```
**********************************************************
   15588 Fri Mar 27 10:14:59 2015
new/usr/src/cmd/hal/hald/hald.c
3792 hald.c:371: error: ‘g_type_init’ is deprecated
Reviewed by: Igor Kozhukhov <ikozhukhov@gmail.com>
Reviewed by: Jon Tibble <meths@btinternet.com>
Reviewed by: Peter Tribble <peter.tribble@gmail.com>
**********************************************************
______unchanged_portion_omitted_

353 /*--------------------------------------------------------------------------

355 /** Entry point for HAL daemon
356  *
357  *  @param  argc                Number of arguments
358  *  @param  argv                Array of arguments
359  *  @return                     Exit code
360  */
361 int
362 main (int argc, char *argv[])
363 {
364         GMainLoop *loop;
365         guint sigterm_iochn_listener_source_id;
366         char *path;
367         char newpath[512];

369         openlog ("hald", LOG_PID, LOG_DAEMON);
370 #if !GLIB_CHECK_VERSION(2,35,0)

371         g_type_init ();
372 #endif

373         if (getenv ("HALD_VERBOSE"))
374                 hald_is_verbose = TRUE;
375         else
376                 hald_is_verbose = FALSE;

378         /* our helpers are installed into libexec, so adjust out $PATH
379          * to include this at the end (since we want to overide in
380          * run-hald.sh and friends)
381          */
382         path = getenv ("PATH");
383         if (path != NULL) {
384                 g_strlcpy (newpath, path, sizeof (newpath));
385                 g_strlcat (newpath, ":", sizeof (newpath));
386         } else {
387                 /* No PATH was set */
388                 newpath[0] = '\0';
389         }

391         g_strlcat (newpath, PACKAGE_LIBEXEC_DIR, sizeof (newpath));
392         g_strlcat (newpath, ":", sizeof (newpath));
393         g_strlcat (newpath, PACKAGE_SCRIPT_DIR, sizeof (newpath));

395         setenv ("PATH", newpath, TRUE);

397         while (1) {
398                 int c;
399                 int option_index = 0;
400                 const char *opt;
401                 static struct option long_options[] = {
402                         {"daemon", 1, NULL, 0},
403                         {"verbose", 1, NULL, 0},
404                         {"use-syslog", 0, NULL, 0},
405                         {"help", 0, NULL, 0},
406                         {"version", 0, NULL, 0},
```



```
407                         {NULL, 0, NULL, 0}
408                 };

410                 c = getopt_long (argc, argv, "",
411                                  long_options, &option_index);
412                 if (c == -1)
413                         break;

415                 switch (c) {
416                 case 0:
417                         opt = long_options[option_index].name;

419                         if (strcmp (opt, "help") == 0) {
420                                 usage ();
421                                 return 0;
422                         } else if (strcmp (opt, "version") == 0) {
423                                 fprintf (stderr, "HAL package version: " PACKAGE
424                                 return 0;
425                         } else if (strcmp (opt, "daemon") == 0) {
426                                 if (strcmp ("yes", optarg) == 0) {
427                                         opt_become_daemon = TRUE;
428                                 } else if (strcmp ("no", optarg) == 0) {
429                                         opt_become_daemon = FALSE;
430                                 } else {
431                                         usage ();
432                                         return 1;
433                                 }
434                         } else if (strcmp (opt, "verbose") == 0) {
435                                 if (strcmp ("yes", optarg) == 0) {
436                                         hald_is_verbose = TRUE;
437                                 } else if (strcmp ("no", optarg) == 0) {
438                                         hald_is_verbose = FALSE;
439                                 } else {
440                                         usage ();
441                                         return 1;
442                                 }
443                         } else if (strcmp (opt, "use-syslog") == 0) {
444                                 hald_use_syslog = TRUE;
445                         }

447                         break;

449                 default:
450                         usage ();
451                         return 1;
452                         break;
453                 }
454         }

456         if (hald_is_verbose)
