ahist - a simple seach history for Acme

(version 0.4.3)

 $Al exander \ Sychev \ (santucco@gmail.com)$

2 INTRODUCTION a hist (version 0.4.3) $\S 1$

1. Introduction. This is an implementation of ahist command for Acme. It tracks all search requests in Acme's window to a separate window.

§2 ahist (version 0.4.3) IMPLEMENTATION

2. Implementation.

```
// This file is part of ahist
   // Copyright (c) 2020 Alexander Sychev. All rights reserved.
   // Redistribution and use in source and binary forms, with or without
   // modification, are permitted provided that the following conditions are
   // met:
      * Redistributions of source code must retain the above copyright
   // notice, this list of conditions and the following disclaimer.
   // * Redistributions in binary form must reproduce the above
   // copyright notice, this list of conditions and the following disclaimer
   // in the documentation and/or other materials provided with the
   // distribution.
   // * The name of author may not be used to endorse or promote products derived from
   // this software without specific prior written permission.
   // THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
   // "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
   // LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
   // A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
   // OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
   // SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
   // LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE.
   // DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
   // THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
   // (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
   // OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
package main
import(
  (Imports 4)
var(
  (Global variables 5)
type(
  \langle \text{Types 44} \rangle
```

4 STARTUP ahist (version 0.4.3) §3

```
3.
       Startup.
   func main(){
       \langle Store a name of the program 13 \rangle
        \langle \text{ Obtaining of } id \text{ of a window } 11 \rangle
        \langle \text{ Open window } w \text{ by } id \text{ 19} \rangle
        \langle Change the name of the program in the tag 14\rangle
       \langle \text{Read } name \text{ of the window } 22 \rangle
       (Start history processing 48)
       (Processing window events 18)
4.
\langle \text{Imports 4} \rangle \equiv
   "fmt"
   "os"
See also sections 9, 17, and 20.
This code is used in section 2.
      Let's define dbg flag and will switch it by ahist + and ahist - .
\langle \text{Global variables 5} \rangle \equiv
   dbg bool
See also sections 10, 12, 21, 35, and 45.
This code is used in section 2.
6.
\langle Switch debug output on 6\rangle \equiv
   dbg = \mathbf{true}
   debug("debug_{\square}has_{\square}been_{\square}switched_{\square}on\n")
This code is used in section 27.
7.
\langle Switch debug output off 7 \rangle \equiv
   debug("\mathtt{debug} \sqcup \mathtt{has} \sqcup \mathtt{been} \sqcup \mathtt{switched} \sqcup \mathtt{off} \setminus \mathtt{n"})
   dbg = \mathbf{false}
This code is used in section 27.
8.
   \mathbf{func} \ \ debug(f \ \mathbf{string}, args \ \dots \mathbf{interface} \{\}) \{
          fmt.Fprintf(os.Stderr, f, args...)
   }
\langle \text{Imports 4} \rangle + \equiv
   "strconv"
10.
\langle Global variables 5\rangle + \equiv
   id int
```

```
§11
       ahist (version 0.4.3)
11.
\langle Obtaining of id of a window 11\rangle \equiv
     var err error
     id, err = strconv.Atoi(os.Getenv("winid"))
     if err \neq nil {
       return
This code is used in section 3.
12.
\langle \text{Global variables 5} \rangle + \equiv
  tagname string
13.
\langle Store a name of the program 13 \rangle \equiv
  tagname = os.Args[0]
  if n := strings.LastIndex(tagname, "/"); n \neq -1  {
     tagname = tagname[n:]
  debug("tagname:%s\n", tagname)
This code is used in section 3.
14. We change ahist into -ahist to add a possibility to switch ahist off.
\langle Change the name of the program in the tag 14\rangle \equiv
     del := [] string \{tagname, "-" + tagname, "-" + tagname + "+", "-" + tagname + "-" \}
     add := []\mathbf{string}\{" \text{-"} + tagname\}
     changeTag(w, del, add)
  }
This code is used in section 3.
15. On exit we should make an opposite change.
\langle \text{Cleanup } 15 \rangle \equiv
     del := [] string \{tagname, "-" + tagname, "-" + tagname + "+", "-" + tagname + "-" \}
     add := []string\{tagname\}
     changeTag(w, del, add)
  }
See also sections 42 and 46.
```

This code is used in sections 18 and 27.

