```
**********************************************************
    2827 Wed Sep 21 15:14:29 2016
new/usr/src/cmd/svc/milestone/process-security.xml
smf: switch to a tri-state for process-security properties true=on,false=off,nil
**********************************************************
<?xml version='1.0'?>
<!DOCTYPE service_bundle SYSTEM '/usr/share/lib/xml/dtd/service_bundle.dtd.1'>

<!--
 Copyright 2015, Richard Lowe.

 CDDL HEADER START

 This file and its contents are supplied under the terms of the
 Common Development and Distribution License ("CDDL"), version 1.0.
 You may only use this file in accordance with the terms of version
 1.0 of the CDDL.

 A full copy of the text of the CDDL should have accompanied this
 source.  A copy of the CDDL is also available via the Internet at
 http://www.illumos.org/license/CDDL.

 CDDL HEADER END

        NOTE:  This service manifest is not editable; its contents will
        be overwritten by package or patch operations, including
        operating system upgrade.  Make customizations in a different
        file.
-->

<service_bundle type="manifest" name="process-security">
        <service name="system/process-security" type="service" version="1">
                <!-- Initial state of the service is disabled -->
                <create_default_instance enabled="false" />

                <single_instance />

                <!-- We don't actually have any methods, but we create a
                     default instance so that we show up in svcs -a -->

                <exec_method type="method" name="start" exec=":true" timeout_sec
                <exec_method type="method" name="stop" exec=":true" timeout_seco

                <property_group name='startd' type='framework'>
                  <propval name='duration' type='astring' value='transient' />
                </property_group>

                <property_group name='default' type='application'>
                  <property name='aslr' type='boolean' />
                  <property name='forbidnullmap' type='boolean' />
                  <property name='noexecstack' type='boolean' />
                  <propval name='aslr' type='boolean' value='false' />
                  <propval name='forbidnullmap' type='boolean' value='false' />
                  <propval name='noexecstack' type='boolean' value='false' />

                  <propval name='value_authorization' type='astring'
                           value='solaris.smf.value.process-security' />
                </property_group>

                <property_group name='lower' type='application'>
                  <property name='aslr' type='boolean' />
                  <property name='forbidnullmap' type='boolean' />
                  <property name='noexecstack' type='boolean' />
                  <propval name='aslr' type='boolean' value='false' />
                  <propval name='forbidnullmap' type='boolean' value='false' />
                  <propval name='noexecstack' type='boolean' value='false' />
```

```
                  <propval name='value_authorization' type='astring'
                           value='solaris.smf.value.process-security' />
                </property_group>

                <property_group name='upper' type='application'>
                  <property name='aslr' type='boolean' />
                  <property name='forbidnullmap' type='boolean' />
                  <property name='noexecstack' type='boolean' />
                  <propval name='aslr' type='boolean' value='true' />
                  <propval name='forbidnullmap' type='boolean' value='true' />
                  <propval name='noexecstack' type='boolean' value='true' />

                  <propval name='value_authorization' type='astring'
                           value='solaris.smf.value.process-security' />
                </property_group>



                <stability value="Unstable" />

                <template>
                        <common_name>
                                <loctext xml:lang='C'>Security Flag Configuratio
                        </common_name>
                        <documentation>
                                <manpage title='security-flags' section='5'
                                         manpath='/usr/share/man' />
                                <manpage title='psecflags' section='1'
                                         manpath='/usr/share/man' />
                        </documentation>
                </template>
        </service>
</service_bundle>
```

****
   97936 Wed Sep 21 15:14:31 2016
new/usr/src/lib/librestart/common/librestart.c
smf: switch to a tri-state for process-security properties true=on,false=off,nil
****
______unchanged_portion_omitted_

```
/*
 * Fetch method context information from the repository, allocate and fill
 * a method_context structure, return it in *mcpp, and return NULL.
 *
 * If no method_context is defined, original init context is provided, where
 * the working directory is '/', and uid/gid are 0/0.  But if a method_context
 * is defined at any level the smf_method(5) method_context defaults are used.
 *
 * Return an error message structure containing the error message
 * with context, and the error so the caller can make a decision
 * on what to do next.
 *
 * Error Types :
 *      E2BIG           Too many values or entry is too big
 *      EINVAL          Invalid value
 *      EIO             an I/O error has occured
 *      ENOENT          no entry for value
 *      ENOMEM          out of memory
 *      ENOTSUP         Version mismatch
 *      ERANGE          value is out of range
 *      EMFILE/ENFILE   out of file descriptors
 *
 *      SCF_ERROR_BACKEND_ACCESS
 *      SCF_ERROR_CONNECTION_BROKEN
 *      SCF_ERROR_DELETED
 *      SCF_ERROR_CONSTRAINT_VIOLATED
 *      SCF_ERROR_HANDLE_DESTROYED
 *      SCF_ERROR_INTERNAL
 *      SCF_ERROR_INVALID_ARGUMENT
 *      SCF_ERROR_NO_MEMORY
 *      SCF_ERROR_NO_RESOURCES
 *      SCF_ERROR_NOT_BOUND
 *      SCF_ERROR_NOT_FOUND
 *      SCF_ERROR_NOT_SET
 *      SCF_ERROR_TYPE_MISMATCH
 *
 */
mc_error_t *
restarter_get_method_context(uint_t version, scf_instance_t *inst,
    scf_snapshot_t *snap, const char *mname, const char *cmdline,
    struct method_context **mcpp)
{
        scf_handle_t *h;
        scf_propertygroup_t *methpg = NULL;
        scf_propertygroup_t *instpg = NULL;
        scf_propertygroup_t *pg = NULL;
        scf_property_t *prop = NULL;
        scf_value_t *val = NULL;
        scf_type_t ty;
        uint8_t use_profile;
        int ret = 0;
        int mc_used = 0;
        mc_error_t *err = NULL;
        struct method_context *cip;

        if ((err = malloc(sizeof (mc_error_t))) == NULL)
                return (mc_error_create(NULL, ENOMEM, NULL));

        /* Set the type to zero to track if an error occured. */
```

```
        err->type = 0;

        if (version != RESTARTER_METHOD_CONTEXT_VERSION)
                return (mc_error_create(err, ENOTSUP,
                    "Invalid client version %d. (Expected %d)",
                    version, RESTARTER_METHOD_CONTEXT_VERSION));

        /* Get the handle before we allocate anything. */
        h = scf_instance_handle(inst);
        if (h == NULL)
                return (mc_error_create(err, scf_error(),
                    scf_strerror(scf_error())));

        cip = malloc(sizeof (*cip));
        if (cip == NULL)
                return (mc_error_create(err, ENOMEM, ALLOCFAIL));

        (void) memset(cip, 0, sizeof (*cip));
        cip->uid = (uid_t)-1;
        cip->euid = (uid_t)-1;
        cip->gid = (gid_t)-1;
        cip->egid = (gid_t)-1;

        cip->vbuf_sz = scf_limit(SCF_LIMIT_MAX_VALUE_LENGTH);
        assert(cip->vbuf_sz >= 0);
        cip->vbuf = malloc(cip->vbuf_sz);
        if (cip->vbuf == NULL) {
                free(cip);
                return (mc_error_create(err, ENOMEM, ALLOCFAIL));
        }

        if ((instpg = scf_pg_create(h)) == NULL ||
            (methpg = scf_pg_create(h)) == NULL ||
            (prop = scf_property_create(h)) == NULL ||
            (val = scf_value_create(h)) == NULL) {
                err = mc_error_create(err, scf_error(),
                    "Failed to create repository object: %s",
                    scf_strerror(scf_error()));
                goto out;
        }

        /*
         * The method environment, and the credentials/profile data,
         * may be found either in the pg for the method (methpg),
         * or in the instance/service SCF_PG_METHOD_CONTEXT pg (named
         * instpg below).
         */

        if (scf_instance_get_pg_composed(inst, snap, mname, methpg) !=
            SCF_SUCCESS) {
                err = mc_error_create(err, scf_error(), "Unable to get the "
                    "\"%s\" method, %s", mname, scf_strerror(scf_error()));
                goto out;
        }

        if (scf_instance_get_pg_composed(inst, snap, SCF_PG_METHOD_CONTEXT,
            instpg) != SCF_SUCCESS) {
                if (scf_error() != SCF_ERROR_NOT_FOUND) {
                        err = mc_error_create(err, scf_error(),
                            "Unable to retrieve the \"%s\" property group, %s",
                            SCF_PG_METHOD_CONTEXT, scf_strerror(scf_error()));
                        goto out;
                }
                scf_pg_destroy(instpg);
                instpg = NULL;
        } else {
```

```
		mc_used++;
	}

	ret = get_environment(h, methpg, cip, prop, val);
	if (ret == ENOENT && instpg != NULL) {
		ret = get_environment(h, instpg, cip, prop, val);
	}

	switch (ret) {
	case 0:
		mc_used++;
		break;
	case ENOENT:
		break;
	case ENOMEM:
		err = mc_error_create(err, ret, "Out of memory.");
		goto out;
	case EINVAL:
		err = mc_error_create(err, ret, "Invalid method environment.");
		goto out;
	default:
		err = mc_error_create(err, ret,
		    "Get method environment failed: %s", scf_strerror(ret));
		goto out;
	}

	pg = methpg;

	ret = scf_pg_get_property(pg, SCF_PROPERTY_USE_PROFILE, prop);
	if (ret && scf_error() == SCF_ERROR_NOT_FOUND && instpg != NULL) {
		pg = NULL;
		ret = scf_pg_get_property(instpg, SCF_PROPERTY_USE_PROFILE,
		    prop);
	}

	if (ret) {
		switch (scf_error()) {
		case SCF_ERROR_NOT_FOUND:
			/* No profile context: use default credentials */
			cip->uid = 0;
			cip->gid = 0;
			break;

		case SCF_ERROR_CONNECTION_BROKEN:
			err = mc_error_create(err, SCF_ERROR_CONNECTION_BROKEN,
			    RCBROKEN);
			goto out;

		case SCF_ERROR_DELETED:
			err = mc_error_create(err, SCF_ERROR_NOT_FOUND,
			    "Could not find property group \"%s\"",
			    pg == NULL ? SCF_PG_METHOD_CONTEXT : mname);
			goto out;

		case SCF_ERROR_HANDLE_MISMATCH:
		case SCF_ERROR_INVALID_ARGUMENT:
		case SCF_ERROR_NOT_SET:
		default:
			bad_fail("scf_pg_get_property", scf_error());
		}
	} else {
		if (scf_property_type(prop, &ty) != SCF_SUCCESS) {
			ret = scf_error();
			switch (ret) {
			case SCF_ERROR_CONNECTION_BROKEN:
				err = mc_error_create(err,
```

```
				    SCF_ERROR_CONNECTION_BROKEN, RCBROKEN);
				break;

			case SCF_ERROR_DELETED:
				err = mc_error_create(err,
				    SCF_ERROR_NOT_FOUND,
				    "Could not find property group \"%s\"",
				    pg == NULL ? SCF_PG_METHOD_CONTEXT : mname);
				break;

			case SCF_ERROR_NOT_SET:
			default:
				bad_fail("scf_property_type", ret);
			}

			goto out;
		}

		if (ty != SCF_TYPE_BOOLEAN) {
			err = mc_error_create(err,
			    SCF_ERROR_TYPE_MISMATCH,
			    "\"%s\" property is not boolean in property group "
			    "\"%s\".", SCF_PROPERTY_USE_PROFILE,
			    pg == NULL ? SCF_PG_METHOD_CONTEXT : mname);
			goto out;
		}

		if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
			ret = scf_error();
			switch (ret) {
			case SCF_ERROR_CONNECTION_BROKEN:
				err = mc_error_create(err,
				    SCF_ERROR_CONNECTION_BROKEN, RCBROKEN);
				break;

			case SCF_ERROR_CONSTRAINT_VIOLATED:
				err = mc_error_create(err,
				    SCF_ERROR_CONSTRAINT_VIOLATED,
				    "\"%s\" property has multiple values.",
				    SCF_PROPERTY_USE_PROFILE);
				break;

			case SCF_ERROR_NOT_FOUND:
				err = mc_error_create(err,
				    SCF_ERROR_NOT_FOUND,
				    "\"%s\" property has no values.",
				    SCF_PROPERTY_USE_PROFILE);
				break;
			default:
				bad_fail("scf_property_get_value", ret);
			}

			goto out;
		}

		mc_used++;
		ret = scf_value_get_boolean(val, &use_profile);
		assert(ret == SCF_SUCCESS);

		/* get ids & privileges */
		if (use_profile)
			err = get_profile(pg, instpg, prop, val, cmdline,
			    cip, err);
		else
			err = get_ids(pg, instpg, prop, val, cip, err);
```

```
                if (err->type != 0)
                        goto out;
        }

        /* get working directory */
        if ((methpg != NULL && scf_pg_get_property(methpg,
            SCF_PROPERTY_WORKING_DIRECTORY, prop) == SCF_SUCCESS) ||
            (instpg != NULL && scf_pg_get_property(instpg,
            SCF_PROPERTY_WORKING_DIRECTORY, prop) == SCF_SUCCESS)) {
                if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
                        ret = scf_error();
                        switch (ret) {
                        case SCF_ERROR_CONNECTION_BROKEN:
                                err = mc_error_create(err, ret, RCBROKEN);
                                break;

                        case SCF_ERROR_CONSTRAINT_VIOLATED:
                                err = mc_error_create(err, ret,
                                    "\"%s\" property has multiple values.",
                                    SCF_PROPERTY_WORKING_DIRECTORY);
                                break;

                        case SCF_ERROR_NOT_FOUND:
                                err = mc_error_create(err, ret,
                                    "\"%s\" property has no values.",
                                    SCF_PROPERTY_WORKING_DIRECTORY);
                                break;

                        default:
                                bad_fail("scf_property_get_value", ret);
                        }

                        goto out;
                }

                mc_used++;
                ret = scf_value_get_astring(val, cip->vbuf, cip->vbuf_sz);
                assert(ret != -1);
        } else {
                ret = scf_error();
                switch (ret) {
                case SCF_ERROR_NOT_FOUND:
                        /* okay if missing. */
                        (void) strcpy(cip->vbuf, ":default");
                        break;

                case SCF_ERROR_CONNECTION_BROKEN:
                        err = mc_error_create(err, ret, RCBROKEN);
                        goto out;

                case SCF_ERROR_DELETED:
                        err = mc_error_create(err, ret,
                            "Property group could not be found");
                        goto out;

                case SCF_ERROR_HANDLE_MISMATCH:
                case SCF_ERROR_INVALID_ARGUMENT:
                case SCF_ERROR_NOT_SET:
                default:
                        bad_fail("scf_pg_get_property", ret);
                }
        }

        if (strcmp(cip->vbuf, ":default") == 0 ||
            strcmp(cip->vbuf, ":home") == 0) {
                switch (ret = lookup_pwd(cip)) {
```

```
                case 0:
                        break;

                case ENOMEM:
                        err = mc_error_create(err, ret, "Out of memory.");
                        goto out;

                case ENOENT:
                case EIO:
                case EMFILE:
                case ENFILE:
                        err = mc_error_create(err, ret,
                            "Could not get passwd entry.");
                        goto out;

                default:
                        bad_fail("lookup_pwd", ret);
                }

                cip->working_dir = strdup(cip->pwd.pw_dir);
                if (cip->working_dir == NULL) {
                        err = mc_error_create(err, ENOMEM, ALLOCFAIL);
                        goto out;
