ahist - a simple seach history for Acme

 $({\rm Version}\ 0.1)$

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

 $2 \qquad \text{INTRODUCTION} \qquad \qquad \text{ahist (version 0.1)} \qquad \S 1$

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 0.1) IMPLEMENTATION

2. Implementation.

```
// This file is part of ahist version 0.1
   // 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 7)
type(
  \langle \text{Types 40} \rangle
```

4 STARTUP ahist (version 0.1) §3

```
3.
      Startup.
   func main(){
      \langle Store a name of the program 10\rangle
      Obtaining of id of a window 8
      \langle \text{ Open window } w \text{ by } id \text{ 16} \rangle
      \langle Change the name of the program in the tag 11\rangle
      \langle \text{Read } name \text{ of the window } 19 \rangle
      ⟨Start history processing 44⟩
      (Processing window events 15)
   }
4.
\langle \text{Imports 4} \rangle \equiv
   "fmt"
   "os"
See also sections 6, 14, and 17.
This code is used in section 2.
5.
   func debug(f \text{ string}, args ... \text{interface}\{\})\{
         // fmt.Fprintf(os.Stderr, f, args...)
6.
\langle \text{Imports 4} \rangle + \equiv
   "strconv"
7.
\langle Global variables 7\rangle
   id int
See also sections 9, 18, 31, and 41.
This code is used in section 2.
8.
\langle Obtaining of id of a window \rangle \equiv
      var err error
      id, err = strconv.Atoi(os.Getenv("winid"))
      if err \neq nil {
         return
   }
This code is used in section 3.
9.
\langle \text{Global variables } 7 \rangle + \equiv
```

tagname string

```
§10
        ahist (version 0.1)
10.
\langle Store a name of the program 10 \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.
      We change ahist into -ahist to add a possibility to switch ahist off.
\langle Change the name of the program in the tag 11\rangle \equiv
     del := \mathbf{append}([]\mathbf{string}\{\}, tagname)
     add := \mathbf{append}([]\mathbf{string}\{\}, "-" + tagname)
     changeTag(w, del, add)
This code is used in section 3.
12. On exit we should make an opposite change.
\langle \text{ Cleanup } 12 \rangle \equiv
     del := \mathbf{append}([]\mathbf{string}\{\}, "-" + tagname)
     add := \mathbf{append}([]\mathbf{string}\{\}, tagname)
     changeTag(w, del, add)
See also sections 38 and 42.
```

This code is used in sections 15 and 24.

STARTUP

5

6 EVENTS HANDLING ahist (version 0.1) $\S13$

13. Events handling.

```
14.
\langle \text{Imports 4} \rangle + \equiv
   "github.com/santucco/goacme"
15.
\langle\, {\rm Processing} \,\, {\rm window} \,\, {\rm events} \,\, {15} \, \rangle \equiv
       ev, err := w.ReadEvent()
       if err \neq nil {
          \langle Cleanup 12\rangle
          return
       \langle \text{Process main window 20} \rangle
This code is used in section 3.
16.
\langle Open window w by id 16 \rangle \equiv
   w, err := goacme.Open(id)
   if err \neq nil {
       debug(\texttt{"cannot}\_\texttt{open}\_\texttt{a}\_\texttt{window}\_\texttt{with}\_\texttt{id}\_\texttt{\%d}:\_\texttt{\%s}\texttt{`n"}, id, err)
       return
   defer w.Close()
This code is used in section 3.
17.
\langle \text{Imports 4} \rangle +\equiv
   "strings"
18.
\langle Global variables 7\rangle + \equiv
   name string
```

7

```
§19
             ahist (version 0.1)
19.
\langle \text{Read } name \text{ of the window } 19 \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"}, 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.
20.
\langle \text{Process main window } 20 \rangle \equiv
     \langle Process and continue if it is not Look in any form 21\rangle
     \langle \text{Process } Look \text{ 26} \rangle
     \langle \text{ Read addr into } b, e | 34 \rangle
     \langle \text{Show dot } 36 \rangle
    ⟨ Write history 48⟩
This code is used in section 15.
```

