

```

*****
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
*****
1 <?xml version='1.0'?>
2 <!DOCTYPE service_bundle SYSTEM '/usr/share/lib/xml/dtd/service_bundle.dtd.1'>

4 <!--
5 Copyright 2015, Richard Lowe.

7 CDDL HEADER START

9 This file and its contents are supplied under the terms of the
10 Common Development and Distribution License ("CDDL"), version 1.0.
11 You may only use this file in accordance with the terms of version
12 1.0 of the CDDL.

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

18 CDDL HEADER END

20 NOTE: This service manifest is not editable; its contents will
21 be overwritten by package or patch operations, including
22 operating system upgrade. Make customizations in a different
23 file.
24 -->

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

31     <single_instance />

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

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

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

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

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

52     <property_group name='lower' type='application'>
53       <property name='aslr' type='boolean' />
54       <property name='forbidnullmap' type='boolean' />
55       <property name='noexecstack' type='boolean' />
53       <propval name='aslr' type='boolean' value='false' />
54       <propval name='forbidnullmap' type='boolean' value='false' />
55       <propval name='noexecstack' type='boolean' value='false' />

```

```

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

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

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

72   <stability value="Unstable" />

74   <template>
75     <common_name>
76       <loctext xml:lang='C'>Security Flag Configuratio
77     </common_name>
78     <documentation>
79       <manpage title='security-flags' section='5'
80       manpath='/usr/share/man' />
81       <manpage title='psecflags' section='1'
82       manpath='/usr/share/man' />
83     </documentation>
84   </template>
85   </service>
86 </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_____

2752 /*
2753 * Fetch method context information from the repository, allocate and fill
2754 * a method_context structure, return it in *mcpp, and return NULL.
2755 *
2756 * If no method_context is defined, original init context is provided, where
2757 * the working directory is '/', and uid/gid are 0/0. But if a method_context
2758 * is defined at any level the smf_method(5) method_context defaults are used.
2759 *
2760 * Return an error message structure containing the error message
2761 * with context, and the error so the caller can make a decision
2762 * on what to do next.
2763 *
2764 * Error Types :
2765 *     E2BIG          Too many values or entry is too big
2766 *     EINVAL        Invalid value
2767 *     EIO           an I/O error has occurred
2768 *     ENOENT        no entry for value
2769 *     ENOMEM        out of memory
2770 *     ENOTSUP       Version mismatch
2771 *     ERANGE        value is out of range
2772 *     EMFILE/ENFILE out of file descriptors
2773 *
2774 *     SCF_ERROR_BACKEND_ACCESS
2775 *     SCF_ERROR_CONNECTION_BROKEN
2776 *     SCF_ERROR_DELETED
2777 *     SCF_ERROR_CONSTRAINT_VIOLATED
2778 *     SCF_ERROR_HANDLE_DESTROYED
2779 *     SCF_ERROR_INTERNAL
2780 *     SCF_ERROR_INVALID_ARGUMENT
2781 *     SCF_ERROR_NO_MEMORY
2782 *     SCF_ERROR_NO_RESOURCES
2783 *     SCF_ERROR_NOT_BOUND
2784 *     SCF_ERROR_NOT_FOUND
2785 *     SCF_ERROR_NOT_SET
2786 *     SCF_ERROR_TYPE_MISMATCH
2787 *
2788 */
2789 mc_error_t *
2790 restarter_get_method_context(uint_t version, scf_instance_t *inst,
2791     scf_snapshot_t *snap, const char *mname, const char *cmdline,
2792     struct method_context **mcpp)
2793 {
2794     scf_handle_t *h;
2795     scf_propertygroup_t *methpg = NULL;
2796     scf_propertygroup_t *instpg = NULL;
2797     scf_propertygroup_t *pg = NULL;
2798     scf_property_t *prop = NULL;
2799     scf_value_t *val = NULL;
2800     scf_type_t ty;
2801     uint8_t use_profile;
2802     int ret = 0;
2803     int mc_used = 0;
2804     mc_error_t *err = NULL;
2805     struct method_context *cip;

2807     if ((err = malloc(sizeof (mc_error_t))) == NULL)
2808         return (mc_error_create(NULL, ENOMEM, NULL));

2810     /* Set the type to zero to track if an error occurred. */

```

```

2811     err->type = 0;

2813     if (version != RESTARTER_METHOD_CONTEXT_VERSION)
2814         return (mc_error_create(err, ENOTSUP,
2815             "Invalid client version %d. (Expected %d)",
2816             version, RESTARTER_METHOD_CONTEXT_VERSION));

2818     /* Get the handle before we allocate anything. */
2819     h = scf_instance_handle(inst);
2820     if (h == NULL)
2821         return (mc_error_create(err, scf_error(),
2822             scf_strerror(scf_error())));

2824     cip = malloc(sizeof (*cip));
2825     if (cip == NULL)
2826         return (mc_error_create(err, ENOMEM, ALLOCFAIL));

2828     (void) memset(cip, 0, sizeof (*cip));
2829     cip->uid = (uid_t)-1;
2830     cip->euid = (uid_t)-1;
2831     cip->gid = (gid_t)-1;
2832     cip->egid = (gid_t)-1;

2834     cip->vbuf_sz = scf_limit(SCF_LIMIT_MAX_VALUE_LENGTH);
2835     assert(cip->vbuf_sz >= 0);
2836     cip->vbuf = malloc(cip->vbuf_sz);
2837     if (cip->vbuf == NULL) {
2838         free(cip);
2839         return (mc_error_create(err, ENOMEM, ALLOCFAIL));
2840     }

2842     if ((instpg = scf_pg_create(h)) == NULL ||
2843         (methpg = scf_pg_create(h)) == NULL ||
2844         (prop = scf_property_create(h)) == NULL ||
2845         (val = scf_value_create(h)) == NULL) {
2846         err = mc_error_create(err, scf_error(),
2847             "Failed to create repository object: %s",
2848             scf_strerror(scf_error()));
2849         goto out;
2850     }

2852     /*
2853     * The method environment, and the credentials/profile data,
2854     * may be found either in the pg for the method (methpg),
2855     * or in the instance/service SCF_PG_METHOD_CONTEXT pg (named
2856     * instpg below).
2857     */

2859     if (scf_instance_get_pg_composed(inst, snap, mname, methpg) !=
2860         SCF_SUCCESS) {
2861         err = mc_error_create(err, scf_error(), "Unable to get the "
2862             "\"%s\" method, %s", mname, scf_strerror(scf_error()));
2863         goto out;
2864     }

2866     if (scf_instance_get_pg_composed(inst, snap, SCF_PG_METHOD_CONTEXT,
2867         instpg) != SCF_SUCCESS) {
2868         if (scf_error() != SCF_ERROR_NOT_FOUND) {
2869             err = mc_error_create(err, scf_error(),
2870                 "Unable to retrieve the \"%s\" property group, %s",
2871                 SCF_PG_METHOD_CONTEXT, scf_strerror(scf_error()));
2872             goto out;
2873         }
2874         scf_pg_destroy(instpg);
2875         instpg = NULL;
2876     } else {

```

```

2877         mc_used++;
2878     }

2880     ret = get_environment(h, methpg, cip, prop, val);
2881     if (ret == ENOENT && instpg != NULL) {
2882         ret = get_environment(h, instpg, cip, prop, val);
2883     }

2885     switch (ret) {
2886     case 0:
2887         mc_used++;
2888         break;
2889     case ENOENT:
2890         break;
2891     case ENOMEM:
2892         err = mc_error_create(err, ret, "Out of memory.");
2893         goto out;
2894     case EINVAL:
2895         err = mc_error_create(err, ret, "Invalid method environment.");
2896         goto out;
2897     default:
2898         err = mc_error_create(err, ret,
2899             "Get method environment failed: %s", scf_strerror(ret));
2900         goto out;
2901     }

2903     pg = methpg;

2905     ret = scf_pg_get_property(pg, SCF_PROPERTY_USE_PROFILE, prop);
2906     if (ret && scf_error() == SCF_ERROR_NOT_FOUND && instpg != NULL) {
2907         pg = NULL;
2908         ret = scf_pg_get_property(instpg, SCF_PROPERTY_USE_PROFILE,
2909             prop);
2910     }

2912     if (ret) {
2913         switch (scf_error()) {
2914         case SCF_ERROR_NOT_FOUND:
2915             /* No profile context: use default credentials */
2916             cip->uid = 0;
2917             cip->gid = 0;
2918             break;

2920         case SCF_ERROR_CONNECTION_BROKEN:
2921             err = mc_error_create(err, SCF_ERROR_CONNECTION_BROKEN,
2922                 RCBROKEN);
2923             goto out;

2925         case SCF_ERROR_DELETED:
2926             err = mc_error_create(err, SCF_ERROR_NOT_FOUND,
2927                 "Could not find property group \"%s\"",
2928                 pg == NULL ? SCF_PG_METHOD_CONTEXT : mname);
2929             goto out;

2931         case SCF_ERROR_HANDLE_MISMATCH:
2932         case SCF_ERROR_INVALID_ARGUMENT:
2933         case SCF_ERROR_NOT_SET:
2934         default:
2935             bad_fail("scf_pg_get_property", scf_error());
2936         }
2937     } else {
2938         if (scf_property_type(prop, &ty) != SCF_SUCCESS) {
2939             ret = scf_error();
2940             switch (ret) {
2941             case SCF_ERROR_CONNECTION_BROKEN:
2942                 err = mc_error_create(err,

