line number can slightly slow down the text scrolling, which is the raison d'ètre of this option.

'`Font...`' brings up the ASL font requester and lets you choose any font for the current window.

'`Wordwrap...`'

brings up a requester which lets you set the number of characters after which a line will be wrapped. Wrapping happens at load time, so you have to reopen the current file if you want to wordwrap it.

'`Page Key...`'

`Leggi` lets you have a variable concept of *page* which is specified by a key which marks the start of a new page. The key can be up to 40 characters long, and can contain any AmigaDOS wildcards. You can move to the next/previous key by using `CTRL-DOWN/UP`. This allows for instance to scan message by message downloads from BBS, BIX and so on. This menu item lets you set the page key.

'`Line Spacing...`'

lets you set the number of pixels that `Leggi` will insert between each text line, thus making the display a little less cluttered. Try some values until you find the one you prefer.

'`Load Settings...`'

brings up the ASL file requester and lets you choose a preferences file to load. The local settings of the current window will immediately be set to the contents of the settings file, but the window won't change its refresh mode (see Section 3.5 [Settings], page 8, the '`Smart Refresh`' menu item description). The window will be resized and moved following the indications of the settings file, and the old macro key assignments will be substituted by the new ones.

‘`Save Settings`’

saves the local settings on the default settings file ‘`PROGDIR:Leggi.prefs`’, unless differently specified via the ‘`Settings`’ keyword/tooltype.

‘`Save Settings As...`’

acts as the previous menu item, but lets you specify a file name through the ASL file requester.

4 The ARexx Interface

`Leggi` features an ARexx interface which allows full control of each window and of the main program. When `Leggi` is started, it opens a main port named ‘`LEGGI`’ (or differently if you used the tooltype/keyword ‘`PortName`’). Moreover, each window opened gets a port with a unique numbered name: ‘`LEGGI.1`’, ‘`LEGGI.2`’ and so on, or, if you asked, say, for a ‘`PortName PIPPO`’, ‘`PIPPO.1`’, ‘`PIPPO.2`’ and so on, unless of course you override locally this name using again the tooltype/keyword ‘`PortName`’.

In general, there are two classes of ARexx commands:

1. Commands which can be sent to both main and window ports, and which will usually act differently in the two cases;
2. Command which can be sent only to window ports (they generate an error if sent to the main port).

The first class includes all commands which set a parameter having both a local and a global copy. The second class includes all document-specific commands, such as navigation commands, edit commands, search commands and so on. After the AmigaDOS style template for each command, it is specified if it can be issued at the main port. A short description of the command and a list of the return codes follow (including under the case “result” the content of the `result` variable if `Options Results` was set). Common return codes are described in Section 4.1 [Conventions], page 11. Many commands correspond directly to menu items, and behave accordingly.

4.1 Conventions

- The syntax of each command is described using a standard AmigaDOS template style; if you don’t know anything about templates, you can look at the *Using the System Software* manual.

- whenever a command is told to change something globally or locally, it is understood this fact depends on the command arriving at the main port or at a window port, respectively;
- if a command which can only be sent to a window port is said to do something to a window, it is understood to be the window which owns the port (unless specified otherwise);
- all syntax errors, including sending a forbidden command to the main port, missing parameters, or plainly issueing an unexisting command, produce an error code of 10;
- all commands which involve requesters return 5 if the requester was canceled;
- all commands which move through the text return 6 if the movement couldn't be performed (usually because you tried to move after the end or before the start of the file), and return the current line or column number in the result variable on success (that is, the number that was affected by the command is returned). Note that when the text is placed in standard position, the column number is 0.

4.2 File Commands

There is a single command for loading a file in a (possibly brand new) window.

4.2.1 Open

Template: `FileName/K,PortName/K,PubScreen/K,Settings/K,Wait/S`
Main port: Yes

If received at the main port, this command creates a new window duplicating the current global settings (unless you override this choice using the suitable keywords) and loads it with the specified filename. If no filename is specified, a file requester appears.

If received at a window port, this command loads in the window the specified filename, or brings up a file requester if no filename is specified. The last three keywords are ignored.

In any case, if the '`Wait`' switch is specified the command will not return until the window has been closed.