                }
        } else {
                cip->working_dir = strdup(cip->vbuf);
                if (cip->working_dir == NULL) {
                        err = mc_error_create(err, ENOMEM, ALLOCFAIL);
                        goto out;
                }
        }

        /* get security flags */
        if ((methpg != NULL && scf_pg_get_property(methpg,
            SCF_PROPERTY_SECFLAGS, prop) == SCF_SUCCESS) ||
            (instpg != NULL && scf_pg_get_property(instpg,
            SCF_PROPERTY_SECFLAGS, prop) == SCF_SUCCESS)) {
                if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
                        ret = scf_error();
                        switch (ret) {
                        case SCF_ERROR_CONNECTION_BROKEN:
                                err = mc_error_create(err, ret, RCBROKEN);
                                break;

                        case SCF_ERROR_CONSTRAINT_VIOLATED:
                                err = mc_error_create(err, ret,
                                    "\"%s\" property has multiple values.",
                                    SCF_PROPERTY_SECFLAGS);
                                break;

                        case SCF_ERROR_NOT_FOUND:
                                err = mc_error_create(err, ret,
                                    "\"%s\" property has no values.",
                                    SCF_PROPERTY_SECFLAGS);
                                break;

                        default:
                                bad_fail("scf_property_get_value", ret);
                        }

                        (void) strlcpy(cip->vbuf, ":default", cip->vbuf_sz);
                } else {
                        ret = scf_value_get_astring(val, cip->vbuf,
                            cip->vbuf_sz);
                        assert(ret != -1);
                }
                mc_used++;
        } else {
                ret = scf_error();
                switch (ret) {
                case SCF_ERROR_NOT_FOUND:
                        /* okay if missing. */
                        (void) strlcpy(cip->vbuf, ":default", cip->vbuf_sz);
                        break;

                case SCF_ERROR_CONNECTION_BROKEN:
                        err = mc_error_create(err, ret, RCBROKEN);
                        goto out;

                case SCF_ERROR_DELETED:
                        err = mc_error_create(err, ret,
                            "Property group could not be found");
                        goto out;

                case SCF_ERROR_HANDLE_MISMATCH:
                case SCF_ERROR_INVALID_ARGUMENT:
                case SCF_ERROR_NOT_SET:
                default:
                        bad_fail("scf_pg_get_property", ret);
                }
        }


        if (scf_default_secflags(h, &cip->def_secflags) != 0) {
                err = mc_error_create(err, EINVAL, "couldn't fetch "
                    "default security-flags");
                goto out;
        }

        if (strcmp(cip->vbuf, ":default") != 0) {
                if (secflags_parse(NULL, cip->vbuf,
        if (strcmp(cip->vbuf, ":default") == 0) {
                if (secflags_parse(&cip->def_secflags.psf_inherit, "default",
                    &cip->secflag_delta) != 0) {
                        err = mc_error_create(err, EINVAL, "couldn't parse "
                            "security flags: %s", cip->vbuf);
                        goto out;
                }
        } else {
                if (secflags_parse(&cip->def_secflags.psf_inherit, cip->vbuf,
                    &cip->secflag_delta) != 0) {
                        err = mc_error_create(err, EINVAL, "couldn't parse "
                            "security flags: %s", cip->vbuf);
                        goto out;
                }
        }

        /* get (optional) corefile pattern */
        if ((methpg != NULL && scf_pg_get_property(methpg,
            SCF_PROPERTY_COREFILE_PATTERN, prop) == SCF_SUCCESS) ||
            (instpg != NULL && scf_pg_get_property(instpg,
            SCF_PROPERTY_COREFILE_PATTERN, prop) == SCF_SUCCESS)) {
                if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
                        ret = scf_error();
                        switch (ret) {
                        case SCF_ERROR_CONNECTION_BROKEN:
                                err = mc_error_create(err, ret, RCBROKEN);
                                break;

                        case SCF_ERROR_CONSTRAINT_VIOLATED:
                                err = mc_error_create(err, ret,
                                    "\"%s\" property has multiple values.",
```

```
                                    SCF_PROPERTY_COREFILE_PATTERN);
                                break;

                        case SCF_ERROR_NOT_FOUND:
                                err = mc_error_create(err, ret,
                                    "\"%s\" property has no values.",
                                    SCF_PROPERTY_COREFILE_PATTERN);
                                break;

                        default:
                                bad_fail("scf_property_get_value", ret);
                        }

                } else {

                        ret = scf_value_get_astring(val, cip->vbuf,
                            cip->vbuf_sz);
                        assert(ret != -1);

                        cip->corefile_pattern = strdup(cip->vbuf);
                        if (cip->corefile_pattern == NULL) {
                                err = mc_error_create(err, ENOMEM, ALLOCFAIL);
                                goto out;
                        }
                }

                mc_used++;
        } else {
                ret = scf_error();
                switch (ret) {
                case SCF_ERROR_NOT_FOUND:
                        /* okay if missing. */
                        break;

                case SCF_ERROR_CONNECTION_BROKEN:
                        err = mc_error_create(err, ret, RCBROKEN);
                        goto out;

                case SCF_ERROR_DELETED:
                        err = mc_error_create(err, ret,
                            "Property group could not be found");
                        goto out;

                case SCF_ERROR_HANDLE_MISMATCH:
                case SCF_ERROR_INVALID_ARGUMENT:
                case SCF_ERROR_NOT_SET:
                default:
                        bad_fail("scf_pg_get_property", ret);
                }
        }

        if (restarter_rm_libs_loadable()) {
                /* get project */
                if ((methpg != NULL && scf_pg_get_property(methpg,
                    SCF_PROPERTY_PROJECT, prop) == SCF_SUCCESS) ||
                    (instpg != NULL && scf_pg_get_property(instpg,
                    SCF_PROPERTY_PROJECT, prop) == SCF_SUCCESS)) {
                        if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
                                ret = scf_error();
                                switch (ret) {
                                case SCF_ERROR_CONNECTION_BROKEN:
                                        err = mc_error_create(err, ret,
                                            RCBROKEN);
                                        break;

                                case SCF_ERROR_CONSTRAINT_VIOLATED:
```

```
					err = mc_error_create(err, ret,
					    "\"%s\" property has multiple "
					    "values.", SCF_PROPERTY_PROJECT);
					break;

				case SCF_ERROR_NOT_FOUND:
					err = mc_error_create(err, ret,
					    "\"%s\" property has no values.",
					    SCF_PROPERTY_PROJECT);
					break;

				default:
					bad_fail("scf_property_get_value", ret);
				}

				(void) strcpy(cip->vbuf, ":default");
			} else {
				ret = scf_value_get_astring(val, cip->vbuf,
				    cip->vbuf_sz);
				assert(ret != -1);
			}

			mc_used++;
		} else {
			(void) strcpy(cip->vbuf, ":default");
		}

		switch (ret = get_projid(cip->vbuf, cip)) {
		case 0:
			break;

		case ENOMEM:
			err = mc_error_create(err, ret, "Out of memory.");
			goto out;

		case ENOENT:
			err = mc_error_create(err, ret,
			    "Missing passwd or project entry for \"%s\".",
			    cip->vbuf);
			goto out;

		case EIO:
			err = mc_error_create(err, ret, "I/O error.");
			goto out;

		case EMFILE:
		case ENFILE:
			err = mc_error_create(err, ret,
			    "Out of file descriptors.");
			goto out;

		case -1:
			err = mc_error_create(err, ret,
			    "Name service switch is misconfigured.");
			goto out;

		case ERANGE:
		case E2BIG:
			err = mc_error_create(err, ret,
			    "Project ID \"%s\" too big.", cip->vbuf);
			goto out;

		case EINVAL:
			err = mc_error_create(err, ret,
			    "Project ID \"%s\" is invalid.", cip->vbuf);
			goto out;
```

```
		default:
			bad_fail("get_projid", ret);
		}

		/* get resource pool */
		if ((methpg != NULL && scf_pg_get_property(methpg,
		    SCF_PROPERTY_RESOURCE_POOL, prop) == SCF_SUCCESS) ||
		    (instpg != NULL && scf_pg_get_property(instpg,
		    SCF_PROPERTY_RESOURCE_POOL, prop) == SCF_SUCCESS)) {
			if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
				ret = scf_error();
				switch (ret) {
				case SCF_ERROR_CONNECTION_BROKEN:
					err = mc_error_create(err, ret,
					    RCBROKEN);
					break;

				case SCF_ERROR_CONSTRAINT_VIOLATED:
					err = mc_error_create(err, ret,
					    "\"%s\" property has multiple "
					    "values.",
					    SCF_PROPERTY_RESOURCE_POOL);
					break;

				case SCF_ERROR_NOT_FOUND:
					err = mc_error_create(err, ret,
					    "\"%s\" property has no "
					    "values.",
					    SCF_PROPERTY_RESOURCE_POOL);
					break;

				default:
					bad_fail("scf_property_get_value", ret);
				}

				(void) strcpy(cip->vbuf, ":default");
			} else {
				ret = scf_value_get_astring(val, cip->vbuf,
				    cip->vbuf_sz);
				assert(ret != -1);
			}

			mc_used++;
		} else {
			ret = scf_error();
			switch (ret) {
			case SCF_ERROR_NOT_FOUND:
				/* okay if missing. */
				(void) strcpy(cip->vbuf, ":default");
				break;

			case SCF_ERROR_CONNECTION_BROKEN:
				err = mc_error_create(err, ret, RCBROKEN);
				goto out;

			case SCF_ERROR_DELETED:
				err = mc_error_create(err, ret,
				    "property group could not be found.");
				goto out;

			case SCF_ERROR_HANDLE_MISMATCH:
			case SCF_ERROR_INVALID_ARGUMENT:
			case SCF_ERROR_NOT_SET:
			default:
				bad_fail("scf_pg_get_property", ret);
```

```
                        }
                }

                if (strcmp(cip->vbuf, ":default") != 0) {
                        cip->resource_pool = strdup(cip->vbuf);
                        if (cip->resource_pool == NULL) {
                                err = mc_error_create(err, ENOMEM, ALLOCFAIL);
                                goto out;
                        }
                }
        }

        /*
         * A method_context was not used for any configurable
         * elements or attributes, so reset and use the simple
         * defaults that provide historic init behavior.
         */
        if (mc_used == 0) {
                free(cip->pwbuf);
                free(cip->vbuf);
                free(cip->working_dir);

                (void) memset(cip, 0, sizeof (*cip));
                cip->uid = 0;
                cip->gid = 0;
                cip->euid = (uid_t)-1;
                cip->egid = (gid_t)-1;

                if (scf_default_secflags(h, &cip->def_secflags) != 0) {
                        err = mc_error_create(err, EINVAL, "couldn’t fetch "
                            "default security-flags");
                        goto out;
                }

                if (secflags_parse(&cip->def_secflags.psf_inherit, "default",
                    &cip->secflag_delta) != 0) {
                        err = mc_error_create(err, EINVAL, "couldn’t parse "
                            "security flags: %s", cip->vbuf);
                        goto out;
                }
        }

        *mcpp = cip;

out:
        (void) scf_value_destroy(val);
        scf_property_destroy(prop);
        scf_pg_destroy(instpg);
        scf_pg_destroy(methpg);

        if (cip->pwbuf != NULL) {
                free(cip->pwbuf);
                cip->pwbuf = NULL;
        }

        free(cip->vbuf);

        if (err->type != 0) {
                restarter_free_method_context(cip);
        } else {
                restarter_mc_error_destroy(err);
                err = NULL;
        }

        return (err);
}
```

```
/*
 * Modify the current process per the given method_context.  On success, returns
 * 0.  Note that the environment is not modified by this function to include the
 * environment variables in cip->env.
 *
 * On failure, sets *fp to NULL or the name of the function which failed,
 * and returns one of the following error codes.  The words in parentheses are
 * the values to which *fp may be set for the error case.
 *   ENOMEM - malloc() failed
 *   EIO - an I/O error occurred (getpwuid_r, chdir)
 *   EMFILE - process is out of file descriptors (getpwuid_r)
 *   ENFILE - system is out of file handles (getpwuid_r)
 *   EINVAL - gid or egid is out of range (setregid)
 *            ngroups is too big (setgroups)
 *            project’s project id is bad (setproject)
 *            uid or euid is out of range (setreuid)
 *            poolname is invalid (pool_set_binding)
 *   EPERM - insufficient privilege (setregid, initgroups, setgroups, setppriv,
 *               setproject, setreuid, settaskid)
 *   ENOENT - uid has a passwd entry but no shadow entry
 *            working_dir does not exist (chdir)
 *            uid has no passwd entry
 *            the pool could not be found (pool_set_binding)
 *   EFAULT - lpriv_set or priv_set has a bad address (setppriv)
 *            working_dir has a bad address (chdir)
 *   EACCES - could not access working_dir (chdir)
 *            in a TASK_FINAL task (setproject, settaskid)
 *            no resource pool accepting default binding exists (setproject)
 *   ELOOP - too many symbolic links in working_dir (chdir)
 *   ENAMETOOLONG - working_dir is too long (chdir)
 *   ENOLINK - working_dir is on an inaccessible remote machine (chdir)
 *   ENOTDIR - working_dir is not a directory (chdir)
 *   ESRCH - uid is not a user of project (setproject)
 *           project is invalid (setproject)
 *           the resource pool specified for project is unknown (setproject)
 *   EBADF - the configuration for the pool is invalid (pool_set_binding)
 *   -1 - core_set_process_path() failed (core_set_process_path)
 *        a resource control assignment failed (setproject)
 *        a system error occurred during pool_set_binding (pool_set_binding)
 */
int
restarter_set_method_context(struct method_context *cip, const char **fp)
{
        pid_t mypid = -1;
        int r, ret;
        secflagdelta_t delta = {0};

        cip->pwbuf = NULL;
        *fp = NULL;

        if (cip->gid != (gid_t)-1) {
                if (setregid(cip->gid,
                    cip->egid != (gid_t)-1 ? cip->egid : cip->gid) != 0) {
                        *fp = "setregid";

                        ret = errno;
                        assert(ret == EINVAL || ret == EPERM);
                        goto out;
                }
        } else {
                if (cip->pwbuf == NULL) {
                        switch (ret = lookup_pwd(cip)) {
                        case 0:
                                break;
                        case ENOMEM:
                        case ENOENT:
                                *fp = NULL;
                                goto out;

                        case EIO:
                        case EMFILE:
                        case ENFILE:
                                *fp = "getpwuid_r";
                                goto out;

                        default:
                                bad_fail("lookup_pwd", ret);
                        }
                }

                if (setregid(cip->pwd.pw_gid,
                    cip->egid != (gid_t)-1 ?