```

```

2943                 SCF_ERROR_CONNECTION_BROKEN, RCBROKEN);
2944                 break;

2946         case SCF_ERROR_DELETED:
2947             err = mc_error_create(err,
2948                 SCF_ERROR_NOT_FOUND,
2949                 "Could not find property group \"%s\"",
2950                 pg == NULL ? SCF_PG_METHOD_CONTEXT : mname);
2951             break;

2953         case SCF_ERROR_NOT_SET:
2954         default:
2955             bad_fail("scf_property_type", ret);
2956         }
2958     }
2959     goto out;

2961     if (ty != SCF_TYPE_BOOLEAN) {
2962         err = mc_error_create(err,
2963             SCF_ERROR_TYPE_MISMATCH,
2964             "\"%s\" property is not boolean in property group "
2965             "\"%s\".", SCF_PROPERTY_USE_PROFILE,
2966             pg == NULL ? SCF_PG_METHOD_CONTEXT : mname);
2967         goto out;
2968     }

2970     if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
2971         ret = scf_error();
2972         switch (ret) {
2973         case SCF_ERROR_CONNECTION_BROKEN:
2974             err = mc_error_create(err,
2975                 SCF_ERROR_CONNECTION_BROKEN, RCBROKEN);
2976             break;

2978         case SCF_ERROR_CONSTRAINT_VIOLATED:
2979             err = mc_error_create(err,
2980                 SCF_ERROR_CONSTRAINT_VIOLATED,
2981                 "\"%s\" property has multiple values.",
2982                 SCF_PROPERTY_USE_PROFILE);
2983             break;

2985         case SCF_ERROR_NOT_FOUND:
2986             err = mc_error_create(err,
2987                 SCF_ERROR_NOT_FOUND,
2988                 "\"%s\" property has no values.",
2989                 SCF_PROPERTY_USE_PROFILE);
2990             break;
2991         default:
2992             bad_fail("scf_property_get_value", ret);
2993         }

2995     }
2996     goto out;

3000     mc_used++;
3001     ret = scf_value_get_boolean(val, &use_profile);
3002     assert(ret == SCF_SUCCESS);

3003     /* get ids & privileges */
3004     if (use_profile)
3005         err = get_profile(pg, instpg, prop, val, cmdline,
3006             cip, err);
3007     else
3008         err = get_ids(pg, instpg, prop, val, cip, err);

```

```

3009         if (err->type != 0)
3010             goto out;
3011     }

3013     /* get working directory */
3014     if ((methpg != NULL && scf_pg_get_property(methpg,
3015         SCF_PROPERTY_WORKING_DIRECTORY, prop) == SCF_SUCCESS) ||
3016         (instpg != NULL && scf_pg_get_property(instpg,
3017         SCF_PROPERTY_WORKING_DIRECTORY, prop) == SCF_SUCCESS)) {
3018         if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
3019             ret = scf_error();
3020             switch (ret) {
3021             case SCF_ERROR_CONNECTION_BROKEN:
3022                 err = mc_error_create(err, ret, RCBROKEN);
3023                 break;

3025             case SCF_ERROR_CONSTRAINT_VIOLATED:
3026                 err = mc_error_create(err, ret,
3027                     "\"%s\" property has multiple values.",
3028                     SCF_PROPERTY_WORKING_DIRECTORY);
3029                 break;

3031             case SCF_ERROR_NOT_FOUND:
3032                 err = mc_error_create(err, ret,
3033                     "\"%s\" property has no values.",
3034                     SCF_PROPERTY_WORKING_DIRECTORY);
3035                 break;

3037             default:
3038                 bad_fail("scf_property_get_value", ret);
3039             }

3041             goto out;
3042         }

3044         mc_used++;
3045         ret = scf_value_get_astring(val, cip->vbuf, cip->vbuf_sz);
3046         assert(ret != -1);
3047     } else {
3048         ret = scf_error();
3049         switch (ret) {
3050         case SCF_ERROR_NOT_FOUND:
3051             /* okay if missing. */
3052             (void) strcpy(cip->vbuf, ":default");
3053             break;

3055         case SCF_ERROR_CONNECTION_BROKEN:
3056             err = mc_error_create(err, ret, RCBROKEN);
3057             goto out;

3059         case SCF_ERROR_DELETED:
3060             err = mc_error_create(err, ret,
3061                 "Property group could not be found");
3062             goto out;

3064         case SCF_ERROR_HANDLE_MISMATCH:
3065         case SCF_ERROR_INVALID_ARGUMENT:
3066         case SCF_ERROR_NOT_SET:
3067             default:
3068                 bad_fail("scf_pg_get_property", ret);
3069             }
3070     }

3072     if (strcmp(cip->vbuf, ":default") == 0 ||
3073         strcmp(cip->vbuf, ":home") == 0) {
3074         switch (ret = lookup_pwd(cip)) {

```

```

3075         case 0:
3076             break;

3078         case ENOMEM:
3079             err = mc_error_create(err, ret, "Out of memory.");
3080             goto out;

3082         case ENOENT:
3083         case EIO:
3084         case EMFILE:
3085         case ENFILE:
3086             err = mc_error_create(err, ret,
3087                 "Could not get passwd entry.");
3088             goto out;

3090         default:
3091             bad_fail("lookup_pwd", ret);
3092         }

3094         cip->working_dir = strdup(cip->pwd.pw_dir);
3095         if (cip->working_dir == NULL) {
3096             err = mc_error_create(err, ENOMEM, ALLOCFAIL);
3097             goto out;
3098         }
3099     } else {
3100         cip->working_dir = strdup(cip->vbuf);
3101         if (cip->working_dir == NULL) {
3102             err = mc_error_create(err, ENOMEM, ALLOCFAIL);
3103             goto out;
3104         }
3105     }

3107     /* get security flags */
3108     if ((methpg != NULL && scf_pg_get_property(methpg,
3109         SCF_PROPERTY_SECFLAGS, prop) == SCF_SUCCESS) ||
3110         (instpg != NULL && scf_pg_get_property(instpg,
3111         SCF_PROPERTY_SECFLAGS, prop) == SCF_SUCCESS)) {
3112         if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
3113             ret = scf_error();
3114             switch (ret) {
3115             case SCF_ERROR_CONNECTION_BROKEN:
3116                 err = mc_error_create(err, ret, RCBROKEN);
3117                 break;

3119             case SCF_ERROR_CONSTRAINT_VIOLATED:
3120                 err = mc_error_create(err, ret,
3121                     "\"%s\" property has multiple values.",
3122                     SCF_PROPERTY_SECFLAGS);
3123                 break;

3125             case SCF_ERROR_NOT_FOUND:
3126                 err = mc_error_create(err, ret,
3127                     "\"%s\" property has no values.",
3128                     SCF_PROPERTY_SECFLAGS);
3129                 break;

3131             default:
3132                 bad_fail("scf_property_get_value", ret);
3133             }

3135         (void) strcpy(cip->vbuf, ":default", cip->vbuf_sz);
3136     } else {
3137         ret = scf_value_get_astring(val, cip->vbuf,
3138             cip->vbuf_sz);
3139         assert(ret != -1);
3140     }

```

```

3141         mc_used++;
3142     } else {
3143         ret = scf_error();
3144         switch (ret) {
3145             case SCF_ERROR_NOT_FOUND:
3146                 /* okay if missing. */
3147                 (void) strcpy(cip->vbuf, ":default", cip->vbuf_sz);
3148                 break;
3149
3150             case SCF_ERROR_CONNECTION_BROKEN:
3151                 err = mc_error_create(err, ret, RCBROKEN);
3152                 goto out;
3153
3154             case SCF_ERROR_DELETED:
3155                 err = mc_error_create(err, ret,
3156                     "Property group could not be found");
3157                 goto out;
3158
3159             case SCF_ERROR_HANDLE_MISMATCH:
3160             case SCF_ERROR_INVALID_ARGUMENT:
3161             case SCF_ERROR_NOT_SET:
3162             default:
3163                 bad_fail("scf_pg_get_property", ret);
3164         }
3165     }
3166
3167     if (scf_default_secflags(h, &cip->def_secflags) != 0) {
3168         err = mc_error_create(err, EINVAL, "couldn't fetch "
3169             "default security-flags");
3170         goto out;
3171     }
3172
3173     if (strcmp(cip->vbuf, ":default") != 0) {
3174         if (secflags_parse(NULL, cip->vbuf,
3175             if (strcmp(cip->vbuf, ":default") == 0) {
3176                 if (secflags_parse(&cip->def_secflags.psf_inherit, "default",
3177                     &cip->secflag_delta) != 0) {
3178                         err = mc_error_create(err, EINVAL, "couldn't parse "
3179                             "security flags: %s", cip->vbuf);
3180                         goto out;
3181                     }
3182                 } else {
3183                     if (secflags_parse(&cip->def_secflags.psf_inherit, cip->vbuf,
3184                         &cip->secflag_delta) != 0) {
3185                             err = mc_error_create(err, EINVAL, "couldn't parse "
3186                                 "security flags: %s", cip->vbuf);
3187                             goto out;
3188                         }
3189                     }
3190                 }
3191             }
3192         }
3193     }
3194
3195     /* get (optional) corefile pattern */
3196     if ((methpg != NULL && scf_pg_get_property(methpg,
3197         SCF_PROPERTY_COREFILE_PATTERN, prop) == SCF_SUCCESS) ||
3198         (instpg != NULL && scf_pg_get_property(instpg,
3199         SCF_PROPERTY_COREFILE_PATTERN, prop) == SCF_SUCCESS)) {
3200         if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
3201             ret = scf_error();
3202             switch (ret) {
3203                 case SCF_ERROR_CONNECTION_BROKEN:
3204                     err = mc_error_create(err, ret, RCBROKEN);
3205                     break;
3206
3207             case SCF_ERROR_CONSTRAINT_VIOLATED:
3208                 err = mc_error_create(err, ret,
3209                     "\"%s\" property has multiple values.",

```

```

3198         SCF_PROPERTY_COREFILE_PATTERN);
3199         break;
3200
3201     case SCF_ERROR_NOT_FOUND:
3202         err = mc_error_create(err, ret,
3203             "\"%s\" property has no values.",
3204             SCF_PROPERTY_COREFILE_PATTERN);
3205         break;
3206
3207     default:
3208         bad_fail("scf_property_get_value", ret);
3209     }
3210 } else {
3211
3212     ret = scf_value_get_astring(val, cip->vbuf,
3213         cip->vbuf_sz);
3214     assert(ret != -1);
3215
3216     cip->corefile_pattern = strdup(cip->vbuf);
3217     if (cip->corefile_pattern == NULL) {
3218         err = mc_error_create(err, ENOMEM, ALLOCFAIL);
3219         goto out;
3220     }
3221 }
3222
3223     mc_used++;
3224 } else {
3225     ret = scf_error();
3226     switch (ret) {
3227         case SCF_ERROR_NOT_FOUND:
3228             /* okay if missing. */
3229             break;
3230
3231     case SCF_ERROR_CONNECTION_BROKEN:
3232         err = mc_error_create(err, ret, RCBROKEN);
3233         goto out;
3234
3235     case SCF_ERROR_DELETED:
3236         err = mc_error_create(err, ret,
3237             "Property group could not be found");
3238         goto out;
3239
3240     case SCF_ERROR_HANDLE_MISMATCH:
3241     case SCF_ERROR_INVALID_ARGUMENT:
3242     case SCF_ERROR_NOT_SET:
3243     default:
3244         bad_fail("scf_pg_get_property", ret);
3245     }
3246 }
3247
3248     if (restarter_rm_libs_loadable()) {
3249         /* get project */
3250         if ((methpg != NULL && scf_pg_get_property(methpg,
3251             SCF_PROPERTY_PROJECT, prop) == SCF_SUCCESS) ||
3252             (instpg != NULL && scf_pg_get_property(instpg,
3253             SCF_PROPERTY_PROJECT, prop) == SCF_SUCCESS)) {
3254             if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
3255                 ret = scf_error();
3256                 switch (ret) {
3257                     case SCF_ERROR_CONNECTION_BROKEN:
3258                         err = mc_error_create(err, ret,
3259                             RCBROKEN);
3260                         break;
3261
3262             case SCF_ERROR_CONSTRAINT_VIOLATED:

```

```

3264         err = mc_error_create(err, ret,
3265             "\\%s\" property has multiple "
3266             "values.", SCF_PROPERTY_PROJECT);
3267         break;

3269     case SCF_ERROR_NOT_FOUND:
3270         err = mc_error_create(err, ret,
3271             "\\%s\" property has no values.",
3272             SCF_PROPERTY_PROJECT);
3273         break;

3275     default:
3276         bad_fail("scf_property_get_value", ret);
3277     }

3279     (void) strcpy(cip->vbuf, ":default");
3280 } else {
3281     ret = scf_value_get_astring(val, cip->vbuf,
3282         cip->vbuf_sz);
3283     assert(ret != -1);
3284 }

3286     mc_used++;
3287 } else {
3288     (void) strcpy(cip->vbuf, ":default");
3289 }

3291 switch (ret = get_projid(cip->vbuf, cip)) {
3292 case 0:
3293     break;

3295 case ENOMEM:
3296     err = mc_error_create(err, ret, "Out of memory.");
3297     goto out;

3299 case ENOENT:
3300     err = mc_error_create(err, ret,
3301         "Missing passwd or project entry for \"%s\".",
3302         cip->vbuf);
3303     goto out;

3305 case EIO:
3306     err = mc_error_create(err, ret, "I/O error.");
3307     goto out;

3309 case EMFILE:
3310 case ENFILE:
3311     err = mc_error_create(err, ret,
3312         "Out of file descriptors.");
3313     goto out;

3315 case -1:
3316     err = mc_error_create(err, ret,
3317         "Name service switch is misconfigured.");
3318     goto out;

3320 case ERANGE:
3321 case E2BIG:
3322     err = mc_error_create(err, ret,
3323         "Project ID \"%s\" too big.", cip->vbuf);
3324     goto out;

3326 case EINVAL:
3327     err = mc_error_create(err, ret,
3328         "Project ID \"%s\" is invalid.", cip->vbuf);
3329     goto out;

```

```

3331     default:
3332         bad_fail("get_projid", ret);
3333     }

3335     /* get resource pool */
3336     if ((methpg != NULL && scf_pg_get_property(methpg,
3337         SCF_PROPERTY_RESOURCE_POOL, prop) == SCF_SUCCESS) ||
3338         (instpg != NULL && scf_pg_get_property(instpg,
3339         SCF_PROPERTY_RESOURCE_POOL, prop) == SCF_SUCCESS)) {
3340         if (scf_property_get_value(prop, val) != SCF_SUCCESS) {
3341             ret = scf_error();
3342             switch (ret) {
3343             case SCF_ERROR_CONNECTION_BROKEN:
3344                 err = mc_error_create(err, ret,
3345                     RCBROKEN);
3346                 break;

3348             case SCF_ERROR_CONSTRAINT_VIOLATED:
3349                 err = mc_error_create(err, ret,
3350                     "\\%s\" property has multiple "
3351                     "values.",
3352                     SCF_PROPERTY_RESOURCE_POOL);
3353                 break;

3355             case SCF_ERROR_NOT_FOUND:
3356                 err = mc_error_create(err, ret,
3357                     "\\%s\" property has no "
3358                     "values.",
3359                     SCF_PROPERTY_RESOURCE_POOL);
3360                 break;

3362             default:
3363                 bad_fail("scf_property_get_value", ret);
3364             }

3366         } else {
3367             (void) strcpy(cip->vbuf, ":default");
3368             ret = scf_value_get_astring(val, cip->vbuf,
3369                 cip->vbuf_sz);
3370             assert(ret != -1);
3371         }

3373     mc_used++;
3374 } else {
3375     ret = scf_error();
3376     switch (ret) {
3377     case SCF_ERROR_NOT_FOUND:
3378         /* okay if missing. */
3379         (void) strcpy(cip->vbuf, ":default");
3380         break;

3382     case SCF_ERROR_CONNECTION_BROKEN:
3383         err = mc_error_create(err, ret, RCBROKEN);
3384         goto out;

3386     case SCF_ERROR_DELETED:
3387         err = mc_error_create(err, ret,
3388             "property group could not be found.");
3389         goto out;

3391     case SCF_ERROR_HANDLE_MISMATCH:
3392     case SCF_ERROR_INVALID_ARGUMENT:
3393     case SCF_ERROR_NOT_SET:
3394     default:
3395         bad_fail("scf_pg_get_property", ret);

```

```

3396     }
3397 }
3399     if (strcmp(cip->vbuf, "::default") != 0) {
3400         cip->resource_pool = strdup(cip->vbuf);
3401         if (cip->resource_pool == NULL) {
3402             err = mc_error_create(err, ENOMEM, ALLOCFAIL);
3403             goto out;
3404         }
3405     }
3406 }
3408 /*
3409  * A method_context was not used for any configurable
3410  * elements or attributes, so reset and use the simple
3411  * defaults that provide historic init behavior.
3412  */
3413 if (mc_used == 0) {
3414     free(cip->pwbuf);
3415     free(cip->vbuf);
3416     free(cip->working_dir);
3418     (void) memset(cip, 0, sizeof (*cip));
3419     cip->uid = 0;
3420     cip->gid = 0;
3421     cip->euid = (uid_t)-1;
3422     cip->egid = (gid_t)-1;
3424     if (scf_default_secflags(h, &cip->def_secflags) != 0) {
3425         err = mc_error_create(err, EINVAL, "couldn't fetch "
3426             "default security-flags");
3427         goto out;
3428     }
3437     if (scf_parse(&cip->def_secflags.psf_inherit, "default",
3438         &cip->secflag_delta) != 0) {
3439         err = mc_error_create(err, EINVAL, "couldn't parse "
3440             "security flags: %s", cip->vbuf);
3441         goto out;
3442     }
3429 }
3431 *mcpp = cip;
3433 out:
3434     (void) scf_value_destroy(val);
3435     scf_property_destroy(prop);
3436     scf_pg_destroy(instpg);
3437     scf_pg_destroy(methpg);
3439     if (cip->pwbuf != NULL) {
3440         free(cip->pwbuf);
3441         cip->pwbuf = NULL;
3442     }
3444     free(cip->vbuf);
3446     if (err->type != 0) {
3447         restarter_free_method_context(cip);
3448     } else {
3449         restarter_mc_error_destroy(err);
3450         err = NULL;
3451     }
3453     return (err);
3454 }

```

```

3456 /*
3457  * Modify the current process per the given method_context.  On success, returns
3458  * 0.  Note that the environment is not modified by this function to include the
3459  * environment variables in cip->env.
3460  *
3461  * On failure, sets *fp to NULL or the name of the function which failed,
3462  * and returns one of the following error codes.  The words in parentheses are
3463  * the values to which *fp may be set for the error case.
3464  * ENOMEM - malloc() failed
3465  * EIO - an I/O error occurred (getpwuid_r, chdir)
3466  * EMFILE - process is out of file descriptors (getpwuid_r)
3467  * ENFILE - system is out of file handles (getpwuid_r)
3468  * EINVAL - gid or egid is out of range (setregid)
3469  *          ngroups is too big (setgroups)
3470  *          project's project id is bad (setproject)
3471  *          uid or euid is out of range (setreuid)
3472  *          poolname is invalid (pool_set_binding)
3473  * EPERM - insufficient privilege (setregid, initgroups, setgroups, setppriv,
3474  *          setproject, setreuid, settaskid)
3475  * ENOENT - uid has a passwd entry but no shadow entry
3476  *          working_dir does not exist (chdir)
3477  *          uid has no passwd entry
3478  *          the pool could not be found (pool_set_binding)
3479  * EFAULT - lpriv_set or priv_set has a bad address (setppriv)
3480  *          working_dir has a bad address (chdir)
3481  * EACCES - could not access working_dir (chdir)
3482  *          in a TASK_FINAL task (setproject, settaskid)
3483  *          no resource pool accepting default binding exists (setproject)
3484  * ELOOP - too many symbolic links in working_dir (chdir)
3485  * ENAMETOOLONG - working_dir is too long (chdir)
3486  * ENOLINK - working_dir is on an inaccessible remote machine (chdir)
3487  * ENOTDIR - working_dir is not a directory (chdir)
3488  * ESRCH - uid is not a user of project (setproject)
3489  *          project is invalid (setproject)
3490  *          the resource pool specified for project is unknown (setproject)
3491  * EBADF - the configuration for the pool is invalid (pool_set_binding)
3492  * -1 - core_set_process_path() failed (core_set_process_path)
3493  *          a resource control assignment failed (setproject)
3494  *          a system error occurred during pool_set_binding (pool_set_binding)
3495  */
3496 int
3497 restarter_set_method_context(struct method_context *cip, const char **fp)
3498 {
3499     pid_t mypid = -1;
3500     int r, ret;
3515     secflagdelta_t delta = {0};
3502     cip->pwbuf = NULL;
3503     *fp = NULL;
3505     if (cip->gid != (gid_t)-1) {
3506         if (setregid(cip->gid,
3507             cip->egid != (gid_t)-1 ? cip->egid : cip->gid) != 0) {
3508             *fp = "setregid";
3510             ret = errno;
3511             assert(ret == EINVAL || ret == EPERM);
3512             goto out;
3513         }
3514     } else {
3515         if (cip->pwbuf == NULL) {
3516             switch (ret = lookup_pwd(cip)) {
3517                 case 0:
3518                     break;