STARTUP

5

6 EVENTS HANDLING ahist (version 0.4.3) $\S16$

16. Events handling.

```
17.
\langle \text{Imports 4} \rangle + \equiv
    "github.com/santucco/goacme"
18.
\langle Processing window events 18 \rangle \equiv
   \langle Fix tag of the window 41\rangle
       ev, err := w.ReadEvent()
       if err \neq nil {
          \langle \text{Cleanup } 15 \rangle
          return
       ⟨ Process main window 23⟩
This code is used in section 3.
19.
\langle \text{ Open window } w \text{ by } id \text{ 19} \rangle \equiv
   w, err := goacme.Open(id)
   if err \neq nil {
       debug(\texttt{"cannot} \sqcup \texttt{open} \sqcup \texttt{a} \sqcup \texttt{window} \sqcup \texttt{with} \sqcup \texttt{id} \sqcup \texttt{\%d} : \sqcup \texttt{\%s} \backslash \texttt{n"}, id, err)
       return
   defer w.Close()
This code is used in section 3.
20.
\langle \text{Imports 4} \rangle + \equiv
   "strings"
21.
\langle Global variables 5\rangle + \equiv
   name string
```

```
22.
\langle \text{Read } name \text{ of the window } 22 \rangle \equiv
         f, err := w.File("tag")
         if err \neq nil {
             debug(\texttt{"cannot}_{\square}\texttt{read}_{\square}\texttt{from}_{\square}\texttt{'tag'}_{\square}\texttt{of}_{\square}\texttt{the}_{\square}\texttt{window}_{\square}\texttt{with}_{\square}\texttt{id}_{\square}\%\texttt{d}:_{\square}\%\texttt{s}\\ \land \texttt{m}^{"}, id, err)
             return
         if \_, err := f.Seek(0,0); err \neq nil  {
             debug(\texttt{"cannot}_{\square} \texttt{seek}_{\square} \texttt{to}_{\square} \texttt{the}_{\square} \texttt{start}_{\square} \texttt{'tag'}_{\square} \texttt{of}_{\square} \texttt{the}_{\square} \texttt{window}_{\square} \texttt{with}_{\square} \texttt{id}_{\square} \% \texttt{d} :_{\square} \% \texttt{s} \\ \texttt{n} \texttt{"}, id, err)
             return
         var b [1000]byte
         n, err := f.Read(b[:])
         if err \neq nil {
             debug("cannot_{\square}read_{\square}tag_{\square}of_{\square}the_{\square}window_{\square}with_{\square}id_{\square}%d:_{\square}%s\n", id, err)
             return
         ss := strings.Split(\mathbf{string}(b[:n]), " \sqcup ")
         if len(ss) \equiv 0 {
             return
         name = \mathbf{string}(ss[0])
    }
This code is used in section 3.
23.
\langle \text{Process main window 23} \rangle \equiv
     \langle Process and continue if it is not Look in any form 24\rangle
     \langle \text{Process } Look \ 30 \rangle
     \langle \text{ Read addr into } b, e | 38 \rangle
     \langle \text{Show dot } 40 \rangle
     ⟨ Write history 53⟩
This code is used in section 18.
```

