LuaFAR for Editor (version 3.7)

LuaFAR for Editor (a Far Manager plugin) is a collection of utilities working in the Far Editor. The plugin also allows to add the unlimited number of new utilities. Both the plugin and the utilities are written in Lua programming language.

For description of the built-in utilities and the Configuration Dialog, see the plugin's help file.

What's new

See the Changelog.

Changelog

```
-------------------------------------------------------------------
  Legend:   [+] added;   [-] deleted;   [*] changed;   [!] fixed;
-------------------------------------------------------------------

2012-04-09, v3.0.6
2012-03-26, v3.0.5
2012-03-24, v3.0.4
      [!] Maintenance versions (minor fixes).
      [+] Examples added.

2012-03-04, v3.0.3
  PLUGIN:
      [*] Function lf4ed.version moved to `utils' module and renamed
          GetPluginVersion.

2012-01-04, v3.0.0
  PLUGIN:
      [*] Adaptation to Far 3.0 / LuaFAR 3.0 API.
      [*] Calling from macros via GUID instead of SysId.
      [-] "searchmenu" is not used by the plugin (but remains availa
          to user scripts).
  Sort Lines:
      [*] Column pattern is Far regex (was: Lua regex).
      [*] Added flag SORT_STRINGSORT to case-insensitive sort.
      [+] "Case-sensitive" check boxes made 3-state.
  Block Sum:
      [+] If no text is selected, the current line is processed.
  Lua Script:
      [+] External script can be specified (via Script Parameters di

2011-02-02, v2.8.0
  PLUGIN:
      [+] Got a SysId (0x10000) for calling from macros. The call sy
          the same as for command line calls (but the prefix is not
          New switch -a for asynchronous calls.
      [+] Function MakeResident: parameter can be a table. That make
          possible to require() files containing event handlers.
  Sort Lines:
      [+] a GUID added to the dialog.
  Reformat Block:
      [+] a GUID added to the dialog.

2010-12-24, v2.7.1
      [!] The build of 2.7.0 was broken due to an error in Makefile.
```

```
2010-12-23, v2.7.0
  PLUGIN:
      [+] 64-bit build added (works with 64-bit LuaFAR).
      [*] Plugin keeps its data in "%APPDATA%\LuaFAR for Editor" dir
          (was: in the plugin directory).

2010-11-26, v2.6.0
  PLUGIN:
      [*] Lua modules used by the plugin do not set global variables
          Use the value returned by `require`.
  Sort Lines:
      [*] A single "undo" in Editor undoes the entire operation.
  Reformat Block:
      [*] A single "undo" in Editor undoes the entire operation.
  Block Sum:
      [*] The dialog moved from the configuration to the utility.
      [+] The result can be edited in the dialog.
      [*] Items immediately followed by [,;:] are considered valid.
  Lua Expression:
      [*] The dialog moved from the configuration to the utility.
      [+] The result can be edited in the dialog.
      [+] If there's no selection, the current line is processed.

2010-10-14, v2.5.0
  PLUGIN:
      [!] Error when calling "Block Sum" or "Lua Expression" setting
      [!] Utilities could interfere with each other via parameters t
  Lua Script:
      [+] Parameters can be passed to the script. Parameters dialog

2010-08-30, v2.4.0
  PLUGIN:
      [+] AddToMenu: parameter "where" can include letter "d" (for "
      [*] AddToMenu, AddCommand: unlimited number of additional argu
      [*] Installed scripts get their parameters in a table (was: 2
          The table may also include fields "From" and "hDlg".
      [!] Fix "Reload user file" with default plugin settings.

2010-07-25, v2.3.0
  PLUGIN:
      [*] 'Reload User File' operation resets `package.loaded', in o
          behave as close as possible to the initial loading.
      [+] new callback ("resident") function ExitScript.
      [+] function AddToMenu supports localization.

2010-06-25, v2.2.0
  PLUGIN:
      [*] LuaFAR 2.3 required.
```

```
      [+] Localization of the configuration dialog.
      [+] Utility for localization of script packets.
  Sort Lines:
      [!] Fixed shortcut keys in the dialogs.
      [!] Sorting made stable.
  Block Sum:
      [*] Made compatible with LuaFAR 2.3.

2010-04-24, v2.1.0
  PLUGIN:
      [!] Fixes to work with Unicode file names and paths (LuaFAR 2.
  Sort Lines:
      [+] Added constant `I' (number of lines in selection). Help fi

2010-02-25, v2.0.1
  PLUGIN:
      [+] Added function unicode.utf8.cfind (see the manual).
  Sort Lines:
      [+] Added variable `i' (number of selected line). Help files u

2010-02-20, v2.0.0
  PLUGIN:
      [*] First Unicode version.
      [*] Requires LuaFAR version 2.0.0.
      [*] s:find() accesses unicode.utf8.find(s), the same goes for
          string function names. To access the standard string libra
          write string.find(s), etc.
  Sort Lines:
      [*] "Case sensitive" checkboxes added. Help files updated.

2010-02-04, v0.12.1
  PLUGIN:
      [!] LuaFAR version check was done after the exported functions
          already connected (that is, too late).
      [*] Requires LuaFAR version 1.1.

2010-01-16, v0.12.0
  PLUGIN:
      [!] When errors occured in a utility called from Editor via sh
          the shortcut was reported to Far as needing further proces
      [!] Broken stack traceback in error messages (since version 0.
      [!] Error message window: could not jump to an already open ed
          from the panels.
      [*] Command line syntax changed: see the manual.
  Lua Script:
      [*] When running on the whole Editor buffer, ignore the 1-st l
          if it starts with a # character.

2010-01-03, v0.11.0
```

```
  PLUGIN:
      [+] Plugin can be called from the command line, via `lfe' pref
          A new function `AddCommand' was added.
      [!] lf4ed:config: changes to the "ReturnToMainMenu" option did
      [!] lf4ed:config: changes were not reverted in case of error.

2009-12-02, v0.10.0
  PLUGIN:
      [!] `far2.history' module was raising error given an empty his