                    cip->egid : cip->pwd.pw_gid) != 0) {
                        *fp = "setregid";

                        ret = errno;
                        assert(ret == EINVAL || ret == EPERM);
                        goto out;
                }
        }

        if (cip->ngroups == -1) {
                if (cip->pwbuf == NULL) {
                        switch (ret = lookup_pwd(cip)) {
                        case 0:
                                break;

                        case ENOMEM:
                        case ENOENT:
                                *fp = NULL;
                                goto out;

                        case EIO:
                        case EMFILE:
                        case ENFILE:
                                *fp = "getpwuid_r";
                                goto out;

                        default:
                                bad_fail("lookup_pwd", ret);
                        }
                }

                /* Ok if cip->gid == -1 */
                if (initgroups(cip->pwd.pw_name, cip->gid) != 0) {
                        *fp = "initgroups";
                        ret = errno;
                        assert(ret == EPERM);
                        goto out;
                }
        } else if (cip->ngroups > 0 &&
            setgroups(cip->ngroups, cip->groups) != 0) {
                *fp = "setgroups";

                ret = errno;
                assert(ret == EINVAL || ret == EPERM);
                goto out;
        }

        if (cip->corefile_pattern != NULL) {
```

```
                mypid = getpid();

                if (core_set_process_path(cip->corefile_pattern,
                    strlen(cip->corefile_pattern) + 1, mypid) != 0) {
                        *fp = "core_set_process_path";
                        ret = -1;
                        goto out;
                }
        }


        delta.psd_ass_active = B_TRUE;
        secflags_copy(&delta.psd_assign, &cip->def_secflags.psf_inherit);
        if (psecflags(P_PID, P_MYID, PSF_INHERIT,
            &cip->def_secflags.ss_default) != 0) {
                *fp = "psecflags (default inherit)";
            &delta) != 0) {
                *fp = "psecflags (inherit defaults)";
                ret = errno;
                goto out;
        }

        if (psecflags(P_PID, P_MYID, PSF_LOWER,
            &cip->def_secflags.ss_lower) != 0) {
                *fp = "psecflags (default lower)";
        if (psecflags(P_PID, P_MYID, PSF_INHERIT,
            &cip->secflag_delta) != 0) {
                *fp = "psecflags (inherit)";
                ret = errno;
                goto out;
        }

        if (psecflags(P_PID, P_MYID, PSF_UPPER,
            &cip->def_secflags.ss_upper) != 0) {
                *fp = "psecflags (default upper)";
        secflags_copy(&delta.psd_assign, &cip->def_secflags.psf_lower);
        if (psecflags(P_PID, P_MYID, PSF_LOWER,
            &delta) != 0) {
                *fp = "psecflags (lower)";
                ret = errno;
                goto out;
        }

        if (psecflags(P_PID, P_MYID, PSF_INHERIT,
            &cip->secflag_delta) != 0) {
                *fp = "psecflags (from manifest)";
        secflags_copy(&delta.psd_assign, &cip->def_secflags.psf_upper);
        if (psecflags(P_PID, P_MYID, PSF_UPPER,
            &delta) != 0) {
                *fp = "psecflags (upper)";
                ret = errno;
                goto out;
        }

        if (restarter_rm_libs_loadable()) {
                if (cip->project == NULL) {
                        if (settaskid(getprojid(), TASK_NORMAL) == -1) {
                                switch (errno) {
                                case EACCES:
                                case EPERM:
                                        *fp = "settaskid";
                                        ret = errno;
                                        goto out;

                                case EINVAL:
                                default:
                                        bad_fail("settaskid", errno);
                                }
                        }
                } else {
                        switch (ret = lookup_pwd(cip)) {
                        case 0:
                                break;

                        case ENOMEM:
                        case ENOENT:
                                *fp = NULL;
                                goto out;

                        case EIO:
                        case EMFILE:
                        case ENFILE:
                                *fp = "getpwuid_r";
                                goto out;

                        default:
                                bad_fail("lookup_pwd", ret);
                        }

                        *fp = "setproject";

                        switch (setproject(cip->project, cip->pwd.pw_name,
                            TASK_NORMAL)) {
                        case 0:
                                break;

                        case SETPROJ_ERR_TASK:
                        case SETPROJ_ERR_POOL:
                                ret = errno;
                                goto out;

                        default:
                                ret = -1;
                                goto out;
                        }
                }

                if (cip->resource_pool != NULL) {
                        if (mypid == -1)
                                mypid = getpid();

                        *fp = "pool_set_binding";

                        if (pool_set_binding(cip->resource_pool, P_PID,
                            mypid) != PO_SUCCESS) {
                                switch (pool_error()) {
                                case POE_INVALID_SEARCH:
                                        ret = ENOENT;
                                        break;

                                case POE_BADPARAM:
                                        ret = EINVAL;
                                        break;

                                case POE_INVALID_CONF:
                                        ret = EBADF;
                                        break;

                                case POE_SYSTEM:
                                        ret = -1;
                                        break;
```

```
                                default:
                                        bad_fail("pool_set_binding",
                                            pool_error());
                                }

                                goto out;
                        }
                }
        }

        /*
         * Now, we have to assume our ID. If the UID is 0, we want it to be
         * privilege-aware, otherwise the limit set gets used instead of E/P.
         * We can do this by setting P as well, which keeps
         * PA status (see priv_can_clear_PA()).
         */

        *fp = "setppriv";

        if (cip->lpriv_set != NULL) {
                if (setppriv(PRIV_SET, PRIV_LIMIT, cip->lpriv_set) != 0) {
                        ret = errno;
                        assert(ret == EFAULT || ret == EPERM);
                        goto out;
                }
        }
        if (cip->priv_set != NULL) {
                if (setppriv(PRIV_SET, PRIV_INHERITABLE, cip->priv_set) != 0) {
                        ret = errno;
                        assert(ret == EFAULT || ret == EPERM);
                        goto out;
                }
        }

        /*
         * If the limit privset is already set, then must be privilege
         * aware.  Otherwise, don’t assume anything, and force privilege
         * aware status.
         */

        if (cip->lpriv_set == NULL && cip->priv_set != NULL) {
                ret = setpflags(PRIV_AWARE, 1);
                assert(ret == 0);
        }

        *fp = "setreuid";
        if (setreuid(cip->uid,
            cip->euid != (uid_t)-1 ? cip->euid : cip->uid) != 0) {
                ret = errno;
                assert(ret == EINVAL || ret == EPERM);
                goto out;
        }

        *fp = "setppriv";
        if (cip->priv_set != NULL) {
                if (setppriv(PRIV_SET, PRIV_PERMITTED, cip->priv_set) != 0) {
                        ret = errno;
                        assert(ret == EFAULT || ret == EPERM);
                        goto out;
                }
        }

        /*
         * The last thing to do is chdir to the specified working directory.
         * This should come after the uid switching as only the user might
         * have access to the specified directory.
```

```
	 */
	if (cip->working_dir != NULL) {
		do {
			r = chdir(cip->working_dir);
		} while (r != 0 && errno == EINTR);
		if (r != 0) {
			*fp = "chdir";
			ret = errno;
			goto out;
		}
	}

	ret = 0;
out:
	free(cip->pwbuf);
	cip->pwbuf = NULL;
	return (ret);
}
```
______***unchanged_portion_omitted_***

***
****
   11594 Wed Sep 21 15:14:33 2016
new/usr/src/lib/librestart/common/librestart.h
smf: switch to a tri-state for process-security properties true=on,false=off,nil
****
______unchanged_portion_omitted_***

```
/*
 * Functions for updating the repository.
 */

/*
 * When setting state to "maintenance", callers of restarter_set_states() can
 * set aux_state to "service_request" to communicate that another service has
 * requested maintenance state for the target service.
 *
 * Callers should use restarter_inst_validate_aux_fmri() to validate the fmri
 * of the requested service and pass "service_request" for aux_state when
 * calling restarter_set_states(). See inetd and startd for examples.
 */
int restarter_set_states(restarter_event_handle_t *, const char *,
    restarter_instance_state_t, restarter_instance_state_t,
    restarter_instance_state_t, restarter_instance_state_t, restarter_error_t,
    restarter_str_t);
int restarter_event_publish_retry(evchan_t *, const char *, const char *,
    const char *, const char *, nvlist_t *, uint32_t);

/*
 * functions for retrieving the state transition reason messages
 */

#define RESTARTER_STRING_VERSION        1

uint32_t restarter_str_version(void);
const char *restarter_get_str_short(restarter_str_t);
const char *restarter_get_str_long(restarter_str_t);

int restarter_store_contract(scf_instance_t *, ctid_t,
    restarter_contract_type_t);
int restarter_remove_contract(scf_instance_t *, ctid_t,
    restarter_contract_type_t);

ssize_t restarter_state_to_string(restarter_instance_state_t, char *, size_t);
restarter_instance_state_t restarter_string_to_state(char *);

#define RESTARTER_METHOD_CONTEXT_VERSION        8

struct method_context {
        /* Stable */
        uid_t           uid, euid;
        gid_t           gid, egid;
        int             ngroups;                /* -1 means use initgroups(). */
        gid_t           groups[NGROUPS_MAX];
        scf_secflags_t  def_secflags;
        psecflags_t     def_secflags;
        secflagdelta_t  secflag_delta;
        priv_set_t      *lpriv_set, *priv_set;
        char            *corefile_pattern;      /* Optional. */
        char            *project;               /* NULL for no change */
        char            *resource_pool;         /* NULL for project default */
        char            *working_dir;           /* NULL for :default */
        char            **env;                  /* NULL for no env */
        size_t          env_sz;                 /* size of env array */

        /* Private */
        char            *vbuf;
```

```
        ssize_t         vbuf_sz;
        struct passwd   pwd;
        char            *pwbuf;
        ssize_t         pwbufsz;
};
```

______unchanged_portion_omitted_

```
************************************************************
   10635 Wed Sep 21 15:14:34 2016
new/usr/src/lib/libscf/common/highlevel.c
smf: switch to a tri-state for process-security properties true=on,false=off,nil
************************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2010 Sun Microsystems, Inc.  All rights reserved.
 * Use is subject to license terms.
 */

/*
 * This file contains high level functions used by multiple utilities.
 */

#include "libscf_impl.h"

#include <assert.h>
#include <libuutil.h>
#include <string.h>
#include <strings.h>
#endif /* ! codereview */
#include <stdlib.h>
#include <sys/systeminfo.h>
#include <sys/uadmin.h>
#include <sys/utsname.h>
#include <sys/secflags.h>

#ifdef  __x86
#include <smbios.h>

/*
 * Check whether the platform is on the fastreboot_blacklist.
 * Return 1 if the platform has been blacklisted, 0 otherwise.
 */
static int
scf_is_fb_blacklisted(void)
{
        smbios_hdl_t *shp;
        smbios_system_t sys;
        smbios_info_t info;

        id_t id;
        int err;
        int i;
```

```
        scf_simple_prop_t *prop = NULL;
        ssize_t numvals;
        char *platform_name;

        int blacklisted = 0;

        /*
         * If there's no SMBIOS, assume it's blacklisted.
         */
        if ((shp = smbios_open(NULL, SMB_VERSION, 0, &err)) == NULL)
                return (1);

        /*
         * If we can't read system info, assume it's blacklisted.
         */
        if ((id = smbios_info_system(shp, &sys)) == SMB_ERR ||
            smbios_info_common(shp, id, &info) == SMB_ERR) {
                blacklisted = 1;
                goto fb_out;
        }

        /*
         * If we can't read the "platforms" property from property group
         * BOOT_CONFIG_PG_FBBLACKLIST, assume no platforms have
         * been blacklisted.
         */
        if ((prop = scf_simple_prop_get(NULL, FMRI_BOOT_CONFIG,
            BOOT_CONFIG_PG_FBBLACKLIST, "platforms")) == NULL)
                goto fb_out;

        numvals = scf_simple_prop_numvalues(prop);

        for (i = 0; i < numvals; i++) {
                platform_name = scf_simple_prop_next_astring(prop);
                if (platform_name == NULL)
                        break;
                if (strcmp(platform_name, info.smbi_product) == 0) {
                        blacklisted = 1;
                        break;
                }
        }

fb_out:
        smbios_close(shp);
        scf_simple_prop_free(prop);

        return (blacklisted);
}

/*
 * Add or get a property group given an FMRI.
 * Return SCF_SUCCESS on success, SCF_FAILED on failure.
 */
static int
scf_fmri_pg_get_or_add(const char *fmri, const char *pgname,
    const char *pgtype, uint32_t pgflags, int add)
{
        scf_handle_t    *handle = NULL;
        scf_instance_t  *inst = NULL;
        int             rc = SCF_FAILED;
        int             error;

        if ((handle = scf_handle_create(SCF_VERSION)) == NULL ||
            scf_handle_bind(handle) != 0 ||
            (inst = scf_instance_create(handle)) == NULL ||
            scf_handle_decode_fmri(handle, fmri, NULL, NULL,
            inst, NULL, NULL, SCF_DECODE_FMRI_EXACT) != SCF_SUCCESS)
                goto scferror;

        if (add) {
                rc = scf_instance_add_pg(inst, pgname, pgtype, pgflags, NULL);
                /*
                 * If the property group already exists, return SCF_SUCCESS.
                 */
                if (rc != SCF_SUCCESS && scf_error() == SCF_ERROR_EXISTS)
                        rc = SCF_SUCCESS;
        } else {
                rc = scf_instance_get_pg(inst, pgname, NULL);
        }

scferror:
        if (rc != SCF_SUCCESS)
                error = scf_error();

        scf_instance_destroy(inst);
        if (handle)
                (void) scf_handle_unbind(handle);
        scf_handle_destroy(handle);

        if (rc != SCF_SUCCESS)
                (void) scf_set_error(error);

        return (rc);
}
#endif  /* __x86 */

/*
 * Get config properties from svc:/system/boot-config:default.
 * It prints errors with uu_warn().
 */
void
scf_get_boot_config(uint8_t *boot_config)
{
        uint64_t ret = 0;

        assert(boot_config);
        *boot_config = 0;

        {
                /*
                 * Property vector for BOOT_CONFIG_PG_PARAMS property group.
                 */
                scf_propvec_t ua_boot_config[] = {
                        { FASTREBOOT_DEFAULT, NULL, SCF_TYPE_BOOLEAN, NULL,
                            UA_FASTREBOOT_DEFAULT },
                        { FASTREBOOT_ONPANIC, NULL, SCF_TYPE_BOOLEAN, NULL,
                            UA_FASTREBOOT_ONPANIC },
                        { NULL }
                };
                scf_propvec_t   *prop;

                for (prop = ua_boot_config; prop->pv_prop != NULL; prop++)
                        prop->pv_ptr = &ret;
                prop = NULL;
                if (scf_read_propvec(FMRI_BOOT_CONFIG, BOOT_CONFIG_PG_PARAMS,
                    B_TRUE, ua_boot_config, &prop) != SCF_FAILED) {

#ifdef  __x86
                        /*
                         * Unset both flags if the platform has been
                         * blacklisted.