8 EVENTS HANDLING ahist (version 0.4.3) $\S24$

24. b, e address pair is taken from the ev event. $\langle \text{Process and continue if it is not } Look \text{ in any form } 24 \rangle \equiv$ $debug("incoming_uevent:_u%+v\n", ev)$ s := ""b := ev.Begine := ev.End $type_switch$: switch{ **case** $ev.Type \equiv goacme.Look \mid goacme.Tag:$ (Process in case of a request by B3 mouse button in the tag 25) **case** $ev.Type \equiv goacme.Look$: (Process in case of a request by B3 command in the body 26) **case** $ev.Type \equiv goacme.Execute \lor ev.Type \equiv goacme.Execute \mid goacme.Tag:$ (Process in case of executing a command in the body or tag 27) **case** $ev.Type \equiv goacme.Insert \lor ev.Type \equiv goacme.Delete$: $\langle \text{Fix tag of the window 41} \rangle$ continue default: (Unread event and continue 29) This code is used in section 23. We take a search string from ev event and set dot. Also we have to clean b, e because it is an address in the tag. $\langle \text{Process in case of a request by B3 mouse button in the tag 25} \rangle \equiv$ b, e = 0, 0s = ev. Textif $len(ev.Arg)\rangle 0$ { $s += " \sqcup " + ev.Arg$ \langle Set addr to dot 32 \rangle This code is used in section 24. We take a search string and address from ev event. $\langle \text{Process in case of a request by B3 command in the body 26} \rangle \equiv$ s = ev. Textif $len(ev.Arg)\rangle 0$ { $s += " \sqcup " + ev.Arg$ \langle Set addr to b, e 34 \rangle This code is used in section 24.

27. For Look command we set address and continue processing. ahist command we just ignore to avoid duplicates. -ahist command makes cleanups and processes to exit. ahist + and ahist - switch debug output on and off. All other commands are written back to "event" file and **fallthrough** to the next case, where a status of the window is checked.

```
\langle Process in case of executing a command in the body or tag 27\rangle \equiv
  switch strings. TrimSpace (ev. Text) {
    case "Look":
       \langle \text{Process in case of executing } Look \text{ command } 28 \rangle
       break type_switch
    case tagname:
       continue
    case "-" + tagname + "+":
       fallthrough
    case "-" + tagname + "-":
       fallthrough
    case "-" + tagname:
       debug("exiting\n")
       \langle \text{Cleanup } 15 \rangle
       return
    case tagname + "+":
       (Switch debug output on 6)
       continue
    case tagname + "-":
       (Switch debug output off 7)
       continue
  w.UnreadEvent(ev)
  fallthrough
This code is used in section 24.
```

28. We take a search string from an argument of Look command. Current address is set to dot, then b, e pair is set to the current address.

```
\langle Process in case of executing Look command 28 \rangle \equiv s = ev. Arg
\langle Set addr to dot 32 \rangle
\langle Read addr into b, e 38 \rangle
This code is used in section 27.

29.
\langle Unread event and continue 29 \rangle \equiv w. UnreadEvent(ev)
continue
This code is used in sections 24, 32, 33, 34, 37, 38, 39, and 40.
```

10 EVENTS HANDLING ahist (version 0.4.3) §30

30. If the ev event contains a search string, use it. Otherwise we should read selected the string from the window's body and read its address into b, e.

```
\langle \text{Process } Look | 30 \rangle \equiv
       if len(s) > 0 {
          \langle Make a search of s 37\,\rangle
       } else {
           (Look for selected string 31)
           \langle \text{Read addr into } b, e | 38 \rangle
This code is used in section 23.
31.
\langle \text{Look for selected string 31} \rangle \equiv
       \langle \text{Read selected string from "xdata" file to } s \mid 33 \rangle
       \langle \text{ Make a search of } s \text{ } 37 \rangle
This code is used in section 30.
32.
\langle \text{ Set addr to dot } 32 \rangle \equiv
   if w.WriteCtl("addr=dot") \neq nil  {
       (Unread event and continue 29)
   debug("set_laddr_lto_ldot\n")
This code is used in sections 25 and 28.
33.
\langle \text{Read selected string from "xdata" file to } s \mid 33 \rangle \equiv
       d, err := w.File("xdata")
       if err \neq nil {
          debug("cannot_{\square}read_{\square}from_{\square}'xdata'_{\square}of_{\square}the_{\square}window_{\square}with_{\square}id_{\square}%d:_{\square}%s\n", id, err)
           (Unread event and continue 29)
       buf := \mathbf{make}([]\mathbf{byte}, e - b + 1)
       \textbf{for} \ \ n, \square := d.Read(\mathit{buf}\,); \ \ n \rangle 0; \ \ n, \square = d.Read(\mathit{buf}\,) \ \ \{
          s += \mathbf{string}(\mathit{buf}[:n])
       debug(\texttt{"read}\_\texttt{address}\_\texttt{from}\_\texttt{xdata}\_\texttt{b}: \_\%\texttt{v}, \_\texttt{e}: \_\%\texttt{v} \land \texttt{n}", b, e)
This code is used in section 31.
```

