```
**********************************************************
   15588 Thu Mar 26 21:29:54 2015
new/usr/src/cmd/hal/hald/hald.c
3792 hald.c:371: error: 'g_type_init' is deprecated
Reviewed by: Jon Tibble <meths@btinternet.com>
**********************************************************
______unchanged_portion_omitted_

/*--------------------------------------------------------------------------------

/** Entry point for HAL daemon
 *
 *  @param  argc                Number of arguments
 *  @param  argv                Array of arguments
 *  @return                     Exit code
 */
int
main (int argc, char *argv[])
{
        GMainLoop *loop;
        guint sigterm_iochn_listener_source_id;
        char *path;
        char newpath[512];

        openlog ("hald", LOG_PID, LOG_DAEMON);
#if !defined(GLIB_VERSION_2_36)

        g_type_init ();
#endif

        if (getenv ("HALD_VERBOSE"))
                hald_is_verbose = TRUE;
        else
                hald_is_verbose = FALSE;

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

        g_strlcat (newpath, PACKAGE_LIBEXEC_DIR, sizeof (newpath));
        g_strlcat (newpath, ":", sizeof (newpath));
        g_strlcat (newpath, PACKAGE_SCRIPT_DIR, sizeof (newpath));

        setenv ("PATH", newpath, TRUE);

        while (1) {
                int c;
                int option_index = 0;
                const char *opt;
                static struct option long_options[] = {
                        {"daemon", 1, NULL, 0},
                        {"verbose", 1, NULL, 0},
                        {"use-syslog", 0, NULL, 0},
                        {"help", 0, NULL, 0},
                        {"version", 0, NULL, 0},
                        {NULL, 0, NULL, 0}
                };
```



```
                c = getopt_long (argc, argv, "",
                                 long_options, &option_index);
                if (c == -1)
                        break;

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

                        if (strcmp (opt, "help") == 0) {
                                usage ();
                                return 0;
                        } else if (strcmp (opt, "version") == 0) {
                                fprintf (stderr, "HAL package version: " PACKAGE
                                return 0;
                        } else if (strcmp (opt, "daemon") == 0) {
                                if (strcmp ("yes", optarg) == 0) {
                                        opt_become_daemon = TRUE;
                                } else if (strcmp ("no", optarg) == 0) {
                                        opt_become_daemon = FALSE;
                                } else {
                                        usage ();
                                        return 1;
                                }
                        } else if (strcmp (opt, "verbose") == 0) {
                                if (strcmp ("yes", optarg) == 0) {
                                        hald_is_verbose = TRUE;
                                } else if (strcmp ("no", optarg) == 0) {
                                        hald_is_verbose = FALSE;
                                } else {
                                        usage ();
                                        return 1;
                                }
                        } else if (strcmp (opt, "use-syslog") == 0) {
                                hald_use_syslog = TRUE;
                        }

                        break;

                default:
                        usage ();
                        return 1;
                        break;
                }
        }

        if (hald_is_verbose)
                logger_enable ();
        else
                logger_disable ();

        if (hald_use_syslog)
                logger_enable_syslog ();
        else
                logger_disable_syslog ();

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

        loop = g_main_loop_new (NULL, FALSE);

        HAL_INFO ((PACKAGE_STRING));

        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_***