```
**********************************************************
   26024 Thu Jun 28 19:00:56 2012
new/usr/src/cmd/sgs/elfdump/common/main.c
2940 elfdump should return non-null on invalid file type
**********************************************************
______unchanged_portion_omitted_

int
main(int argc, char **argv, char **envp)
{
        Elf             *elf;
        int             var, fd, wfd = 0;
        char            *wname = NULL;
        uint_t          flags = 0;
        match_rec_t     match_data;
        int             ret;
        uchar_t         osabi;

        /*
         * If we're on a 64-bit kernel, try to exec a full 64-bit version of
         * the binary.  If successful, conv_check_native() won't return.
         */
        (void) conv_check_native(argv, envp);

        /*
         * Establish locale.
         */
        (void) setlocale(LC_MESSAGES, MSG_ORIG(MSG_STR_EMPTY));
        (void) textdomain(MSG_ORIG(MSG_SUNW_OST_SGS));

        (void) setvbuf(stdout, NULL, _IOLBF, 0);
        (void) setvbuf(stderr, NULL, _IOLBF, 0);

        opterr = 0;
        while ((var = getopt(argc, argv, MSG_ORIG(MSG_STR_OPTIONS))) != EOF) {
                switch (var) {
                case 'C':
                        flags |= FLG_CTL_DEMANGLE;
                        break;
                case 'c':
                        flags |= FLG_SHOW_SHDR;
                        break;
                case 'd':
                        flags |= FLG_SHOW_DYNAMIC;
                        break;
                case 'e':
                        flags |= FLG_SHOW_EHDR;
                        break;
                case 'G':
                        flags |= FLG_SHOW_GOT;
                        break;
                case 'g':
                        flags |= FLG_SHOW_GROUP;
                        break;
                case 'H':
                        flags |= FLG_SHOW_CAP;
                        break;
                case 'h':
                        flags |= FLG_SHOW_HASH;
                        break;
                case 'I':
                        if (!process_index_opt(optarg, &match_data))
                                goto usage_brief;
                        if (!add_match_record(argv[0], &match_data))
                                return (1);
                        flags |= FLG_CTL_MATCH;
```

```
                        break;
                case 'i':
                        flags |= FLG_SHOW_INTERP;
                        break;
                case 'k':
                        flags |= FLG_CALC_CHECKSUM;
                        break;
                case 'l':
                        flags |= FLG_CTL_LONGNAME;
                        break;
                case 'm':
                        flags |= FLG_SHOW_MOVE;
                        break;
                case 'N':
                        match_data.opt_type = MATCH_OPT_NAME;
                        match_data.value.name = optarg;
                        if (!add_match_record(argv[0], &match_data))
                                return (1);
                        flags |= FLG_CTL_MATCH;
                        break;
                case 'n':
                        flags |= FLG_SHOW_NOTE;
                        break;
                case 'O':
                        {
                                uint32_t val;

                                /*
                                 * osabi is a uchar_t in the ELF header.
                                 * Don't accept any value that exceeds
                                 * that range.
                                 */
                                if ((atoui(optarg, ATOUI_OSABI, &val) == 0) ||
                                    (val > 255)) {
                                        (void) fprintf(stderr,
                                            MSG_INTL(MSG_ERR_BAD_T_OSABI),
                                            basename(argv[0]), optarg);
                                        return (1);
                                }
                                osabi = val;
                        }
                        flags |= FLG_CTL_OSABI;
                        break;
                case 'P':
                        flags |= FLG_CTL_FAKESHDR;
                        break;
                case 'p':
                        flags |= FLG_SHOW_PHDR;
                        break;
                case 'r':
                        flags |= FLG_SHOW_RELOC;
                        break;
                case 'S':
                        flags |= FLG_SHOW_SORT;
                        break;
                case 's':
                        flags |= FLG_SHOW_SYMBOLS;
                        break;
                case 'T':
                        /*
                         * We can't evaluate the value yet, because
                         * we need to know if -p is used or not in
                         * order to tell if we're seeing section header
                         * or program header types. So, we save the
                         * string in the name field, and then convert
                         * it to a type integer in a following pass.
