

```

*****
51356 Mon May 5 22:34:22 2014
new/usr/src/cmd/hal/hald/solaris/devinfo_storage.c
4846 HAL partition names don't match real partition names
*****
1 /*****
2 *
3 * devinfo_storage.c : storage devices
4 *
5 * Copyright (c) 2006, 2010, Oracle and/or its affiliates. All rights reserved.
6 * Copyright 2013 Garrett D'Amore <garrett@damore.org>
7 * Copyright 2014 Andrew Stormont.
8 #endif /* ! codereview */
9 *
10 * Licensed under the Academic Free License version 2.1
11 *
12 *****/

14 #ifdef HAVE_CONFIG_H
15 # include <config.h>
16 #endif

18 #include <stdio.h>
19 #include <string.h>
20 #include <strings.h>
21 #include <ctype.h>
22 #include <libdevinfo.h>
23 #include <sys/types.h>
24 #include <sys/mkdev.h>
25 #include <sys/stat.h>
26 #include <sys/mntent.h>
27 #include <sys/mnttab.h>

29 #include "../osspec.h"
30 #include "../logger.h"
31 #include "../hald.h"
32 #include "../hald_dbus.h"
33 #include "../device_info.h"
34 #include "../util.h"
35 #include "../hald_runner.h"
36 #include "hotplug.h"
37 #include "devinfo.h"
38 #include "devinfo_misc.h"
39 #include "devinfo_storage.h"
40 #include "osspec_solaris.h"

42 #ifdef sparc
43 #define WHOLE_DISK "s2"
44 #else
45 #define WHOLE_DISK "p0"
46 #endif

48 /* some devices, especially CDROMs, may take a while to be probed (values in ms)
49 #define DEVINFO_PROBE_STORAGE_TIMEOUT 60000
50 #define DEVINFO_PROBE_VOLUME_TIMEOUT 60000

52 typedef struct devinfo_storage_minor {
53     char *devpath;
54     char *devlink;
55     char *slice;
56     dev_t dev;
57     int dosnum; /* dos disk number or -1 */
58 } devinfo_storage_minor_t;

60 HalDevice *devinfo_ide_add(HalDevice *parent, di_node_t node, char *devfs_path,
61 static HalDevice *devinfo_ide_host_add(HalDevice *parent, di_node_t node, char *

```

```

62 static HalDevice *devinfo_ide_device_add(HalDevice *parent, di_node_t node, char
63 static HalDevice *devinfo_ide_storage_add(HalDevice *parent, di_node_t node, cha
64 HalDevice *devinfo_scsi_add(HalDevice *parent, di_node_t node, char *devfs_path,
65 static HalDevice *devinfo_scsi_storage_add(HalDevice *parent, di_node_t node, ch
66 HalDevice *devinfo_blkdev_add(HalDevice *parent, di_node_t node, char *devfs_pat
67 static HalDevice *devinfo_blkdev_storage_add(HalDevice *parent, di_node_t node,
68 HalDevice *devinfo_floppy_add(HalDevice *parent, di_node_t node, char *devfs_pat
69 static void devinfo_floppy_add_volume(HalDevice *parent, di_node_t node);
70 static HalDevice *devinfo_lofi_add(HalDevice *parent, di_node_t node, char *devf
71 static void devinfo_lofi_add_minor(HalDevice *parent, di_node_t node, char *mino
72 static void devinfo_storage_minors(HalDevice *parent, di_node_t node, gchar *dev
73 static struct devinfo_storage_minor *devinfo_storage_new_minor(char *maindev_pat
74     char *devlink, dev_t dev, int dosnum);
75 static void devinfo_storage_free_minor(struct devinfo_storage_minor *m);
76 HalDevice *devinfo_volume_add(HalDevice *parent, di_node_t node, devinfo_storage
77 static void devinfo_volume_preprobing_done(HalDevice *d, gpointer userdata1, gpo
78 static void devinfo_volume_hotplug_begin_add (HalDevice *d, HalDevice *parent, D
79 static void devinfo_storage_hotplug_begin_add (HalDevice *d, HalDevice *parent,
80 static void devinfo_storage_probing_done (HalDevice *d, guint32 exit_type, gint
81 const gchar *devinfo_volume_get_prober (HalDevice *d, int *timeout);
82 const gchar *devinfo_storage_get_prober (HalDevice *d, int *timeout);

84 static char *devinfo_scsi_dtype2str(int dtype);
85 static char *devinfo_volume_get_slice_name (char *devlink);
86 static boolean_t is_dos_slice(const char *slice, int *partnum);
87 static gboolean dos_to_dev(char *path, char **devpath, int *partnum);
88 static gboolean is_dos_path(char *path, int *partnum);

88 static void devinfo_storage_set_nicknames (HalDevice *d);

90 DevinfoDevHandler devinfo_ide_handler = {
91     devinfo_ide_add,
92     NULL,
93     NULL,
94     NULL,
95     NULL,
96     NULL,
97 };
_____unchanged_portion_omitted_____

796 /*
797 * Storage minor nodes are potential "volume" objects.
798 * This function also completes building the parent object (main storage device)
799 */
800 static void
801 devinfo_storage_minors(HalDevice *parent, di_node_t node, gchar *devfs_path, gbo
802 {
803     di_devlink_handle_t devlink_hdl;
804     gboolean is_cdrom;
805     const char *whole_disk;
806     int major;
807     di_minor_t minor;
808     dev_t dev;
809     char *minor_path = NULL;
810     char *maindev_path = NULL;
811     char *devpath, *devlink;
812     int doslink_len;
813     char *doslink;
814     char dospath[64];
815     char *slice;
816     int pathlen;
817     int i;
818     char *raw;
819     boolean_t maindev_is_d0;
820     GQueue *mq;
821     HalDevice *volume;