Return codes

5	No file selected

6	File not found or out of memory
20	Can't open window (only when sent to the main port)
result	The ARexx port name assigned to the window

4.3 Window Handling Commands

These commands allows to modify the position and size of any window.

4.3.1 ActivateWindow

Template: ,
Main port: No

This command activates a window (in the Intuition sense, of course).

4.3.2 ChangeWindow

Template: `LeftEdge/N,TopEdge/N,Width/N,Height/N`
Main port: No

This command sets both the position and the dimensions of a window. Any parameter omitted won't be changed. The result could be different from what you expected if the window couldn't fit into the screen.

4.3.3 Close

Template: ,
Main Port: No

This command closes a window.

4.3.4 MoveWindow

Template: `LeftEdge/N,TopEdge/N`
Main port: No

This command sets the position of a window. You can specify one or both parameters. The result could be different from what you expected if the window couldn't fit into the screen.

4.3.5 New

Template: `PortName/K,PubScreen/K,Settings/K,Wait/S`
Main port: Yes

This command creates a new window duplicating the current global or local settings (unless you override this choice using the suitable keywords). If the '`Wait`' switch is specified the command will not return until the window has been closed.

Return codes

20	Can't open window
result	The ARexx port name assigned to the window

4.3.6 RedisplayWindow

Template: ,
Main port: No

This command refreshes the window (the text contained will be redisplayed).

4.3.7 Quit

Template: ,
Main port: Yes

This command closes all opened windows. If `Leggi` wasn't requested to stay in the background (see Section 2.3 [The CLI Interface], page 4 and see Section 2.2 [The Workbench Interface], page 3), it exits.

4.3.8 SizeWindow

Template: `Width/N,Height/N`
Main port: No

This command sets the width and height of a window. You can specify one or both parameters. The result could be different from what you expected if the window couldn't fit into the screen.

4.3.9 UnZoomWindow

Template: `,`
Main port: No

This command set the window to its normal position and size (see Section 4.3.12 [ZoomWindow], page 16).

4.3.10 WindowToBack

Template: `,`
Main port: No

This command depth arranges a window, sending it behind all other ones.

4.3.11 WindowToFront

Template: `,`
Main port: No

This command depth arranges a window, bringing it to the front.

4.3.12 ZoomWindow

Template: ,
Main port: No

This command sets the window to its alternate position and size (just like the zoom gadget).

4.4 Edit Commands

These commands control the Clipboard. Note that from ARexx you can access *any* clip.

4.4.1 Copy

Template: `/N`
Main port: No

This commands copies the contents of the window (i.e., the whole document) to the clipboard in the clip specified by the argument. If no number is specified, 0 (the primary clip) is assumed.

Return codes

20 Couldn't copy to clipboard

4.4.2 Cut

Template: `/N`
Main port: No

This commands cuts the contents of the window (i.e., the whole document) to the clipboard in the clip specified by the argument. If no number is specified, 0 (the primary clip) is assumed.

Return codes

20 Couldn't cut to clipboard

4.4.3 Erase

Template: ,
Main port: No

This command erases the contents of the window, thus freeing the associated memory.

4.4.4 Paste

Template: `/N`
Main port: No

This commands pastes the contents of the clipboard in the window using the the clip specified by the argument. If no number is specified, 0 (the primary clip) is assumed.

Return codes

20 Couldn't paste from clipboard

4.5 Search Commands

These commands control the search system.

4.5.1 Find

Template: `Text/F`
Main port: No

This command will set the current search string to the argument '`Text`', and will start a forward search; if no argument is specified, the user will be prompted with a text requester.

Return codes

0	Successful search
5	No search string was specified or selected
6	No match found
result	The line containing the match

4.5.2 FindNext

Template: ,
Main port: No

Iterates forward the search with the current search string. See Section 4.5.1 [Find], page 17.

4.5.3 FindPrevious

Template: ,
Main port: No

Iterates backward the search with the current search string. See Section 4.5.1 [Find], page 17.

4.6 Settings Commands

These commands allows to set the global settings, and to control some feature which is not available via GUI.

4.6.1 Background

Template: `On/S,Off/S`
Main port: Yes

This command sets globally or locally the background flag. Note that unless you plan to save your local settings, changing a local background flag has absolutely no effect (this is why there is no menu item for the background flag).