      [*] `far2.history' module renamed to `history'.
      [*] configuration changes done by a user script are reverted b
          plugin after the script exits.
      [*] lf4ed.config: always returns configuration existed prior t
      [!] far.OnError: eliminate jumps to embedded scripts.
      [!] far.OnError: jump to incorrect line in another file, when
          was selection in the current file (the bug existed since v
      [*] Main Menu, Config. Menu: removed separators after the buil
          (They can be added via files _usermenu.lua).
      [*] The plugin's DLL, as supplied, now embeds scripts and modu
          source code form (not compiled). That allows to work with

2009-11-02, v0.9.1
  PLUGIN:
      [!] configuration changes were not saved in the history file
          (the bug existed since version 0.8.0).

2009-11-01, v0.9.0
  PLUGIN:
      [*] all utilities added by AddToMenu calls within a single use
          share a common environment that does not change until the
          "Reload User File" operation (or until FAR termination);
      [+] plugin checks LuaFAR version;
      [*] error handler (far.OnError) improved;
      [*] plugin (as it is supplied) does not embed LuaFAR library;

2009-10-02, v0.8.0
  PLUGIN:
      [*] signature and functionality of AddToMenu function changed;
          [*] removed "global functions requirement" for user script
          [+] become possible to add items to Configuration Menu;
          [+] menu separators can be added;
      [+] added function AddUserFile (callable from _usermenu.lua);
      [+] added function AutoInstall (callable from _usermenu.lua);
      [*] '<plugin_path>/scripts/?.lua' is added to package.path;
      [+] added function lf4ed.config: get/set plugin settings from
      [+] added function lf4ed.version: get plugin version;
      [+] built-in Lua modules (dialog, history, searchmenu) used in
          are documented; user scripts can now "officially" use them
      [*] "embedded" versions of the plugin contain LuaFAR 0.8.0;
```

```
      [!] unneeded "require 'strict'" removed from far2/sortlines.lu
      [-] lf4ed_emb.dll is not supplied any more;

2009-09-01, v0.7.4
  PLUGIN:
      [!] function far.OnError can be reliably replaced from _userme
      [*] plugin built on base LuaFAR 0.7.3

2009-08-28, v0.7.3
  PLUGIN:
      [*] plugin built on base LuaFAR 0.7.2

2009-06-18, v0.7.2
  PLUGIN:
      [*] plugin built on base LuaFAR 0.7.1

2009-02-14, v0.7.1
  Reformat Block:
      [!] error loading and saving start and end column data

2009-02-13, v0.7
  PLUGIN:
      [*] plugin built on base LuaFAR 0.7

2009-01-14, v0.6
  PLUGIN:
      [*] plugin built on base LuaFAR 0.6

2008-12-31, v0.5.2
  PLUGIN:
      [!] improvements in jumping to error lines
      [*] _usermenu.lua (and event handler files) are run in 2 cases
             a) on plugin start-up
             b) on configuration menu command "Reload User File"
      [*] _usermenu.lua: input arguments are deprecated
      [+] configuration menu command "Reload User File"
      [+] user is able to install event handlers, by means
          of calling new function MakeResident in _usermenu.lua

2008-12-27, v0.5.1
  PLUGIN:
      [!] did not work if file _usermenu.lua was missing
      [!] option "always reload on require" was not independent from
          "always reload default script"
      [!] several problems related to jumping to error lines
      [*] plugin built on base LuaFAR 0.5.1

2008-12-22, v0.5
  PLUGIN:
```

```
      [+] configuration dialog "Plugin Settings"
      [+] work from viewer and panels
      [+] file farkeys.lua
      [*] _usermenu.lua: AddToMenu API changed
      [*] added <plugin directory>\?.lua to initial value of package
      [*] env. variable LUAFAR_INIT is processed on start up (was: L
      [*] plugin menus can be made "searchable" (thanks to maxdrfl)
      [*] indicator of memory used by the Lua State on the error mes
      [*] buttons on error message boxes for jumping to error lines
      [*] plugin built on base LuaFAR 0.5

2008-12-13, v0.4
  Reformat Block:
      [+] Russian interface translation
  Block Sum:
      [!] editor window not redrawn after hotkey-initiated execution
  Lua Expression:
      [!] editor window not redrawn after hotkey-initiated execution
  PLUGIN:
      [!] hot keys: multiple executions after a single key press
      [!] hot keys: did not work when they were part of a macro
      [*] _usermenu.lua: is passed 2 arguments (event type and edito
      [*] plugin built on base LuaFAR 0.4

2008-12-06, v0.3d
  PLUGIN:
      [!] hot keys: worked only after the menu was called
      [!] hot keys: triple combinations didn't work
      [+] hot keys: can be assigned without adding items to the menu
      [+] hot keys: can be assigned to the built-in utilities
      [*] _usermenu.lua: is (almost) not restricted by its scope
      [*] _usermenu.lua: is run when either of 4 different events oc
          and is passed an argument (event type)

2008-12-04, v0.3c
  PLUGIN:
      [!] incorrect processing of editor input

2008-12-04, v0.3b
  PLUGIN:
      [+] hot keys can be assigned to user utilities
      [+] can work with external Lua DLL
      [*] plugin built on base LuaFAR 0.3

2008-11-26, v0.3
  Sort Lines:
      [!] handling different types of EOLs.
      [!] sorting "only selected" in vertical blocks.
  Block Sum:
```

```
      [*] the dialog moved to Configuration Menu.
  Lua Expression:
      [*] the dialog moved to Configuration Menu.
      [*] a semicolon is appended to result when inserting into the
  Lua Script:
      [*] works either on selection or on the whole editor buffer.
  PLUGIN:
      [+] adding utilities by the user.
      [+] configuration menu added

2008-11-06, v0.2.1
  Sort Lines:
      [!] sorting "only selected" in vertical blocks.

2008-11-05, v0.2
  PLUGIN:
      [+] first public release.
```