```

```

822     struct devinfo_storage_minor *m;
823     struct devinfo_storage_minor *maindev = NULL;

825     /* for cdroms whole disk is always s2 */
826     is_cdrom = hal_device_has_capability (parent, "storage.cdrom");
827     whole_disk = is_cdrom ? "s2" : WHOLE_DISK;

829     major = di_driver_major(node);

831     /* the "whole disk" p0/s2/d0 node must come first in the hotplug queue
832     * so we put other minor nodes on the local queue and move to the
833     * hotplug queue up in the end
834     */
835     if ((mq = g_queue_new()) == NULL) {
836         goto err;
837     }
838     if ((devlink_hdl = di_devlink_init(NULL, 0)) == NULL) {
839         g_queue_free (mq);
840         goto err;
841     }
842     minor = DI_MINOR_NIL;
843     while ((minor = di_minor_next(node, minor)) != DI_MINOR_NIL) {
844         dev = di_minor_devt(minor);
845         if ((major != major(dev)) ||
846             (di_minor_type(minor) != DDM_MINOR) ||
847             (di_minor_spectype(minor) != S_IFBLK) ||
848             ((minor_path = di_devfs_minor_path(minor)) == NULL)) {
849             continue;
850         }
851         if ((devlink = get_devlink(devlink_hdl, NULL, minor_path)) == NU
852             di_devfs_path_free (minor_path);
853             continue;
854     }

856     slice = devinfo_volume_get_slice_name (devlink);
857     if (strlen (slice) < 2) {
858         free (devlink);
859         di_devfs_path_free (minor_path);
860         continue;
861     }

863     /* ignore p1..N - we'll use p0:N instead */
864     if ((strlen (slice) > 1) && (slice[0] == 'p') && isdigit(slice[1
865         ((atol(&slice[1])) > 0)) {
866         free (devlink);
867         di_devfs_path_free (minor_path);
868         continue;
869     }

870     m = devinfo_storage_new_minor(minor_path, slice, devlink, dev, -
871     if (m == NULL) {
872         free (devlink);
873         di_devfs_path_free (minor_path);
874         continue;
875     }
876     maindev_path = minor_path;
877     maindev = m;
878     g_queue_push_head (mq, maindev);
879     } else {