8 EVENTS HANDLING ahist (version 0.1) $\S 21$

```
21.
```

```
\langle \text{Process and continue if it is not } Look \text{ in any form } 21 \rangle \equiv
  debug("ev: \_\%#v\n", ev)
  s := ""
  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 22)
     case ev.Type \equiv goacme.Look:
       (Process in case of a request by B3 command in the body 23)
     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 24)
     case ev.Type \equiv goacme.Insert \lor ev.Type \equiv goacme.Delete:
       \langle Fix tag of the window 37\rangle
       continue
     default:
        (Unread event and continue 25)
  }
This code is used in section 20.
22. We take a search string from ev event and set dot
\langle \text{Process in case of a request by B3 mouse button in the tag } 22 \rangle \equiv
  s = ev. Text
  if len(ev.Arg)\rangle 0 {
     s += "\Box" + ev.Arg
  (Set addr to dot 28)
This code is used in section 21.
23. We take a search string and address from ev event.
\langle Process in case of a request by B3 command in the body 23\rangle \equiv
  s=\mathit{ev.Text}
  if len(ev.Arg)\rangle 0 {
     s += "\Box" + ev.Arg
  b := ev.Begin
  e := ev.End
  \langle Set addr to b, e 30 \rangle
This code is used in section 21.
```

 $\S 24$ ahist (version 0.1) EVENTS HANDLING

9

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. 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 24\rangle \equiv
  switch ev.Text {
     case "Look":
       s = ev.Arg
        \langle Set addr to dot 28\rangle
        break type_switch
     case tagname:
       continue
     case "-" + tagname:
        \langle \text{Cleanup } 12 \rangle
        return
  w.UnreadEvent(ev)
  fallthrough
This code is used in section 21.
\langle \text{Unread event and continue 25} \rangle \equiv
  w.UnreadEvent(ev)
  continue
This code is used in sections 21, 28, 29, 30, 33, 34, 35, and 36.
```

26. If the ev event contains a search string, use it. Otherwise we should read selected the string from the window's body.

```
\langle \text{Read addr into } b, e | 34 \rangle
       if len(s) > 0 {
          \langle \text{ Make a search of } s \text{ 33} \rangle
       } else {
          (Look for selected string 27)
   }
This code is used in section 20.
27.
\langle \text{Look for selected string } 27 \rangle \equiv
       \langle \text{Read selected string from "xdata" file to } s \text{ 29} \rangle
       \langle \text{ Make a search of } s \text{ 33} \rangle
This code is used in section 26.
```

 $\langle \text{Process } Look \ 26 \rangle \equiv$

10 EVENTS HANDLING ahist (version 0.1) §28

```
28.
\langle Set addr to dot 28\rangle \equiv
  if w.WriteCtl("addr=dot") \neq nil  {
      (Unread event and continue 25)
  debug("set_{\square}addr_{\square}to_{\square}dot\n")
This code is used in sections 22 and 24.
\langle\, {\rm Read} \,\, {\rm selected} \,\, {\rm string} \,\, {\rm from} \,\, "{\tt xdata"} \,\, {\rm file} \,\, {\rm to} \,\, s \,\, {\tt 29} \, \rangle \equiv
      d, err := w.File("xdata")
     if err \neq nil {
         debug("cannot\_read\_from\_'xdata'\_of\_the\_window\_with\_id\_'kd:\_'ks\n", id, err)
         (Unread event and continue 25)
      buf := \mathbf{make}([]\mathbf{byte}, e - b + 1)
      for n, := d.Read(buf); n > 0; n, = d.Read(buf) {
        s += \mathbf{string}(\mathit{buf}[:n])
      debug("read_{\square}address_{\square}from_{\square}xdata_{\square}b:_{\square}%v,_{\square}e:_{\square}%v\n",b,e)
This code is used in section 27.
30.
\langle Set addr to b, e | 30 \rangle \equiv
  if err := w.WriteAddr("#%d,#%d",b,e); err \neq nil  {
      debug("cannot_write_to_l'addr'_lof_the_window_with_lid_\%d:_\%s\n", id, err)
      (Unread event and continue 25)
  }
  debug("\mathtt{set}_{\sqcup}\mathtt{addr}_{\sqcup}\mathtt{to}_{\sqcup}\%\mathtt{d},_{\sqcup}\%\mathtt{d}\mathtt{'n}",b,e)
This code is used in sections 23 and 33.
31. 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 \text{Global variables } 7 \rangle + \equiv
  lentr entry
32. Let's add empty function for entry
  func (this entry) empty() bool{
      return this.b \equiv this.e
```

}

§33 ahist (version 0.1) EVENTS HANDLING 11

33. 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\, {\rm Make} \ {\rm a} \ {\rm search} \ {\rm of} \ s \ {\scriptstyle {\bf 33}} \,\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 \text{ Set addr to } b, e | \mathbf{30} \rangle
      } else if b \neq e {
         lentr = entry\{b, e, s\}
         ⟨ Write history 48⟩
      }
      es := ""
      for \_, v := \mathbf{range} \ s \ \{
         if strings.ContainsRune("|\\/[].+?()*^$",v) {
            es += " \ "
         es += \mathbf{string}(v)
      debug("es: \_ %q\n", es)
      \mathbf{if} \ \ err := w. \textit{WriteAddr}(" \textit{/\subsetss}, es); \ \ err \neq \mathbf{nil} \ \ \{
         debug("cannot_write_to_l'addr'_lof_the_window_with_lid_%d:_%s\n", id, err)
         (Unread event and continue 25)
      }
   }
This code is used in sections 26 and 27.
34.
\langle \text{Read addr into } b, e \mid 34 \rangle \equiv
   b, e, err := w.ReadAddr()
   if err \neq nil {
      (Unread event and continue 25)
   debug(\texttt{"read}\_\texttt{address}\_\texttt{b}:\_\texttt{%v,}\_\texttt{e}:\_\texttt{%v}\texttt{`n"},b,e)
This code is used in sections 20 and 26.
35.
\langle \text{ Set dot to addr } 35 \rangle \equiv
   if w.WriteCtl("dot=addr\nshow") \neq nil  {
      debug("cannot | write | to | `ctl' | of | the | window | with | id | %d: | %s | n", id, err)
      (Unread event and continue 25)
   debug("set_{\sqcup}dot_{\sqcup}to_{\sqcup}addr\n")
This code is used in section 36.
```