4.6.2 Font

Template: `Name/K,Height/K`
Main port: Yes

This command sets globally or locally the display font. If no argument is specified, the ASL font requester is issued. If you specify only one of the two arguments, the other one will be taken from the current font.

Return codes

5	The user canceled the requester
20	Couldn't open the specified font

4.6.3 LineNumbers

Template: `On/S,Off/S`
Main port: Yes

This commands sets globally or locally the line numbers flag.

4.6.4 LineSpacing

Template: `/N`
Main port: Yes

This command sets globally or locally the number of pixels `Leggi` will put inbetween two lines. If no argument is specified, a requester is issued. Allowed values are between 0 and 99, included.

Return codes

5	The user canceled the requester
20	Value out of range

4.6.5 LoadSettings

Template: `Settings/K`
Main port: Yes

This commands loads in the local or global settings the specified file.

4.6.6 MouseLeft

Template: `On/S,Off/S`
Main port: Yes

This command sets globally or locally the mouse left scrolling flag. If this flag is set (the default), when the mouse is positioned horizontally in the leftmost eighth of a window, and vertically more or less in the center, the text will be scrolled horizontally. Note that if you want to disable left/right scrolling through the keyboard, too, you can override the definition of the left/right cursor keys with a `NOP` command or with some command of your choice.

4.6.7 MouseRight

Template: `On/S,Off/S`
Main port: Yes

This command sets globally or locally the mouse right scrolling flag. See Section 4.6.6 [MouseLeft], page 20.

4.6.8 PageKey

Template: `Key/F`
Main port: Yes

This command sets globally or locally the key `Leggi` will use when performing a `Next Key` or a `Previous Key` (which in the default configuration is equivalent to `CTRL-UP/DOWN`). If no argument is specified, a requester is issued. The page key has a maximum length of 39 characters—longer keys will be silently truncated.

Return codes

5 The user canceled the requester

4.6.9 SaveSettings

Template: `,`
Main port: Yes

This commands saves the local or global settings to the default settings file.

4.6.10 SaveSettingsAs

Template: `Settings/K`
Main port: Yes

This commands saves the local or global settings to the specified file. If the keyword '`Settings`' is not specified, the ASL file requester pops up.

4.6.11 SmartRefresh

Template: `On/S,Off/S`
Main port: Yes

This commands sets globally or locally the smart refresh flag.

4.6.12 SmoothScrolling

Template: `On/S,Off/S`
Main port: Yes

This commands sets globally or locally the smooth scrolling flag.

4.6.13 TabSize

Template: `/N`
Main port: Yes

This command sets globally or locally the number of spaces `Leggi` will use when expanding a TAB. If no argument is specified, a requester is issued. Allowed values are between 0 and 999, included.

Return codes

5	The user canceled the requester
20	Value out of range

4.6.14 WordWrap

Template: `/N`
Main port: Yes

This command sets globally or locally the number of characters after which a word wrap will happen. A value of 0 means no wordwrap. Note that since wordwrap is done at load time, the change has no visible effect until you load a file. If no argument is specified, a requester is issued. Allowed values are between 0 and 9999, included.

Return codes

5	The user canceled the requester
20	Value out of range

4.7 Navigation Commands

These commands allows you to move in a document.

4.7.1 Column

Template: `/N/A`
Main port: No

This command changes the horizontal position of the text displayed in the window, by adding the argument to the number of the first displayed column. Thus, a positive argument moves towards left, while a negative one moves towards right. An argument of 0 can be used to just get the current setting.

4.7.2 GoToBookmark

Template: `/N`
Main port: No

This command sets the top line number to the bookmark specified by the argument. If no argument is specified, 1 is assumed as bookmark number. Valid values for bookmark numbers are from 1 to 10, included.

Return codes

5	Bookmark number out of range
6	Bookmark position greater than file length

4.7.3 GoToColumn

Template: `/N`
Main port: No

This command changes the horizontal position of the text displayed in the window, by setting the first displayed column number to the argument. If no argument is specified, a requester is issued.

4.7.4 GoToLine

Template: `/N`
Main port: No

This command changes the vertical position of the text displayed in the window, by setting it to the argument. If no argument is specified, a requester is issued.