```

```

880         di_devfs_path_free (minor_path);
881         g_queue_push_tail (mq, m);
882     }

884     free (devlink);
885     }
886     di_devlink_fini (&devlink_hdl);

888     if (maindev == NULL) {
889         /* shouldn't typically happen */
890         while (!g_queue_is_empty (mq)) {
891             devinfo_storage_free_minor (g_queue_pop_head (mq));
892         }
893         goto err;
894     }

896     /* first enqueue main storage device */
897     if (!rescan) {
898         hal_device_property_set_int (parent, "block.major", major);
899         hal_device_property_set_int (parent, "block.minor", minor(mainde
900         hal_device_property_set_string (parent, "block.device", maindev-
901         raw = dsk_to_rdisk (maindev->devlink);
902         hal_device_property_set_string (parent, "block.solaris.raw_devic
903         free (raw);
904         hal_device_property_set_bool (parent, "block.is_volume", FALSE);
905         hal_device_property_set_string (parent, "solaris.devfs_path", ma
906         devinfo_add_enqueue (parent, maindev_path, &devinfo_storage_hand
907     }

909     /* add virtual dos volumes to enable pcfs probing */
910     if (!is_cdrom) {
911         doslink_len = strlen (maindev->devlink) + sizeof (":NNN") + 1;
912         if ((doslink = (char *)calloc (1, doslink_len)) != NULL) {
913             for (i = 1; i < 16; i++) {
914                 snprintf(dospath, sizeof (dospath), "%s:%d", mai
915                 snprintf(doslink, doslink_len, "%s:%d", maindev-
916                 m = devinfo_storage_new_minor(maindev_path, dosp
917                 g_queue_push_tail (mq, m);
918             }
919             free (doslink);
920         }
921     }

923     maindev_is_d0 = (strcmp (maindev->slice, "d0") == 0);

925     /* enqueue all volumes */
926     while (!g_queue_is_empty (mq)) {
927         m = g_queue_pop_head (mq);

929         /* if main device is d0, we'll throw away s2/p0 */
930         if (maindev_is_d0 && (strcmp (m->slice, whole_disk) == 0)) {
931             devinfo_storage_free_minor (m);
932             continue;
933         }
934         /* don't do p0 on cdrom */
935         if (is_cdrom && (strcmp (m->slice, "p0") == 0)) {
936             devinfo_storage_free_minor (m);
937             continue;
938         }
939         if (rescan) {
940             /* in rescan mode, don't reprobe existing volumes */
941             /* XXX detect volume removal? */
942             volume = hal_device_store_match_key_value_string (hald_g
943                 "solaris.devfs_path", m->devpath);
944             if ((volume == NULL) || !hal_device_has_capability(volum
945                 devinfo_volume_add (parent, node, m);

```

```

932         } else {
933             HAL_INFO(("rescan volume exists %s", m->devpath)
934         }
935     } else {
936         devinfo_volume_add (parent, node, m);
937     }
938     devinfo_storage_free_minor (m);
939 }

941 if (maindev_path != NULL) {
942     di_devfs_path_free (maindev_path);
943 }

945 return;

947 err:
948 if (maindev_path != NULL) {
949     di_devfs_path_free (maindev_path);
950 }
951 if (!rescan) {
952     devinfo_add_enqueue (parent, devfs_path, &devinfo_storage_handle
953 }
954 }

```

unchanged portion omitted

```

1023 static void
1024 devinfo_volume_preprobing_done (HalDevice *d, gpointer userdatal, gpointer userd
1025 {
1026     void *end_token = (void *) userdatal;
1027     char *whole_disk;
1028     char *block_device;
1029     const char *storage_udi;
1030     HalDevice *storage_d;
1031     const char *slice;
1032     int dos_num;

1034     if (hal_device_property_get_bool (d, "info.ignore")) {
1035         HAL_INFO (("Preprobing merged info.ignore==TRUE %s", hal_device_
1036             goto skip;
1037     }

1039     /*
1040     * Optimizations: only probe if there's a chance to find something
1041     */
1042     block_device = (char *)hal_device_property_get_string (d, "block.device"
1043     storage_udi = hal_device_property_get_string (d, "block.storage_device"
1044     slice = hal_device_property_get_string(d, "block.solaris.slice");
1045     if ((block_device == NULL) || (storage_udi == NULL) ||
1046         (slice == NULL) || (strlen (slice) < 2)) {
1047         HAL_INFO (("Malformed volume properties %s", hal_device_get_udi
1048             goto skip;
1049     }
1050     storage_d = hal_device_store_match_key_value_string (hald_get_gdl (), "i
1051     if (storage_d == NULL) {
1052         HAL_INFO (("Storage device not found %s", hal_device_get_udi (d)
1053             goto skip;
1054     }

1056     whole_disk = hal_device_has_capability (storage_d,
1057         "storage.cdrom") ? "s2" : WHOLE_DISK;

1059     if (is_dos_slice(slice, &dos_num)) {
1060     if (is_dos_path(block_device, &dos_num)) {
1061         /* don't probe more dos volumes than probe-storage found */
1062         if ((hal_device_property_get_bool (storage_d, "storage.no_partit
            (dos_num > hal_device_property_get_int (storage_d, "storage.