                         */
```

```
                        if (scf_is_fb_blacklisted())
                                return;
#endif  /* __x86 */
                        *boot_config = (uint8_t)ret;
                        return;
                }
#if defined(FASTREBOOT_DEBUG)
                if (prop != NULL) {
                        (void) uu_warn("Service %s property ’%s/%s’ "
                            "not found.\n", FMRI_BOOT_CONFIG,
                            BOOT_CONFIG_PG_PARAMS, prop->pv_prop);
                } else {
                        (void) uu_warn("Unable to read service %s "
                            "property ’%s’: %s\n", FMRI_BOOT_CONFIG,
                            BOOT_CONFIG_PG_PARAMS, scf_strerror(scf_error()));
                }
#endif  /* FASTREBOOT_DEBUG */
        }
}

/*
 * Get or set properties in non-persistent "config_ovr" property group
 * in svc:/system/boot-config:default.
 * It prints errors with uu_warn().
 */
/*ARGSUSED*/
static int
scf_getset_boot_config_ovr(int set, uint8_t *boot_config_ovr)
{
        int rc = SCF_SUCCESS;

        assert(boot_config_ovr);

#ifndef __x86
        return (rc);
#else
        {
                /*
                 * Property vector for BOOT_CONFIG_PG_OVR property group.
                 */
                scf_propvec_t ua_boot_config_ovr[] = {
                        { FASTREBOOT_DEFAULT, NULL, SCF_TYPE_BOOLEAN, NULL,
                            UA_FASTREBOOT_DEFAULT },
                        { FASTREBOOT_ONPANIC, NULL, SCF_TYPE_BOOLEAN, NULL,
                            UA_FASTREBOOT_ONPANIC },
                        { NULL }
                };
                scf_propvec_t   *prop;

                rc = scf_fmri_pg_get_or_add(FMRI_BOOT_CONFIG,
                    BOOT_CONFIG_PG_OVR, SCF_GROUP_APPLICATION,
                    SCF_PG_FLAG_NONPERSISTENT, set);

                if (rc != SCF_SUCCESS) {
#if defined(FASTREBOOT_DEBUG)
                        if (set)
                                (void) uu_warn("Unable to add service %s "
                                    "property group ’%s’\n",
                                    FMRI_BOOT_CONFIG, BOOT_CONFIG_PG_OVR);
#endif  /* FASTREBOOT_DEBUG */
                        return (rc);
                }

                for (prop = ua_boot_config_ovr; prop->pv_prop != NULL; prop++)
                        prop->pv_ptr = boot_config_ovr;
                prop = NULL;
```

```
		if (set)
			rc = scf_write_propvec(FMRI_BOOT_CONFIG,
			    BOOT_CONFIG_PG_OVR, ua_boot_config_ovr, &prop);
		else
			rc = scf_read_propvec(FMRI_BOOT_CONFIG,
			    BOOT_CONFIG_PG_OVR, B_FALSE, ua_boot_config_ovr,
			    &prop);

#if defined(FASTREBOOT_DEBUG)
		if (rc != SCF_SUCCESS) {
			if (prop != NULL) {
				(void) uu_warn("Service %s property '%s/%s' "
				    "not found.\n", FMRI_BOOT_CONFIG,
				    BOOT_CONFIG_PG_OVR, prop->pv_prop);
			} else {
				(void) uu_warn("Unable to %s service %s "
				    "property '%s': %s\n", set ? "set" : "get",
				    FMRI_BOOT_CONFIG, BOOT_CONFIG_PG_OVR,
				    scf_strerror(scf_error()));
			}
		}
#endif	/* FASTREBOOT_DEBUG */

		if (set)
			(void) smf_refresh_instance(FMRI_BOOT_CONFIG);

		return (rc);

	}
#endif	/* __x86 */
}

/*
 * Get values of properties in non-persistent "config_ovr" property group.
 */
void
scf_get_boot_config_ovr(uint8_t *boot_config_ovr)
{
	(void) scf_getset_boot_config_ovr(B_FALSE, boot_config_ovr);
}

/*
 * Set value of "config_ovr/fastreboot_default".
 */
int
scf_fastreboot_default_set_transient(boolean_t value)
{
	uint8_t boot_config_ovr = 0;

	if (value == B_TRUE)
		boot_config_ovr = UA_FASTREBOOT_DEFAULT | UA_FASTREBOOT_ONPANIC;

	return (scf_getset_boot_config_ovr(B_TRUE, &boot_config_ovr));
}

/*
 * Check whether Fast Reboot is the default operating mode.
 * Return 0 if
 *   1. the platform is xVM
 * or
 *   2. svc:/system/boot-config:default service doesn't exist,
 * or
 *   3. property "config/fastreboot_default" doesn't exist,
 * or
 *   4. value of property "config/fastreboot_default" is set to "false"
```

```
 *      and "config_ovr/fastreboot_default" is not set to "true",
 * or
 *   5. the platform has been blacklisted.
 * or
 *   6. value of property "config_ovr/fastreboot_default" is set to "false".
 * Return non-zero otherwise.
 */
int
scf_is_fastboot_default(void)
{
	uint8_t boot_config = 0, boot_config_ovr;
	char procbuf[SYS_NMLN];

	/*
	 * If we are on xVM, do not fast reboot by default.
	 */
	if (sysinfo(SI_PLATFORM, procbuf, sizeof (procbuf)) == -1 ||
	    strcmp(procbuf, "i86xpv") == 0)
		return (0);

	/*
	 * Get property values from "config" property group
	 */
	scf_get_boot_config(&boot_config);

	/*
	 * Get property values from non-persistent "config_ovr" property group
	 */
	boot_config_ovr = boot_config;
	scf_get_boot_config_ovr(&boot_config_ovr);

	return (boot_config & boot_config_ovr & UA_FASTREBOOT_DEFAULT);
}

/*
 * Read the default security-flags from system/process-security and return a
 * secflagset_t suitable for psecflags(2)
 *
 * Unfortunately, this symbol must _exist_ in the native build, for the sake
 * of the mapfile, even though we don't ever use it, and it will never work.
 */
struct group_desc {
	secflagdelta_t *delta;
	secflagset_t *set;
	char *fmri;
};

int
scf_default_secflags(scf_handle_t *hndl, scf_secflags_t *flags)
scf_default_secflags(scf_handle_t *hndl, psecflags_t *flags)
{
#if !defined(NATIVE_BUILD)
	scf_property_t *prop;
	scf_value_t *val;
	const char *flagname;
	int flag;
	struct group_desc *g;
	struct group_desc groups[] = {
		{NULL, "svc:/system/process-security/"
		    ":properties/default"},
		{NULL, "svc:/system/process-security/"
		    ":properties/lower"},
		{NULL, "svc:/system/process-security/"
		    ":properties/upper"},
		{NULL, NULL}
	};
```

```
        bzero(flags, sizeof (*flags));
        groups[0].set = &flags->psf_inherit;
        groups[1].set = &flags->psf_lower;
        groups[2].set = &flags->psf_upper;

        groups[0].delta = &flags->ss_default;
        groups[1].delta = &flags->ss_lower;
        groups[2].delta = &flags->ss_upper;
        /* Ensure sane defaults */
        psecflags_default(flags);

        for (g = groups; g->delta != NULL; g++) {
        for (g = groups; g->set != NULL; g++) {
                for (flag = 0; (flagname = secflag_to_str(flag)) != NULL;
                    flag++) {
                        char *pfmri;
                        uint8_t flagval = 0;

                        if ((val = scf_value_create(hndl)) == NULL)
                                return (-1);

                        if ((prop = scf_property_create(hndl)) == NULL) {
                                scf_value_destroy(val);
                                return (-1);
                        }

                        if ((pfmri = uu_msprintf("%s/%s", g->fmri,
                            flagname)) == NULL)
                                uu_die("Allocation failure\n");

                        if (scf_handle_decode_fmri(hndl, pfmri,
                            NULL, NULL, NULL, NULL, prop, NULL) != 0)
                                goto next;

                        if (scf_property_get_value(prop, val) != 0)
                                goto next;

                        (void) scf_value_get_boolean(val, &flagval);

                        if (flagval != 0)
                                secflag_set(&g->delta->psd_add, flag);
                                secflag_set(g->set, flag);
                        else
                                secflag_set(&g->delta->psd_rem, flag);
                                secflag_clear(g->set, flag);

next:
                        uu_free(pfmri);
                        scf_value_destroy(val);
                        scf_property_destroy(prop);
                }
        }

        if (!psecflags_validate(flags))
                return (-1);

        return (0);
#else
        assert(0);
        abort();
#endif /* !NATIVE_BUILD */
}
```

______***unchanged_portion_omitted_***

```
**********************************************************
   34769 Wed Sep 21 15:14:35 2016
new/usr/src/lib/libscf/inc/libscf.h
smf: switch to a tri-state for process-security properties true=on,false=off,nil
**********************************************************
   1 /*
   2  * CDDL HEADER START
   3  *
   4  * The contents of this file are subject to the terms of the
   5  * Common Development and Distribution License (the "License").
   6  * You may not use this file except in compliance with the License.
   7  *
   8  * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
   9  * or http://www.opensolaris.org/os/licensing.
  10  * See the License for the specific language governing permissions
  11  * and limitations under the License.
  12  *
  13  * When distributing Covered Code, include this CDDL HEADER in each
  14  * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
  15  * If applicable, add the following below this CDDL HEADER, with the
  16  * fields enclosed by brackets "[]" replaced with your own identifying
  17  * information: Portions Copyright [yyyy] [name of copyright owner]
  18  *
  19  * CDDL HEADER END
  20  */

  22 /*
  23  * Copyright (c) 2004, 2010, Oracle and/or its affiliates. All rights reserved.
  24  */

  26 #ifndef _LIBSCF_H
  27 #define _LIBSCF_H


  30 #include <stddef.h>
  31 #include <libnvpair.h>

  33 #ifndef NATIVE_BUILD
  34 #include <sys/secflags.h>
  35 #endif  /* NATIVE_BUILD */
  36 #endif /* ! codereview */
  37 #include <sys/types.h>
  31 #include <libnvpair.h>

  39 #ifdef  __cplusplus
  40 extern "C" {
  41 #endif

  43 typedef struct scf_version *scf_version_t;
  44 #define SCF_VERSION     ((scf_version_t)1UL)

  46 /*
  47  * Opaque structures
  48  */
  49 typedef struct scf_handle scf_handle_t;
  50 typedef struct scf_scope scf_scope_t;
  51 typedef struct scf_service scf_service_t;
  52 typedef struct scf_instance scf_instance_t;
  53 typedef struct scf_propertygroup scf_propertygroup_t;
  54 typedef struct scf_property scf_property_t;

  56 typedef struct scf_snapshot scf_snapshot_t;
  57 typedef struct scf_snaplevel scf_snaplevel_t;

  59 typedef struct scf_transaction scf_transaction_t;
  60 typedef struct scf_transaction_entry scf_transaction_entry_t;
```

```
  61 typedef struct scf_value scf_value_t;

  63 typedef struct scf_iter scf_iter_t;

  65 typedef struct scf_pg_tmpl scf_pg_tmpl_t;
  66 typedef struct scf_prop_tmpl scf_prop_tmpl_t;
  67 typedef struct scf_tmpl_errors scf_tmpl_errors_t;

  69 typedef struct scf_simple_app_props scf_simple_app_props_t;
  70 typedef struct scf_simple_prop scf_simple_prop_t;

  72 /*
  73  * Types
  74  */
  75 typedef enum {
  76         SCF_TYPE_INVALID = 0,

  78         SCF_TYPE_BOOLEAN,
  79         SCF_TYPE_COUNT,
  80         SCF_TYPE_INTEGER,
  81         SCF_TYPE_TIME,
  82         SCF_TYPE_ASTRING,
  83         SCF_TYPE_OPAQUE,

  85         SCF_TYPE_USTRING = 100,

  87         SCF_TYPE_URI = 200,
  88         SCF_TYPE_FMRI,

  90         SCF_TYPE_HOST = 300,
  91         SCF_TYPE_HOSTNAME,
  92         SCF_TYPE_NET_ADDR_V4,
  93         SCF_TYPE_NET_ADDR_V6,
  94         SCF_TYPE_NET_ADDR
  95 } scf_type_t;
______unchanged_portion_omitted_

 202 typedef struct scf_tmpl_error scf_tmpl_error_t;

 204 /*
 205  * This unfortunately needs to be public, because consumers of librestart must
 206  * deal with it
 207  */
 208 typedef struct {
 209 #ifndef NATIVE_BUILD
 210         secflagdelta_t ss_default;
 211         secflagdelta_t ss_lower;
 212         secflagdelta_t ss_upper;
 213 #else
 214         /*
 215          * This is never used, but is necessary for bootstrapping.
 216          * Not even the size matters.