4.7.5 Line

Template: `/N/A`
Main port: No

This command changes the vertical position of the text displayed in the window, by adding the argument to the number of the top line. Thus, a positive argument moves forward, while a negative one moves backward. An argument of 0 can be used to just get the current setting.

4.7.6 Next

Template: `Page/S,Key/S,Windowful/S`
Main port: No

This command moves the display position to the next element, where element can be a page, i.e., the number of lines displayed in the window minus one will be added to the top line, a page as defined by a key, i.e., a search for the next key is performed and if a match is found the line containing the match becomes the top line (see Section 4.6.8 [PageKey], page 21), or a horizontal

page, i.e., the number of displayed columns minus one is added to the horizontal position. You specify one of the three available switch depending of the kind of displacement you need.

4.7.7 Position

Template: `SOF/S,EOF/S`
Main port: No

This command moves the display position to the start of file (`SOF`) or to the end of file (`EOF`). Note that in the last case the top line won't be the last line of the file, which will be the last displayed line at the bottom of the window.

4.7.8 Previous

Template: `Page/S,Key/S,Windowful/S`
Main port: No

This command moves the display position to the previous element. See Section 4.7.6 [Next], page 24.

4.7.9 SetBookmark

Template: `/N`
Main port: No

This command set the bookmark specified by the argument to the current top line. If no argument is specified, 1 is assumed as bookmark number. Valid values for bookmark numbers are from 1 to 10, included.

Return codes

5 Bookmark number out of range

4.8 Support Commands

These commands offer miscellaneous services which can be useful to an ARexx macro.

4.8.1 About

Template: ,
Main port: Yes

This command displays a simple information requester about `Leggi`.

4.8.2 GetName

Template: ,
Main port: No

This command returns in the result variable the complete pathname of the file currently containted in the window.

Return codes

5	No file is currently loaded

4.8.3 Help

Template: `Command`
Main port: Yes

This command returns in the result variable the template of the given command. It is very useful for Command Shell ARexx scripts. If no command is specified, the whole list of available command is printed (you will need an I/O console to see it).

Return codes

5	No such command
result	The template of the given command

4.8.4 NOP

Template: ,
Main port: Yes

It does nothing. Mainly useful for inhibiting standard key definitions.

4.8.5 RequestFile

Template: `Title/K,Path/K,File/K,Pattern/K`
Main port: Yes

This command issues an ASL file requester on the global or local public screen. The requester title is specified by the '`Title`' argument, while the remaining keywords allow to specify default values for the filename, the pattern and the path displayed by the requester.

Return codes

5	The user canceled the requester
result	The complete pathname of the file selected

4.8.6 RequestNotify

Template: `Prompt/K/A`
Main port: Yes

This command issues a requester on the global or local public screen. The user is prompted with the '`Prompt`' argument, and can only confirm the requester.

4.8.7 RequestNumber

Template: `Prompt/K,Default/N/K`
Main port: Yes

This command issues a requester on the global or local public screen asking for a number. The prompt string lets you select the label appearing at the left of the string gadget. If the first letter of the '`Prompt`' argument is a '`_`', the following letter will be considered a shortcut for the activation of the string gadget. The default number will appear in the gadget.

Return codes

5	The user canceled the requester
result	The number

4.8.8 RequestResponse

Template: `Prompt/K/A`
Main port: Yes

This command issues a requester on the global or local public screen. The user is prompted with the '`Prompt`' argument, and can confirm or cancel the requester.

4.8.9 RequestString

Template: `Prompt/K,Default/K`
Main port: Yes

This command issues a requester on the global or local public screen asking for a string. The prompt string lets you select the label appearing at the left of the string gadget. If the first letter of the '`Prompt`' argument is a '`_`', the following letter will be considered a shortcut for the activation of the string gadget. The default string will appear in the gadget.

Return codes

5	The user canceled the requester
result	The string

4.8.10 RX

Template: `Console/S,Command/F`
Main port: Yes

This commands starts an external ARexx macro. If the '`Console`' switch is specified, an I/O `AUTO` console is opened for I/O (otherwise, the standard I/O streams are inherited from the program). If the '`Command`' argument is surrounded by quotes (which aren't strictly necessary even if you use a filename with spaces in it, because of the '`/F`' template specification) it is assumed to be a program string (i.e., an ARexx macro itself). This allows to program complex functions inside `Leggi` by using ARexx without invoking external files. If no command is specified, a file requester will appear. Note that in order to pass a quoted argument, you have to diddle with the quoting escape conventions of both AmigaDOS and ARexx, which are quite complex. If you don't need to use quotes inside the string macro, you can use '`''''`' as delimiter (for instance, '`RX '''do for 3; line 1; end'''`'). Otherwise, you have to escape the quotes (for instance, '`RX '"*"do for 3; line 1; end*""'`'). The first set of quotes is stripped by ARexx, the second one by the system parser.