```

```

1063         HAL_INFO (("d > %d %s", dos_num, hal_device_propert
1064             "storage.solaris.num_dos_partitions"), hal_devic
1065             goto skip;
1066     }
1067 } else {
1068     /* if no VTOC slices found, don't probe slices except s2 */
1069     if ((slice[0] == 's') && (isdigit(slice[1])) && ((strcmp (slice,
1070         !hal_device_property_get_bool (storage_d, "storage.solaris.v
1071         HAL_INFO (("Not probing slice %s", hal_device_get_udi (d
1072             goto skip;
1073     }
1074 }

1076     HAL_INFO(("Probing udi=%s", hal_device_get_udi (d));
1077     hald_runner_run (d,
1078         "hald-probe-volume", NULL,
1079         DEVINFO_PROBE_VOLUME_TIMEOUT,
1080         devinfo_callouts_probing_done,
1081         (gpointer) end_token, userdata2);

1083     return;

1085 skip:
1086     hal_device_store_remove (hald_get_tdl (), d);
1087     g_object_unref (d);
1088     hotplug_event_end (end_token);
1089 }

```

unchanged portion omitted

```

1380 static boolean_t
1381 is_dos_slice(const char *slice, int *partnum)
1382 {
1383     char *p;
1384     if ((p = strrchr(slice, 'p')) == NULL &&
1385         (p = strrchr(slice, ':')) == NULL) {
1386         return (B_FALSE);
1387     }
1388     if ((p = strrchr (path, ':')) == NULL) {
1389         return (FALSE);
1390     }
1391 }

1390 #endif /* ! codereview */
1391 return ((*partnum = atoi(p + 1)) != 0);
1392 }

1393 static gboolean
1394 dos_to_dev(char *path, char **devpath, int *partnum)
1395 {
1396     char *p;
1397     if ((p = strrchr (path, ':')) == NULL) {
1398         return (FALSE);
1399     }
1400     if ((*partnum = atoi(p + 1)) == 0) {
1401         return (FALSE);
1402     }
1403     p[0] = '\0';
1404     *devpath = strdup(path);
1405     p[0] = ':';
1406     return (*devpath != NULL);
1407 }

1394 static void
1395 devinfo_storage_cleanup_mountpoint_cb (HalDevice *d, guint32 exit_type,

```

new/usr/src/cmd/hal/hald/solaris/devinfo_storage.c

7

```
1396         gint return_code, gchar **error,  
1397         gpointer data1, gpointer data2)  
1398 {  
1399     char *mount_point = (char *) data1;  
  
1401     HAL_INFO (("Cleaned up mount point '%s'", mount_point));  
1402     g_free (mount_point);  
1403 }  
_____unchanged_portion_omitted_____
```