```

```

3520         case ENOMEM:
3521         case ENOENT:
3522             *fp = NULL;
3523             goto out;
3524
3525         case EIO:
3526         case EMFILE:
3527         case ENFILE:
3528             *fp = "getpwuid_r";
3529             goto out;
3530
3531         default:
3532             bad_fail("lookup_pwd", ret);
3533     }
3534 }
3535
3536 if (setregid(cip->pwd.pw_gid,
3537             cip->egid != (gid_t)-1 ?
3538             cip->egid : cip->pwd.pw_gid) != 0) {
3539     *fp = "setregid";
3540
3541     ret = errno;
3542     assert(ret == EINVAL || ret == EPERM);
3543     goto out;
3544 }
3545
3546 if (cip->ngroups == -1) {
3547     if (cip->pwbuf == NULL) {
3548         switch (ret = lookup_pwd(cip)) {
3549             case 0:
3550                 break;
3551
3552             case ENOMEM:
3553             case ENOENT:
3554                 *fp = NULL;
3555                 goto out;
3556
3557             case EIO:
3558             case EMFILE:
3559             case ENFILE:
3560                 *fp = "getpwuid_r";
3561                 goto out;
3562
3563             default:
3564                 bad_fail("lookup_pwd", ret);
3565         }
3566     }
3567 }
3568
3569 /* Ok if cip->gid == -1 */
3570 if (initgroups(cip->pwd.pw_name, cip->gid) != 0) {
3571     *fp = "initgroups";
3572     ret = errno;
3573     assert(ret == EPERM);
3574     goto out;
3575 }
3576 } else if (cip->ngroups > 0 &&
3577            setgroups(cip->ngroups, cip->groups) != 0) {
3578     *fp = "setgroups";
3579
3580     ret = errno;
3581     assert(ret == EINVAL || ret == EPERM);
3582     goto out;
3583 }
3584
3585 if (cip->corefile_pattern != NULL) {

```

```

3586         mypid = getpid();
3587
3588         if (core_set_process_path(cip->corefile_pattern,
3589                                 strlen(cip->corefile_pattern) + 1, mypid) != 0) {
3590             *fp = "core_set_process_path";
3591             ret = -1;
3592             goto out;
3593         }
3594     }
3595
3596     delta.psd_ass_active = B_TRUE;
3597     secflags_copy(&delta.psd_assign, &cip->def_secflags.psf_inherit);
3598     if (psecflags(P_PID, P_MYID, PSF_INHERIT,
3599                 &cip->def_secflags.ss_default) != 0) {
3600         *fp = "psecflags (default inherit)";
3601         &delta) != 0) {
3602             *fp = "psecflags (inherit defaults)";
3603             ret = errno;
3604             goto out;
3605         }
3606     }
3607
3608     if (psecflags(P_PID, P_MYID, PSF_LOWER,
3609                 &cip->def_secflags.ss_lower) != 0) {
3610         *fp = "psecflags (default lower)";
3611         if (psecflags(P_PID, P_MYID, PSF_INHERIT,
3612                     &cip->secflag_delta) != 0) {
3613             *fp = "psecflags (inherit)";
3614             ret = errno;
3615             goto out;
3616         }
3617     }
3618
3619     if (psecflags(P_PID, P_MYID, PSF_UPPER,
3620                 &cip->def_secflags.ss_upper) != 0) {
3621         *fp = "psecflags (default upper)";
3622         secflags_copy(&delta.psd_assign, &cip->def_secflags.psf_lower);
3623         if (psecflags(P_PID, P_MYID, PSF_LOWER,
3624                     &delta) != 0) {
3625             *fp = "psecflags (lower)";
3626             ret = errno;
3627             goto out;
3628         }
3629     }
3630
3631     if (psecflags(P_PID, P_MYID, PSF_INHERIT,
3632                 &cip->secflag_delta) != 0) {
3633         *fp = "psecflags (from manifest)";
3634         secflags_copy(&delta.psd_assign, &cip->def_secflags.psf_upper);
3635         if (psecflags(P_PID, P_MYID, PSF_UPPER,
3636                     &delta) != 0) {
3637             *fp = "psecflags (upper)";
3638             ret = errno;
3639             goto out;
3640         }
3641     }
3642
3643     if (restarter_rm_libs_loadable() {
3644         if (cip->project == NULL) {
3645             if (settaskid(getprojid(), TASK_NORMAL) == -1) {
3646                 switch (errno) {
3647                     case EACCES:
3648                     case EPERM:
3649                         *fp = "settaskid";
3650                         ret = errno;
3651                         goto out;
3652                 }
3653             }
3654         }
3655     }
3656
3657     case EINVAL:
3658     default:

```



```

3637         bad_fail("settaskid", errno);
3638     }
3639 } else {
3640     switch (ret = lookup_pwd(cip)) {
3641     case 0:
3642         break;
3643
3644     case ENOMEM:
3645     case ENOENT:
3646         *fp = NULL;
3647         goto out;
3648
3649     case EIO:
3650     case EMFILE:
3651     case ENFILE:
3652         *fp = "getpwuid_r";
3653         goto out;
3654
3655     default:
3656         bad_fail("lookup_pwd", ret);
3657     }
3658
3659     *fp = "setproject";
3660
3661     switch (setproject(cip->project, cip->pwd.pw_name,
3662         TASK_NORMAL)) {
3663     case 0:
3664         break;
3665
3666     case SETPROJ_ERR_TASK:
3667     case SETPROJ_ERR_POOL:
3668         ret = errno;
3669         goto out;
3670
3671     default:
3672         ret = -1;
3673         goto out;
3674     }
3675 }
3676
3677 if (cip->resource_pool != NULL) {
3678     if (mypid == -1)
3679         mypid = getpid();
3680
3681     *fp = "pool_set_binding";
3682
3683     if (pool_set_binding(cip->resource_pool, P_PID,
3684         mypid) != PO_SUCCESS) {
3685         switch (pool_error()) {
3686         case POE_INVALID_SEARCH:
3687             ret = ENOENT;
3688             break;
3689
3690         case POE_BADPARAM:
3691             ret = EINVAL;
3692             break;
3693
3694         case POE_INVALID_CONF:
3695             ret = EBADF;
3696             break;
3697
3698         case POE_SYSTEM:
3699             ret = -1;
3700             break;
3701

```

```

3703         default:
3704             bad_fail("pool_set_binding",
3705                 pool_error());
3706     }
3707
3708     goto out;
3709 }
3710 }
3711
3712 /*
3713  * Now, we have to assume our ID. If the UID is 0, we want it to be
3714  * privilege-aware, otherwise the limit set gets used instead of E/P.
3715  * We can do this by setting P as well, which keeps
3716  * PA status (see priv_can_clear_PA()).
3717  */
3718
3719 *fp = "setppriv";
3720
3721 if (cip->lpriv_set != NULL) {
3722     if (setppriv(PRIV_SET, PRIV_LIMIT, cip->lpriv_set) != 0) {
3723         ret = errno;
3724         assert(ret == EFAULT || ret == EPERM);
3725         goto out;
3726     }
3727 }
3728
3729 if (cip->priv_set != NULL) {
3730     if (setppriv(PRIV_SET, PRIV_INHERITABLE, cip->priv_set) != 0) {
3731         ret = errno;
3732         assert(ret == EFAULT || ret == EPERM);
3733         goto out;
3734     }
3735 }
3736
3737 /*
3738  * If the limit privset is already set, then must be privilege
3739  * aware. Otherwise, don't assume anything, and force privilege
3740  * aware status.
3741  */
3742
3743 if (cip->lpriv_set == NULL && cip->priv_set != NULL) {
3744     ret = setpflags(PRIV_AWARE, 1);
3745     assert(ret == 0);
3746 }
3747
3748 *fp = "setreuid";
3749 if (setreuid(cip->uid,
3750     cip->euid != (uid_t)-1 ? cip->euid : cip->uid) != 0) {
3751     ret = errno;
3752     assert(ret == EINVAL || ret == EPERM);
3753     goto out;
3754 }
3755
3756 *fp = "setppriv";
3757 if (cip->priv_set != NULL) {
3758     if (setppriv(PRIV_SET, PRIV_PERMITTED, cip->priv_set) != 0) {
3759         ret = errno;
3760         assert(ret == EFAULT || ret == EPERM);
3761         goto out;
3762     }
3763 }
3764
3765 /*
3766  * The last thing to do is chdir to the specified working directory.
3767  * This should come after the uid switching as only the user might
3768  * have access to the specified directory.

```

```
3769     */
3770     if (cip->working_dir != NULL) {
3771         do {
3772             r = chdir(cip->working_dir);
3773         } while (r != 0 && errno == EINTR);
3774         if (r != 0) {
3775             *fp = "chdir";
3776             ret = errno;
3777             goto out;
3778         }
3779     }
3781     ret = 0;
3782 out:
3783     free(cip->pwbuf);
3784     cip->pwbuf = NULL;
3785     return (ret);
3786 }
unchanged_portion_omitted
```

new/usr/src/lib/librestart/common/librestart.h

1

```
*****
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_____

231 /*
232  * Functions for updating the repository.
233  */

235 /*
236  * When setting state to "maintenance", callers of restarter_set_states() can
237  * set aux_state to "service_request" to communicate that another service has
238  * requested maintenance state for the target service.
239  *
240  * Callers should use restarter_inst_validate_aux_fmri() to validate the fmri
241  * of the requested service and pass "service_request" for aux_state when
242  * calling restarter_set_states(). See inetd and startd for examples.
243  */
244 int restarter_set_states(restarter_event_handle_t *, const char *,
245     restarter_instance_state_t, restarter_instance_state_t,
246     restarter_instance_state_t, restarter_instance_state_t, restarter_error_t,
247     restarter_str_t);
248 int restarter_event_publish_retry(evchan_t *, const char *, const char *,
249     const char *, const char *, nvlist_t *, uint32_t);

251 /*
252  * functions for retrieving the state transition reason messages
253  */

255 #define RESTARTER_STRING_VERSION      1

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

261 int restarter_store_contract(scf_instance_t *, ctid_t,
262     restarter_contract_type_t);
263 int restarter_remove_contract(scf_instance_t *, ctid_t,
264     restarter_contract_type_t);

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

269 #define RESTARTER_METHOD_CONTEXT_VERSION      8

271 struct method_context {
272     /* Stable */
273     uid_t      uid, euid;
274     gid_t      gid, egid;
275     int        ngroups;          /* -1 means use initgroups(). */
276     gid_t      groups[NGROUPS_MAX];
277     scf_secflags_t  def_secflags;
277     psecflags_t  def_secflags;
278     secflagdelta_t  secflag_delta;
279     priv_set_t  *lpriv_set, *priv_set;
280     char        *corefile_pattern; /* Optional. */
281     char        *project;          /* NULL for no change */
282     char        *resource_pool;    /* NULL for project default */
283     char        *working_dir;     /* NULL for :default */
284     char        **env;            /* NULL for no env */
285     size_t      env_sz;          /* size of env array */

287     /* Private */
288     char        *vbuf;
```

new/usr/src/lib/librestart/common/librestart.h

2

```
289     ssize_t      vbuf_sz;
290     struct passwd pwd;
291     char        *pwbuff;
292     ssize_t      pwbufsz;
293 };
_____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
*****
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 2010 Sun Microsystems, Inc. All rights reserved.
24  * Use is subject to license terms.
25 */
27 /*
28  * This file contains high level functions used by multiple utilities.
29 */
31 #include "libscf_impl.h"
33 #include <assert.h>
34 #include <libutil.h>
35 #include <string.h>
36 #include <strings.h>
37 #endif /* ! codereview */
38 #include <stdlib.h>
39 #include <sys/systeminfo.h>
40 #include <sys/uadmin.h>
41 #include <sys/utsname.h>
42 #include <sys/secflags.h>
44 #ifdef __x86
45 #include <smbios.h>
47 /*
48  * Check whether the platform is on the fastreboot_blacklist.
49  * Return 1 if the platform has been blacklisted, 0 otherwise.
50  */
51 static int
52 scf_is_fb_blacklisted(void)
53 {
54     smbios_hdl_t *shp;
55     smbios_system_t sys;
56     smbios_info_t info;
58     id_t id;
59     int err;
60     int i;

