atag - tag extender for Acme

(Version 0.1)

Alexander Sychev (santucco@gmail.com)

1. Introduction. This is an implementation of `atag` command for `Acme`. It adds specified commands to a tag of every `Acme`'s window

2. Implementation.

```
    // This file is part of atag 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 3⟩
  )
```

3.

```
⟨Imports 3⟩ ≡
  "fmt"
  "os"
```

See also sections 5 and 9.

This code is used in section 2.

4. At first, if no commands are specified, let's print the usage info and exit. Then an enumeration of opened windows is processed in a separated goroutine. Then pooling of `Acme`'s log is started. Start of the enumeration is syncronized with the start of pulling `Acme`'s log.

```
func main(){
  if len(os.Args) ≡ 1 {
    fmt.Fprintf(os.Stderr, "Tag␣extender\nExtends␣tags␣of␣Acme␣with␣specified␣commands\\
        nUsage:␣%s␣<commands>\n", os.Args[0])
    return
  }
  sync := make(chan bool)
  ⟨Enumerate the opened windows 7⟩
  ⟨Start polling of Acme's log 6⟩
}
```

5.

```
⟨Imports 3⟩ +≡
  "github.com/santucco/goacme"
```

6.

```
⟨Start polling of Acme's log 6⟩ ≡
  log, err := goacme.OpenLog()
  if err ≠ nil {
    fmt.Fprint(os.Stderr, err)
    return
  }
  defer log.Close()
  close(sync)
  for ev, err := log.Read(); err ≡ nil; ev, err = log.Read() {
    if ev.Type ≡ goacme.NewWin {
      id := ev.Id
      ⟨Write specified commands to a tag of the new window with id after pipe simbol 8⟩
    }
  }
```

This code is used in section 4.

7.

```
⟨Enumerate the opened windows 7⟩ ≡
  go func(){
    ← sync
    ids, err := goacme.WindowsInfo()
    if err ≠ nil {
      fmt.Fprintf(os.Stderr, "cannot␣get␣a␣list␣of␣the␣opened␣windows␣of␣Acme:␣%v\n", err)
      return
    }
    for _, v := range ids {
      id := v.Id
      ⟨Write specified commands to a tag of the new window with id after pipe simbol 8⟩
    }
  }()
```

This code is used in section 4.

8.

⟨ Write specified commands to a tag of the new window with *id* after pipe simbol 8 ⟩ ≡

```
if err := writeTag(id, os.Args[1:]); err ≠ nil {
  fmt.Fprint(os.Stderr, err)
}
```

This code is used in sections 6 and 7.

9.

⟨ Imports 3 ⟩ +≡

```
"strings"
```

10. Let's describe a writing of tag like a function

```
func writeTag(id int, list []string) error{
  ⟨ Check if list is empty 11 ⟩
  ⟨ Open a window w by id 12 ⟩
  ⟨ Read the tag into s 13 ⟩
  ⟨ Remove the tag content before the pipe symbol 14 ⟩
  ⟨ Compose a new tag 15 ⟩
  ⟨ Clear the tag and write the new tag 16 ⟩
  return nil
}
```

11.

⟨ Check if *list* is empty 11 ⟩ ≡

```
if len(list) ≡ 0 {
  return nil
}
```

This code is used in section 10.

12.

⟨ Open a window *w* by *id* 12 ⟩ ≡

```
w, err := goacme.Open(id)
if err ≠ nil {
  return fmt.Errorf("cannot␣open␣a␣window␣with␣id␣%d:␣%s\n", id, err)
}
defer w.Close()
```

This code is used in section 10.

13.

⟨ Read the tag into *s* 13 ⟩ ≡
```
f, err := w.File("tag")
if err ≠ nil {
  return fmt.Errorf("cannot␣get␣tag␣file␣of␣the␣window␣with␣id␣%d:␣%s\n", id, err)
}
var b [200]byte
n, err := f.Read(b[:])
if err ≠ nil {
  return fmt.Errorf("cannot␣read␣the␣tag␣of␣the␣window␣with␣id␣%d:␣%s\n", id, err)
}
s := string(b[:n])
```
This code is used in section 10.

14.

⟨ Remove the tag content before the pipe symbol 14 ⟩ ≡
```
if n = strings.Index(s, "|"); n ≡ −1 {
  n = 0
} else {
  n++
}
s = s[n:]
```
This code is used in section 10.

15.

⟨ Compose a new tag 15 ⟩ ≡
```
s = "␣" + strings.Join(list, "␣") + s
```
This code is used in section 10.

16.

⟨ Clear the tag and write the new tag 16 ⟩ ≡
```
if err := w.WriteCtl("cleartag"); err ≠ nil {
  return fmt.Errorf("cannot␣clear␣the␣tag␣of␣the␣window␣with␣id␣%d:␣%s\n", id, err)
}
if _, err := f.Write([]byte(s)); err ≠ nil {
  return fmt.Errorf("cannot␣write␣the␣tag␣of␣the␣window␣with␣id␣%d:␣%s\n", id, err)
}
```
This code is used in section 10.

atag - tag extender for Acme

(version 0.1)