new/usr/src/cmd/hal/utlils/fsutlils.c

1

```
*****
5151 Mon May 5 22:34:22 2014
new/usr/src/cmd/hal/utlils/fsutlils.c
4846 HAL partition names don't match real partition names
*****
1 /*
2 *
3 * fsutlils.c : filesystem utilities
4 *
5 * Copyright 2008 Sun Microsystems, Inc. All rights reserved.
6 * Use is subject to license terms.
7 *
8 * Copyright 2014 Andrew Stormont.
9 *
10 #endif /* ! codereview */
11 * Licensed under the Academic Free License version 2.1
12 *
13 */

15 #ifdef HAVE_CONFIG_H
16 #include <config.h>
17 #endif

19 #include <stdio.h>
20 #include <sys/types.h>
21 #include <sys/scsi/impl/uscsi.h>
22 #include <string.h>
23 #include <strings.h>
24 #include <ctype.h>
25 #include <unistd.h>
26 #include <stdlib.h>
27 #include <errno.h>
28 #include <fcntl.h>
29 #include <sys/dkio.h>
30 #include <libintl.h>
31 #include <sys/dktp/fdisk.h>
32 #include <sys/fs/pc_label.h>

34 #include <libhal.h>
35 #include "fsutlils.h"

37 /*
38 * Separates dos notation device spec into device and drive number
39 * pN partition names are rewritten to point to p0
40 * :N partition names are dropped
41 #endif /* ! codereview */
42 */
43 boolean_t
44 dos_to_dev(char *path, char **devpath, int *num)
45 {
46     int i;
47     char *buf;
48     boolean_t found = B_FALSE;
49     char *p;

50     for (i = strlen(path); i > 0; i--) {
51         if (path[i] == 'p' || path[i] == ':') {
52             found = B_TRUE;
53             break;
54         }
55     }
56     if ((p = strrchr(path, ':')) == NULL) {
57         return (B_FALSE);
58     }
59     if (found == B_FALSE || (*num = atoi(path + i + 1)) == 0 ||
60         (buf = strdup(path)) == NULL) {
```