```

```

62     scf_simple_prop_t *prop = NULL;
63     ssize_t numvals;
64     char *platform_name;
66     int blacklisted = 0;
68     /*
69      * If there's no SMBIOS, assume it's blacklisted.
70      */
71     if ((shp = smbios_open(NULL, SMB_VERSION, 0, &err)) == NULL)
72         return (1);
74     /*
75      * If we can't read system info, assume it's blacklisted.
76      */
77     if ((id = smbios_info_system(shp, &sys)) == SMB_ERR ||
78         smbios_info_common(shp, id, &info) == SMB_ERR) {
79         blacklisted = 1;
80         goto fb_out;
81     }
83     /*
84      * If we can't read the "platforms" property from property group
85      * BOOT_CONFIG_PG_FBBLACKLIST, assume no platforms have
86      * been blacklisted.
87      */
88     if ((prop = scf_simple_prop_get(NULL, FMRI_BOOT_CONFIG,
89         BOOT_CONFIG_PG_FBBLACKLIST, "platforms")) == NULL)
90         goto fb_out;
92     numvals = scf_simple_prop_numvalues(prop);
94     for (i = 0; i < numvals; i++) {
95         platform_name = scf_simple_prop_next_astring(prop);
96         if (platform_name == NULL)
97             break;
98         if (strcmp(platform_name, info.smbi_product) == 0) {
99             blacklisted = 1;
100             break;
101         }
102     }
104 fb_out:
105     smbios_close(shp);
106     scf_simple_prop_free(prop);
108     return (blacklisted);
109 }
111 /*
112  * Add or get a property group given an FMRI.
113  * Return SCF_SUCCESS on success, SCF_FAILED on failure.
114  */
115 static int
116 scf_fmri_pg_get_or_add(const char *fmri, const char *pgname,
117     const char *pgtype, uint32_t pgflags, int add)
118 {
119     scf_handle_t *handle = NULL;
120     scf_instance_t *inst = NULL;
121     int rc = SCF_FAILED;
122     int error;
124     if ((handle = scf_handle_create(SCF_VERSION)) == NULL ||
125         scf_handle_bind(handle) != 0 ||
126         (inst = scf_instance_create(handle)) == NULL ||
127         scf_handle_decode_fmri(handle, fmri, NULL, NULL,

```

```

128     inst, NULL, NULL, SCF_DECODE_FMRI_EXACT) != SCF_SUCCESS)
129     goto scferror;

131     if (add) {
132         rc = scf_instance_add_pg(inst, pgname, pgtype, pgflags, NULL);
133         /*
134          * If the property group already exists, return SCF_SUCCESS.
135          */
136         if (rc != SCF_SUCCESS && scf_error() == SCF_ERROR_EXISTS)
137             rc = SCF_SUCCESS;
138     } else {
139         rc = scf_instance_get_pg(inst, pgname, NULL);
140     }

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

146     scf_instance_destroy(inst);
147     if (handle)
148         (void) scf_handle_unbind(handle);
149     scf_handle_destroy(handle);

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

154     return (rc);
155 }
156 #endif /* __x86 */

158 /*
159  * Get config properties from svc:/system/boot-config:default.
160  * It prints errors with uu_warn().
161  */
162 void
163 scf_get_boot_config(uint8_t *boot_config)
164 {
165     uint64_t ret = 0;

167     assert(boot_config);
168     *boot_config = 0;

170     {
171         /*
172          * Property vector for BOOT_CONFIG_PG_PARAMS property group.
173          */
174         scf_propvec_t ua_boot_config[] = {
175             { FASTREBOOT_DEFAULT, NULL, SCF_TYPE_BOOLEAN, NULL,
176               UA_FASTREBOOT_DEFAULT },
177             { FASTREBOOT_ONPANIC, NULL, SCF_TYPE_BOOLEAN, NULL,
178               UA_FASTREBOOT_ONPANIC },
179             { NULL }
180         };
181         scf_propvec_t *prop;

183         for (prop = ua_boot_config; prop->pv_prop != NULL; prop++)
184             prop->pv_ptr = &ret;
185         prop = NULL;
186         if (scf_read_propvec(FMRI_BOOT_CONFIG, BOOT_CONFIG_PG_PARAMS,
187                             B_TRUE, ua_boot_config, &prop) != SCF_FAILED) {
189 #ifdef __x86
190             /*
191              * Unset both flags if the platform has been
192              * blacklisted.
193              */

```

```

194         if (scf_is_fb_blacklisted())
195             return;
196 #endif /* __x86 */
197         *boot_config = (uint8_t)ret;
198         return;
199     }
200 #if defined(FASTREBOOT_DEBUG)
201     if (prop != NULL) {
202         (void) uu_warn("Service %s property '%s/%s' "
203                       "not found.\n", FMRI_BOOT_CONFIG,
204                       BOOT_CONFIG_PG_PARAMS, prop->pv_prop);
205     } else {
206         (void) uu_warn("Unable to read service %s "
207                       "property '%s': %s\n", FMRI_BOOT_CONFIG,
208                       BOOT_CONFIG_PG_PARAMS, scf_strerror(scf_error()));
209     }
210 #endif /* FASTREBOOT_DEBUG */
211 }
212 }

214 /*
215  * Get or set properties in non-persistent "config_ovr" property group
216  * in svc:/system/boot-config:default.
217  * It prints errors with uu_warn().
218  */
219 /*ARGSUSED*/
220 static int
221 scf_getset_boot_config_ovr(int set, uint8_t *boot_config_ovr)
222 {
223     int rc = SCF_SUCCESS;

225     assert(boot_config_ovr);

227 #ifndef __x86
228     return (rc);
229 #else
230     {
231         /*
232          * Property vector for BOOT_CONFIG_PG_OVR property group.
233          */
234         scf_propvec_t ua_boot_config_ovr[] = {
235             { FASTREBOOT_DEFAULT, NULL, SCF_TYPE_BOOLEAN, NULL,
236               UA_FASTREBOOT_DEFAULT },
237             { FASTREBOOT_ONPANIC, NULL, SCF_TYPE_BOOLEAN, NULL,
238               UA_FASTREBOOT_ONPANIC },
239             { NULL }
240         };
241         scf_propvec_t *prop;

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

247         if (rc != SCF_SUCCESS) {
248             #if defined(FASTREBOOT_DEBUG)
249                 if (set)
250                     (void) uu_warn("Unable to add service %s "
251                                     "property group '%s'\n",
252                                     FMRI_BOOT_CONFIG, BOOT_CONFIG_PG_OVR);
253             #endif /* FASTREBOOT_DEBUG */
254             return (rc);
255         }

257         for (prop = ua_boot_config_ovr; prop->pv_prop != NULL; prop++)
258             prop->pv_ptr = boot_config_ovr;
259         prop = NULL;

```

```

261         if (set)
262             rc = scf_write_propvec(FMRI_BOOT_CONFIG,
263                                   BOOT_CONFIG_PG_OVR, ua_boot_config_ovr, &prop);
264         else
265             rc = scf_read_propvec(FMRI_BOOT_CONFIG,
266                                   BOOT_CONFIG_PG_OVR, B_FALSE, ua_boot_config_ovr,
267                                   &prop);
268
269 #if defined(FASTREBOOT_DEBUG)
270     if (rc != SCF_SUCCESS) {
271         if (prop != NULL) {
272             (void) uu_warn("Service %s property '%s/%s' "
273                           "not found.\n", FMRI_BOOT_CONFIG,
274                           BOOT_CONFIG_PG_OVR, prop->pv_prop);
275         } else {
276             (void) uu_warn("Unable to %s service %s "
277                           "property '%s': %s\n", set ? "set" : "get",
278                           FMRI_BOOT_CONFIG, BOOT_CONFIG_PG_OVR,
279                           scf_strerror(scf_error()));
280         }
281     }
282 #endif /* FASTREBOOT_DEBUG */
283
284     if (set)
285         (void) smf_refresh_instance(FMRI_BOOT_CONFIG);
286
287     return (rc);
288
289 }
290 #endif /* __x86 */
291
292
293 /*
294  * Get values of properties in non-persistent "config_ovr" property group.
295  */
296 void
297 scf_get_boot_config_ovr(uint8_t *boot_config_ovr)
298 {
299     (void) scf_getset_boot_config_ovr(B_FALSE, boot_config_ovr);
300 }
301
302 /*
303  * Set value of "config_ovr/fastreboot_default".
304  */
305 int
306 scf_fastreboot_default_set_transient(boolean_t value)
307 {
308     uint8_t boot_config_ovr = 0;
309
310     if (value == B_TRUE)
311         boot_config_ovr = UA_FASTREBOOT_DEFAULT | UA_FASTREBOOT_ONPANIC;
312
313     return (scf_getset_boot_config_ovr(B_TRUE, &boot_config_ovr));
314 }
315
316 /*
317  * Check whether Fast Reboot is the default operating mode.
318  * Return 0 if
319  * 1. the platform is xVM
320  * or
321  * 2. svc:/system/boot-config:default service doesn't exist,
322  * or
323  * 3. property "config/fastreboot_default" doesn't exist,
324  * or
325  * 4. value of property "config/fastreboot_default" is set to "false"