12 EVENTS HANDLING ahist (version 0.1) §36

```
36.
\langle \text{Show dot } 36 \rangle \equiv
   \langle Set dot to addr 35\rangle
   if w.WriteCtl("show") \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 25)
   debug("show_dot\n")
This code is used in section 20.
37. 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 } 37 \rangle \equiv
      \_, \_, \_, \_, d, \_, \_, \_, err := w.ReadCtl()
      if err \neq nil {
         debug(\texttt{"cannot}_{\square}\texttt{read}_{\square}\texttt{from}_{\square}\texttt{'ctl'}_{\square}\texttt{of}_{\square}\texttt{the}_{\square}\texttt{window}_{\square}\texttt{with}_{\square}\texttt{id}_{\square}\%\texttt{d}:_{\square}\%\texttt{s}\\ \land \texttt{n}", id, err)
         continue
      debug("dirty: \_%v\n", d)
      var add, del [string
      if d {
         add = \mathbf{append}(add, "Put")
      } else {
         del = \mathbf{append}(del, "Put")
      add = \mathbf{append}(add, "Undo", "Redo")
      changeTag(w, del, add)
This code is used in section 21.
38. Removing added commands on exit
\langle \text{Cleanup } 12 \rangle + \equiv
      del := \mathbf{append}([]\mathbf{string}\{\}, "Put", "Undo", "Redo")
      changeTag(w, del, \mathbf{nil})
   }
```

Tracking search requests.

 $\mathbf{var}\ h\ *goacme.Window$

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

```
40.
\langle \, {\rm Types} \, \, {\color{red} 40} \, \rangle \equiv
  entry struct{
     b, e int
     s string
  }
This code is used in section 2.
41. Special histch channel is received entry to print them in the window
\langle \text{Global variables } 7 \rangle + \equiv
  histch chan entry = make(chan entry)
42. On exit we should signal the goroutine to stop processing. It is made by closing histch channel
\langle \text{Cleanup } 12 \rangle + \equiv
  close(histch)
43.
\langle \text{ Variables outside the loop 43} \rangle \equiv
  \mathbf{var}\ hch\ \leftarrow \mathbf{chan}\ *goacme.Event
See also sections 45 and 49.
This code is used in section 44.
44. The goroutine handles two variants of events.
\langle Start history processing 44\rangle \equiv
  go func(){
      ⟨ Variables outside the loop 43⟩
      for{
         select
           case entr, ok := \leftarrow histch:
               ⟨ Process entr entry from histch 46⟩
           case ev, ok := \leftarrow hch:
               \langle \text{Process } ev \text{ event from } hch \text{ event channel of the window } 47 \rangle
  }()
This code is used in section 3.
45.
\langle \text{ Variables outside the loop } 43 \rangle + \equiv
```

ahist (version 0.1)

