```
**********************************************************
   24930 Fri Jul 12 15:06:09 2019
new/usr/src/lib/brand/lx/lx_init/lxinit.c
want lxinit to support link-local routes
**********************************************************
______unchanged_portion_omitted_

static int lxi_route_send_msg(struct rt_msghdr * rtm)
{
    int sockfd, len;

    if ((sockfd = socket(PF_ROUTE, SOCK_RAW, AF_INET)) < 0) {
        lxi_warn("socket(PF_ROUTE): %s\n", strerror(errno));
        return (-1);
    }

    if ((len = write(sockfd, (const void *)rtm, rtm->rtm_msglen)) < 0) {
        lxi_warn("could not write rtmsg: %s", strerror(errno));
        close(sockfd);
        return (-1);
    } else if (len < rtm->rtm_msglen) {
        lxi_warn("write() rtmsg incomplete");
        close(sockfd);
        return (-1);
    }

    close(sockfd);
    return (0);
}

static int
lxi_iface_gateway(const char *iface, const char *dst, int dstpfx,
    const char *gwaddr, int llroute)
    const char *gwaddr)
{
        int idx;
        int idx, len, sockfd;
        char rtbuf[RTMBUFSZ];
        struct rt_msghdr *rtm = (struct rt_msghdr *)rtbuf;
        struct sockaddr_in *dst_sin = (struct sockaddr_in *)
            (rtbuf + sizeof (struct rt_msghdr));
        struct sockaddr_in *gw_sin = (struct sockaddr_in *)(dst_sin + 1);
        struct sockaddr_in *netmask_sin = (struct sockaddr_in *)(gw_sin + 1);

        (void) bzero(rtm, RTMBUFSZ);
        rtm->rtm_addrs = RTA_DST | RTA_GATEWAY | RTA_NETMASK;

        if(!llroute)
          rtm->rtm_flags = RTF_UP | RTF_STATIC | RTF_GATEWAY;

        rtm->rtm_msglen = sizeof (rtbuf);
        rtm->rtm_pid = getpid();
        rtm->rtm_type = RTM_ADD;
        rtm->rtm_version = RTM_VERSION;


        /*
         * The destination and netmask components have already been zeroed,
         * which represents the default gateway.  If we were passed a more
         * specific destination network, use that instead.
         */

        if(dstpfx == -1) {
          /*
           * no prefix was specified; assume a prefix length of 32,
           * which seems in line with the behavior of vmadm.
```



```
           */
            dstpfx = 32;
        }

        dst_sin->sin_family = AF_INET;
        netmask_sin->sin_family = AF_INET;
        if (dst != NULL) {
                struct sockaddr *mask = (struct sockaddr *)netmask_sin;

                if ((inet_pton(AF_INET, dst, &(dst_sin->sin_addr))) != 1 ||
                    plen2mask(dstpfx, AF_INET, mask) != 0) {
                        lxi_warn("bad destination network %s/%d: %s", dst,
                            dstpfx, strerror(errno));
                        return (-1);
                }
        }

        if ((inet_pton(AF_INET, gwaddr, &(gw_sin->sin_addr))) != 1) {
                lxi_warn("bad gateway %s: %s", gwaddr, strerror(errno));
                return (-1);
        }

        if (iface != NULL) {
                if ((idx = if_nametoindex(iface)) == 0) {
                        lxi_warn("unable to get interface index for %s: %s\n",
                            iface, strerror(errno));
                        return (-1);
                }
                rtm->rtm_index = idx;
        }

        return lxi_route_send_msg(rtm);
        if ((sockfd = socket(PF_ROUTE, SOCK_RAW, AF_INET)) < 0) {
                lxi_warn("socket(PF_ROUTE): %s\n", strerror(errno));
                return (-1);
        }

        if ((len = write(sockfd, rtbuf, rtm->rtm_msglen)) < 0) {
                lxi_warn("could not write rtmsg: %s", strerror(errno));
                close(sockfd);
                return (-1);
        } else if (len < rtm->rtm_msglen) {
                lxi_warn("write() rtmsg incomplete");
                close(sockfd);
                return (-1);
        }

        close(sockfd);
        return (0);
}
______unchanged_portion_omitted_

static void
lxi_net_setup(zone_dochandle_t handle)
{
        struct zone_nwiftab lookup;
        boolean_t do_addrconf = B_FALSE;

        if (zonecfg_setnwifent(handle) != Z_OK)
                return;
        while (zonecfg_getnwifent(handle, &lookup) == Z_OK) {
                const char *iface = lookup.zone_nwif_physical;
                struct zone_res_attrtab *attrs = lookup.zone_nwif_attrp;
                const char *ipaddrs;
                const char *ipaddrs, *primary, *gateway;
                char ipaddrs_copy[MAXNAMELEN], cidraddr[BUFSIZ],
```

```
                    *ipaddr, *tmp, *lasts;
                boolean_t first_ipv4_configured = B_FALSE;
                boolean_t *ficp = &first_ipv4_configured;

                lxi_net_plumb(iface);
                if (zone_find_attr(attrs, "ips", &ipaddrs) != 0 &&
                    lxi_get_old_ip(attrs, &ipaddrs, cidraddr, BUFSIZ) != 0) {
                        lxi_warn("Could not find a valid network configuration "
                            "for the %s interface", iface);
                        continue;
                }

                if (lxi_iface_ipv6_link_local(iface) != 0) {
                        lxi_warn("unable to bring up link-local address on "
                            "interface %s", iface);
                }

                /*
                 * If we're going to be doing DHCP, we have to do it first since
                 * dhcpagent doesn't like to operate on non-zero logical
                 * interfaces.