 217          */
 218         void *ss_default;
 219         void *ss_lower;
 220         void *ss_upper;
 221 #endif /* NATIVE_BUILD */
 222 } scf_secflags_t;

 224 /*
 225 #endif /* ! codereview */
 226  * scf_tmpl_strerror() human readable flag
 227  */
 228 #define SCF_TMPL_STRERROR_HUMAN 0x1

 230 /*
 * Standard services
 */
#define SCF_SERVICE_CONFIGD     ((const char *) \
                                    "svc:/system/svc/repository:default")
#define SCF_INSTANCE_GLOBAL     ((const char *) \
                                    "svc:/system/svc/global:default")
#define SCF_SERVICE_GLOBAL      ((const char *) \
                                    "svc:/system/svc/global")
#define SCF_SERVICE_STARTD      ((const char *) \
                                    "svc:/system/svc/restarter:default")
#define SCF_INSTANCE_EMI        ((const char *) \
                                    "svc:/system/early-manifest-import:default")
#define SCF_INSTANCE_FS_MINIMAL ((const char *) \
                                    "svc:/system/filesystem/minimal:default")
#define SCF_INSTANCE_MI         ((const char *) \
                                    "svc:/system/manifest-import:default")

/*
 * Major milestones
 */
#define SCF_MILESTONE_SINGLE_USER \
        ((const char *) "svc:/milestone/single-user:default")
#define SCF_MILESTONE_MULTI_USER \
        ((const char *) "svc:/milestone/multi-user:default")
#define SCF_MILESTONE_MULTI_USER_SERVER \
        ((const char *) "svc:/milestone/multi-user-server:default")

/*
 * standard scope names
 */
#define SCF_SCOPE_LOCAL                 ((const char *)"localhost")

/*
 * Property group types
 */
#define SCF_GROUP_APPLICATION           ((const char *)"application")
#define SCF_GROUP_FRAMEWORK             ((const char *)"framework")
#define SCF_GROUP_DEPENDENCY            ((const char *)"dependency")
#define SCF_GROUP_METHOD                ((const char *)"method")
#define SCF_GROUP_TEMPLATE              ((const char *)"template")
#define SCF_GROUP_TEMPLATE_PG_PATTERN   ((const char *)"template_pg_pattern")
#define SCF_GROUP_TEMPLATE_PROP_PATTERN ((const char *)"template_prop_pattern")

/*
 * Dependency types
 */
#define SCF_DEP_REQUIRE_ALL             ((const char *)"require_all")
#define SCF_DEP_REQUIRE_ANY             ((const char *)"require_any")
#define SCF_DEP_EXCLUDE_ALL             ((const char *)"exclude_all")
#define SCF_DEP_OPTIONAL_ALL            ((const char *)"optional_all")

#define SCF_DEP_RESET_ON_ERROR          ((const char *)"error")
#define SCF_DEP_RESET_ON_RESTART        ((const char *)"restart")
#define SCF_DEP_RESET_ON_REFRESH        ((const char *)"refresh")
#define SCF_DEP_RESET_ON_NONE           ((const char *)"none")

/*
 * Standard property group names
 */
#define SCF_PG_GENERAL                  ((const char *)"general")
#define SCF_PG_GENERAL_OVR              ((const char *)"general_ovr")
#define SCF_PG_RESTARTER                ((const char *)"restarter")
#define SCF_PG_RESTARTER_ACTIONS        ((const char *)"restarter_actions")
#define SCF_PG_METHOD_CONTEXT           ((const char *)"method_context")
#define SCF_PG_APP_DEFAULT              ((const char *)"application")
#define SCF_PG_DEPENDENTS               ((const char *)"dependents")
```

```
#define SCF_PG_OPTIONS                  ((const char *)"options")
#define SCF_PG_OPTIONS_OVR              ((const char *)"options_ovr")
#define SCF_PG_STARTD                   ((const char *)"startd")
#define SCF_PG_STARTD_PRIVATE           ((const char *)"svc-startd-private")
#define SCF_PG_DEATHROW                 ((const char *)"deathrow")
#define SCF_PG_MANIFESTFILES            ((const char *)"manifestfiles")

/*
 * Template property group names and prefixes
 */
#define SCF_PG_TM_COMMON_NAME           ((const char *)"tm_common_name")
#define SCF_PG_TM_DESCRIPTION           ((const char *)"tm_description")

#define SCF_PG_TM_MAN_PREFIX            ((const char *)"tm_man_")
#define SCF_PG_TM_DOC_PREFIX            ((const char *)"tm_doc_")

/*
 * Standard property names
 */
#define SCF_PROPERTY_ACTIVE_POSTFIX     ((const char *)"active")
#define SCF_PROPERTY_AUX_STATE          ((const char *)"auxiliary_state")
#define SCF_PROPERTY_AUX_FMRI           ((const char *)"auxiliary_fmri")
#define SCF_PROPERTY_AUX_TTY            ((const char *)"auxiliary_tty")
#define SCF_PROPERTY_CONTRACT           ((const char *)"contract")
#define SCF_PROPERTY_COREFILE_PATTERN   ((const char *)"corefile_pattern")
#define SCF_PROPERTY_DEGRADED           ((const char *)"degraded")
#define SCF_PROPERTY_DEGRADE_IMMEDIATE  ((const char *)"degrade_immediate")
#define SCF_PROPERTY_DODUMP             ((const char *)"do_dump")
#define SCF_PROPERTY_DURATION           ((const char *)"duration")
#define SCF_PROPERTY_ENABLED            ((const char *)"enabled")
#define SCF_PROPERTY_DEATHROW           ((const char *)"deathrow")
#define SCF_PROPERTY_ENTITY_STABILITY   ((const char *)"entity_stability")
#define SCF_PROPERTY_ENTITIES           ((const char *)"entities")
#define SCF_PROPERTY_EXEC               ((const char *)"exec")
#define SCF_PROPERTY_GROUP              ((const char *)"group")
#define SCF_PROPERTY_GROUPING           ((const char *)"grouping")
#define SCF_PROPERTY_IGNORE             ((const char *)"ignore_error")
#define SCF_PROPERTY_INTERNAL_SEPARATORS ((const char *)"internal_separators")
#define SCF_PROPERTY_LIMIT_PRIVILEGES   ((const char *)"limit_privileges")
#define SCF_PROPERTY_MAINT_OFF          ((const char *)"maint_off")
#define SCF_PROPERTY_MAINT_ON           ((const char *)"maint_on")
#define SCF_PROPERTY_MAINT_ON_IMMEDIATE ((const char *)"maint_on_immediate")
#define SCF_PROPERTY_MAINT_ON_IMMTEMP   ((const char *)"maint_on_immtemp")
#define SCF_PROPERTY_MAINT_ON_TEMPORARY ((const char *)"maint_on_temporary")
#define SCF_PROPERTY_METHOD_PID         ((const char *)"method_pid")
#define SCF_PROPERTY_MILESTONE          ((const char *)"milestone")
#define SCF_PROPERTY_NEED_SESSION       ((const char *)"need_session")
#define SCF_PROPERTY_NEXT_STATE         ((const char *)"next_state")
#define SCF_PROPERTY_PACKAGE            ((const char *)"package")
#define SCF_PROPERTY_PRIVILEGES         ((const char *)"privileges")
#define SCF_PROPERTY_PROFILE            ((const char *)"profile")
#define SCF_PROPERTY_PROJECT            ((const char *)"project")
#define SCF_PROPERTY_REFRESH            ((const char *)"refresh")
#define SCF_PROPERTY_RESOURCE_POOL      ((const char *)"resource_pool")
#define SCF_PROPERTY_ENVIRONMENT        ((const char *)"environment")
#define SCF_PROPERTY_RESTART            ((const char *)"restart")
#define SCF_PROPERTY_RESTARTER          ((const char *)"restarter")
#define SCF_PROPERTY_RESTART_INTERVAL   ((const char *)"restart_interval")
#define SCF_PROPERTY_RESTART_ON         ((const char *)"restart_on")
#define SCF_PROPERTY_RESTORE            ((const char *)"restore")
#define SCF_PROPERTY_SECFLAGS           ((const char *)"security_flags")
#define SCF_PROPERTY_SINGLE_INSTANCE    ((const char *)"single_instance")
#define SCF_PROPERTY_START_METHOD_TIMESTAMP     \
        ((const char *)"start_method_timestamp")
#define SCF_PROPERTY_START_METHOD_WAITSTATUS    \
        ((const char *)"start_method_waitstatus")
```

```c
#define SCF_PROPERTY_START_PID          ((const char *)"start_pid")
#define SCF_PROPERTY_STATE              ((const char *)"state")
#define SCF_PROPERTY_STABILITY          ((const char *)"stability")
#define SCF_PROPERTY_STATE_TIMESTAMP    ((const char *)"state_timestamp")
#define SCF_PROPERTY_SUPP_GROUPS        ((const char *)"supp_groups")
#define SCF_PROPERTY_TIMEOUT            ((const char *)"timeout_seconds")
#define SCF_PROPERTY_TIMEOUT_RETRY      ((const char *)"timeout_retry")
#define SCF_PROPERTY_TRANSIENT_CONTRACT ((const char *)"transient_contract")
#define SCF_PROPERTY_TYPE               ((const char *)"type")
#define SCF_PROPERTY_USE_PROFILE        ((const char *)"use_profile")
#define SCF_PROPERTY_USER               ((const char *)"user")
#define SCF_PROPERTY_UTMPX_PREFIX       ((const char *)"utmpx_prefix")
#define SCF_PROPERTY_WORKING_DIRECTORY  ((const char *)"working_directory")

/*
 * Template property names
 */
#define SCF_PROPERTY_TM_CARDINALITY_MIN ((const char *)"cardinality_min")
#define SCF_PROPERTY_TM_CARDINALITY_MAX ((const char *)"cardinality_max")
#define SCF_PROPERTY_TM_CHOICES_INCLUDE_VALUES ((const char *) \
                                            "choices_include_values")
#define SCF_PROPERTY_TM_CHOICES_NAME    ((const char *)"choices_name")
#define SCF_PROPERTY_TM_CHOICES_RANGE   ((const char *)"choices_range")
#define SCF_PROPERTY_TM_CONSTRAINT_NAME ((const char *)"constraint_name")
#define SCF_PROPERTY_TM_CONSTRAINT_RANGE ((const char *)"constraint_range")
#define SCF_PROPERTY_TM_MANPATH         ((const char *)"manpath")
#define SCF_PROPERTY_TM_NAME            ((const char *)"name")
#define SCF_PROPERTY_TM_PG_PATTERN      ((const char *)"pg_pattern")
#define SCF_PROPERTY_TM_REQUIRED        ((const char *)"required")
#define SCF_PROPERTY_TM_SECTION         ((const char *)"section")
#define SCF_PROPERTY_TM_TARGET          ((const char *)"target")
#define SCF_PROPERTY_TM_TITLE           ((const char *)"title")
#define SCF_PROPERTY_TM_TYPE            ((const char *)"type")
#define SCF_PROPERTY_TM_URI             ((const char *)"uri")
#define SCF_PROPERTY_TM_VALUE_PREFIX    ((const char *)"value_")
#define SCF_PROPERTY_TM_VALUES_NAME     ((const char *)"values_name")
#define SCF_PROPERTY_TM_VISIBILITY      ((const char *)"visibility")
#define SCF_PROPERTY_TM_COMMON_NAME_PREFIX      ((const char *)"common_name_")
#define SCF_PROPERTY_TM_DESCRIPTION_PREFIX      ((const char *)"description_")
#define SCF_PROPERTY_TM_UNITS_PREFIX            ((const char *)"units_")

/*
 * Templates wildcard string
 */
#define SCF_TMPL_WILDCARD       ((const char *)"*")

/*
 * Strings used by restarters for state and next_state properties.
 * MAX_SCF_STATE_STRING holds the max length of a state string, including the
 * terminating null.
 */

#define MAX_SCF_STATE_STRING_SZ         14

#define SCF_STATE_STRING_NONE           ((const char *)"none")
#define SCF_STATE_STRING_UNINIT         ((const char *)"uninitialized")
#define SCF_STATE_STRING_MAINT          ((const char *)"maintenance")
#define SCF_STATE_STRING_OFFLINE        ((const char *)"offline")
#define SCF_STATE_STRING_DISABLED       ((const char *)"disabled")
#define SCF_STATE_STRING_ONLINE         ((const char *)"online")
#define SCF_STATE_STRING_DEGRADED       ((const char *)"degraded")
#define SCF_STATE_STRING_LEGACY         ((const char *)"legacy_run")

#define SCF_STATE_UNINIT                0x00000001
#define SCF_STATE_MAINT                 0x00000002
#define SCF_STATE_OFFLINE               0x00000004
```

```c
#define SCF_STATE_DISABLED              0x00000008
#define SCF_STATE_ONLINE                0x00000010
#define SCF_STATE_DEGRADED              0x00000020
#define SCF_STATE_ALL                   0x0000003F

/*
 * software fma svc-transition class
 */
#define SCF_NOTIFY_PARAMS_VERSION       0X0
#define SCF_NOTIFY_NAME_FMRI            ((const char *)"fmri")
#define SCF_NOTIFY_NAME_VERSION         ((const char *)"version")
#define SCF_NOTIFY_NAME_TSET            ((const char *)"tset")
#define SCF_NOTIFY_PG_POSTFIX           ((const char *)"fmnotify")
#define SCF_NOTIFY_PARAMS               ((const char *)"notify-params")
#define SCF_NOTIFY_PARAMS_INST \
        ((const char *)"svc:/system/fm/notify-params:default")
#define SCF_SVC_TRANSITION_CLASS \
        ((const char *)"ireport.os.smf.state-transition")
#define SCF_NOTIFY_PARAMS_PG_TYPE       ((const char *)"notify_params")

/*
 * Useful transition macros
 */
#define SCF_TRANS_SHIFT_INITIAL_STATE(s)        ((s) << 16)
#define SCF_TRANSITION_ALL \
        (SCF_TRANS_SHIFT_INITIAL_STATE(SCF_STATE_ALL) | SCF_STATE_ALL)
#define SCF_TRANS(f, t) (SCF_TRANS_SHIFT_INITIAL_STATE(f) | (t))
#define SCF_TRANS_VALID(t)      (!((t) & ~SCF_TRANSITION_ALL))
#define SCF_TRANS_INITIAL_STATE(t)      ((t) >> 16 & SCF_STATE_ALL)
#define SCF_TRANS_FINAL_STATE(t)        ((t) & SCF_STATE_ALL)