457                 logger_enable ();
458         else
459                 logger_disable ();

461         if (hald_use_syslog)
462                 logger_enable_syslog ();
463         else
464                 logger_disable_syslog ();

466         /* will fork into two; only the child will return here if we are success
467         /*master_slave_setup ();
468           sleep (100000000);*/

470         loop = g_main_loop_new (NULL, FALSE);

472         HAL_INFO ((PACKAGE_STRING));
```

```
474         if (opt_become_daemon) {
475                 int child_pid;
476                 int dev_null_fd;
477                 int pf;
478                 ssize_t written;
479                 char pid[9];
480
481                 HAL_INFO (("Will daemonize"));
482                 HAL_INFO (("Becoming a daemon"));

484                 if (pipe (startup_daemonize_pipe) != 0) {
485                         fprintf (stderr, "Could not setup pipe: %s\n", strerror(
486                         exit (1);
487                 }


490                 if (chdir ("/") < 0) {
491                         fprintf (stderr, "Could not chdir to /: %s\n", strerror(
492                         exit (1);
493                 }

495                 child_pid = fork ();
496                 switch (child_pid) {
497                 case -1:
498                         fprintf (stderr, "Cannot fork(): %s\n", strerror(errno))
499                         break;

501                 case 0:
502                         /* child */

504                         dev_null_fd = open ("/dev/null", O_RDWR);
505                         /* ignore if we can't open /dev/null */
506                         if (dev_null_fd >= 0) {
507                                 /* attach /dev/null to stdout, stdin, stderr */
508                                 dup2 (dev_null_fd, 0);
509                                 dup2 (dev_null_fd, 1);
510                                 dup2 (dev_null_fd, 2);
511                                 close (dev_null_fd);
512                         }

514                         umask (022);
515                         break;

517                 default:
518                         /* parent, block until child writes */
519                         exit (parent_wait_for_child (startup_daemonize_pipe[0],
520                         break;
521                 }

523                 /* Create session */
524                 setsid ();

526                 /* remove old pid file */
527                 unlink (HALD_PID_FILE);

529                 /* Make a new one */
530                 if ((pf= open (HALD_PID_FILE, O_WRONLY|O_CREAT|O_TRUNC|O_EXCL, 0
531                         snprintf (pid, sizeof(pid), "%lu\n", (long unsigned) get
532                         written = write (pf, pid, strlen(pid));
533                         close (pf);
534                         atexit (delete_pid);
535                 }
536         } else {
537                 HAL_INFO (("Will not daemonize"));
538         }
```



```
541         /* we need to do stuff when we are expected to terminate, thus
542          * this involves looking for SIGTERM; UNIX signal handlers are
543          * evil though, so set up a pipe to transmit the signal.
544          */

546         /* create pipe */
547         if (pipe (sigterm_unix_signal_pipe_fds) != 0) {
548                 DIE (("Could not setup pipe, errno=%d", errno));
549         }
550
551         /* setup glib handler - 0 is for reading, 1 is for writing */
552         sigterm_iochn = g_io_channel_unix_new (sigterm_unix_signal_pipe_fds[0]);
553         if (sigterm_iochn == NULL)
554                 DIE (("Could not create GIOChannel"));
555
556         /* get callback when there is data to read */
557         sigterm_iochn_listener_source_id = g_io_add_watch (
558                 sigterm_iochn, G_IO_IN, sigterm_iochn_data, NULL);
559
560         /* Finally, setup unix signal handler for TERM */
561         signal (SIGTERM, handle_sigterm);

563         /* set up the local dbus server */
564         if (!hald_dbus_local_server_init ())
565                 return 1;
566         /* Start the runner helper daemon */
567         if (!hald_runner_start_runner ()) {
568                 return 1;
569         }

571         drop_privileges(0);

573         /* initialize operating system specific parts */
574         osspec_init ();

576         hald_is_initialising = TRUE;

578         /* detect devices */
579         osspec_probe ();

581         /* run the main loop and serve clients */
582         g_main_loop_run (loop);

584         return 0;
585 }
```

_____***unchanged_portion_omitted_***