```

```
			 */
			match_data.opt_type = MATCH_OPT_TYPE;
			match_data.value.name = optarg;
			if (!add_match_record(argv[0], &match_data))
				return (1);
			flags |= FLG_CTL_MATCH;
			break;
		case 'u':
			flags |= FLG_SHOW_UNWIND;
			break;
		case 'v':
			flags |= FLG_SHOW_VERSIONS;
			break;
		case 'w':
			wname = optarg;
			break;
		case 'y':
			flags |= FLG_SHOW_SYMINFO;
			break;
		case '?':
			(void) fprintf(stderr, MSG_INTL(MSG_USAGE_BRIEF),
			    basename(argv[0]));
			detail_usage();
			return (1);
		default:
			break;
		}
	}

	/* -p and -w are mutually exclusive. -w only works with sections */
	if (((flags & FLG_SHOW_PHDR) != 0) && (wname != NULL))
		goto usage_brief;

	/* If a match argument is present, prepare the match state */
	if ((match_state.list != NULL) && (match_prepare(argv[0], flags) == 0))
		return (1);

	/*
	 * Decide what to do if no options specifying something to
	 * show or do are present.
	 *
	 * If there is no -w and no match options, then we will set all
	 * the show flags, causing a full display of everything in the
	 * file that we know how to handle.
	 *
	 * Otherwise, if there is no match list, we generate a usage
	 * error and quit.
	 *
	 * In the case where there is a match list, we go ahead and call
	 * regular() anyway, leaving it to decide what to do. If -w is
	 * present, regular() will use the match list to handle it.
	 * In addition, in the absence of explicit show/calc flags, regular()
	 * will compare the section headers to the match list and use
	 * that to generate the FLG_ bits that will display the information
	 * specified by the match list.
	 */
	if ((flags & ~FLG_MASK_CTL) == 0) {
		if (!wname && (match_state.list == NULL))
			flags |= FLG_MASK_SHOW;
		else if (match_state.list == NULL)
			goto usage_brief;
	}

	/* There needs to be at least 1 filename left following the options */
	if ((var = argc - optind) == 0)
		goto usage_brief;
```

```
	/*
	 * If the -l/-C option is specified, set up the liblddbg.so.
	 */
	if (flags & FLG_CTL_LONGNAME)
		dbg_desc->d_extra |= DBG_E_LONG;
	if (flags & FLG_CTL_DEMANGLE)
		dbg_desc->d_extra |= DBG_E_DEMANGLE;

	/*
	 * If the -w option has indicated an output file open it.  It's
	 * arguable whether this option has much use when multiple files are
	 * being processed.
	 *
	 * If wname is non-NULL, we know that -p was not specified, due
	 * to the test above.
	 */
	if (wname) {
		if ((wfd = open(wname, (O_RDWR | O_CREAT | O_TRUNC),
		    0666)) < 0) {
			int err = errno;
			(void) fprintf(stderr, MSG_INTL(MSG_ERR_OPEN),
			    wname, strerror(err));
			return (1);
		}
	}

	/*
	 * Open the input file, initialize the elf interface, and
	 * process it.
	 */
	ret = 0;
	for (; (optind < argc) && (ret == 0); optind++) {
		const char	*file = argv[optind];

		if ((fd = open(argv[optind], O_RDONLY)) == -1) {
			int err = errno;
			(void) fprintf(stderr, MSG_INTL(MSG_ERR_OPEN),
			    file, strerror(err));
			continue;
		}
		(void) elf_version(EV_CURRENT);
		if ((elf = elf_begin(fd, ELF_C_READ, NULL)) == NULL) {
			failure(file, MSG_ORIG(MSG_ELF_BEGIN));
			(void) close(fd);
			continue;
		}

		if (var > 1)
			dbg_print(0, MSG_ORIG(MSG_FMT_NLSTRNL), file);

		switch (elf_kind(elf)) {
		case ELF_K_AR:
			ret = archive(file, fd, elf, flags, wname, wfd, osabi);
			break;
		case ELF_K_ELF:
			ret = decide(file, fd, elf, flags, wname, wfd, osabi);
			break;
		default:
			(void) fprintf(stderr, MSG_INTL(MSG_ERR_BADFILE), file);
			ret = 1;
#endif /* ! codereview */
			break;
		}

		(void) close(fd);
		(void) elf_end(elf);
	}

	if (wfd)
		(void) close(wfd);
	return (ret);

usage_brief:
	/* Control comes here for a simple usage message and exit */
	(void) fprintf(stderr, MSG_INTL(MSG_USAGE_BRIEF),
	    basename(argv[0]));
	return (1);

}
```