5 Keyboard Macros

`Leggi` allows you to associate to any keystroke any ARexx command. This means you can:

1. Assign to a keystroke a specific command; for instance, assigning to `CTRL-U` the command `Line -12` would emulate `Ed`'s behaviour. This does not require ARexx.
2. Assign to a keystroke an ARexx program via the inline program facility; for instance, assigning to `ALT-R` the command

   ```
   RX 'Options Results; RequestNumber; N=Result; If RC==0 Do For N FindNext; Loop'
   ```

   would cause a number to be requested and then a `FindNext` being performed as many times as specified.
3. Assign to a keystroke a full ARexx program stored in a separate file.

The macro assignment process is done outside `Leggi` through the `LeggiPrefs` program (see Section 7.1 [The Settings Editor], page 31). Macros can be local or global as any other setting (remember however that if a macro is present globally it is not callable from a window; global macros are simply used for cloning when opening a new window; also, see Section 3.5 [Settings], page 8). The only way to modify the local macros is via the '`Load Settings...`' menu item or the ARexx command `LoadSettings`. Global macros can be changed by issueing a `LoadSettings` command to the main ARexx port.

The `Leggi.prefs` file included in the `Leggi` distribution archive has a series of macros which assign to `ALT+`the *n*th function key the action of bringing to front the *n*th window (if it exists). Moreover, `CTRL-F1` edits the file currently contained in a window with your preferred editor (via the `EDITOR` environment variable). You can take a look at these macros in order to have an idea of what can be done.

6 ISO/ANSI Compliance

`Leggi` is able to parse completely a small subset of the ISO/ANSI color/style sequences; let's take a look at it:

```
Color/Style Specifications

<ESC>[0m    normal char set (all reverts to plain)
<ESC>[3m    italics on
<ESC>[23m   italics off
<ESC>[4m    underline on
<ESC>[24m   underline off
<ESC>[1m    boldface on
<ESC>[22m   boldface off
<ESC>[3xm   set foreground color to x (from 0 to 9)
<ESC>[4xm   set background color to x (from 0 to 9)
```

You can freely mix together indications. For instance,

```
<ESC>[4;1;31;40m
```

sets underline and boldface on, and sets the color to 1 (fore) and 0 (back).

As a final note, `Leggi` supports TABs (you can set their size via the corresponding menu item) and backspaces, which move one space backwards the cursor; note that backspaces won't be displayed properly with a proportional font, for obvious reasons.

7 The Settings Editor and File Specification

This chapter focuses on the preferences program that lets you edit a settings file, and on the IFF specification of such a file.

7.1 The Settings Editor

The `LeggiPrefs` program allows you to directly edit each setting in `Leggi`'s configuration. In particular, it is the place where you can define the keyboard macros.

The usage of the editor is straightforward: simply put in each gadget the value you want, or click on the checkbox gadget to toggle or on off an option. Remember that writing '`-1`' in the '`Width`' or '`Height`' field will make the window fit the whole screen (starting from the '`LeftEdge`'/'`TopEdge`' position, of course). Moreover, a '`Wordwrap`' of '`0`' means no word wrapping. Note also that when the editor is started, no font is displayed above the '`Set Font...`' gadget. If you don't select any font, `Leggi` will open the system default font when using that settings file. The editor doesn't currently check for out of range values, so please be careful.

The keyboard assignment system is very simple to use: if you activate the string gadget just under the list and type in a keystroke expression (try, for instance, '`ALT 80`'), the keystroke will be added to the list. You can then select it and fill in the '`Command`' gadget, which describes the command associated to the keystroke. Whatever is in the '`Command`' gadget is also copied when you create a new keystroke mapping, so it's very easy to copy an assignment many times. If you select an item from the list and click on the '`Delete`' gadget, the assignment will be deleted.