Version 2.5.0

PLUGIN: [!] Error when calling “Block Sum” or “Lua Expression” settings dialogs.
[!] Utilities could interfere with each other via parameters table.

Lua Script:
[+] Parameters can be passed to the script. Parameters dialog added.

Version 2.4.0

[+] AddToMenu: parameter “where” can include letter “d” (for “dialog”).
[*] AddToMenu, AddCommand: unlimited number of additional arguments.
[*] Installed scripts get their parameters in a table (was: 2 parameters). The table may also include fields “From” and “hDlg”.
[!] Fix “Reload user file” with default plugin settings.

Version 2.3.0

Plugin

`[*]` ‘Reload User File’ operation resets `package.loaded`, in order to behave as close as possible to the initial loading.
`[+]` new callback (“resident”) function **`ExitScript`** (see `MakeResident` function).
`[+]` function `AddToMenu` supports localization.

Version 2.2.0

Plugin

[+] Localization of the configuration dialog.
[+] Utility for localization of script packets.

Sort Lines

[!] Fixed shortcut keys in the dialogs.
[!] Sorting made stable.

Version 2.1.0

Plugin:

[!] Fixes to work with Unicode file names and paths (LuaFAR 2.2 required).

Sort Lines:

[+] Added constant **I** (number of lines in selection). Help files updated.

Version 2.0.1

See also Changelog.

1. `unicode.utf8.cfind` function added.

Version 0.12.1

- **Bug fix:** LuaFAR version check was done after the exported functions were already connected (that is, too late).
- Requires LuaFAR version 1.1.

Version 0.12.0

- Command line syntax extended.
- far2.searchmenu:
  - produce data allowing custom highlighting of the matched part of the item text
  - added property `Map`
- Several bugs fixed: see Changelog.

Version 0.11.0

Plugin

- Plugin can be called from the command line, via the prefix **`lfe`** (see User's utilities and `AddCommand`).

Modules

- `history`: method `setfield` added.
- `far2.searchmenu`: added properties `AllowEmpty`, `Menu`, `CheckItem`, `SearchText`; property `SearchPlain` replaced by `SearchMethod`.

Version 0.10.0

PLUGIN:

- `far2.history` module renamed to `history`.
- configuration changes done by a user script are reverted by the plugin after the script exits.
- `lf4ed.config`: always returns configuration existed prior to the call.
- Main Menu, Config. Menu: removed separators after the built-in items. (They can be added via files `_usermenu.lua`).
- The plugin's DLL, as supplied, now embeds scripts and modules in source code form (not compiled). That allows to work with LuaJIT.

Version 0.9.0

Installing user’s utilities

- All utilities added by `AddToMenu` calls within a single usermenu file share a common environment that does not change until the next *Reload User File* operation (or until FAR termination). See `AddUserFile`.

Version 0.8.0

Installing user's utilities

- Improved `AddToMenu` function
- Added `AddUserFile` function
- Added `AutoInstall` function
- **`<plugin_path>/scripts/?.lua`** is added to `package.path`.
  (See Packets of scripts).

Plugin API

- Added `lf4ed.config` function
- Added `lf4ed.version` function

Lua modules

- Added documentation for a few built-in Lua modules used in plugin; user utilities can use those modules.

Version 0.7.3

- Plugin is built on base LuaFAR 0.7.2

Version 0.7

- Plugin is built on base LuaFAR 0.7

Version 0.6

- Plugin is built on base LuaFAR 0.6

Version 0.5.2

- User is able to install event handlers, by means of calling new function `MakeResident` in `_usermenu.lua`
- Configuration menu command “Reload User File”

Version 0.5.1

This version is mainly a bug-fixing release.

Also, since this version embeds LuaFAR 0.5.1, it is possible to assign hot keys that were unavailable to previous versions (keys with punctuation characters).

Incompatibilities with the previous version

Version 2.6.0

---

1. Lua modules used by the plugin do not set global variables. Use the value returned by `require`.

Version 2.4.0

---

1. Installed scripts get their parameters in a table (was: 2 parameters).

Version 2.1.0

Module `history`

1. Methods `hobj:field` and `hobj:setfield` – API changed; the code using those methods must be fixed.

Version 2.0.0

1. See *Incompatibilities* section of LuaFAR 2.0.0 manual.

Version 0.12.0

- Command line syntax changed.
- `MakeResident`: handlers' return values are treated slightly differently.

Version 0.11.0

---

1. `far2.searchmenu`: property `SearchPlain` replaced by `SearchMethod`.

Version 0.10.0

---

1. Module `far2.history` renamed to `history`.

Version 0.9.0

1. LuaFAR version 0.9 required.

Version 0.8.0

---

1. `AddToMenu`: function signature changed. Both the existing *_usermenu.lua* files and user script files **must be adapted** in order to work correctly with the new API.

Version 0.7

1. See Incompatibilities section of LuaFAR 0.7 manual.

Version 0.6

1. See Incompatibilities section of LuaFAR 0.6 manual.

Version 0.5.2

---

1. **_usermenu.lua**: no input arguments are passed.

Version 0.5.1

---

1. Incompatibilities of LuaFAR-0.5.1 versus LuaFAR-0.5 (see LuaFAR 0.5.1 manual)

Version 0.5

---

1. Function `AddToMenu`: a new first parameter; all existing parameters are shifted toward the right.

2. `"<plugin directory>\?.lua;"` is prepended to the initial value of `package.path`

3. Environment variable `LUAFAR_INIT` (if it exists) is processed on start up (was: `LUA_INIT`)

4. Incompatibilities of LuaFAR-0.5 versus LuaFAR-0.4 (see LuaFAR 0.5 manual)

User's utilities

Installing user’s utilities

The plugin treats the file **_usermenu.lua** lying in the plugin’s directory as the installation script for user’s utilities. This script is run in two cases:

- When Far calls `SetStartupInfoW` of the plugin
- When the command “Reload User File” from the Configuration Menu is executed

There are a few installation functions that are intended to be called from **_usermenu.lua**.

Running installed user’s utilities

User utilities can be configured to run in the following ways:

- Via a menu item; see `AddToMenu`.
- Via a shortcut (from Editor only); see `AddToMenu`.
- From the command line (the plugin registers a command line prefix **`lfe`**); see `AddCommand`.

Running scripts without installation

- From the Editor: use the included utility *Lua Script*.
- From the command line.

Installation functions

AddToMenu AddCommand
AddUserFile
AutoInstall
MakeResident

AddToMenu

`AddToMenu (where, text, hotkey, file [, ...])`

Function `AddToMenu` is available to the *_usermenu.lua* script.
It allows to add a specified utility to the plugin's menu and assign it a hot key.