11

```
\S34 ahist (version 0.4.3)
```

```
34.
```

```
 \langle \text{Set addr to } b, e \mid 34 \rangle \equiv \\ \text{if } err := w.WriteAddr("\#\%d,\#\%d",b,e); err \neq \text{nil } \{ \\ debug("cannot \sqsubseteq \text{write} \sqsubseteq \text{to} \sqsubseteq \text{'addr'} \sqsubseteq \text{of} \sqsubseteq \text{the} \sqsubseteq \text{window} \sqsubseteq \text{with} \sqsubseteq \text{id} \sqsubseteq \%d: \sqsubseteq \%s \upharpoonright, id, err) \\ \langle \text{Unread event and continue } 29 \rangle \\ \} \\ debug("set \sqsubseteq \text{addr} \sqsubseteq \text{to} \sqsubseteq \%d, \sqsubseteq \%d \urcorner, b,e) \\ \text{This code is used in sections } 26 \text{ and } 37.
```

35. We need to story previous history *entry* for the case, when *Look* in a tag is executed but without selected text. In the case a search string is taken from Acme. We take it from *lentr*

```
\langle Global variables 5\rangle +\equiv lentr \ entry
```

36. Let's add *empty* function for *entry*

```
\begin{array}{ll} \mathbf{func} & (\mathit{this\,entry}) & \mathit{empty}() & \mathbf{bool} \{ \\ & \mathbf{return} & \mathit{this}.b \equiv \mathit{this}.e \\ \} \end{array}
```

37. Search is processed by writing "/<regex>/" to "addr" file, but before regex-specific symbols of s have to be escaped In the case of empty search string we take it from lentr. Also we write the current position with the string to the history to track the search, because it already has a place.

```
\langle Make a search of s 37\rangle \equiv
   {
       debug("last\_entry\_: \_%v\n", lentr)
       if len(s) \equiv 0 {
          if \neg lentr.empty() {
              b = lentr.b
              e = lentr.e
              s = lentr.s
              \langle Set addr to b, e 34 \rangle
       } else if b \neq e {
          lentr = entry\{b, e, s\}
          ⟨Write history 53⟩
       es := escapeSymbols(s)
       debug("escaped\_search\_string:\_%q\n", es)
       if err := w.WriteAddr("/%s/", es); err \neq nil  {
          debug(\texttt{"cannot}_{\square} \texttt{write}_{\square} \texttt{to}_{\square} \texttt{'addr'}_{\square} \texttt{of}_{\square} \texttt{the}_{\square} \texttt{window}_{\square} \texttt{with}_{\square} \texttt{id}_{\square} \% \texttt{d} : \underline{\ \ \ } \texttt{vn''}, id, err)
          (Unread event and continue 29)
   }
```

This code is used in sections 30 and 31.