14

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

```
\langle \text{Process } entr \text{ entry from } histch | 46 \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 50)
   if ee, ok := history[entr.b]; ok \land ee \equiv entr.e {
      continue
   history[entr.b] = entr.e
   debug(\texttt{"writing} \texttt{\_to} \texttt{\_the} \texttt{\_history} \texttt{\_\%d,\%d} \texttt{\land 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")
This code is used in section 44.
```

47. 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 } 47 \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)
This code is used in section 44.
48.
\langle \text{Write history 48} \rangle \equiv
  debug("request_to_store_a_history:_\%v, %v_\%q\n", b, e, s)
  histch \leftarrow entry\{b: b, e: e, s: s\}
This code is used in sections 20 and 33.
49.
\langle \text{ Variables outside the loop } 43 \rangle + \equiv
  var history map[int]int
```

```
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 50\rangle \equiv
   if h \equiv \text{nil} {
      var err error
      if h, err = goacme.New(); err \neq nil  {
         return
      h.WriteCtl("name_{\sqcup}\%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 46.
51. change Tag function.
   We read the tag of w window, remove all commands from del list and add all commands from add list.
   func changeTag(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{ 52} \rangle
      \langle \text{Split tag into } tag \text{ fields after the pipe symbol } 53 \rangle
      \langle \text{Compose } newtag 54 \rangle
      \langle \text{ Clear the tag and write } newtag \text{ to the tag 57} \rangle
   }
52.
\langle \text{ Read a tag of } w \text{ into } s \text{ 52} \rangle \equiv
   f, err := w.File("tag")
   if err \neq nil {
      debug("cannot_{\square}read_{\square}from_{\square}'tag'_{\square}of_{\square}the_{\square}window_{\square}with_{\square}id_{\square}%d:_{\square}%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)
      return
   s := \mathbf{string}(b[:n])
This code is used in section 51.
```

```
53.
```

```
\langle \text{Split tag into } tag \text{ fields after the pipe symbol } 53 \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, """)
This code is used in section 51.
54.
\langle \text{ Compose } newtag | 54 \rangle \equiv
  newtag := \mathbf{append}([[\mathbf{string}\{\},""])
  \langle Every part is contained in del we remove from tag 55\rangle
  \langle Every part is contained in add we remove from tag 56\rangle
  newtag = \mathbf{append}(newtag, add \dots)
  newtag = \mathbf{append}(newtag, tag...)
This code is used in section 51.
\langle Every part is contained in del we remove from tag 55\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 54.
\langle Every part is contained in add we remove from tag 56\rangle \equiv
  for \_, v := \mathbf{range} \ add \ \{
     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 54.
```