Parameters

1. `where`
   Where the utility is intended to run from: any combination of letters **[cdepv]** (**c**=configuration menu, **d**=dialog, **e**=editor, **p**=panels, **v**=viewer).

2. `text`
   Text that should appear in the menu.
   — To assign a hot key without adding a menu item, supply a `nil`.
   — To add a separator, specify a string beginning with `":sep:"` following by optional text, e.g. `":sep:Block commands"`.
   — If the utility for localization is used, and `text` begins with `"::"`, then the rest of the text is treated as a message identifier. (In the case of separator, prefix `":sep:::"` can be used).

3. `hotkey`
   A key combination for calling the utility, e.g. `"Alt+Shift+F4"`.
   Supply `nil` if no hot key is needed.
   NOTE: a hot key assigned this way works **only from Editor**, provided that no macro is assigned to that key.

4. `file`
   A specification for a Lua script to run upon this item activation. The exact file name is determined according to the same rules as with `require` function, that is, `package.path` is used to search for the file. But contrary to `require`, the value returned by the script is not cached in `package.loaded`.

5. Extra parameters (optional)
   Values that will be passed to user script (they are passed to the script in a table). There can be any number of values of any Lua type.

Examples

```
AddToMenu ("e",   "Count words",   nil,      "farscripts.edit.coun
AddToMenu ("e",   "Count letters", "Alt+F2", "farscripts.edit.coun
AddToMenu ("e",   "Search",        "Ctrl+F", "farscripts.edit.sear
AddToMenu ("p",   "Rename Files",  nil,      "farscripts.rename")
AddToMenu ("evp", "Calendar",     "Alt+F12", "farscripts.calendar"
```

To assign a hot key to a built-in utility, supply `text` as `true` and specify the menu position via `file`:

```
AddToMenu ("e",   true,           "Ctrl+1",  1)
```

AddCommand

`AddCommand (command, file [, ...])`

Function `AddCommand` is available to the *_usermenu.lua* script.
It establishes a correspondence between the given `command` and the specified utility (`file`). This allows to call that utility from the command line (or a macro).

Parameters

1. `command`
   The first command line parameter after the plugin's prefix.

2. `file`
   A specification for a Lua script to run upon this item activation. The exact file name is determined according to the same rules as with `require` function, that is, `package.path` is used to search for the file. But contrary to `require`, the value returned by the script is not cached in `package.loaded`.

3. Extra parameters (optional)
   Values that will be passed to user script (they are passed to the script in a table). There can be any number of values of any Lua type.

Examples

```
AddCommand("calc",  "scripts.fl_scripts.common.calc")
AddCommand("umenu", "scripts.Rh_Scripts.LuaPUM.LuaPUM")
```

AddUserFile

`AddUserFile (filename)`

Function `AddUserFile` is available to the *_usermenu.lua* script.

Parameter `filename` (its path is relative to the plugin's directory) specifies a file that is treated as an additional *_usermenu.lua* file.

All utilities added by `AddToMenu` and `AddCommand` calls within the file specified by `filename` share a common environment that does not change until the next *Reload User File* operation (or until FAR termination).

In fact, the plugin itself executes `AddUserFile("_usermenu.lua")`.

AutoInstall

`AutoInstall (startpath, [filepattern], [depth])`

Function `AutoInstall` is available to the *_usermenu.lua* script.
It looks for files whose names match `filepattern` in the directory given by `startpath` and its subdirectories recursively, and calls `AddUserFile` for each matching file found.

Parameters

1. `startpath` (string)
   A path relative to the plugin's directory

2. `filepattern` (string, or nil)
   Lua regular expression specifying what files to install. Only file name (without path) is matched against this regular expression. If the parameter is not specified, the value `"^_usermenu%.lua$"` is used.

3. `depth` (number, or nil)
   The maximum depth of subdirectories to recurse into. When it is 0, the search is conducted only in `startpath`. When it is not specified, the recursion depth is unlimited.

Warning

This function should be used with caution since it runs every matching file found.

Examples

```
AutoInstall ("scripts")
AutoInstall ("scripts", nil, 1)
AutoInstall ("scripts/cool", "^_.*menu%.lua$", 0)
```

MakeResident

`MakeResident (file)`

Function `MakeResident` is available to the *_usermenu.lua* script.
It allows to add a file, containing one or more handlers that will be further called on some FAR events.

The following handlers are supported:

- `ProcessEditorInput`
- `ProcessEditorEvent`
- `ProcessViewerEvent`
- `ExitScript`

Parameters

1. `file`
   A specification for a Lua script that is run by this function. The exact file name is determined according to the same rules as with `require` function, that is, `package.path` is used to search for the file. But contrary to `require`, the value returned by the script is not cached in `package.loaded`.

Notes about the handlers

1. Handlers must be defined as global functions.
2. There can be multiple handlers for the same event type. They will be called in the order their files are specified in `_usermenu.lua`.
3. The handlers' input parameters correspond to the *exported functions* with the same names, e.g. `ProcessViewerEvent` corresponds to `export.ProcessViewerEvent`, etc. (see LuaFAR manual for details). `ExitScript` has no parameters, no return value.
4. If the return value of a handler's `ProcessEditorInput` is true, the rest of `ProcessEditorInput` handlers are not called, and true is returned to Far.
5. The return values of `ProcessEditorEvent` and `ProcessViewerEvent` are ignored.

6. `ProcessEditorInput` handlers are not called when a user-defined hot key is pressed.
7. `ExitScript` is called when the plugin is about to be unloaded. It is also called before the “Reload User file” operation.

Passing data to installed scripts

An installed script, when it is run conventionally, always receives a single argument of *table* type.

Scripts installed via `AddToMenu` call (run from plugin menu or shortcut)

- The array part of the table contains the additional arguments specified in the `AddToMenu` call.
- The hash part of the table has the field **`From`** that contains either of the following strings: **“config”**, **“dialog”**, **“editor”**, **“panels”** or **“viewer”**.
- In the case `From=="dialog"`, the table also has the field **`hDlg`**, that contains the dialog handle (a userdata value).