                 */
                if (strstr(ipaddrs, "dhcp") != NULL &&
                    lxi_iface_dhcp(iface, ficp) != 0) {
                        lxi_warn("Failed to start DHCP on %s\n", iface);
                }

                /*
                 * Copy the ipaddrs string, since strtok_r will write NUL
                 * characters into it.
                 */
                (void) strlcpy(ipaddrs_copy, ipaddrs, MAXNAMELEN);
                tmp = ipaddrs_copy;

                /*
                 * Iterate over each IP and then set it up on the interface.
                 */
                while ((ipaddr = strtok_r(tmp, ",", &lasts)) != NULL) {
                        tmp = NULL;
                        if (strcmp(ipaddr, "addrconf") == 0) {
                                do_addrconf = B_TRUE;
                        } else if (strcmp(ipaddr, "dhcp") == 0) {
                                continue;
                        } else if (lxi_iface_ip(iface, ipaddr, ficp) < 0) {
                                lxi_warn("Unable to add new IP address (%s) "
                                    "to interface %s", ipaddr, iface);
                        }
                }
        }

        if (do_addrconf) {
                lxi_net_ndpd_start();
        }

        (void) zonecfg_endnwifent(handle);
}

static void
lxi_net_setup_gateways(zone_dochandle_t handle)
{
    struct zone_nwiftab lookup;

    if (zonecfg_setnwifent(handle) != Z_OK)
        return;
    while (zonecfg_getnwifent(handle, &lookup) == Z_OK) {
        const char *iface = lookup.zone_nwif_physical;
```



```
        struct zone_res_attrtab *attrs = lookup.zone_nwif_attrp;
        const char *primary, *gateway;

        if (zone_find_attr(attrs, "primary", &primary) == 0 &&
            strncmp(primary, "true", MAXNAMELEN) == 0 &&
            zone_find_attr(attrs, "gateway", &gateway) == 0) {
            lxi_iface_gateway(iface, NULL, 0, gateway, 0);
                        lxi_iface_gateway(iface, NULL, 0, gateway);
        }
    }

        if (do_addrconf) {
                lxi_net_ndpd_start();
        }

    (void) zonecfg_endnwifent(handle);
}

static void
lxi_parse_route_line(const char *line, custr_t *cu, char *gw, char *dst, int* pf
lxi_net_static_route(const char *line)
{
        /*
         * Each static route line is a string of the form:
         *
         *      "10.77.77.2|10.1.1.0/24|false"
         *
         * i.e. gateway address, destination network, and whether this is
         * a "link local" route or a next hop route.
         */
        custr_t *cu = NULL;
        char *gw = NULL, *dst = NULL;
        int pfx = -1;
    int i;
    int havegw = 0, havedst = 0, nopfx = 0;

        if (custr_alloc(&cu) != 0) {
                lxi_err("custr_alloc failure");
        }

    for (i = 0; line[i] != '\0'; i++) {
        if (!havegw) {
                if (gw == NULL) {
            if (line[i] == '|') {
                if ((strlcpy(gw, custr_cstr(cu), gwlen) ) == 0) {
                    lxi_err("strlcpy failure");
                } else {
                    havegw = 1;
                                if ((gw = strdup(custr_cstr(cu))) == NULL) {
                                        lxi_err("strdup failure");
                }
                custr_reset(cu);
            } else {
                if (custr_appendc(cu, line[i]) != 0) {
                    lxi_err("custr_appendc failure");
                }
            }
            continue;
        }

        if (!havedst) {
            if (line[i] == '/' || line[i] == '|') {
                if ((strlcpy(dst, custr_cstr(cu), dstlen)) == 0) {
                    lxi_err("strlcpy failure");
                } else {
                    havedst = 1;
```

```
                    if(line[i] == '|') {
                        nopfx = 1;
                if (dst == NULL) {
                        if (line[i] == '/') {
                                if ((dst = strdup(custr_cstr(cu))) == NULL) {
                                        lxi_err("strdup failure");
                    }
                }
                custr_reset(cu);
            } else {
                if (custr_appendc(cu, line[i]) != 0) {
                    lxi_err("custr_appendc failure");
                }
            }
            continue;
        }

        if (*pfx == -1 && !nopfx) {
                if (pfx == -1) {
            if (line[i] == '|') {
                *pfx = atoi(custr_cstr(cu));
                                pfx = atoi(custr_cstr(cu));
                custr_reset(cu);
            } else {
                if (custr_appendc(cu, line[i]) != 0) {
                    lxi_err("custr_appendc failure");
                }
            }
            continue;
        }

        if (custr_appendc(cu, line[i]) != 0) {
            lxi_err("custr_appendc failure");
        }
    }

    return;
}

static void
lxi_net_process_route_line(const char *line, int llonly)
{
        /*
         * Each static route line is a string of the form:
         *
         *      "10.77.77.2|10.1.1.0/24|false"
         *
         * i.e. gateway address, destination network, and whether this is
         * a "link local" route or a next hop route.