12 EVENTS HANDLING ahist (version 0.4.3) §38

```
38.
\langle \text{Read addr into } b, e | 38 \rangle \equiv
   b, e, err = w.ReadAddr()
   if err \neq nil {
       (Unread event and continue 29)
   debug(\texttt{"read}\_\texttt{address}\_\texttt{b}:\_\texttt{\%v,}\_\texttt{e}:\_\texttt{\%v}\texttt{\n"},b,e)
This code is used in sections 23, 28, and 30.
39.
\langle \text{ Set dot to addr } 39 \rangle \equiv
   if w.WriteCtl("dot=addr") \neq nil  {
       debug("cannot_{\sqcup}write_{\sqcup}to_{\sqcup}'ctl'_{\sqcup}of_{\sqcup}the_{\sqcup}window_{\sqcup}with_{\sqcup}id_{\sqcup}%d:_{\sqcup}%s\n",id,err)
       (Unread event and continue 29)
   debug("set_{\sqcup}dot_{\sqcup}to_{\sqcup}addr\n")
This code is used in section 40.
40.
\langle \text{Show dot } 40 \rangle \equiv
   \langle Set dot to addr 39\rangle
   if w.WriteCtl("show") \neq nil {
       debug(\texttt{"cannot} \sqcup \texttt{write} \sqcup \texttt{to} \sqcup \texttt{'ctl'} \sqcup \texttt{of} \sqcup \texttt{the} \sqcup \texttt{window} \sqcup \texttt{with} \sqcup \texttt{id} \sqcup \texttt{\%d} : \sqcup \texttt{\%s} \setminus \texttt{n"}, id, err)
       (Unread event and continue 29)
   debug("{\tt show} \sqcup {\tt dot} \")
This code is used in section 23.
41. Acme does not produce standard commands in case of opened "event" file. So we have to add command
"Put" in case of the window is modified and "Undo" and "Redo" commands too.
\langle \text{Fix tag of the window 41} \rangle \equiv
       -, -, -, -, d, -, -, err := w.ReadCtl()
      if err \neq nil {
          debug("cannot_{\square}read_{\square}from_{\square}'ctl'_{\square}of_{\square}the_{\square}window_{\square}with_{\square}id_{\square}%d:_{\square}%s\n", id, err)
       } else {
          debug(\texttt{"dirty:} \llcorner \texttt{\%v} \verb|\| n \texttt{"}, d)
          del := []string{"Put", "Undo", "Redo"}
          var add [string
          if d {
              add = \mathbf{append}(\mathit{add}, \texttt{"Put"})
          add = \mathbf{append}(add, "Undo", "Redo")
          changeTag(w, del, add)
   }
This code is used in sections 18 and 24.
```

 $\S42$ ahist (version 0.4.3) EVENTS HANDLING 13

```
42. Removing added commands on exit  \langle \text{Cleanup 15} \rangle +\equiv \\ \{ \\ del := \mathbf{append}([]\mathbf{string}\{\}, "Put", "Undo", "Redo") \\ change Tag(w, del, \mathbf{nil}) \\ \}
```

ahist (version 0.4.3)

Tracking search requests.

We create a window with history of search requests and make separated goroutine to process events from the window.

```
44.
\langle \, {\rm Types} \, \, {\color{red} 44} \, \rangle \equiv
  entry struct{
     b, e int
     s string
  }
This code is used in section 2.
45. Special histch channel is received entry to print them in the window
\langle \text{Global variables 5} \rangle + \equiv
  histch chan entry = make(chan entry)
46. On exit we should signal the goroutine to stop processing. It is made by closing histch channel
\langle \text{Cleanup } 15 \rangle + \equiv
  close(histch)
47.
\langle \text{ Variables outside the loop 47} \rangle \equiv
  \mathbf{var}\ hch\ \leftarrow \mathbf{chan}\ *goacme.Event
See also sections 49 and 54.
This code is used in section 48.
48. The goroutine handles two variants of events.
\langle Start history processing 48\rangle \equiv
  go func(){
      ⟨ Variables outside the loop 47⟩
      for{
         select
            case entr, ok := \leftarrow histch:
               \langle \text{Process } entr \text{ entry from } histch 50 \rangle
            case ev, ok := \leftarrow hch:
               \langle \text{Process } ev \text{ event from } hch \text{ event channel of the window 51} \rangle
  }()
This code is used in section 3.
49.
\langle \text{ Variables outside the loop } 47 \rangle + \equiv
  \mathbf{var}\ h\ *goacme.Window
```

50. Events from *histch* channel is written to the history.

```
\langle \text{Process } entr \text{ entry from } histch | 50 \rangle \equiv
  if \neg ok {
     if h \neq \mathbf{nil} {
        h.Del(\mathbf{true})
        h.Close()
        h = \mathbf{nil}
     return
  }
  (Open history window, if it does not exist 55)
  if history[entr.b] \neq entr.e {
     history[entr.b] = entr.e
     debug("writing_{\sqcup}to_{\sqcup}the_{\sqcup}history_{\sqcup}%d,%d\n", entr.b, entr.e)
     h.Write([]\mathbf{byte}(fmt.Sprintf("\%s:\#\%d,\#\%d_\%q\n",name,entr.b,entr.e,entr.s)))
     h. WriteCtl("clean")
  debug("selecting_the_current_position_#%d,#%d_in_the_history\n", entr.b, entr.e)
  es := fmt.Sprintf("#%d,#%d", entr.b, entr.e)
  (Make a selection of the current search request 52)
This code is used in section 48.
```