Scripts installed via `AddCommand` call (run via plugin command)

- The array part of the table contains the additional arguments specified in the `AddCommand` call followed by the command line arguments.
- The hash part of the table has the field **`From`** that contains the string **“panels”**.

Packets of scripts

If there is a set of utilities that is distributed as a single unit (“packet of scripts”), it makes sense to install it separately from other scripts.

- The standard location for adding packets of scripts is **`<plugin_path>/scripts`**. It is recommended to install the packet in subdirectory **`<plugin_path>/scripts/<packet_name>`**.

- The plugin modifies `package.path` by adding at the beginning: **`<plugin_path>/scripts/?.lua;`**. (So the user doesn’t have to.)

- Due to the danger of module names collision, it is **not** recommended for packet writers to further modify `package.path`. Instead, start the argument of every `require` call with `<packet_name>`. E.g., if the packet is named *fl_scripts* then do: `require 'fl_scripts/utils/read_config'`

Utility for localization

The plugin contains a utility for adding localization to user packets of scripts. The utility consists of two files: <plugin_directory>/**`lf4ed_lang.lua`** and **`far2/makelang.lua`**.

(1) Create a "language template" file (similar to **`lf4ed_lang.templ`** file in the plugin directory), let's assume it is *scripts/my_package/lang.templ*.

- The exact syntax of "language template" files is described in the file **`far2/makelang.lua`**. The template file should be in UTF-8 encoding, with or without BOM.

- Choose some prefix for all your message identifiers (e.g. "mp"), to avoid conflicts with the existing message identifiers (if conflicts occur, they are detected by the program).

- Every script using this message system should **`require "lf4ed_message"`**. This returns a table that can be accessed for retrieving localized messages.

(2) Run the following command from the plugin's directory:

```
lua lf4ed_lang.lua scripts/my_package/lang.templ
```

If no errors occured, this will extend **`*.lng`** files and **`lf4ed_message.lua`** file with the localized messages of your script package.

(3) Restart Far.

Example of use:

```
local M = require "lf4ed_message"
........
far.Message(M.mpSomeMsgText, M.mpSomeMsgTitle, M.mpSomeMsgButton
```

Binary modules

Sometimes, user scripts may need some binary module (e.g., *LuaFileSystem*) for its functioning. There are two ways of installing the binary modules:

- If there is no specific setup for changing `package.cpath` (usually via the environment variable `LUAFAR_CPATH`), then put the binary modules into `%FARHOME%` directory.
- Otherwise, put the binary modules into any directory listed in `package.cpath`.

Example of use

_usermenu.lua

```
AddToMenu("e",   "Count words",   "Alt+F2",  "edit.count", "words")
AddToMenu("e",   "Count letters", "Alt+F12", "edit.count", "letters"
AddToMenu("evp", "Calendar",      nil,       "calendar",   "show")
AddToMenu("c",   "Calendar",      nil,       "calendar",   "config")
AddCommand("calen", "calendar", "show")
AddUserFile("scripts/fl_scripts/_usermenu.lua")
AddUserFile("scripts/Rh_Scripts/_testmenu.lua")
MakeResident("handlers")
```

handlers.lua

```
local F = far.Flags

function ProcessEditorInput (Rec)
  if (Rec.EventType == F.FARMACRO_KEY_EVENT) or
     (Rec.EventType == F.KEY_EVENT and Rec.bKeyDown)
  then
    if Rec.AsciiChar == ("t"):byte() then
      editor.InsertText(nil, "X")
      editor.Redraw()
      return true
    end
  end
end

function ProcessEditorEvent (Event, Param)
  if Event == F.EE_READ then
    require 'fl_scripts/editor/template'
    templates_menu()
  end
end
```

Command line calls

Syntax

```
lfe: [<options>] <command>|-r<filename> [<arguments>]
```

Options

```
-a          asynchronous execution
-e <str>    execute string <str>
-l <lib>    load library <lib>
```

Command

Any command added via `AddCommand` function in *_usermenu.lua*.

Filename

Name of a Lua script file. It can be either absolute, or relative to the current directory.

Example

```
lfe: calc 2+2
```

Macro calls

Macro call syntax

1. `Plugin.Call(guid, "code", <code> [,<arguments>])` Execute string containing Lua code `<code>`.

2. `Plugin.Call(guid, "file", <filename> [,<arguments>])`
   Execute Lua script `<filename>`.
   `<filename>` may contain environment variables.

3. `Plugin.Call(guid, "command", <command> [,<arguments>])`
   Execute `<command>` (any command added via `AddCommand` function in *_usermenu.lua*).

4. `Plugin.Call(guid, "own", <command> [,<arguments>])`
   Execute own (internal) plugin's command `<command>`.

Examples

```
local guid = "6F332978-08B8-4919-847A-EFBB6154C99A"

Plugin.Call(guid, "code", "return 2+2,3+3")

Plugin.Call(guid, "file", "%farprofile%\\tests\\test1.lua", "ful

Plugin.Call(guid, "command", "calc", "2+2")
```

Plugin API

---

1. There is an important thing to know when writing scripts for *LuaFAR for Editor*: indexing string variables accesses functions in **`unicode.utf8`** rather than in **`string`** namespace.
   - For example, `s:sub(1,2)` means `unicode.utf8.sub(s,1,2)`.
   - To use `string` library, specify that explicitly, e.g., `string.sub(s,1,2)`.
   - **`#s`** refers to `string.len(s)`. Use **`s:len()`** to obtain number of characters.
2. The plugin has a few functions that are available to user scripts. They are placed under **lf4ed** namespace.
   - `lf4ed.config`
   - `lf4ed.version`
   - `unicode.utf8.cfind`

lf4ed.config

Get or set the plugin configuration.

```
cfg = lf4ed.config ([newcfg])
```

Parameters:

newcfg: table
Fields of *newcfg* (every field is optional):

ReloadDefaultScript	boolean
RequireWithReload	boolean
UseStrict	boolean
ReturnToMainMenu	boolean

Returns:

cfg: table (the configuration as it was before the call)

Description:

If *newcfg* is given, it is a table with configuration parameters to be set. Parameters not contained in this table will remain unchanged. This means that (newcfg.param == false) will set 'param' to false, but (newcfg.param == nil) will leave 'param' as it was before the call.
Returned is a copy of the "old" configuration table (as it was before the call).