/*
 * Prefixes for states in state transition notification
 */
#define SCF_STN_PREFIX_FROM             ((const char *)"from-")
#define SCF_STN_PREFIX_TO               ((const char *)"to-")

#define SCF_PG_FLAG_NONPERSISTENT       0x1

#define SCF_TRACE_LIBRARY               0x1
#define SCF_TRACE_DAEMON                0x2

#define SMF_IMMEDIATE                   0x1
#define SMF_TEMPORARY                   0x2
#define SMF_AT_NEXT_BOOT                0x4

scf_error_t scf_error(void);
const char *scf_strerror(scf_error_t);

ssize_t scf_limit(uint32_t code);
#define SCF_LIMIT_MAX_NAME_LENGTH       -2000U
#define SCF_LIMIT_MAX_VALUE_LENGTH      -2001U
#define SCF_LIMIT_MAX_PG_TYPE_LENGTH    -2002U
#define SCF_LIMIT_MAX_FMRI_LENGTH       -2003U

scf_handle_t *scf_handle_create(scf_version_t);

int scf_handle_decorate(scf_handle_t *, const char *, scf_value_t *);
#define SCF_DECORATE_CLEAR      ((scf_value_t *)0)

int scf_handle_bind(scf_handle_t *);
int scf_handle_unbind(scf_handle_t *);
void scf_handle_destroy(scf_handle_t *);

int scf_type_base_type(scf_type_t type, scf_type_t *out);
const char *scf_type_to_string(scf_type_t);
```

```
scf_type_t scf_string_to_type(const char *);

/* values */
scf_value_t *scf_value_create(scf_handle_t *);
scf_handle_t *scf_value_handle(const scf_value_t *);
void scf_value_destroy(scf_value_t *);

scf_type_t scf_value_base_type(const scf_value_t *);
scf_type_t scf_value_type(const scf_value_t *);
int scf_value_is_type(const scf_value_t *, scf_type_t);

void scf_value_reset(scf_value_t *);

int scf_value_get_boolean(const scf_value_t *, uint8_t *);
int scf_value_get_count(const scf_value_t *, uint64_t *);
int scf_value_get_integer(const scf_value_t *, int64_t *);
int scf_value_get_time(const scf_value_t *, int64_t *, int32_t *);
ssize_t scf_value_get_astring(const scf_value_t *, char *, size_t);
ssize_t scf_value_get_ustring(const scf_value_t *, char *, size_t);
ssize_t scf_value_get_opaque(const scf_value_t *, void *, size_t);

void scf_value_set_boolean(scf_value_t *, uint8_t);
void scf_value_set_count(scf_value_t *, uint64_t);
void scf_value_set_integer(scf_value_t *, int64_t);
int scf_value_set_time(scf_value_t *, int64_t, int32_t);
int scf_value_set_astring(scf_value_t *, const char *);
int scf_value_set_ustring(scf_value_t *, const char *);
int scf_value_set_opaque(scf_value_t *, const void *, size_t);

ssize_t scf_value_get_as_string(const scf_value_t *, char *, size_t);
ssize_t scf_value_get_as_string_typed(const scf_value_t *, scf_type_t,
    char *, size_t);
int scf_value_set_from_string(scf_value_t *, scf_type_t, const char *);

scf_iter_t *scf_iter_create(scf_handle_t *);
scf_handle_t *scf_iter_handle(const scf_iter_t *);
void scf_iter_reset(scf_iter_t *);
void scf_iter_destroy(scf_iter_t *);

int scf_iter_handle_scopes(scf_iter_t *, const scf_handle_t *);
int scf_iter_scope_services(scf_iter_t *, const scf_scope_t *);
int scf_iter_service_instances(scf_iter_t *, const scf_service_t *);
int scf_iter_service_pgs(scf_iter_t *, const scf_service_t *);
int scf_iter_instance_pgs(scf_iter_t *, const scf_instance_t *);
int scf_iter_instance_pgs_composed(scf_iter_t *, const scf_instance_t *,
    const scf_snapshot_t *);
int scf_iter_service_pgs_typed(scf_iter_t *, const scf_service_t *,
    const char *);
int scf_iter_instance_pgs_typed(scf_iter_t *, const scf_instance_t *,
    const char *);
int scf_iter_instance_pgs_typed_composed(scf_iter_t *, const scf_instance_t *,
    const scf_snapshot_t *, const char *);
int scf_iter_snaplevel_pgs(scf_iter_t *, const scf_snaplevel_t *);
int scf_iter_snaplevel_pgs_typed(scf_iter_t *, const scf_snaplevel_t *,
    const char *);
int scf_iter_instance_snapshots(scf_iter_t *, const scf_instance_t *);
int scf_iter_pg_properties(scf_iter_t *, const scf_propertygroup_t *);
int scf_iter_property_values(scf_iter_t *, const scf_property_t *);

int scf_iter_next_scope(scf_iter_t *, scf_scope_t *);
int scf_iter_next_service(scf_iter_t *, scf_service_t *);
int scf_iter_next_instance(scf_iter_t *, scf_instance_t *);
int scf_iter_next_pg(scf_iter_t *, scf_propertygroup_t *);
int scf_iter_next_property(scf_iter_t *, scf_property_t *);
int scf_iter_next_snapshot(scf_iter_t *, scf_snapshot_t *);
int scf_iter_next_value(scf_iter_t *, scf_value_t *);
```

```
scf_scope_t *scf_scope_create(scf_handle_t *);
scf_handle_t *scf_scope_handle(const scf_scope_t *);

/* XXX eventually remove this */
#define scf_handle_get_local_scope(h, s) \
        scf_handle_get_scope((h), SCF_SCOPE_LOCAL, (s))

int scf_handle_get_scope(scf_handle_t *, const char *, scf_scope_t *);
void scf_scope_destroy(scf_scope_t *);
ssize_t scf_scope_get_name(const scf_scope_t *, char *, size_t);

ssize_t scf_scope_to_fmri(const scf_scope_t *, char *, size_t);

scf_service_t *scf_service_create(scf_handle_t *);
scf_handle_t *scf_service_handle(const scf_service_t *);
void scf_service_destroy(scf_service_t *);
int scf_scope_get_parent(const scf_scope_t *, scf_scope_t *);
ssize_t scf_service_get_name(const scf_service_t *, char *, size_t);
ssize_t scf_service_to_fmri(const scf_service_t *, char *, size_t);
int scf_service_get_parent(const scf_service_t *, scf_scope_t *);
int scf_scope_get_service(const scf_scope_t *, const char *, scf_service_t *);
int scf_scope_add_service(const scf_scope_t *, const char *, scf_service_t *);
int scf_service_delete(scf_service_t *);

scf_instance_t *scf_instance_create(scf_handle_t *);
scf_handle_t *scf_instance_handle(const scf_instance_t *);
void scf_instance_destroy(scf_instance_t *);
ssize_t scf_instance_get_name(const scf_instance_t *, char *, size_t);
ssize_t scf_instance_to_fmri(const scf_instance_t *, char *, size_t);
int scf_service_get_instance(const scf_service_t *, const char *,
    scf_instance_t *);
int scf_service_add_instance(const scf_service_t *, const char *,
    scf_instance_t *);
int scf_instance_delete(scf_instance_t *);

scf_snapshot_t *scf_snapshot_create(scf_handle_t *);
scf_handle_t *scf_snapshot_handle(const scf_snapshot_t *);
void scf_snapshot_destroy(scf_snapshot_t *);
ssize_t scf_snapshot_get_name(const scf_snapshot_t *, char *, size_t);
int scf_snapshot_get_parent(const scf_snapshot_t *, scf_instance_t *);
int scf_instance_get_snapshot(const scf_instance_t *, const char *,
    scf_snapshot_t *);
int scf_snapshot_update(scf_snapshot_t *);

scf_snaplevel_t *scf_snaplevel_create(scf_handle_t *);
scf_handle_t *scf_snaplevel_handle(const scf_snaplevel_t *);
void scf_snaplevel_destroy(scf_snaplevel_t *);
int scf_snaplevel_get_parent(const scf_snaplevel_t *, scf_snapshot_t *);
ssize_t scf_snaplevel_get_scope_name(const scf_snaplevel_t *, char *, size_t);
ssize_t scf_snaplevel_get_service_name(const scf_snaplevel_t *, char *, size_t);
ssize_t scf_snaplevel_get_instance_name(const scf_snaplevel_t *, char *,
    size_t);
int scf_snaplevel_get_pg(const scf_snaplevel_t *, const char *,
    scf_propertygroup_t *pg);
int scf_snapshot_get_base_snaplevel(const scf_snapshot_t *, scf_snaplevel_t *);
int scf_snaplevel_get_next_snaplevel(const scf_snaplevel_t *,
    scf_snaplevel_t *);

scf_propertygroup_t *scf_pg_create(scf_handle_t *);
scf_handle_t *scf_pg_handle(const scf_propertygroup_t *);
void scf_pg_destroy(scf_propertygroup_t *);
ssize_t scf_pg_to_fmri(const scf_propertygroup_t *,  char *, size_t);
ssize_t scf_pg_get_name(const scf_propertygroup_t *, char *, size_t);
ssize_t scf_pg_get_type(const scf_propertygroup_t *, char *, size_t);
int scf_pg_get_flags(const scf_propertygroup_t *, uint32_t *);
int scf_pg_get_parent_service(const scf_propertygroup_t *, scf_service_t *);
int scf_pg_get_parent_instance(const scf_propertygroup_t *, scf_instance_t *);
int scf_pg_get_parent_snaplevel(const scf_propertygroup_t *, scf_snaplevel_t *);
int scf_service_get_pg(const scf_service_t *, const char *,
    scf_propertygroup_t *);
int scf_instance_get_pg(const scf_instance_t *, const char *,
    scf_propertygroup_t *);
int scf_instance_get_pg_composed(const scf_instance_t *, const scf_snapshot_t *,
    const char *, scf_propertygroup_t *);
int scf_service_add_pg(const scf_service_t *,  const char *, const char *,
    uint32_t, scf_propertygroup_t *);
int scf_instance_add_pg(const scf_instance_t *,  const char *, const char *,
    uint32_t, scf_propertygroup_t *);
int scf_pg_delete(scf_propertygroup_t *);

int scf_pg_get_underlying_pg(const scf_propertygroup_t *,
    scf_propertygroup_t *);
int scf_instance_get_parent(const scf_instance_t *, scf_service_t *);

int scf_pg_update(scf_propertygroup_t *);

scf_property_t *scf_property_create(scf_handle_t *);
scf_handle_t *scf_property_handle(const scf_property_t *);
void scf_property_destroy(scf_property_t *);
int scf_property_is_type(const scf_property_t *, scf_type_t);
int scf_property_type(const scf_property_t *, scf_type_t *);
ssize_t scf_property_get_name(const scf_property_t *, char *, size_t);
int scf_property_get_value(const scf_property_t *, scf_value_t *);
ssize_t scf_property_to_fmri(const scf_property_t *, char *, size_t);
int scf_pg_get_property(const scf_propertygroup_t *,  const char *,
    scf_property_t *);

scf_transaction_t *scf_transaction_create(scf_handle_t *);
scf_handle_t *scf_transaction_handle(const scf_transaction_t *);
int scf_transaction_start(scf_transaction_t *, scf_propertygroup_t *);
void scf_transaction_destroy(scf_transaction_t *);
void scf_transaction_destroy_children(scf_transaction_t *);

void scf_transaction_reset(scf_transaction_t *);
void scf_transaction_reset_all(scf_transaction_t *);

int scf_transaction_commit(scf_transaction_t *);

scf_transaction_entry_t *scf_entry_create(scf_handle_t *);
scf_handle_t *scf_entry_handle(const scf_transaction_entry_t *);
void scf_entry_reset(scf_transaction_entry_t *);
void scf_entry_destroy(scf_transaction_entry_t *);
void scf_entry_destroy_children(scf_transaction_entry_t *);

int scf_transaction_property_change(scf_transaction_t *,
    scf_transaction_entry_t *, const char *, scf_type_t);
int scf_transaction_property_delete(scf_transaction_t *,
    scf_transaction_entry_t *, const char *);
int scf_transaction_property_new(scf_transaction_t *,
    scf_transaction_entry_t *, const char *, scf_type_t);
int scf_transaction_property_change_type(scf_transaction_t *,
    scf_transaction_entry_t *, const char *, scf_type_t);

int scf_entry_add_value(scf_transaction_entry_t *, scf_value_t *);

int scf_handle_decode_fmri(scf_handle_t *, const char *, scf_scope_t *,
    scf_service_t *, scf_instance_t *, scf_propertygroup_t *, scf_property_t *,
    int);
#define SCF_DECODE_FMRI_EXACT                   0x00000001
#define SCF_DECODE_FMRI_TRUNCATE                0x00000002
#define SCF_DECODE_FMRI_REQUIRE_INSTANCE        0x00000004
```

```
#define SCF_DECODE_FMRI_REQUIRE_NO_INSTANCE     0x00000008

ssize_t scf_myname(scf_handle_t *, char *, size_t);

/*
 * Property group template interfaces.
 */
scf_pg_tmpl_t *scf_tmpl_pg_create(scf_handle_t *);
void scf_tmpl_pg_destroy(scf_pg_tmpl_t *);
void scf_tmpl_pg_reset(scf_pg_tmpl_t *);
int scf_tmpl_get_by_pg(scf_propertygroup_t *, scf_pg_tmpl_t *, int);
int scf_tmpl_get_by_pg_name(const char *, const char *,
    const char *, const char *, scf_pg_tmpl_t *, int);
int scf_tmpl_iter_pgs(scf_pg_tmpl_t *, const char *, const char *,
    const char *, int);
#define SCF_PG_TMPL_FLAG_REQUIRED       0x1
#define SCF_PG_TMPL_FLAG_EXACT          0x2
#define SCF_PG_TMPL_FLAG_CURRENT        0x4

ssize_t scf_tmpl_pg_name(const scf_pg_tmpl_t *, char **);
ssize_t scf_tmpl_pg_common_name(const scf_pg_tmpl_t *, const char *, char **);
ssize_t scf_tmpl_pg_description(const scf_pg_tmpl_t *, const char *, char **);
ssize_t scf_tmpl_pg_type(const scf_pg_tmpl_t *, char **);

ssize_t scf_tmpl_pg_target(const scf_pg_tmpl_t *, char **);
#define SCF_TM_TARGET_ALL               ((const char *)"all")
#define SCF_TM_TARGET_DELEGATE          ((const char *)"delegate")
#define SCF_TM_TARGET_INSTANCE          ((const char *)"instance")
#define SCF_TM_TARGET_THIS              ((const char *)"this")

int scf_tmpl_pg_required(const scf_pg_tmpl_t *, uint8_t *);

/*
 * Property template interfaces.