51. Event from *hch* channel is checked for a case the channel is close. In the case that means the history window is closed and we clear *h*, *hch* and *history*. Otherwise we just write the event back.

```
\langle \text{Process } ev \text{ event from } hch \text{ event channel of the window 51} \rangle \equiv
  if \neg ok {
     debug("history_is_closed\n")
     h.Del(\mathbf{true})
     h.Close()
     h = \mathbf{nil}
     hch = \mathbf{nil}
     history = nil
     continue
  h.UnreadEvent(ev)
  if ev.Type \equiv goacme.Look {
     debug("incoming\_event: \_\%+v\n", ev)
     debug ("selecting_the_current_position_\%q_in_the_history\n", ev.Text)
     es := escapeSymbols(ev.Text)
     \langle Make a selection of the current search request 52 \rangle
  }
This code is used in section 48.
52.
\langle \text{Make a selection of the current search request } 52 \rangle \equiv
  if err := h.WriteAddr("/%s/-+", es); err \neq nil  {
     debug("writing_lof_laddr_lfailed:_l%s\n", err)
    else if err := h.WriteCtl("dot=addr\nshow"); err \neq nil {
     debug("writing_of_ctl_failed: \%s\n", err)
  }
This code is used in sections 50 and 51.
```

```
53.
\langle \text{Write history } 53 \rangle \equiv
   debug(\texttt{"request}_{\sqcup} \texttt{to}_{\sqcup} \texttt{store}_{\sqcup} \texttt{a}_{\sqcup} \texttt{history} : _{\sqcup} \% \texttt{v} \, , \% \texttt{v}_{\sqcup} \% \texttt{q} \backslash \texttt{n} " \, , b, e, s)
   histch \leftarrow entry\{b: b, e: e, s: s\}
This code is used in sections 23 and 37.
54.
\langle \text{ Variables outside the loop } 47 \rangle + \equiv
   var history map[int]int
55. If the history window h does not exist, we create it and (re)create history map too.
\langle Open history window, if it does not exist 55\rangle \equiv
   {\bf if} \ h \equiv {\bf nil} \ \{
      var err error
      if h, err = goacme.New(); err \neq nil  {
         return
      h. WriteCtl("name_\%s", name + "+History")
      if hch, err = h.EventChannel(1, goacme.AllTypes); err \neq nil  {
         return
      history = \mathbf{make}(\mathbf{map}[\mathbf{int}]\mathbf{int})
   }
This code is used in section 50.
        change Tag function.
   We read the tag of w window, remove all commands from del list and add all commands from add list.
   func change Tag(w * goacme. Window, del [string, add [string)]
      if add \equiv \mathbf{nil} \wedge del \equiv \mathbf{nil} {
         return
       \langle \text{Read a tag of } w \text{ into } s \text{ 57} \rangle
      \langle Split tag into tag fields after the pipe symbol 58\rangle
       \langle \text{ Compose } newtag | 59 \rangle
       \langle Clear the tag and write newtag to the tag 61\rangle
```

```
ahist (version 0.4.3)
57.
\langle \text{Read a tag of } w \text{ into } s \text{ 57} \rangle \equiv
   f, err := w.File("tag")
   if err \neq nil {
      debug("cannot_read_from_r'tag'_lof_the_window_with_id_%d:_%s\n", id, err)
     return
   if \_, err := f.Seek(0,0); err \neq nil  {
      debug("cannot_seek_to_the_start_', tag'_of_the_window_with_id_%d:_%s\n", id, err)
     return
   var b [1000] byte
   n, err := f.Read(b[:])
   if err \neq nil {
      debug("cannot_{\square}read_{\square}tag_{\square}of_{\square}the_{\square}window_{\square}with_{\square}id_{\square}%d:_{\square}%s\n", id, err)