new/usr/src/cmd/hal/utlils/fsutlils.c

2

```
13     if ((*num = atoi(p + 1)) == 0) {
14         return (B_FALSE);
15     }
16     (void) strcpy(buf + i, path[i] == 'p' ? "p0" : "");
17     *devpath = buf;
18     return (B_TRUE);
19     p[0] = '\0';
20     *devpath = strdup(path);
21     p[0] = ':';
22     return (*devpath != NULL);
23 }
24 }
25 }
26 }
27 }
28 }
29 }
30 }
31 }
32 }
33 }
34 }
35 }
36 }
37 }
38 }
39 }
40 }
41 }
42 }
43 }
44 }
45 }
46 }
47 }
48 }
49 }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
101 }
102 }
103 }
104 }
105 }
106 }
107 }
108 }
109 }
110 }
111 }
112 }
113 }
114 }
115 }
116 }
117 }
118 }
119 }
120 }
121 }
122 }
123 }
124 }
125 }
126 }
127 }
128 }
129 }
130 }
131 }
132 }
133 }
134 }
135 }
136 }
137 }
138 }
139 }
140 }
141 }
142 }
143 }
144 }
145 }
146 }
147 }
148 }
149 }
150 }
151 }
152 }
153 }
154 }
155 }
156 }
157 }
158 }
159 }
160 }
161 }
162 }
163 }
164 }
165 }
166 }
167 }
168 }
169 }
170 }
171 }
172 }
173 }
174 }
175 }
176 }
177 }
178 }
179 }
180 }
181 }
182 }
183 }
184 }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }
195 }
196 }
197 }
198 }
199 }
200 }
201 }
202 }
203 }
204 }
205 }
206 }
207 }
208 }
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
501 }
502 }
503 }
504 }
505 }
506 }
507 }
508 }
509 }
510 }
511 }
512 }
513 }
514 }
515 }
516 }
517 }
518 }
519 }
520 }
521 }
522 }
523 }
524 }
525 }
526 }
527 }
528 }
529 }
530 }
531 }
532 }
533 }
534 }
535 }
536 }
537 }
538 }
539 }
540 }
541 }
542 }
543 }
544 }
545 }
546 }
547 }
548 }
549 }
550 }
551 }
552 }
553 }
554 }
555 }
556 }
557 }
558 }
559 }
560 }
561 }
562 }
563 }
564 }
565 }
566 }
567 }
568 }
569 }
570 }
571 }
572 }
573 }
574 }
575 }
576 }
577 }
578 }
579 }
580 }
581 }
582 }
583 }
584 }
585 }
586 }
587 }
588 }
589 }
590 }
591 }
592 }
593 }
594 }
595 }
596 }
597 }
598 }
599 }
600 }
601 }
602 }
603 }
604 }
605 }
606 }
607 }
608 }
609 }
610 }
611 }
612 }
613 }
614 }
615 }
616 }
617 }
618 }
619 }
620 }
621 }
622 }
623 }
624 }
625 }
626 }
627 }
628 }
629 }
630 }
631 }
632 }
633 }
634 }
635 }
636 }
637 }
638 }
639 }
640 }
641 }
642 }
643 }
644 }
645 }
646 }
647 }
648 }
649 }
650 }
651 }
652 }
653 }
654 }
655 }
656 }
657 }
658 }
659 }
660 }
661 }
662 }
663 }
664 }
665 }
666 }
667 }
668 }
669 }
670 }
671 }
672 }
673 }
674 }
675 }
676 }
677 }
678 }
679 }
680 }
681 }
682 }
683 }
684 }
685 }
686 }
687 }
688 }
689 }
690 }
691 }
692 }
693 }
694 }
695 }
696 }
697 }
698 }
699 }
700 }
701 }
702 }
703 }
704 }
705 }
706 }
707 }
708 }
709 }
710 }
711 }
712 }
713 }
714 }
715 }
716 }
717 }
718 }
719 }
720 }
721 }
722 }
723 }
724 }
725 }
726 }
727 }
728 }
729 }
730 }
731 }
732 }
733 }
734 }
735 }
736 }
737 }
738 }
739 }
740 }
741 }
742 }
743 }
744 }
745 }
746 }
747 }
748 }
749 }
750 }
751 }
752 }
753 }
754 }
755 }
756 }
757 }
758 }
759 }
760 }
761 }
762 }
763 }
764 }
765 }
766 }
767 }
768 }
769 }
770 }
771 }
772 }
773 }
774 }
775 }
776 }
777 }
778 }
779 }
780 }
781 }
782 }
783 }
784 }
785 }
786 }
787 }
788 }
789 }
790 }
791 }
792 }
793 }
794 }
795 }
796 }
797 }
798 }
799 }
800 }
801 }
802 }
803 }
804 }
805 }
806 }
807 }
808 }
809 }
810 }
811 }
812 }
813 }
814 }
815 }
816 }
817 }
818 }
819 }
820 }
821 }
822 }
823 }
824 }
825 }
826 }
827 }
828 }
829 }
830 }
831 }
832 }
833 }
834 }
835 }
836 }
837 }
838 }
839 }
840 }
841 }
842 }
843 }
844 }
845 }
846 }
847 }
848 }
849 }
850 }
851 }
852 }
853 }
854 }
855 }
856 }
857 }
858 }
859 }
860 }
861 }
862 }
863 }
864 }
865 }
866 }
867 }
868 }
869 }
870 }
871 }
872 }
873 }
874 }
875 }
876 }
877 }
878 }
879 }
880 }
881 }
882 }
883 }
884 }
885 }
886 }
887 }
888 }
889 }
890 }
891 }
892 }
893 }
894 }
895 }
896 }
897 }
898 }
899 }
900 }
901 }
902 }
903 }
904 }
905 }
906 }
907 }
908 }
909 }
910 }
911 }
912 }
913 }
914 }
915 }
916 }
917 }
918 }
919 }
920 }
921 }
922 }
923 }
924 }
925 }
926 }
927 }
928 }
929 }
930 }
931 }
932 }
933 }
934 }
935 }
936 }
937 }
938 }
939 }
940 }
941 }
942 }
943 }
944 }
945 }
946 }
947 }
948 }
949 }
950 }
951 }
952 }
953 }
954 }
955 }
956 }
957 }
958 }
959 }
960 }
961 }
962 }
963 }
964 }
965 }
966 }
967 }
968 }
969 }
970 }
971 }
972 }
973 }
974 }
975 }
976 }
977 }
978 }
979 }
980 }
981 }
982 }
983 }
984 }
985 }
986 }
987 }
988 }
989 }
990 }
991 }
992 }
993 }
994 }
995 }
996 }
997 }
998 }
999 }
```

unchanged_portion_omitted_