If *newcfg* is not given, an up-to-date copy of the configuration table is returned. This table contains the same fields as the *newcfg* table described above.

Note:

Configuration changes done by a user script via this function are reverted by the plugin after the user script exits.

unicode.utf8.cfind

It is a helper function. It behaves like `unicode.utf8.find` except that it treats its input offset and expresses its output offsets in **characters** rather than **bytes**. (The only exception are “position captures” that are still returned expressed in bytes).

Lua Modules

There are a few Lua modules that can be used in the utilities added by the user:

- far2.dialog
- far2.history
- far2.searchmenu

far2.dialog

Module `far2.dialog` makes common operations with FAR dialogs easier. It contains the following functions: `NewDialog`, `LoadData` and `SaveData`.

The module is loaded as follows:

```
require "far2.dialog"
```

NewDialog

```
dlg = far2_dialog.NewDialog()
```

Parameters:
  none

Returns:
  dlg: Dialog object (a table).
       It represents the full set of dialog items, and is eventually
       passed to function *far.Dialog* as its 6-th parameter.

Description:
The dialog object has the following features:

1. To add an item, assign it to some string field of the object, e.g

```
  dlg.cbxCase = {"DI_CHECKBOX",10,4,0,0, 0, "","",0, "&Case sensi
  dlg.cbxWord = {"DI_CHECKBOX",10,5,0,0, 0, "","",0, "&Whole word
```

   The added items are now accessible by their names: dlg.cbxCase,
   dlg.cbxWord. If there are items that need not to be accessed
   after their adding, they can be assigned the same name, e.g.,
   dlg.label or dlg._

2. The properties of the added items are accessible in two ways:
   either by index, or by name.

```
  print(dlg.cbxCase[3]) --> 4
  print(dlg.cbxCase.Y1) --> 4
  dlg.cbxCase.Y1 = 6
  print(dlg.cbxCase[3]) --> 6
  print(dlg.cbxCase.Y1) --> 6
```

2.1. Here is the correspondence between indexes and names of dialog
   item properties (wherever multiple names are listed for an index,
   any of them may be used):

```
   1 : Type
   2 : X1
   3 : Y1
   4 : X2
   5 : Y2
   6 : Selected, ListItems, VBuf
   7 : History
   8 : Mask
```

```
 9 : Flags
10 : Data
11 : MaxLength
12 : UserData
```

LoadData

```
far2_dialog.LoadData(aDialog, aData)
```

Parameters:
  aDialog : a dialog object created by a NewDialog call
  aData   : a table with data to load into *aDialog*

Returns:
  nothing

Description:
  The function copies input data *aData* into a dialog object *aDialog*.
  The dialog items must be added to the object *before* this function
  is called, since this function loads data only to existing dialog
  items.

  - The following item types are supported by the function:
    DI_CHECKBOX, DI_RADIOBUTTON, DI_EDIT, DI_FIXEDIT, DI_LISTBOX,
    DI_COMBOBOX.

  - The following properties are loaded by the function:
     - For DI_CHECKBOX, DI_RADIOBUTTON: only index 6 ("Selected").
     - For DI_LISTBOX, DI_COMBOBOX: only field "SelectIndex"
       of index 6 ("ListItems").
     - For DI_EDIT, DI_FIXEDIT: only index 10 ("Data").

  - Data are loaded to items whose names are identical to the names
    of the *aData* fields.

  - If an item has either of fields *_noautoload* or *_noauto* set to
    true, it is not loaded.

Example:

```
local dlg = far2_dialog.NewDialog()
dlg.cbxCase = {"DI_CHECKBOX",10,4,0,0, 0, "","",0, "&Case sensitiv
dlg.cbxWord = {"DI_CHECKBOX",10,5,0,0, 0, "","",0, "&Whole words"}
far2_dialog.LoadData(dlg, {cbxCase=true, cbxWord=false})
```

SaveData

```
far2_dialog.SaveData(aDialog, aData)
```

Parameters:
  aDialog : a dialog object created by a NewDialog call
  aData   : a table to save data in from *aDialog*

Returns:
  nothing

Description:
  The function copies data from a dialog object *aDialog* to *aData*.

  - The following item types are supported by the function:
    DI_CHECKBOX, DI_RADIOBUTTON, DI_EDIT, DI_FIXEDIT, DI_LISTBOX,
    DI_COMBOBOX.

  - The following properties are saved by the function:
     - For DI_CHECKBOX, DI_RADIOBUTTON: only index 6 ("Selected").
     - For DI_LISTBOX, DI_COMBOBOX: only field "SelectIndex"
       of index 6 ("ListItems").
     - For DI_EDIT, DI_FIXEDIT: only index 10 ("Data").

  - Data are saved by the names identical to the item names.

  - If an item has either of fields *_noautosave* or *_noauto* set to
    true, it is not saved.

Example:

```
local dlg = far2_dialog.NewDialog()
dlg.cbxCase = {"DI_CHECKBOX",10,4,0,0, 0, "","",0, "&Case sensitiv
dlg.cbxWord = {"DI_CHECKBOX",10,5,0,0, 0, "","",0, "&Whole words"}
-- add other items
-- call far.Dialog(...)
local data = {}
far2.dialog.SaveData(dlg, data)
return data
```

far2.history

Module `history` saves specified plugin data to files and loads the data from files. The module API consists of functions that create objects and methods of those objects.

Functions

- `newfile`
- `newsettings`

Methods

- `hobj:field`
- `hobj:setfield`
- `hobj:serialize`
- `hobj:save`

The module is loaded as follows:

```
require "far2.history"
```

newfile

```
hobj = far2_history.newfile (filename)
```

Parameters:
```
  filename : string
```

Returns:
```
  hobj     : history object (a table).
```

Description:

- The function executes *filename* as a Lua script, in an empty environment table.

- The script is assumed to contain Lua data in the global variable *Data* (a table).

- The environment table is returned, with its field *FileName* set to the value of *filename* argument.