```

```

326  * and "config_ovr/fastreboot_default" is not set to "true",
327  * or
328  * 5. the platform has been blacklisted.
329  * or
330  * 6. value of property "config_ovr/fastreboot_default" is set to "false".
331  * Return non-zero otherwise.
332  */
333 int
334 scf_is_fastboot_default(void)
335 {
336     uint8_t boot_config = 0, boot_config_ovr;
337     char procbuf[SYS_NMLN];
338
339     /*
340      * If we are on xVM, do not fast reboot by default.
341      */
342     if (sysinfo(SI_PLATFORM, procbuf, sizeof (procbuf)) == -1 ||
343         strcmp(procbuf, "i86xpv") == 0)
344         return (0);
345
346     /*
347      * Get property values from "config" property group
348      */
349     scf_get_boot_config(&boot_config);
350
351     /*
352      * Get property values from non-persistent "config_ovr" property group
353      */
354     boot_config_ovr = boot_config;
355     scf_get_boot_config_ovr(&boot_config_ovr);
356
357     return (boot_config & boot_config_ovr & UA_FASTREBOOT_DEFAULT);
358 }
359
360 /*
361  * Read the default security-flags from system/process-security and return a
362  * secflagset_t suitable for psecflags(2)
363  */
364 /* Unfortunately, this symbol must exist in the native build, for the sake
365  * of the mapfile, even though we don't ever use it, and it will never work.
366  */
367 struct group_desc {
368     secflagdelta_t *delta;
369     secflagset_t *set;
370     char *fmri;
371 };
372
373 int
374 scf_default_secflags(scf_handle_t *hndl, scf_secflags_t *flags)
375 {
376     scf_default_secflags(scf_handle_t *hndl, psecflags_t *flags)
377 {
378     #if !defined(NATIVE_BUILD)
379     scf_property_t *prop;
380     scf_value_t *val;
381     const char *flagname;
382     int flag;
383     struct group_desc *g;
384     struct group_desc groups[] = {
385         {NULL, "svc:/system/process-security/"
386          "properties/default"},
387         {NULL, "svc:/system/process-security/"
388          "properties/lower"},
389         {NULL, "svc:/system/process-security/"
390          "properties/upper"},
391         {NULL, NULL}
392     };
393     #endif

```

```

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

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

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

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

406                 if ((prop = scf_property_create(hndl)) == NULL) {
407                     scf_value_destroy(val);
408                     return (-1);
409                 }

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

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

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

422                 (void) scf_value_get_boolean(val, &flagval);

424                 if (flagval != 0)
425                     secflag_set(&g->delta->psd_add, flag);
94                     secflag_set(g->set, flag);
426                 else
427                     secflag_set(&g->delta->psd_rem, flag);
96                     secflag_clear(g->set, flag);

429 next:
430                 uu_free(pfmri);
431                 scf_value_destroy(val);
432                 scf_property_destroy(prop);
433             }
434         }

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

436     return (0);
437 #else
438     assert(0);
439     abort();
440 #endif /* !NATIVE_BUILD */
441 }

```

unchanged_portion_omitted

new/usr/src/lib/libscf/inc/libscf.h

1

```
*****
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;
```

new/usr/src/lib/libscf/inc/libscf.h

2

```
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 /*
```



```

231 * Standard services
232 */
233 #define SCF_SERVICE_CONFIG ((const char *) \
234     "svc:/system/svc/repository:default")
235 #define SCF_INSTANCE_GLOBAL ((const char *) \
236     "svc:/system/svc/global:default")
237 #define SCF_SERVICE_GLOBAL ((const char *) \
238     "svc:/system/svc/global")
239 #define SCF_SERVICE_STARTD ((const char *) \
240     "svc:/system/svc/restarter:default")
241 #define SCF_INSTANCE_EMI ((const char *) \
242     "svc:/system/early-manifest-import:default")
243 #define SCF_INSTANCE_FS_MINIMAL ((const char *) \
244     "svc:/system/filesystem/minimal:default")
245 #define SCF_INSTANCE_MI ((const char *) \
246     "svc:/system/manifest-import:default")
247
248 /*
249 * Major milestones
250 */
251 #define SCF_MILESTONE_SINGLE_USER \
252     ((const char *) "svc:/milestone/single-user:default")
253 #define SCF_MILESTONE_MULTI_USER \
254     ((const char *) "svc:/milestone/multi-user:default")
255 #define SCF_MILESTONE_MULTI_USER_SERVER \
256     ((const char *) "svc:/milestone/multi-user-server:default")
257
258 /*
259 * standard scope names
260 */
261 #define SCF_SCOPE_LOCAL ((const char *) "localhost")
262
263 /*
264 * Property group types
265 */
266 #define SCF_GROUP_APPLICATION ((const char *) "application")
267 #define SCF_GROUP_FRAMEWORK ((const char *) "framework")
268 #define SCF_GROUP_DEPENDENCY ((const char *) "dependency")
269 #define SCF_GROUP_METHOD ((const char *) "method")
270 #define SCF_GROUP_TEMPLATE ((const char *) "template")
271 #define SCF_GROUP_TEMPLATE_PG_PATTERN ((const char *) "template_pg_pattern")
272 #define SCF_GROUP_TEMPLATE_PROP_PATTERN ((const char *) "template_prop_pattern")
273
274 /*
275 * Dependency types
276 */
277 #define SCF_DEP_REQUIRE_ALL ((const char *) "require_all")
278 #define SCF_DEP_REQUIRE_ANY ((const char *) "require_any")
279 #define SCF_DEP_EXCLUDE_ALL ((const char *) "exclude_all")
280 #define SCF_DEP_OPTIONAL_ALL ((const char *) "optional_all")
281
282 #define SCF_DEP_RESET_ON_ERROR ((const char *) "error")
283 #define SCF_DEP_RESET_ON_RESTART ((const char *) "restart")
284 #define SCF_DEP_RESET_ON_REFRESH ((const char *) "refresh")
285 #define SCF_DEP_RESET_ON_NONE ((const char *) "none")
286
287 /*
288 * Standard property group names
289 */
290 #define SCF_PG_GENERAL ((const char *) "general")
291 #define SCF_PG_GENERAL_OVR ((const char *) "general_ovr")
292 #define SCF_PG_RESTARTER ((const char *) "restarter")
293 #define SCF_PG_RESTARTER_ACTIONS ((const char *) "restarter_actions")
294 #define SCF_PG_METHOD_CONTEXT ((const char *) "method_context")
295 #define SCF_PG_APP_DEFAULT ((const char *) "application")
296 #define SCF_PG_DEPENDENTS ((const char *) "dependents")

```

```

297 #define SCF_PG_OPTIONS ((const char *) "options")
298 #define SCF_PG_OPTIONS_OVR ((const char *) "options_ovr")
299 #define SCF_PG_STARTD ((const char *) "startd")
300 #define SCF_PG_STARTD_PRIVATE ((const char *) "svc-startd-private")
301 #define SCF_PG_DEATHROW ((const char *) "deathrow")
302 #define SCF_PG_MANIFESTFILES ((const char *) "manifestfiles")
303
304 /*
305 * Template property group names and prefixes
306 */
307 #define SCF_PG_TM_COMMON_NAME ((const char *) "tm_common_name")
308 #define SCF_PG_TM_DESCRIPTION ((const char *) "tm_description")
309
310 #define SCF_PG_TM_MAN_PREFIX ((const char *) "tm_man_")
311 #define SCF_PG_TM_DOC_PREFIX ((const char *) "tm_doc_")
312
313 /*
314 * Standard property names
315 */
316 #define SCF_PROPERTY_ACTIVE_POSTFIX ((const char *) "active")
317 #define SCF_PROPERTY_AUX_STATE ((const char *) "auxiliary_state")
318 #define SCF_PROPERTY_AUX_FMRI ((const char *) "auxiliary_fmri")
319 #define SCF_PROPERTY_AUX_TTY ((const char *) "auxiliary_tty")
320 #define SCF_PROPERTY_CONTRACT ((const char *) "contract")
321 #define SCF_PROPERTY_COREFILE_PATTERN ((const char *) "corefile_pattern")
322 #define SCF_PROPERTY_DEGRADED ((const char *) "degraded")
323 #define SCF_PROPERTY_DEGRADE_IMMEDIATE ((const char *) "degrade_immediate")
324 #define SCF_PROPERTY_DODUMP ((const char *) "do_dump")
325 #define SCF_PROPERTY_DURATION ((const char *) "duration")
326 #define SCF_PROPERTY_ENABLED ((const char *) "enabled")
327 #define SCF_PROPERTY_DEATHROW ((const char *) "deathrow")
328 #define SCF_PROPERTY_ENTITY_STABILITY ((const char *) "entity_stability")
329 #define SCF_PROPERTY_ENTITIES ((const char *) "entities")
330 #define SCF_PROPERTY_EXEC ((const char *) "exec")
331 #define SCF_PROPERTY_GROUP ((const char *) "group")
332 #define SCF_PROPERTY_GROUPING ((const char *) "grouping")
333 #define SCF_PROPERTY_IGNORE ((const char *) "ignore_error")
334 #define SCF_PROPERTY_INTERNAL_SEPARATORS ((const char *) "internal_separators")
335 #define SCF_PROPERTY_LIMIT_PRIVILEGES ((const char *) "limit_privileges")
336 #define SCF_PROPERTY_MAINT_OFF ((const char *) "maint_off")
337 #define SCF_PROPERTY_MAINT_ON ((const char *) "maint_on")
338 #define SCF_PROPERTY_MAINT_ON_IMMEDIATE ((const char *) "maint_on_immediate")
339 #define SCF_PROPERTY_MAINT_ON_IMMTEMP ((const char *) "maint_on_immtemp")
340 #define SCF_PROPERTY_MAINT_ON_TEMPORARY ((const char *) "maint_on_temporary")
341 #define SCF_PROPERTY_METHOD_PID ((const char *) "method_pid")
342 #define SCF_PROPERTY_MILESTONE ((const char *) "milestone")
343 #define SCF_PROPERTY_NEED_SESSION ((const char *) "need_session")
344 #define SCF_PROPERTY_NEXT_STATE ((const char *) "next_state")
345 #define SCF_PROPERTY_PACKAGE ((const char *) "package")
346 #define SCF_PROPERTY_PRIVILEGES ((const char *) "privileges")
347 #define SCF_PROPERTY_PROFILE ((const char *) "profile")
348 #define SCF_PROPERTY_PROJECT ((const char *) "project")
349 #define SCF_PROPERTY_REFRESH ((const char *) "refresh")
350 #define SCF_PROPERTY_RESOURCE_POOL ((const char *) "resource_pool")
351 #define SCF_PROPERTY_ENVIRONMENT ((const char *) "environment")
352 #define SCF_PROPERTY_RESTART ((const char *) "restart")
353 #define SCF_PROPERTY_RESTARTER ((const char *) "restarter")
354 #define SCF_PROPERTY_RESTART_INTERVAL ((const char *) "restart_interval")
355 #define SCF_PROPERTY_RESTART_ON ((const char *) "restart_on")
356 #define SCF_PROPERTY_RESTORE ((const char *) "restore")
357 #define SCF_PROPERTY_SECFLAGS ((const char *) "security_flags")
358 #define SCF_PROPERTY_SINGLE_INSTANCE ((const char *) "single_instance")
359 #define SCF_PROPERTY_START_METHOD_TIMESTAMP \
360     ((const char *) "start_method_timestamp")
361 #define SCF_PROPERTY_START_METHOD_WAITSTATUS \
362     ((const char *) "start_method_waitstatus")