   }
   s := \mathbf{string}(b[:n])
This code is used in section 56.
58.
\langle \text{Split tag into } tag \text{ fields after the pipe symbol } 58 \rangle \equiv
   if n = strings.LastIndex(s, "|"); n \equiv -1  {
     n = 0
   } else {
     n+\!\!+\!\!
   }
   s = s[n:]
   s = strings.TrimLeft(s, " \sqcup ")
   tag := strings.Split(s, " \sqcup ")
This code is used in section 56.
59.
\langle \text{ Compose } newtag | 59 \rangle \equiv
   newtag := \mathbf{append}([]\mathbf{string}\{\},"")
   \langle Every part is contained in del we remove from tag 60\rangle
   newtag = \mathbf{append}(newtag, add \dots)
   newtag = \mathbf{append}(newtag, tag...)
This code is used in section 56.
```

```
60.
```

```
\langle Every part is contained in del we remove from tag 60\rangle \equiv
  for _{-},v:= range del {
     for i := 0; i\langle \mathbf{len}(tag); \{
       if tag[i] \neq v {
          i++
          continue
        }
       \mathbf{copy}(tag[i:], tag[i+1:])
       tag = tag[:\mathbf{len}(tag) - 1]
  }
This code is used in section 59.
61.
\langle Clear the tag and write newtag to the tag 61\rangle \equiv
  s = strings.Join(newtag, " \sqcup ")
  if err := w.WriteCtl("cleartag"); err \neq nil  {
     debug("cannot_{\square}clear_{\square}tag_{\square}of_{\square}the_{\square}window_{\square}with_{\square}id_{\square}%d:_{\square}%s\n", id, err)
     return
  if \_, err := f.Write([]\mathbf{byte}(s)); err \neq \mathbf{nil} \{
     debug("cannot | write | tag | of | the | window | with | id | %d: | %s | n", id, err)
     return
  }
This code is used in section 56.
62.
  func escapeSymbols(s string) (es string){
     for _{-},v:= range s \ \{
       \textbf{if} \ \ strings.ContainsRune("\\/[].+?()*^$",v) \ \{
          es += " \ " 
        es += \mathbf{string}(v)
     return
  }
add: 14, 15, 41, 56, 59.
                                                                  dbg: 5, 6, 7, 8.
addr: 37.
                                                                  debug: 6, 7, 8, 13, 19, 22, 24, 27, 32, 33, 34, 37,
ahist: 5, 27.
                                                                       38, 39, 40, 41, 50, 51, 52, 53, 57, 61.
All Types: 55.
                                                                  del: 14, 15, 41, 42, 56, 60.
Arg: 25, 26, 28.
                                                                  Del: 50, 51.
                                                                  Delete: 24.
Args: 13.
args: 8.
                                                                  empty: \underline{36}, \underline{37}.
                                                                  End: 24.
Atoi: 11.
Begin: 24.
                                                                  entr: 48, 50.
buf: 33.
                                                                  entry: 35, 36, 37, 44, 45, 53.
change Tag: 14, 15, 41, 42, \underline{56}.
                                                                  err: 11, 18, 19, 22, 33, 34, 37, 38, 39, 40, 41,
Close: 19, 50, 51.
                                                                       52, 55, 57, 61.
Contains Rune: 62.
                                                                  es: 37, 50, 51, 52, 62.
```