         *
         * passing a true value  to the llonly argument to this method causes on
         * those lines with the link-local flag to be processed and vice-versa
         * We currently only support "next hop" routes, so ensure that
         * "linklocal" is false:
         */
        custr_t *cu = NULL;
        int pfx = -1;
        char gw[INET6_ADDRSTRLEN];
        char dst[INET6_ADDRSTRLEN];

        if (custr_alloc(&cu) != 0) {
                lxi_err("custr_alloc failure");
        if (strcmp(custr_cstr(cu), "false") != 0) {
                lxi_warn("invalid static route: %s", line);
        }
```



```
        lxi_parse_route_line(line, cu, gw, dst, &pfx, sizeof(gw), sizeof(dst));

        if (llonly && strcmp(custr_cstr(cu), "true") == 0) {
          if (lxi_iface_gateway(NULL, dst, pfx, gw, 1) != 0) {
                lxi_err("failed to add link-local route: %s/%d -> %s", dst, pfx,
        if (lxi_iface_gateway(NULL, dst, pfx, gw) != 0) {
                lxi_err("failed to add route: %s/%d -> %s", dst, pfx, gw);
          }
        } else if(!llonly && strcmp(custr_cstr(cu), "false") == 0) {
          if (lxi_iface_gateway(NULL, dst, pfx, gw, 0) != 0) {
                lxi_err("failed to add next-hop route: %s/%d -> %s", dst, pfx, g
          }
        } else if(strcmp(custr_cstr(cu), "true") != 0 && strcmp(custr_cstr(cu),
          /*
           * try to be helpful when we encounter something we don't expect.
           */
          lxi_warn("skipping unknown static route defined in line %s, parsed lin
        }

        custr_free(cu);
        free(gw);
        free(dst);
}

static void
lxi_net_static_route(const char * line) {
        /*
         * process only those lines where the link-local field is false.
         */
        lxi_net_process_route_line(line, 0);
}

static void
lxi_net_linklocal_route(const char * line) {
        /*
         * processes only those lines where the link-local field is true.
         */
        lxi_net_process_route_line(line, 1);
}

static void
lxi_run_routeinfo(void * callback)
lxi_net_static_routes(void)
{
    const char *cmd = "/native/usr/lib/brand/lx/routeinfo";
    char *const argv[] = { "routeinfo", NULL };
    char *const envp[] = { NULL };
    int code;
    struct stat st;
    char errbuf[512];

    if (stat(cmd, &st) != 0 || !S_ISREG(st.st_mode)) {
        /*
         * This binary is (potentially) shipped from another
         * consolidation.  If it does not exist, then the platform does
         * not currently support link-local or static routes for LX-branded zone
                 * not currently support static routes for LX-branded zones.
         */
        return;
    }

    /*
     * Run the command, firing the callback for each line that it
     * outputs.  When this function returns, static route processing
     * is complete.
     */
```

```
	if (run_command(cmd, argv, envp, errbuf, sizeof (errbuf),
			callback, &code) != 0 || code != 0) {
		lxi_net_static_route, &code) != 0 || code != 0) {
		lxi_err("failed to run \"%s\": %s", cmd, errbuf);
	}
}

static void
lxi_net_linklocal_routes(void)
{
	lxi_run_routeinfo(lxi_net_linklocal_route);
}


static void
lxi_net_static_routes(void)
{
	lxi_run_routeinfo(lxi_net_static_route);
}

static void
lxi_config_close(zone_dochandle_t handle)
{
	zonecfg_fini_handle(handle);
}
______unchanged_portion_omitted_

/*ARGSUSED*/
int
main(int argc, char *argv[])
{
	zone_dochandle_t handle;

	lxi_log_open();

	lxi_net_ipmgmtd_start();
	lxi_net_ipadm_open();

	handle = lxi_config_open();
	lxi_net_loopback();
	lxi_net_setup(handle);

	lxi_net_linklocal_routes();
	lxi_net_setup_gateways(handle);

	lxi_config_close(handle);

	lxi_net_static_routes();

	lxi_net_ipadm_close();

	lxi_hook_postnet();

	lxi_log_close();

	lxi_init_exec(argv);

	/* NOTREACHED */
	return (0);
}
______unchanged_portion_omitted_
```