```

```

363 #define SCF_PROPERTY_START_PID ((const char *)"start_pid")
364 #define SCF_PROPERTY_STATE ((const char *)"state")
365 #define SCF_PROPERTY_STABILITY ((const char *)"stability")
366 #define SCF_PROPERTY_STATE_TIMESTAMP ((const char *)"state_timestamp")
367 #define SCF_PROPERTY_SUPP_GROUPS ((const char *)"supp_groups")
368 #define SCF_PROPERTY_TIMEOUT ((const char *)"timeout_seconds")
369 #define SCF_PROPERTY_TIMEOUT_RETRY ((const char *)"timeout_retry")
370 #define SCF_PROPERTY_TRANSIENT_CONTRACT ((const char *)"transient_contract")
371 #define SCF_PROPERTY_TYPE ((const char *)"type")
372 #define SCF_PROPERTY_USE_PROFILE ((const char *)"use_profile")
373 #define SCF_PROPERTY_USER ((const char *)"user")
374 #define SCF_PROPERTY_UTMPX_PREFIX ((const char *)"utmpx_prefix")
375 #define SCF_PROPERTY_WORKING_DIRECTORY ((const char *)"working_directory")

377 /*
378 * Template property names
379 */
380 #define SCF_PROPERTY_TM_CARDINALITY_MIN ((const char *)"cardinality_min")
381 #define SCF_PROPERTY_TM_CARDINALITY_MAX ((const char *)"cardinality_max")
382 #define SCF_PROPERTY_TM_CHOICES_INCLUDE_VALUES ((const char *) \
383     "choices_include_values")
384 #define SCF_PROPERTY_TM_CHOICES_NAME ((const char *)"choices_name")
385 #define SCF_PROPERTY_TM_CHOICES_RANGE ((const char *)"choices_range")
386 #define SCF_PROPERTY_TM_CONSTRAINT_NAME ((const char *)"constraint_name")
387 #define SCF_PROPERTY_TM_CONSTRAINT_RANGE ((const char *)"constraint_range")
388 #define SCF_PROPERTY_TM_MANPATH ((const char *)"manpath")
389 #define SCF_PROPERTY_TM_NAME ((const char *)"name")
390 #define SCF_PROPERTY_TM_PG_PATTERN ((const char *)"pg_pattern")
391 #define SCF_PROPERTY_TM_REQUIRED ((const char *)"required")
392 #define SCF_PROPERTY_TM_SECTION ((const char *)"section")
393 #define SCF_PROPERTY_TM_TARGET ((const char *)"target")
394 #define SCF_PROPERTY_TM_TITLE ((const char *)"title")
395 #define SCF_PROPERTY_TM_TYPE ((const char *)"type")
396 #define SCF_PROPERTY_TM_URI ((const char *)"uri")
397 #define SCF_PROPERTY_TM_VALUE_PREFIX ((const char *)"value_")
398 #define SCF_PROPERTY_TM_VALUES_NAME ((const char *)"values_name")
399 #define SCF_PROPERTY_TM_VISIBILITY ((const char *)"visibility")
400 #define SCF_PROPERTY_TM_COMMON_NAME_PREFIX ((const char *)"common_name_")
401 #define SCF_PROPERTY_TM_DESCRIPTION_PREFIX ((const char *)"description_")
402 #define SCF_PROPERTY_TM_UNITS_PREFIX ((const char *)"units_")

404 /*
405 * Templates wildcard string
406 */
407 #define SCF_TMPL_WILDCARD ((const char *)"")

409 /*
410 * Strings used by restarters for state and next_state properties.
411 * MAX_SCF_STATE_STRING holds the max length of a state string, including the
412 * terminating null.
413 */

415 #define MAX_SCF_STATE_STRING_SZ 14

417 #define SCF_STATE_STRING_NONE ((const char *)"none")
418 #define SCF_STATE_STRING_UNINIT ((const char *)"uninitialized")
419 #define SCF_STATE_STRING_MAINT ((const char *)"maintenance")
420 #define SCF_STATE_STRING_OFFLINE ((const char *)"offline")
421 #define SCF_STATE_STRING_DISABLED ((const char *)"disabled")
422 #define SCF_STATE_STRING_ONLINE ((const char *)"online")
423 #define SCF_STATE_STRING_DEGRADED ((const char *)"degraded")
424 #define SCF_STATE_STRING_LEGACY ((const char *)"legacy_run")

426 #define SCF_STATE_UNINIT 0x00000001
427 #define SCF_STATE_MAINT 0x00000002
428 #define SCF_STATE_OFFLINE 0x00000004

```

```

429 #define SCF_STATE_DISABLED 0x00000008
430 #define SCF_STATE_ONLINE 0x00000010
431 #define SCF_STATE_DEGRADED 0x00000020
432 #define SCF_STATE_ALL 0x0000003F

434 /*
435 * software fma svc-transition class
436 */
437 #define SCF_NOTIFY_PARAMS_VERSION 0X0
438 #define SCF_NOTIFY_NAME_FMRI ((const char *)"fmri")
439 #define SCF_NOTIFY_NAME_VERSION ((const char *)"version")
440 #define SCF_NOTIFY_NAME_TSET ((const char *)"tset")
441 #define SCF_NOTIFY_PG_POSTFIX ((const char *)"fmnotify")
442 #define SCF_NOTIFY_PARAMS ((const char *)"notify_params")
443 #define SCF_NOTIFY_PARAMS_INST \
444     ((const char *)"svc:/system/fm/notify-params:default")
445 #define SCF_SVC_TRANSITION_CLASS \
446     ((const char *)"ireport.os.smf.state-transition")
447 #define SCF_NOTIFY_PARAMS_PG_TYPE ((const char *)"notify_params")

449 /*
450 * Useful transition macros
451 */
452 #define SCF_TRANS_SHIFT_INITIAL_STATE(s) ((s) << 16)
453 #define SCF_TRANSITION_ALL \
454     (SCF_TRANS_SHIFT_INITIAL_STATE(SCF_STATE_ALL) | SCF_STATE_ALL)
455 #define SCF_TRANS(f, t) (SCF_TRANS_SHIFT_INITIAL_STATE(f) | (t))
456 #define SCF_TRANS_VALID(t) (!(t) & ~SCF_TRANSITION_ALL)
457 #define SCF_TRANS_INITIAL_STATE(t) ((t) >> 16 & SCF_STATE_ALL)
458 #define SCF_TRANS_FINAL_STATE(t) ((t) & SCF_STATE_ALL)

460 /*
461 * Prefixes for states in state transition notification
462 */
463 #define SCF_STN_PREFIX_FROM ((const char *)"from-")
464 #define SCF_STN_PREFIX_TO ((const char *)"to-")

466 #define SCF_PG_FLAG_NONPERSISTENT 0x1

468 #define SCF_TRACE_LIBRARY 0x1
469 #define SCF_TRACE_DAEMON 0x2

471 #define SMF_IMMEDIATE 0x1
472 #define SMF_TEMPORARY 0x2
473 #define SMF_AT_NEXT_BOOT 0x4

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

478 ssize_t scf_limit(uint32_t code);
479 #define SCF_LIMIT_MAX_NAME_LENGTH -2000U
480 #define SCF_LIMIT_MAX_VALUE_LENGTH -2001U
481 #define SCF_LIMIT_MAX_PG_TYPE_LENGTH -2002U
482 #define SCF_LIMIT_MAX_FMRI_LENGTH -2003U

484 scf_handle_t *scf_handle_create(scf_version_t);

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

489 int scf_handle_bind(scf_handle_t *);
490 int scf_handle_unbind(scf_handle_t *);
491 void scf_handle_destroy(scf_handle_t *);

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

```

```

495 scf_type_t scf_string_to_type(const char *);

497 /* values */
498 scf_value_t *scf_value_create(scf_handle_t *);
499 scf_handle_t *scf_value_handle(const scf_value_t *);
500 void scf_value_destroy(scf_value_t *);

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

506 void scf_value_reset(scf_value_t *);

508 int scf_value_get_boolean(const scf_value_t *, uint8_t *);
509 int scf_value_get_count(const scf_value_t *, uint64_t *);
510 int scf_value_get_integer(const scf_value_t *, int64_t *);
511 int scf_value_get_time(const scf_value_t *, int64_t *, int32_t *);
512 ssize_t scf_value_get_astring(const scf_value_t *, char *, size_t);
513 ssize_t scf_value_get_ustring(const scf_value_t *, char *, size_t);
514 ssize_t scf_value_get_opaque(const scf_value_t *, void *, size_t);

516 void scf_value_set_boolean(scf_value_t *, uint8_t);
517 void scf_value_set_count(scf_value_t *, uint64_t);
518 void scf_value_set_integer(scf_value_t *, int64_t);
519 int scf_value_set_time(scf_value_t *, int64_t, int32_t);
520 int scf_value_set_astring(scf_value_t *, const char *);
521 int scf_value_set_ustring(scf_value_t *, const char *);
522 int scf_value_set_opaque(scf_value_t *, const void *, size_t);

524 ssize_t scf_value_get_as_string(const scf_value_t *, char *, size_t);
525 ssize_t scf_value_get_as_string_typed(const scf_value_t *, scf_type_t,
526 char *, size_t);
527 int scf_value_set_from_string(scf_value_t *, scf_type_t, const char *);

529 scf_iter_t *scf_iter_create(scf_handle_t *);
530 scf_handle_t *scf_iter_handle(const scf_iter_t *);
531 void scf_iter_reset(scf_iter_t *);
532 void scf_iter_destroy(scf_iter_t *);

534 int scf_iter_handle_scopes(scf_iter_t *, const scf_handle_t *);
535 int scf_iter_scope_services(scf_iter_t *, const scf_scope_t *);
536 int scf_iter_service_instances(scf_iter_t *, const scf_service_t *);
537 int scf_iter_service_pgs(scf_iter_t *, const scf_service_t *);
538 int scf_iter_instance_pgs(scf_iter_t *, const scf_instance_t *);
539 int scf_iter_instance_pgs_composed(scf_iter_t *, const scf_instance_t *,
540 const scf_snapshot_t *);
541 int scf_iter_service_pgs_typed(scf_iter_t *, const scf_service_t *,
542 const char *);
543 int scf_iter_instance_pgs_typed(scf_iter_t *, const scf_instance_t *,
544 const char *);
545 int scf_iter_instance_pgs_typed_composed(scf_iter_t *, const scf_instance_t *,
546 const scf_snapshot_t *, const char *);
547 int scf_iter_snaplevel_pgs(scf_iter_t *, const scf_snaplevel_t *);
548 int scf_iter_snaplevel_pgs_typed(scf_iter_t *, const scf_snaplevel_t *,
549 const char *);
550 int scf_iter_instance_snapshots(scf_iter_t *, const scf_instance_t *);
551 int scf_iter_pg_properties(scf_iter_t *, const scf_propertygroup_t *);
552 int scf_iter_property_values(scf_iter_t *, const scf_property_t *);

554 int scf_iter_next_scope(scf_iter_t *, scf_scope_t *);
555 int scf_iter_next_service(scf_iter_t *, scf_service_t *);
556 int scf_iter_next_instance(scf_iter_t *, scf_instance_t *);
557 int scf_iter_next_pg(scf_iter_t *, scf_propertygroup_t *);
558 int scf_iter_next_property(scf_iter_t *, scf_property_t *);
559 int scf_iter_next_snapshot(scf_iter_t *, scf_snapshot_t *);
560 int scf_iter_next_value(scf_iter_t *, scf_value_t *);