The keystroke expressions accepted by `LeggiPrefs` have the following template:

```
Shift/S,Alt/S,Control/S,LCommand/S,RCommand/S,Code/N/A
```

The '`Code`' part is the raw key code assigned to a particular key on the keyboard. For a list of those, you can peek at the '`KeyCodes`' file which is supplied with `Leggi`, or at the *Amiga Rom*

Kernel Manuals. For instance, the ten function keys have codes from 80 to 89. You can add to the code as many qualifier as you like. '`LCommand`' and '`RCommand`' refer to the left and right Amiga keys, of course.

`LeggiPrefs` has a single menu with the basic Open/Save operations, which are rather obvious:

'`Open`' opens a settings file (brings up the file requester);

'`Save`' saves the current configuration with the current filename;

'`Save As...`'
saves the current configuration with a name (brings up the file requester);

'`About...`'
displays some info about `LeggiPrefs`;

'`Quit`' exits `LeggiPrefs`.

7.2 The Settings File Specification

The settings file of `Leggi` is an IFF file, namely a LEGG FORM which can contain only two chunk types:

KASS a keyboard assignment; you can have many of those;

SETT a settings chunk; there must be just one, and it has to be the last chunk in the FORM (just like the BODY in an ILBM FORM).

The content of a KASS is given by:

```
typedef struct {
   USHORT Code, Qualifier;
   char Command[];
} KASS;
```

i.e., it's a variable-length structure containing two `USHORT`s (`RAWKEY` code and qualifier of the macro) and a 0-terminated string which specifies the ARexx command to execute. The `Qualifier` field has to contain both or none of the `SHIFT` (or `ALT`) left/right qualifier flags (i.e., you can't set a macro for the left `SHIFT` only).

The content of the SETT chunk is a `Settings` structure:

```
#define PAGEKEYLENGTH 40
#define FONTNAMELENGTH 64

typedef struct {
   USHORT   Flags;
   USHORT   WordWrap;
   USHORT   TabSize;
   USHORT   LineSpacing;
   WORD     LeftEdge, TopEdge, Width, Height;
   struct   MinList KAList;
   struct   TextAttr TextAttr;
   UBYTE    PageKey[PAGEKEYLENGTH];
   UBYTE    FontName[FONTNAMELENGTH];
}  Settings;
```

The `Flags` field can have the following (obvious) flags set:

```
#define LEGGIF_SMOOTHSCROLLING   (1<<0)
#define LEGGIF_SMARTREFRESH      (1<<1)
#define LEGGIF_LINENUMBERS       (1<<2)
#define LEGGIF_BACKGROUND        (1<<3)
#define LEGGIF_MOUSELEFT         (1<<4)
#define LEGGIF_MOUSERIGHT        (1<<5)
```

The `WordWrap`, `TabSize`..., `Height` and `PageKey` fields contain the values for the corresponding options. The `KAList` field is irrelevant, and can be set to anything. The `TextAttr` field contains information about the desired font, but the `ta_Name` field is irrelevant—the font name is stored in the `FontName` array. Note that a `-1` in the `Width` or `Height` fields will set them to the screen width or height (minus the `LeftEdge` or `TopEdge` values, if they're nonzero).

8 Acknowledgments

Many people contributed to this program: the writers of the *Amiga User Interface Style Guide*, my beta testers, Ilya Shubentsov for designing the icon, Steve Tibbett for giving his '`StringReq.c`' code, Ewout Walraven for useful suggestions, and many others. Thanks to them all, and to the people who sent me bug reports.

But a most special thank goes to Gayle Noble, for her outstanding contribution, which exceeds everything I received in my life for my work.

Comments, complaints, desiderata are welcome.

9 Author Info

```
Sebastiano Vigna
Via California 22
I-20144 Milano MI

BIX: svigna
INTERNET: vigna@imiucca.csi.unimi.it
          vigna@ghost.sm.dsi.unimi.it
UUCP:cbmehq!cbmita!sebamiga!seba@cbmvax.cbm.commodore.com
     ...{uunet|pyramid|rutgers}!cbmvax!cbmehq!cbmita!sebamiga!seba
FIDO: 2:332/607.28
```

ARexx Command Index

T

U

Z

W

Concept Index

Table of Contents