ahist (version 0.4.3)

19

Write: 50, 61.

 $xdata: \underline{31}, \underline{33}.$

WriteAddr: 34, 37, 52.

WriteCtl: 32, 39, 40, 50, 52, 55, 61.

```
escapeSymbols: 37, 51, \underline{62}.
ev: 18, 24, 25, 26, 27, 28, 29, 30, 48, 51.
event: \underline{27}, \underline{41}.
Event: 47.
Event Channel: 55.
Execute: 24.
File: 22, 33, 57.
fmt: \ \ \underline{4}, \ \ 8, \ \ 50.
Fprintf: 8.
Getenv: 11.
goacme: 17, 19, 24, 47, 49, 51, 55, 56.
hch: 47, 48, 51, 55.
histch: 45, 46, 48, 50, 53.
history: 50, 51, 54, 55.
id: 10, 11, 19, 22, 33, 34, 37, 39, 40, 41, 57, 61.
Insert: 24.
Join: 61.
LastIndex: 13, 58.
lentr: 35, 37.
Look: 24, 27, 28, 35, 51.
main: 2, \underline{3}.
name: 21, 22, 50, 55.
New: 55.
newtaq: 59, 61.
ok: 48, 50, 51.
Open: 19.
os: 4, 8, 11, 13.
Put\colon \ \underline{\textbf{41}}.
Read: 22, 33, 57.
ReadAddr: 38.
ReadCtl: 41.
ReadEvent: 18.
Redo: \underline{41}.
Seek: \overline{22}, 57.
Split: 22, 58.
Sprintf: 50.
ss: 22.
Stderr: 8.
strconv: \underline{9}, 11.
strings\colon \ \ \underline{20},\ 13,\ 22,\ 27,\ 58,\ 61,\ 62.
tag: 58, 59, 60.
Tag: 24.
tagname: 12, 13, 14, 15, 27.
Text: 25, 26, 27, 51.
this: 36.
TrimLeft: 58.
TrimSpace: 27.
Type: 24, 51.
type\_switch: 24, 27.
Undo: \underline{41}.
UnreadEvent: 27, 29, 51.
Window: 49, 56.
```

20 NAMES OF THE SECTIONS ahist (version 0.4.3)

```
\langle Change the name of the program in the tag 14\rangle Used in section 3.
Cleanup 15, 42, 46 \rangle Used in sections 18 and 27.
Clear the tag and write newtag to the tag 61 \ Used in section 56.
Compose newtag 59 Used in section 56.
Every part is contained in del we remove from tag 60 Used in section 59.
Fix tag of the window 41 \rangle Used in sections 18 and 24.
 Global variables 5, 10, 12, 21, 35, 45 Used in section 2.
 Imports 4, 9, 17, 20 Used in section 2.
 Look for selected string 31 Vsed in section 30.
 Make a search of s 37 \ Used in sections 30 and 31.
 Make a selection of the current search request 52 Used in sections 50 and 51.
 Obtaining of id of a window 11 \) Used in section 3.
 Open history window, if it does not exist 55 \ Used in section 50.
 Open window w by id 19 \text{ Used in section 3.}
 Process and continue if it is not Look in any form 24 \ Used in section 23.
 Process in case of a request by B3 command in the body 26) Used in section 24.
 Process in case of a request by B3 mouse button in the tag 25 \ Used in section 24.
 Process in case of executing a command in the body or tag 27 \ Used in section 24.
 Process in case of executing Look command 28 \rangle Used in section 27.
Process main window 23 \ Used in section 18.
 Process Look 30 Used in section 23.
 Process entr entry from histch 50 \ Used in section 48.
 Process ev event from hch event channel of the window 51 \rangle Used in section 48.
 Processing window events 18 \ Used in section 3.
 Read a tag of w into s 57\rangle Used in section 56.
 Read addr into b, e 38 Used in sections 23, 28, and 30.
 Read selected string from "xdata" file to s 33 \ Used in section 31.
 Read name of the window 22 \ Used in section 3.
Set addr to dot 32 \ Used in sections 25 and 28.
Set addr to b, e 34 Used in sections 26 and 37.
 Set dot to addr 39 \ Used in section 40.
 Show dot 40 V Used in section 23.
 Split tag into tag fields after the pipe symbol 58 \rangle Used in section 56.
Start history processing 48 \rangle Used in section 3.
Store a name of the program 13 \ Used in section 3.
Switch debug output off 7 Used in section 27.
Switch debug output on 6 \ Used in section 27.
Types 44 Vsed in section 2.
 Unread event and continue 29 \ Used in sections 24, 32, 33, 34, 37, 38, 39, and 40.
 Variables outside the loop 47, 49, 54 \ Used in section 48.
Write history 53 Used in sections 23 and 37.
```

ahist - a simple search history for Acme

(version 0.4.3)

	Sect	ion	Page
Introduction		. 1	2
Implementation		2	3
Startup		3	4
Events handling		16	6
Tracking search requests		43	14

Copyright © 2020 Alexander Sychev. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- The name of author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.