- If the file *filename* was absent or failed to compile, then the field *Data* of the returned object is an empty table.

- The returned history object *hobj* has four methods: hobj:field, hobj:setfield, hobj:serialize and hobj:save.

newsettings

```
hobj = far2_history.newsettings (Subkey, Name)
```

Parameters:

```
  Subkey: string; nil for the root key
  Name  : string
```

Returns:

```
  hobj  : history object (a table).
```

Description:

- The function reads in the data from Far plugin settings database and executes this data as a Lua script, in an empty environment table.

- The script is assumed to contain Lua data in the global variable *Data* (a table).

- The environment table is returned, with its fields *Subkey* and *Name* set to the value of the respective received arguments.

- If the subkey or data were absent or failed to compile, then the field *Data* of the returned object is an empty table.

- If *Subkey* argument contains dots, then hierarchical subkeys are created in the database. E.g. specifying "key1.key2.key3" will create or access subkey "key3" under subkey "key2" under subkey "key1" under root.

- The returned history object *hobj* has four methods: hobj:field, hobj:setfield, hobj:serialize and hobj:save.

hobj:field

```
val = hobj:field (name)
```

Parameters:

```
  name   : sequence of dot-delimitered fields;
```

Returns:

```
  val    : value of the given nested field in history object hobj
           (will be created if absent).
```

Description:

```
  - hobj:field("key1.key2 ... keyN") returns the equivalent of
    hobj["key1"]["key2"]...["keyN"].

  - If at any stage of retrieving some intermediate nested field
    "keyM", its value is nil, then both that and all subsequent
    fields are created by assigning each of them a new table.
```

Example:

```
  if From == "e" then     hist = _Hist:field ("menu.editor")
  elseif From == "v" then hist = _Hist:field ("menu.viewer")
  elseif From == "p" then hist = _Hist:field ("menu.pluginsmenu")
  else return
```

hobj:getfield

```
val = hobj:getfield (name)
```

Parameters:

```
  name   : sequence of dot-delimitered fields;
```

Returns:

```
  val    : value of the given nested field in history object hobj
```

Description:

```
  - hobj:field("key1.key2 ... keyN") returns the equivalent of
    hobj["key1"]["key2"]...["keyN"].
```

hobj:save

---

hobj:save ()

Parameters:
  none

Returns:
  nothing

Description:
  - The method serializes and saves the "history object". The object
    into a file (see newfile), or into a data base entry (see newset
    corresponding file (or the data base entry) exist, their content

  - Only "Data" field of the *hobj* is saved.
    - Within it, values of the following types are saved:
      numbers, strings, booleans and tables (recursively).
    - Not saved: functions, coroutines and userdatas.
    - Not saved: metatable relations.

hobj:serialize

```
str = hobj:serialize ()
```

Parameters:
  none

Returns:
  str: string

Description:
  - The method serializes the "history object" into a string.

  - Only "Data" field of the *hobj* is saved.
    - Within it, values of the following types are saved:
      numbers, strings, booleans and tables (recursively).
    - Not saved: functions, coroutines and userdatas.
    - Not saved: metatable relations.

hobj:setfield

```
val = hobj:setfield (name, val)
```

Parameters:

```
  name  : sequence of dot-delimitered fields;
  val   : value to set the field with
```

Returns:

```
  val
```

Description:

```
  - hobj:setfield("key1.key2  ... keyN", val) does the equivalent of
    hobj["key1"]["key2"]...["keyN"] = val.

  - If at any stage of retrieving some intermediate nested field
    "keyM", its value is nil, then both that and all subsequent
    fields are created by assigning each of them a new table.
```

far2.message

Message

---

result = far2_message.Message (Text, Title, Buttons, Flags, HelpTopi

Parameters:

```
  Text       : Text elements to display inside the dialog frame.
               Either a string or a table, depending on flag 'c'.
               Sequences '\n', '\r\n' and '\r' are treated as line s
  Title      : Title string; optional.
  Buttons    : Buttons string; ';' and '\n' serve as button separato
               Buttons automatically wrap on multiple lines if don't
               line. Separator '\n' forces new line for the next but
  Flags      : Concatenation of 0 or more character flags; optional.
               'l' - left-align text lines (default: center lines on
               'w' - use "warning" color set for the dialog and its
               'R' - don't wrap long text lines (default: wrap).
               'c' - "color"-mode, that changes treating the aText a
                  this flag set, aText should be an array of individ
                  each of which is either a string or a table.
                  A table elements may have the following fields:
                     "text" (string)
                     "color" (number; optional)
                     "separator" (1 = single line, 2 = double line;
                  Each element's begins at the position next to the
                  element's end. Separators are always put on separa
  HelpTopic  : Help topic string; optional.
  Id         : Dialog Id; binary GUID string; optional.
```

Returns:

```
  result     : negative number when dialog was canceled, button numb
               (1 is the first button).
```

TableBox

---

result = far2_message.TableBox (items, title, buttons, flags, helpto

Display a two-column table.

Parameters:
  items: an array of rows (tables); row[1],row[2] = left and right c
         A row can also be a single or double separator line if row.
         1 or 2; a separator can have optional field row.text.

Other parameters and return value are similar to those of *far.Messag*

far2.searchmenu

Description

This provides a filter for standard menu. Every typed symbol is added to the pattern which is used to check every menu item. Every item which satisfies the pattern is displayed, others become hidden. The pattern is displayed in the menu header.

Parameters

```
Item, Position = far2.searchmenu(Properties, Items [, BreakKeys])
```

All arguments and return values are the same as for **`far.Menu`** (see LuaFAR manual), except some optional additional fields of `Properties` table:

```
  AllowEmpty:    Allow the user to input patterns that make menu
                 empty (boolean)

  CheckItem:     Function for determining if an item should be
                 displayed:
                   from,to = CheckItem(pattern,text[,searchmethod])

  Map:           Table that maps keys and key combinations
                 to characters and actions. It allows to add
                 keys or redefine the treatment of user's input.

  Menu:          Function for displaying the menu
                 (defaults to far.Menu)

  Pattern:       Initial search pattern (string)

  SearchMethod:  "lua"   = use Lua regexps (default)
                 "dos"   = use DOS wildcards * and ?
                 "plain" = plain text search
```

Also, there is an optional field `SearchText` in a menu item. When present, it is used instead of the `text` field when checking the item, while the `text` field is used for displaying the item.