 */
scf_prop_tmpl_t *scf_tmpl_prop_create(scf_handle_t *);
void scf_tmpl_prop_destroy(scf_prop_tmpl_t *);
void scf_tmpl_prop_reset(scf_prop_tmpl_t *);
int scf_tmpl_get_by_prop(scf_pg_tmpl_t *, const char *,
    scf_prop_tmpl_t *, int);
int scf_tmpl_iter_props(scf_pg_tmpl_t *, scf_prop_tmpl_t *, int);
#define SCF_PROP_TMPL_FLAG_REQUIRED     0x1

ssize_t scf_tmpl_prop_name(const scf_prop_tmpl_t *, char **);
int scf_tmpl_prop_type(const scf_prop_tmpl_t *, scf_type_t *);
int scf_tmpl_prop_required(const scf_prop_tmpl_t *, uint8_t *);
ssize_t scf_tmpl_prop_common_name(const scf_prop_tmpl_t *, const char *,
    char **);
ssize_t scf_tmpl_prop_description(const scf_prop_tmpl_t *, const char *,
    char **);
ssize_t scf_tmpl_prop_units(const scf_prop_tmpl_t *, const char *, char **);
int scf_tmpl_prop_cardinality(const scf_prop_tmpl_t *prop, uint64_t *,
    uint64_t *);
int scf_tmpl_prop_internal_seps(const scf_prop_tmpl_t *, scf_values_t *);

int scf_tmpl_prop_visibility(const scf_prop_tmpl_t *, uint8_t *);
#define SCF_TMPL_VISIBILITY_HIDDEN              1
#define SCF_TMPL_VISIBILITY_READONLY            2
#define SCF_TMPL_VISIBILITY_READWRITE           3

const char *scf_tmpl_visibility_to_string(uint8_t);
#define SCF_TM_VISIBILITY_HIDDEN        ((const char *)"hidden")
#define SCF_TM_VISIBILITY_READONLY      ((const char *)"readonly")
#define SCF_TM_VISIBILITY_READWRITE     ((const char *)"readwrite")

int scf_tmpl_value_name_constraints(const scf_prop_tmpl_t *prop,
    scf_values_t *vals);
void scf_count_ranges_destroy(scf_count_ranges_t *);
void scf_int_ranges_destroy(scf_int_ranges_t *);
int scf_tmpl_value_count_range_constraints(const scf_prop_tmpl_t *,
    scf_count_ranges_t *);
int scf_tmpl_value_int_range_constraints(const scf_prop_tmpl_t *,
    scf_int_ranges_t *);
int scf_tmpl_value_count_range_choices(const scf_prop_tmpl_t *,
    scf_count_ranges_t *);
int scf_tmpl_value_int_range_choices(const scf_prop_tmpl_t *,
    scf_int_ranges_t *);
int scf_tmpl_value_name_choices(const scf_prop_tmpl_t *prop,
    scf_values_t *vals);

void scf_values_destroy(scf_values_t *);

ssize_t scf_tmpl_value_common_name(const scf_prop_tmpl_t *, const char *,
    const char *, char **);
ssize_t scf_tmpl_value_description(const scf_prop_tmpl_t *, const char *,
    const char *, char **);

int scf_tmpl_value_in_constraint(const scf_prop_tmpl_t *pt, scf_value_t *value,
    scf_tmpl_errors_t **errs);

/*
 * Template validation interfaces
 */
int scf_tmpl_validate_fmri(scf_handle_t *, const char *,
    const char *, scf_tmpl_errors_t **, int);
#define SCF_TMPL_VALIDATE_FLAG_CURRENT  0x1

void scf_tmpl_errors_destroy(scf_tmpl_errors_t *errs);
scf_tmpl_error_t *scf_tmpl_next_error(scf_tmpl_errors_t *);
void scf_tmpl_reset_errors(scf_tmpl_errors_t *errs);
int scf_tmpl_strerror(scf_tmpl_error_t *err, char *s, size_t n, int flag);
int scf_tmpl_error_source_fmri(const scf_tmpl_error_t *, char **);
int scf_tmpl_error_type(const scf_tmpl_error_t *, scf_tmpl_error_type_t *);
int scf_tmpl_error_pg_tmpl(const scf_tmpl_error_t *, char **, char **);
int scf_tmpl_error_pg(const scf_tmpl_error_t *, char **, char **);
int scf_tmpl_error_prop_tmpl(const scf_tmpl_error_t *, char **, char **);
int scf_tmpl_error_prop(const scf_tmpl_error_t *, char **, char **);
int scf_tmpl_error_value(const scf_tmpl_error_t *, char **);

/*
 * Simplified calls
 */
int smf_enable_instance(const char *, int);
int smf_disable_instance(const char *, int);
int smf_refresh_instance(const char *);
int smf_restart_instance(const char *);
int smf_maintain_instance(const char *, int);
int smf_degrade_instance(const char *, int);
int smf_restore_instance(const char *);
char *smf_get_state(const char *);

int scf_simple_walk_instances(uint_t, void *,
    int (*inst_callback)(scf_handle_t *, scf_instance_t *, void *));

scf_simple_prop_t *scf_simple_prop_get(scf_handle_t *, const char *,
    const char *, const char *);
void scf_simple_prop_free(scf_simple_prop_t *);
scf_simple_app_props_t *scf_simple_app_props_get(scf_handle_t *, const char *);
void scf_simple_app_props_free(scf_simple_app_props_t *);
const scf_simple_prop_t *scf_simple_app_props_next(
    const scf_simple_app_props_t *, scf_simple_prop_t *);
const scf_simple_prop_t *scf_simple_app_props_search(
```

```
    const scf_simple_app_props_t *, const char *, const char *);
ssize_t scf_simple_prop_numvalues(const scf_simple_prop_t *);
scf_type_t scf_simple_prop_type(const scf_simple_prop_t *);
char *scf_simple_prop_name(const scf_simple_prop_t *);
char *scf_simple_prop_pgname(const scf_simple_prop_t *);
uint8_t *scf_simple_prop_next_boolean(scf_simple_prop_t *);
uint64_t *scf_simple_prop_next_count(scf_simple_prop_t *);
int64_t *scf_simple_prop_next_integer(scf_simple_prop_t *);
int64_t *scf_simple_prop_next_time(scf_simple_prop_t *, int32_t *);
char *scf_simple_prop_next_astring(scf_simple_prop_t *);
char *scf_simple_prop_next_ustring(scf_simple_prop_t *);
void *scf_simple_prop_next_opaque(scf_simple_prop_t *, size_t *);
void scf_simple_prop_next_reset(scf_simple_prop_t *);

/*
 * smf_state_from_string()
 * return SCF_STATE_* value for the input
 * -1 on error. String "all" maps to SCF_STATE_ALL macro
 */
int32_t smf_state_from_string(const char *);

/*
 * smf_state_to_string()
 * return SCF_STATE_STRING* value for the input
 * NULL on error.
 */
const char *smf_state_to_string(int32_t);

/*
 * Notification interfaces
 */
int smf_notify_set_params(const char *, nvlist_t *);
int smf_notify_get_params(nvlist_t **, nvlist_t *);
int smf_notify_del_params(const char *, const char *, int32_t);

/*
 * SMF exit status definitions
 */
#define SMF_EXIT_OK               0
#define SMF_EXIT_ERR_FATAL       95
#define SMF_EXIT_ERR_CONFIG      96
#define SMF_EXIT_MON_DEGRADE     97
#define SMF_EXIT_MON_OFFLINE     98
#define SMF_EXIT_ERR_NOSMF       99
#define SMF_EXIT_ERR_PERM       100

#ifdef  __cplusplus
}
#endif

#endif  /* _LIBSCF_H */
```

****
   20895 Wed Sep 21 15:14:36 2016
new/usr/src/lib/libscf/inc/libscf_priv.h
smf: switch to a tri-state for process-security properties true=on,false=off,nil
****
______*unchanged_portion_omitted_*

```
/*
 * The pg_pattern element has two optional attributes that play a part in
 * selecting the appropriate prefix for the name of the pg_pattern property
 * group. The two attributes are name and type.  The appropriate prefix
 * encodes the presence are absence of these attributes.
 *
 *      SCF_PG_TM_PG_PATTERN_PREFIX     neither attribute
 *      SCF_PG_TM_PG_PATTERN_N_PREFIX   name only
 *      SCF_PG_TM_PG_PATTERN_T_PREFIX   type only
 *      SCF_PG_TM_PG_PATTERN_NT_PREFIX  both name and type
 */
#define SCF_PG_TM_PG_PAT_BASE           "tm_pgpat"
#define SCF_PG_TM_PG_PATTERN_PREFIX     ((const char *)SCF_PG_TM_PG_PAT_BASE \
        "_")
#define SCF_PG_TM_PG_PATTERN_N_PREFIX   ((const char *)SCF_PG_TM_PG_PAT_BASE \
        "n_")
#define SCF_PG_TM_PG_PATTERN_T_PREFIX   ((const char *)SCF_PG_TM_PG_PAT_BASE \
        "t_")
#define SCF_PG_TM_PG_PATTERN_NT_PREFIX  ((const char *)SCF_PG_TM_PG_PAT_BASE \
        "nt_")
#define SCF_PG_TM_PROP_PATTERN_PREFIX   ((const char *)"tm_proppat_")

/*
 * Pad character to use when encoding strings for property names.
 */
#define SCF_ENCODE32_PAD                ('-')

/*
 * Functions for base 32 encoding/decoding
 */
int scf_decode32(const char *, size_t, char *, size_t, size_t *, char);
int scf_encode32(const char *, size_t, char *, size_t, size_t *, char);

/*
 * handy functions
 */
/*
 * _scf_sanitize_locale
 * Make sure a locale string has only alpha-numeric or '_' characters
 */
void _scf_sanitize_locale(char *);

/*
 * _scf_read_tmpl_prop_type_as_string()
 * Handy function to get template property type as a string
 */
char *_scf_read_tmpl_prop_type_as_string(const scf_prop_tmpl_t *);
/*
 * _scf_read_single_astring_from_pg()
 * Given a property group (pg) and a property name (pn), this function
 * retrives an astring value from pg/pn.
 */
char *_scf_read_single_astring_from_pg(scf_propertygroup_t *, const char *);

/*
 * scf_instance_delete_prop()
 * Given instance, property group, and property, delete the property.
 */
int
```

```
scf_instance_delete_prop(scf_instance_t *, const char *, const char *);

/*
 * Functions to extract boot config information from FMRI_BOOT_CONFIG
 */
void scf_get_boot_config(uint8_t *);
void scf_get_boot_config_ovr(uint8_t *);
int scf_is_fastboot_default(void);

/*
 * Set value of "config_ovr/fastreboot_default".
 */
int scf_fastreboot_default_set_transient(boolean_t);

/*
 * scf_is_compatible_type()
 * Return true if the second type is the same type, or a subtype of the
 * first.
 */
int scf_is_compatible_type(scf_type_t, scf_type_t);

/*
 * Check an array of services and enable any that don't have the
 * "application/auto_enable" property set to "false", which is
 * the interface to turn off this behaviour (see PSARC 2004/739).
 */
void _check_services(char **);

/*
 * _scf_handle_create_and_bind()
 * convenience function that creates and binds a handle
 */
scf_handle_t *_scf_handle_create_and_bind(scf_version_t);

/*
 * _smf_refresh_all_instances()
 * refresh all intances of a service
 * return SCF_SUCCESS or SCF_FAILED on _PERMISSION_DENIED, _BACKEND_ACCESS
 * or _BACKEND_READONLY.
 */
int _smf_refresh_all_instances(scf_service_t *);

/*
 * _scf_get_fma_notify_params()
 * Specialized fuction to get fma notifitation parameters
 */
int _scf_get_fma_notify_params(const char *, nvlist_t *, int);

/*
 * _scf_get_svc_notify_params()
 * Specialized function to get SMF state transition notification parameters
 */
int _scf_get_svc_notify_params(const char *, nvlist_t *, int32_t, int, int);

/*
 * _scf_notify_get_params()
 * Specialized function to get notification parametes from a pg into an
 * nvlist_t
 */
int _scf_notify_get_params(scf_propertygroup_t *, nvlist_t *);

#if !defined(NATIVE_BUILD)
int scf_default_secflags(scf_handle_t *, scf_secflags_t *);
int scf_default_secflags(scf_handle_t *, psecflags_t *);
#endif
```

```
#define SCF_NOTIFY_PARAMS_SOURCE_NAME   ((const char *)"preference_source")

#ifdef  __cplusplus
}
```
______***unchanged_portion_omitted_***

```
************************************************************
    3840 Wed Sep 21 15:14:37 2016
new/usr/src/man/man5/security-flags.5
smf: switch to a tri-state for process-security properties true=on,false=off,nil
************************************************************
.\"
.\" This file and its contents are supplied under the terms of the
.\" Common Development and Distribution License ("CDDL"), version 1.0.
.\" You may only use this file in accordance with the terms of version
.\" 1.0 of the CDDL.
.\"
.\" A full copy of the text of the CDDL should have accompanied this
.\" source.  A copy of the CDDL is also available via the Internet at
.\" http://www.illumos.org/license/CDDL.
.\"
.\" Copyright 2015, Richard Lowe.
.\"
.TH "SECURITY-FLAGS" "5" "June 6, 2016"
.SH "NAME"
\fBsecurity-flags\fR - process security flags
.SH "DESCRIPTION"
Each process on an illumos system has an associated set of security-flags
which describe additional per-process security and exploit mitigation
features which are enabled for that process.
.P
There are four sets of these flags for each process, the effective set
(abbreviated \fIE\fR) are the set which currently apply to the process and are
immutable. The inheritable set (abbreviated \fII\fR) are the flags which will
become effective the next time the process calls one of the \fBexec(2)\fR
family of functions, and will be inherited as both the effective and
inheritable sets by any child processes. The upper set (abbreviated \fIU\fR)
specify the maximal flags that a process can have in its inheritable set.  The
lower set (abbreviated \fIL\fR) specify the minimal amount of flags that a
process must have in its inheritable set.  The inheritable set may be changed
at any time, subject to permissions and the lower and upper sets.
.P
To change the security-flags of a process one must have both permissions
equivalent to those required to send a signal to the process and have the
\fBPRIV_PROC_SECFLAGS\fR privilege.
.P
Currently available features are:

.sp
.ne 2
.na
Address Space Layout Randomisation (\fBASLR\fR)
.ad
.RS 11n
The base addresses of the stack, heap and shared library (including
\fBld.so\fR) mappings are randomised, the bases of mapped regions other than
those using \fBMAP_FIXED\fR are randomised.
.P
Currently, executable base addresses are \fInot\fR randomised, due to which
the mitigation provided by this feature is currently limited.
.P
This flag may also be enabled by the presence of the \fBDT_SUNW_ASLR\fR
dynamic tag in the \fB.dynamic\fR section of the executable file. If this
tag has a value of 1, ASLR will be enabled. If the flag has a value of
\fB0\fR ASLR will be disabled. If the tag is not present, the value of the
ASLR flag will be inherited as normal.
.RE

.sp
.ne 2
.na
Forbid mappings at NULL (\fBFORBIDNULLMAP\fR)
```

```
.ad
.RS 11n
Mappings with an address of 0 are forbidden, and return EINVAL rather than
being honored.
.RE

.sp
.ne 2
.na
Make the userspace stack non-executable (\fBNOEXECSTACK\fR)
.ad
.RS 11n
The stack will be mapped without executable permission, and attempts to
execute it will fault.
.RE

System default security-flags are configured via properties on the
\fBsvc:/system/process-security\fR service, which contains a boolean property
per-flag in the \fBdefault\fR, \fBlower\fR and \fBupper\fR, property groups.
The value indicates the setting of the flag, flags with no value take their
defaults.  For example, to enable ASLR by default you would execute the
following commands:
For example, to enable ASLR by default you would execute the following
commands:
.sp
.in +2
.nf
# svccfg -s svc:/system/process-security setprop default/aslr = true
.fi
.in -2
.sp
.P
To restore the setting to the defaults you would execute:
.sp
.in +2
.nf
# svccfg -s svc:/system/process-security delpropvalue default/aslr true
.fi
.in -2
.sp
.P
#endif /* ! codereview */
This can be done by any user with the \fBsolaris.smf.value.process-security\fR
authorization.
.P
Since security-flags are strictly inherited, this will not take effect until
the system or zone is next booted.

.SH "SEE ALSO"
.BR psecflags (1),
.BR svccfg (1M),
.BR brk (2),
.BR exec (2),
.BR mmap (2),
.BR mmapobj (2),
.BR privileges (5),
.BR rbac (5)
```

```
**********************************************************
   14843 Wed Sep 21 15:14:39 2016
new/usr/src/man/man5/smf_method.5
smf_method(5): fix description of security_flags
**********************************************************
’\" te
.\" Copyright (c) 2009, Sun Microsystems, Inc. All Rights Reserved.
.\" The contents of this file are subject to the terms of the Common Development
.\"  See the License for the specific language governing permissions and limitat
.\" the fields enclosed by brackets "[]" replaced with your own identifying info
.TH SMF_METHOD 5 "June 6, 2016"
.SH NAME
smf_method \- service management framework conventions for methods
.SH DESCRIPTION
.LP
The class of services managed by \fBsvc.startd\fR(1M) in the service management
framework, \fBsmf\fR(5), consists of applications that fit a simple
\fBfork\fR(2)-\fBexec\fR(2) model. The \fBsvc.startd\fR(1M) master daemon and
other restarters support the \fBfork\fR(2)-\fBexec\fR(2) model, potentially
with additional capabilities. The \fBsvc.startd\fR(1M) daemon and other
restarters require that the methods which activate, manipulate, or examine a
service instance follow the conventions described in this manual page.