```
57.
\langle Clear the tag and write newtag to the tag 57\rangle \equiv
  s = strings.Join(newtag, " \sqcup ")
  if err := w.WriteCtl("cleartag"); err \neq nil {
    debug("cannot_clear_tag_of_the_window_with_id_%d:_%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 51.
add: 11, 12, 37, 51, 54, 56.
                                                            Insert: 21.
                                                            Join: 57.
addr: 33.
                                                            LastIndex: 10, 53.
ahist: 24.
AllTypes: 50.
                                                            lentr: 31, 33.
Arg: 22, 23, 24.
                                                            Look: 21, 24, 31.
                                                            main: 2, \underline{3}.
Args: 10.
                                                            name: 18, 19, 46, 50.
args: 5.
                                                            New: 50.
Atoi: 8.
                                                            newtag: 54, 57.
Begin: 23.
                                                            ok: 44, 46, 47.
buf: 29.
                                                            Open: 16.
change Tag: 11, 12, 37, 38, \underline{51}.
                                                            os: \underline{4}, 8, 10.
Close: 16, 46, 47.
                                                            Put: 37.
Contains Rune \colon \ \ \textbf{33}.
                                                            Read: 19, 29, 52.
debug: 5, 10, 16, 19, 21, 28, 29, 30, 33, 34, 35,
                                                            ReadAddr: 34.
    36, 37, 46, 47, 48, 52, 57.
                                                            ReadCtl: 37.
del: 11, 12, 37, 38, 51, 55.
                                                            ReadEvent: 15.
Del: 46, 47.
                                                            Redo: 37.
Delete: 21.
                                                            Seek: 19, 52.
ee: 46.
                                                            Split: 19, 53.
empty: \underline{32}, \underline{33}.
                                                            Sprintf: 46.
End: 23.
                                                            ss: 19.
entr: 44, 46.
                                                            strconv: \underline{6}, 8.
entry: 31, 32, 33, 40, 41, 48.
                                                            strings: <u>17</u>, 10, 19, 33, 53, 57.
err: 8, 15, 16, 19, 29, 30, 33, 34, 35, 36, 37,
                                                            tag: 53, 54, 55, 56.
    50, 52, 57.
                                                            Tag: 21.
es: 33.
                                                            tagname: 9, 10, 11, 12, 24.
ev: 15, 21, 22, 23, 24, 25, 26, 44, 47.
                                                            Text: 22, 23, 24.
event: \underline{24}, \underline{37}.
                                                            this: 32.
Event: 43.
                                                            TrimLeft: 53.
EventChannel: 50.
                                                            Type: 21.
Execute: 21.
                                                            type\_switch: 21, 24.
File: 19, 29, 52.
                                                            Undo: 37.
fmt: \underline{4}, \underline{46}.
                                                            UnreadEvent: 24, 25, 47.
Getenv: 8.
                                                            Window: 45, 51.
goacme: 14, 16, 21, 43, 45, 50, 51.
                                                            Write: 46, 57.
hch: 43, 44, 47, 50.
                                                            WriteAddr: 30, 33.
                                                            WriteCtl: 28, 35, 36, 46, 50, 57.
histch: 41, 42, 44, 46, 48.
history: 46, 47, 49, 50.
                                                            xdata: 27, 29.
```

 $id\colon \ \ 7,\, 8,\, 16,\, 19,\, 29,\, 30,\, 33,\, 35,\, 36,\, 37,\, 52,\, 57.$

18 NAMES OF THE SECTIONS ahist (version 0.1)

```
\langle Change the name of the program in the tag 11 \rangle Used in section 3.
Cleanup 12, 38, 42 \rangle Used in sections 15 and 24.
Clear the tag and write newtag to the tag 57 Used in section 51.
Compose newtag 54 Used in section 51.
Every part is contained in add we remove from tag\ 56 \ Used in section 54.
Every part is contained in del we remove from taq 55 Used in section 54.
Fix tag of the window 37 Used in section 21.
 Global variables 7, 9, 18, 31, 41 \ Used in section 2.
 Imports 4, 6, 14, 17 Used in section 2.
 Look for selected string 27 Vsed in section 26.
 Make a search of s 33 \ Used in sections 26 and 27.
 Obtaining of id of a window 8 Used in section 3.
 Open history window, if it does not exist 50 \ Used in section 46.
 Open window w by id 16 \rightarrow Used in section 3.
 Process and continue if it is not Look in any form 21 \) Used in section 20.
 Process in case of a request by B3 command in the body 23 \ Used in section 21.
 Process in case of a request by B3 mouse button in the tag 22 \ Used in section 21.
 Process in case of executing a command in the body or tag 24 \( \) Used in section 21.
 Process main window 20 \ Used in section 15.
Process Look\ 26 \rightarrow Used in section 20.
 Process entr entry from histch 46 \ Used in section 44.
Process ev event from hch event channel of the window 47 Used in section 44.
 Processing window events 15 \ Used in section 3.
 Read a tag of w into s 52 \ Used in section 51.
 Read addr into b, e \mid 34 \rangle Used in sections 20 and 26.
 Read selected string from "xdata" file to s 29 \ Used in section 27.
 Read name of the window 19 \ Used in section 3.
Set addr to dot 28 \ Used in sections 22 and 24.
Set addr to b, e \mid 30 \rangle Used in sections 23 and 33.
Set dot to addr 35 \ Used in section 36.
Show dot 36 V used in section 20.
 Split tag into tag fields after the pipe symbol 53 \ Used in section 51.
Start history processing 44 \rightarrow Used in section 3.
Store a name of the program 10 \ Used in section 3.
\langle \text{Types 40} \rangle Used in section 2.
Unread event and continue 25 \ Used in sections 21, 28, 29, 30, 33, 34, 35, and 36.
Variables outside the loop 43, 45, 49 \ Used in section 44.
Write history 48 \ Used in sections 20 and 33.
```

ahist - a simple search history for Acme

(version 0.1)

	Sec	tion	Page
Introduction		. 1	2
Implementation		. 2	3
Startup		. 3	4
Events handling		13	6
Tracking search requests		39	13

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.