Predefined keys

```
Space     - insert a space character
DELETE    - delete the entire pattern
BACKSPACE - delete the last symbol
CtrlV     - insert a pattern from the clipboard
```

Available symbols

```
Small English letters: a-z
Numbers:  0-9
Symbols: .,><=+-_;:/?`~[]{}()\~|'"!@#$%^&*
```

far2.tableview

Диалог-браузер таблиц lua.

Работает так:
showDialog('_G', _G) - отображает таблицу _G (верит, что она находит
                       адресу '_G')
showDialog('_G',)   - отображает таблицу _G, получая её по адресу
showDialog(nil, _G) - отображает таблицу _G, адресная строка в этом
                       содержит запись <internal>

Когда в фокусе находится поле, там можно ввести адрес таблицы и откр
на редактирование.
Когда в фокусе находится список:
    Enter - открыть таблицу под курсором
    BS   - вернутся к предыдущей таблице
    Ins   - Вставить новое поле. Запрашиваются четыре значения: тип
            (number, boolean, string), ключ, тип значения (тоже само
            значение
    Del   - Удалить поле. С подтверждением.
    F4   - Редактировать значение. При этом тип сохраняется.

Когда в фокусе функция, по enter можно её выполнить, передав список

Отображаемая информация: полное количество элементов в текущей табли
так-же поля метатаблицы в заголовке. Строки, числа, булевы переменны
как есть. Функция - function.
Для таблицы отображается число элементов в массиве, наличие метатабл
отдельно пишется, если таблица пуста.

far2.utils

This module is intended for the development of LuaFAR plugins. User scripts usually do not need it.

AddMenuItems

```
trg = utils.AddMenuItems (trg, src, msgtable)
```

Parameters:

```
  trg :      table (array of menu items), or nil
  src :      table (array of menu items)
  msgtable:  table (localization conversion table)
```

Returns:

```
  trg:       table
```

Description:

```
  -- Adds menu items from src table to trg table.
  -- If trg argument is nil, a new empty table is first created.
  -- The items whose text starts with :: are replaced with items
     with their text converted by msgtable (see AddToMenu and
     Utility for localization).
```

GetPluginVersion

```
version = GetPluginVersion()
```

Parameters:
  none

Returns:
  version: string, e.g. "3.0.0"

InitPlugin

```
plugin = utils.InitPlugin()
```

Parameters:
  none

Returns:
  plugin: table

Description:
  The function does the following:
  -- redirects indexing of strings to *unicode.utf8* table, so that e.
     str:len() ends up with *unicode.utf8.len(str)* instead of *string.*
  -- adds function unicode.utf8.cfind.
  -- sets up *export.OnError* function.
  -- returns a table with *ModuleDir* field set to plugin's directory.

LoadUserMenu

```
menuItems, commands, hotKeys, handlers = utils.LoadUserMenu (FileNam
```

Parameters:

```
  FileName:  string (filename relative to the plugin's directory;
                     usually it is "_usermenu.lua")
```

Returns:

```
  menuItems: table with the following structure:
               { editor={},viewer={},panels={},config={},dialog={} }

  commands:  table with the following structure:
               { <command1>={}, ..., <commandN>={} }

  hotKeys:   table with the following structure:
               { <hotkey1>={}, ..., <hotkeyN>={} }

  handlers:  table with the following structure:
               { EditorInput={},EditorEvent={},ViewerEvent={},ExitSc
```

Description:

1. An environment table *env* containing all the Installation functi
2. env.AddUserFile(FileName) is run.

OpenMacro

```
results = utils.OpenMacro (Item, Commands, ConfigFunc)
```

Parameters:

Item:	table
Commands:	table
ConfigFunc:	function, or nil

Returns:

results:	zero or more Lua values

Description:

This function is intended to be called from *export.Open* function. The function does arguments processing as described in the section Macro calls.

This function should not be called when the plugin creates panels line or macro calls.

OpenCommandLine

---

utils.OpenCommandLine (Item, Commands, ConfigFunc)

Parameters:

Item:	integer or string
Commands:	table
ConfigFunc:	function, or nil

Returns:

result:	nothing

Description:

This function is intended to be called from *export.Open* function. The function does arguments processing as described in the section Command line calls.

This function should not be called when the plugin creates panels line or macro calls.

RunInternalScript

```
result = utils.RunInternalScript (name, ...)
```

Parameters:

```
  name:    string
             either field name in package.preload table (without pre
             or file name relative to plugin dir (without suffix ".l
  ... :    additional parameters (optional)
```

Returns:

```
  result:  any type
```

Description:

This function is intended to run utilities coming with the plugin to "user's utilities"). Those utilities can either be part of the ("embedded"), or located in disk files.

For example, given argument *name* == "wrap", the function will firs to run a script from package.preload["<wrap"], then from the file <plugin_directory>/wrap.lua.

RunUserItem

```
results = RunUserItem (Item, Properties, ...)
```

Parameters:

```
  Item:       table
                filename: string; script file specification
                env:      table; environment to run the script in
                arg:      table; array of arguments associated with

  Properties: table
                From:     string ("config", "dialog", "editor", "pan
                hDlg:     userdata (dialog handle), or nil

  ... :       sequence of additional arguments (appended to existing
```

Returns:

```
  results:    zero or more Lua values
```

Description:

The function runs user's menu item, created as a result of executi

Item.filename is a specification for a Lua script to run upon this
The exact file name is determined according to the same rules as w
that is, `package.path` is used to search for the file. But contra
the value returned by the script is not cached in `package.loaded`

For the script's input arguments, see Passing data to installed sc

Credits

Many thanks to:

- **Maxim Gonchar**: ideas, bug reports, *Searchable Menu* and *Table View* scripts.
- **GalS**: bug reports.
- **ccaid**: bug reports.
- **Aidar Rakhmatullin**: ideas, translation of the help file into Russian; bug reports.
- **Grey**: bug reports.
- **Vadim Yegorov**: ideas, code examples, bug reports.