.SS "Invocation form"
.LP
The form of a method invocation is not dictated by convention. In some cases, a
method invocation might consist of the direct invocation of the daemon or other
binary executable that provides the service. For cases in which an executable
script or other mediating executable is used, the convention recommends the
form:
.sp
.in +2
.nf
/path/to/method_executable abbr_method_name
.fi
.in -2

.sp
.LP
The \fIabbr_method_name\fR used for the recommended form is a supported method
such as \fBstart\fR or \fBstop\fR. The set of methods supported by a restarter
is given on the related restarter page. The \fBsvc.startd\fR(1M) daemon
supports \fBstart\fR, \fBstop\fR, and \fBrefresh\fR methods.
.sp
.LP
A restarter might define other kinds of methods beyond those referenced in this
page. The conventions surrounding such extensions are defined by the restarter
and might not be identical to those given here.
.SS "Environment Variables"
.LP
The restarter provides four environment variables to the method that determine
the context in which the method is invoked.
.sp
.ne 2
.na
\fB\fBSMF_FMRI\fR\fR
.ad
.sp .6
.RS 4n
The service fault management resource identifier (FMRI) of the instance for
which the method is invoked.
.RE

.sp
.ne 2
.na
\fB\fBSMF_METHOD\fR\fR
.ad
.sp .6
.RS 4n
The full name of the method being invoked, such as \fBstart\fR or \fBstop\fR.
.RE

.sp
.ne 2
.na
\fB\fBSMF_RESTARTER\fR\fR
.ad
.sp .6
.RS 4n
The service FMRI of the restarter that invokes the method
.RE

.sp
.ne 2
.na
\fB\fBSMF_ZONENAME\fR\fR
.ad
.sp .6
.RS 4n
The name of the zone in which the method is running. This can also be obtained
by using the \fBzonename\fR(1) command.
.RE

.sp
.LP
These variables should be removed from the environment prior to the invocation
of any persistent process by the method. A convenience shell function,
\fBsmf_clear_env\fR, is given for service authors who use Bourne-compatible
shell scripting to compose service methods in the include file described below.
.sp
.LP
The method context can cause other environment variables to be set as described
below.
.SS "Method Definition"
.LP
A method is defined minimally by three properties in a propertygroup of type
\fBmethod\fR.
.sp
.LP
These properties are:
.sp
.ne 2
.na
\fBexec (\fIastring\fR)\fR
.ad
.RS 27n
Method executable string.
.RE

.sp
.ne 2
.na
\fBtimeout_seconds (\fIcount\fR)\fR
.ad
.RS 27n
Number of seconds before method times out. See the \fBTimeouts\fR section for
more detail.
.RE

.sp
.ne 2
.na
```

```
\fBtype (\fIastring\fR)\fR
.ad
.RS 27n
Method type. Currently always set to \fBmethod\fR.
.RE

.sp
.LP
A Method Context can be defined to further refine the execution environment of
the method. See the \fBMethod Context\fR section for more information.
.SS "Method Tokens"
.LP
When defined in the \fBexec\fR string of the method by the restarter
\fBsvc.startd\fR, a set of tokens are parsed and expanded with appropriate
value. Other restarters might not support method tokens. The delegated
restarter for inet services, \fBinetd\fR(1M), does not support the following
method expansions.
.sp
.ne 2
.na
\fB\fB%%\fR\fR
.ad
.sp .6
.RS 4n
%
.RE

.sp
.ne 2
.na
\fB\fB%r\fR\fR
.ad
.sp .6
.RS 4n
Name of the restarter, such as \fBsvc.startd\fR
.RE

.sp
.ne 2
.na
\fB\fB%m\fR\fR
.ad
.sp .6
.RS 4n
The full name of the method being invoked, such as \fBstart\fR or \fBstop\fR.
.RE

.sp
.ne 2
.na
\fB\fB%s\fR\fR
.ad
.sp .6
.RS 4n
Name of the service
.RE

.sp
.ne 2
.na
\fB\fB%i\fR\fR
.ad
.sp .6
.RS 4n
Name of the instance
.RE
```

```

.sp
.ne 2
.na
\fB\fB\fR\fB%f\fR\fR
.ad
.sp .6
.RS 4n
FMRI of the instance
.RE

.sp
.ne 2
.na
\fB\fB%{prop[:,]}\fR\fR
.ad
.sp .6
.RS 4n
Value(s) of a property. The \fBprop\fR might be a property FMRI, a property
group name and a property name separated by a \fB/\fR, or a property name in
the \fBapplication\fR property group. These values can be followed by a \fB,\fR
(comma) or \fB:\fR (colon). If present, the separators are used to separate
multiple values. If absent, a space is used. The following shell metacharacters
encountered in string values are quoted with a \ (backslash):
.sp
.in +2
.nf
; & ( ) | ^ < > newline space tab  \  " ’
.fi
.in -2

An invalid expansion constitutes method failure.
.RE

.sp
.LP
Two explicit tokens can be used in the place of method commands.
.sp
.ne 2
.na
\fB\fB:kill [-signal]\fR\fR
.ad
.sp .6
.RS 4n
Sends the specified signal, which is \fBSIGTERM\fR by default, to all processes
in the primary instance contract. Always returns \fBSMF_EXIT_OK\fR. This token
should be used to replace common \fBpkill\fR invocations.
.RE

.sp
.ne 2
.na
\fB\fB:true\fR\fR
.ad
.sp .6
.RS 4n
Always returns \fBSMF_EXIT_OK\fR. This token should be used for methods that
are required by the restarter but which are unnecessary for the particular
service implementation.
.RE

.SS "Exiting and Exit Status"
.LP
The required behavior of a start method is to delay exiting until the service
instance is ready to answer requests or is otherwise functional.
.sp
```

```
.LP
The following exit status codes are defined in \fB<libscf.h>\fR and in the
shell support file.
.sp

.sp
.TS
l l l
l l l .
\fBSMF_EXIT_OK\fR       \fB0\fR T{
Method exited, performing its operation successfully.
T}
\fBSMF_EXIT_ERR_FATAL\fR        \fB95\fR        T{
Method failed fatally and is unrecoverable without administrative intervention.
T}
\fBSMF_EXIT_ERR_CONFIG\fR       \fB96\fR        T{
Unrecoverable configuration error. A common condition that returns this exit sta
T}
\fBSMF_EXIT_ERR_NOSMF\fR        \fB99\fR        T{
Method has been mistakenly invoked outside the \fBsmf\fR(5) facility. Services t
T}
\fBSMF_EXIT_ERR_PERM\fR \fB100\fR       T{
Method requires a form of permission such as file access, privilege, authorizati
T}
\fBSMF_EXIT_ERR_OTHER\fR        \fBnon-zero\fR  T{
Any non-zero exit status from a method is treated as an unknown error. A series
T}
.TE

.sp
.LP
Use of a precise exit code allows the responsible restarter to categorize an
error response as likely to be intermittent and worth pursuing restart or
permanent and request administrative intervention.
.SS "Timeouts"
.LP
Each method can have an independent timeout, given in seconds. The choice of a
particular timeout should be based on site expectations for detecting a method
failure due to non-responsiveness. Sites with replicated filesystems or other
failover resources can elect to lengthen method timeouts from the default.
Sites with no remote resources can elect to shorten the timeouts. Method
timeout is specified by the \fBtimeout_seconds\fR property.
.sp
.LP
If you specify \fB0 timeout_seconds\fR for a method, it declares to the
restarter that there is no timeout for the service. This setting is not
preferred, but is available for services that absolutely require it.
.sp
.LP
\fB-1 timeout_seconds\fR is also accepted, but is a deprecated specification.
.SS "Shell Programming Support"
.LP
A set of environment variables that define the above exit status values is
provided with convenience shell functions in the file
\fB/lib/svc/share/smf_include.sh\fR. This file is a Bourne shell script
suitable for inclusion via the source operator in any Bourne-compatible shell.
.sp
.LP
To assist in the composition of scripts that can serve as SMF methods as well
as \fB/etc/init.d\fR scripts, the \fBsmf_present()\fR shell function is
provided. If the \fBsmf\fR(5) facility is not available, \fBsmf_present()\fR
returns a non-zero exit status.
.sp
.LP
One possible structure for such a script follows:
.sp
```

```
.in +2
.nf
if smf_present; then
      # Shell code to run application as managed service
      ....

      smf_clear_env
else
      # Shell code to run application as /etc/init.d script
      ....
fi
.fi
.in -2

.sp
.LP
This example shows the use of both convenience functions that are provided.
.SS "Method Context"
.LP
The service management facility offers a common mechanism set the context in
which the \fBfork\fR(2)-\fBexec\fR(2) model services execute.
.sp
.LP
The desired method context should be provided by the service developer. All
service instances should run with the lowest level of privileges possible to
limit potential security compromises.
.sp
.LP
A method context can contain the following properties:
.sp
.ne 2
.na
\fB\fBuse_profile\fR\fR
.ad
.sp .6
.RS 4n
A boolean that specifies whether the profile should be used instead of the
\fBuser\fR, \fBgroup\fR, \fBprivileges\fR, and \fBlimit_privileges\fR
properties.
.RE

.sp
.ne 2
.na
\fBenvironment\fR
.ad
.sp .6
.RS 4n
Environment variables to insert into the environment of the method, in the form
of a number of \fBNAME=value\fR strings.
.RE

.sp
.ne 2
.na
\fB\fBprofile\fR\fR
.ad
.sp .6
.RS 4n
The name of an RBAC (role-based access control) profile which, along with the
method executable, identifies an entry in \fBexec_attr\fR(4).
.RE

.sp
.ne 2
.na
```

```
\fB\fBuser\fR\fR
.ad
.sp .6
.RS 4n
The user ID in numeric or text form.
.RE

.sp
.ne 2
.na
\fB\fBgroup\fR\fR
.ad
.sp .6
.RS 4n
The group ID in numeric or text form.
.RE

.sp
.ne 2
.na
\fB\fBsupp_groups\fR\fR
.ad
.sp .6
.RS 4n
An optional string that specifies the supplemental group memberships by ID, in
numeric or text form.
.RE

.sp
.ne 2
.na
\fB\fBprivileges\fR\fR
.ad
.sp .6
.RS 4n
An optional string specifying the privilege set as defined in
\fBprivileges\fR(5).
.RE

.sp
.ne 2
.na
\fB\fBlimit_privileges\fR\fR
.ad
.sp .6
.RS 4n
An optional string specifying the limit privilege set as defined in
\fBprivileges\fR(5).
.RE

.sp
.ne 2
.na
\fB\fBworking_directory\fR\fR
.ad
.sp .6
.RS 4n
The home directory from which to launch the method. \fB:home\fR can be used as
a token to indicate the home directory of the user whose \fBuid\fR is used to
launch the method. If the property is unset, \fB:home\fR is used.
.RE

.sp
.ne 2
.na
\fB\fBsecurity_flags\fR\fR
```

```
.ad
.sp .6
.RS 4n
The security flags to apply when launching the method.  See \fBsecurity-flags\fR
.sp
.LP
The "default" keyword specifies those flags specified in
\fBsvc:/system/process-security\fR.  The "all" keyword enables all flags, the
"none" keyword enables no flags.  The "current" keyword specifies the current
flags.  Flags may be added by specifying their name (optionally preceded
by '+'), and removed by preceding their name with '-').
"none" keyword enables no flags.  Further flags may be added by specifying
their name, or removed by specifying their name prefixed by '-' or '!'.
.sp
.LP
Use of "all" has associated risks, as future versions of the system may
include further flags which may harm poorly implemented software.
.RE

.sp
.ne 2
.na
\fB\fBcorefile_pattern\fR\fR
.ad
.sp .6
.RS 4n
An optional string that specifies the corefile pattern to use for the service,
as per \fBcoreadm\fR(1M). Most restarters supply a default. Setting this
property overrides local customizations to the global core pattern.
.RE

.sp
.ne 2
.na
\fB\fBproject\fR\fR
.ad
.sp .6
.RS 4n
The project ID in numeric or text form. \fB:default\fR can be used as a token
to indicate a project identified by \fBgetdefaultproj\fR(3PROJECT) for the user
whose \fBuid\fR is used to launch the method.
.RE

.sp
.ne 2
.na
\fB\fBresource_pool\fR\fR
.ad
.sp .6
.RS 4n
The resource pool name on which to launch the method. \fB:default\fR can be
used as a token to indicate the pool specified in the \fBproject\fR(4) entry
given in the \fBproject\fR attribute above.
.RE

.sp
.LP
The method context can be set for the entire service instance by specifying a
\fBmethod_context\fR property group for the service or instance. A method might
override the instance method context by providing the method context properties
on the method property group.
.sp
.LP
Invalid method context settings always lead to failure of the method, with the
exception of invalid environment variables that issue warnings.
.sp
```

```
.LP
In addition to the context defined above, many \fBfork\fR(2)-\fBexec\fR(2)
model restarters also use the following conventions when invoking executables
as methods:
.sp
.ne 2
.na
\fBArgument array\fR
.ad
.sp .6
.RS 4n
The arguments in \fBargv[]\fR are set consistently with the result \fB/bin/sh
-c\fR of the \fBexec\fR string.
.RE

.sp
.ne 2
.na
\fBFile descriptors\fR
.ad
.sp .6
.RS 4n
File descriptor \fB0\fR is \fB/dev/null\fR. File descriptors \fB1\fR and
\fB2\fR are recommended to be a per-service log file.
.RE

.SH FILES
.ne 2
.na
\fB\fB/lib/svc/share/smf_include.sh\fR\fR
.ad
.sp .6
.RS 4n
Definitions of exit status values.
.RE

.sp
.ne 2
.na
\fB\fB/usr/include/libscf.h\fR\fR
.ad
.sp .6
.RS 4n
Definitions of exit status codes.
.RE

.SH SEE ALSO
.LP
\fBzonename\fR(1), \fBcoreadm\fR(1M), \fBinetd\fR(1M), \fBsvccfg\fR(1M),
\fBsvc.startd\fR(1M), \fBexec\fR(2), \fBfork\fR(2),
\fBgetdefaultproj\fR(3PROJECT), \fBexec_attr\fR(4), \fBproject\fR(4),
\fBservice_bundle\fR(4), \fBattributes\fR(5), \fBprivileges\fR(5),
\fBrbac\fR(5), \fBsmf\fR(5), \fBsmf_bootstrap\fR(5), \fBzones\fR(5),
\fBsecurity-flags\fR(5)
.SH NOTES
.LP
The present version of \fBsmf\fR(5) does not support multiple repositories.
.sp
.LP
When a service is configured to be started as root but with privileges
different from \fBlimit_privileges\fR, the resulting process is privilege
aware.  This can be surprising to developers who expect \fBseteuid(<non-zero
UID>)\fR to reduce privileges to basic or less.
```