```

```

562 scf_scope_t *scf_scope_create(scf_handle_t *);
563 scf_handle_t *scf_scope_handle(const scf_scope_t *);

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

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

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

575 scf_service_t *scf_service_create(scf_handle_t *);
576 scf_handle_t *scf_service_handle(const scf_service_t *);
577 void scf_service_destroy(scf_service_t *);
578 int scf_scope_get_parent(const scf_scope_t *, scf_scope_t *);
579 ssize_t scf_service_get_name(const scf_service_t *, char *, size_t);
580 ssize_t scf_service_to_fmri(const scf_service_t *, char *, size_t);
581 int scf_service_get_parent(const scf_service_t *, scf_scope_t *);
582 int scf_scope_get_service(const scf_scope_t *, const char *, scf_service_t *);
583 int scf_scope_add_service(const scf_scope_t *, const char *, scf_service_t *);
584 int scf_service_delete(scf_service_t *);

586 scf_instance_t *scf_instance_create(scf_handle_t *);
587 scf_handle_t *scf_instance_handle(const scf_instance_t *);
588 void scf_instance_destroy(scf_instance_t *);
589 ssize_t scf_instance_get_name(const scf_instance_t *, char *, size_t);
590 ssize_t scf_instance_to_fmri(const scf_instance_t *, char *, size_t);
591 int scf_service_get_instance(const scf_service_t *, const char *,
592 scf_instance_t *);
593 int scf_service_add_instance(const scf_service_t *, const char *,
594 scf_instance_t *);
595 int scf_instance_delete(scf_instance_t *);

597 scf_snapshot_t *scf_snapshot_create(scf_handle_t *);
598 scf_handle_t *scf_snapshot_handle(const scf_snapshot_t *);
599 void scf_snapshot_destroy(scf_snapshot_t *);
600 ssize_t scf_snapshot_get_name(const scf_snapshot_t *, char *, size_t);
601 int scf_snapshot_get_parent(const scf_snapshot_t *, scf_instance_t *);
602 int scf_instance_get_snapshot(const scf_instance_t *, const char *,
603 scf_snapshot_t *);
604 int scf_snapshot_update(scf_snapshot_t *);

606 scf_snaplevel_t *scf_snaplevel_create(scf_handle_t *);
607 scf_handle_t *scf_snaplevel_handle(const scf_snaplevel_t *);
608 void scf_snaplevel_destroy(scf_snaplevel_t *);
609 int scf_snaplevel_get_parent(const scf_snaplevel_t *, scf_snapshot_t *);
610 ssize_t scf_snaplevel_get_scope_name(const scf_snaplevel_t *, char *, size_t);
611 ssize_t scf_snaplevel_get_service_name(const scf_snaplevel_t *, char *, size_t);
612 ssize_t scf_snaplevel_get_instance_name(const scf_snaplevel_t *, char *,
613 size_t);
614 int scf_snaplevel_get_pg(const scf_snaplevel_t *, const char *,
615 scf_propertygroup_t *);
616 int scf_snapshot_get_base_snaplevel(const scf_snapshot_t *, scf_snaplevel_t *);
617 int scf_snaplevel_get_next_snaplevel(const scf_snaplevel_t *,
618 scf_snaplevel_t *);

620 scf_propertygroup_t *scf_pg_create(scf_handle_t *);
621 scf_handle_t *scf_pg_handle(const scf_propertygroup_t *);
622 void scf_pg_destroy(scf_propertygroup_t *);
623 ssize_t scf_pg_to_fmri(const scf_propertygroup_t *, char *, size_t);
624 ssize_t scf_pg_get_name(const scf_propertygroup_t *, char *, size_t);
625 ssize_t scf_pg_get_type(const scf_propertygroup_t *, char *, size_t);
626 int scf_pg_get_flags(const scf_propertygroup_t *, uint32_t *);

```

```

627 int scf_pg_get_parent_service(const scf_propertygroup_t *, scf_service_t *);
628 int scf_pg_get_parent_instance(const scf_propertygroup_t *, scf_instance_t *);
629 int scf_pg_get_parent_snaplevel(const scf_propertygroup_t *, scf_snaplevel_t *);
630 int scf_service_get_pg(const scf_service_t *, const char *,
631     scf_propertygroup_t *);
632 int scf_instance_get_pg(const scf_instance_t *, const char *,
633     scf_propertygroup_t *);
634 int scf_instance_get_pg_composed(const scf_instance_t *, const scf_snapshot_t *,
635     const char *, scf_propertygroup_t *);
636 int scf_service_add_pg(const scf_service_t *, const char *, const char *,
637     uint32_t, scf_propertygroup_t *);
638 int scf_instance_add_pg(const scf_instance_t *, const char *, const char *,
639     uint32_t, scf_propertygroup_t *);
640 int scf_pg_delete(scf_propertygroup_t *);

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

646 int scf_pg_update(scf_propertygroup_t *);

648 scf_property_t *scf_property_create(scf_handle_t *);
649 scf_handle_t *scf_property_handle(const scf_property_t *);
650 void scf_property_destroy(scf_property_t *);
651 int scf_property_is_type(const scf_property_t *, scf_type_t);
652 int scf_property_type(const scf_property_t *, scf_type_t *);
653 ssize_t scf_property_get_name(const scf_property_t *, char *, size_t);
654 int scf_property_get_value(const scf_property_t *, scf_value_t *);
655 ssize_t scf_property_to_fmri(const scf_property_t *, char *, size_t);
656 int scf_pg_get_property(const scf_propertygroup_t *, const char *,
657     scf_property_t *);

659 scf_transaction_t *scf_transaction_create(scf_handle_t *);
660 scf_handle_t *scf_transaction_handle(const scf_transaction_t *);
661 int scf_transaction_start(scf_transaction_t *, scf_propertygroup_t *);
662 void scf_transaction_destroy(scf_transaction_t *);
663 void scf_transaction_destroy_children(scf_transaction_t *);

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

668 int scf_transaction_commit(scf_transaction_t *);

670 scf_transaction_entry_t *scf_entry_create(scf_handle_t *);
671 scf_handle_t *scf_entry_handle(const scf_transaction_entry_t *);
672 void scf_entry_reset(scf_transaction_entry_t *);
673 void scf_entry_destroy(scf_transaction_entry_t *);
674 void scf_entry_destroy_children(scf_transaction_entry_t *);

676 int scf_transaction_property_change(scf_transaction_t *,
677     scf_transaction_entry_t *, const char *, scf_type_t);
678 int scf_transaction_property_delete(scf_transaction_t *,
679     scf_transaction_entry_t *, const char *);
680 int scf_transaction_property_new(scf_transaction_t *,
681     scf_transaction_entry_t *, const char *, scf_type_t);
682 int scf_transaction_property_change_type(scf_transaction_t *,
683     scf_transaction_entry_t *, const char *, scf_type_t);

685 int scf_entry_add_value(scf_transaction_entry_t *, scf_value_t *);

687 int scf_handle_decode_fmri(scf_handle_t *, const char *, scf_scope_t *,
688     scf_service_t *, scf_instance_t *, scf_propertygroup_t *, scf_property_t *,
689     int);
690 #define SCF_DECODE_FMRI_EXACT          0x00000001
691 #define SCF_DECODE_FMRI_TRUNCATE      0x00000002
692 #define SCF_DECODE_FMRI_REQUIRE_INSTANCE 0x00000004

```

```

693 #define SCF_DECODE_FMRI_REQUIRE_NO_INSTANCE 0x00000008

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

697 /*
698  * Property group template interfaces.
699  */
700 scf_pg_tmpl_t *scf_tmpl_pg_create(scf_handle_t *);
701 void scf_tmpl_pg_destroy(scf_pg_tmpl_t *);
702 void scf_tmpl_pg_reset(scf_pg_tmpl_t *);
703 int scf_tmpl_get_by_pg(scf_propertygroup_t *, scf_pg_tmpl_t *, int);
704 int scf_tmpl_get_by_pg_name(const char *, const char *,
705     const char *, const char *, scf_pg_tmpl_t *, int);
706 int scf_tmpl_iter_pgs(scf_pg_tmpl_t *, const char *, const char *,
707     const char *, int);
708 #define SCF_PG_TMPL_FLAG_REQUIRED      0x1
709 #define SCF_PG_TMPL_FLAG_EXACT        0x2
710 #define SCF_PG_TMPL_FLAG_CURRENT      0x4

712 ssize_t scf_tmpl_pg_name(const scf_pg_tmpl_t *, char **);
713 ssize_t scf_tmpl_pg_common_name(const scf_pg_tmpl_t *, const char *, char **);
714 ssize_t scf_tmpl_pg_description(const scf_pg_tmpl_t *, const char *, char **);
715 ssize_t scf_tmpl_pg_type(const scf_pg_tmpl_t *, char **);

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

723 int scf_tmpl_pg_required(const scf_pg_tmpl_t *, uint8_t *);

725 /*
726  * Property template interfaces.
727  */
728 scf_prop_tmpl_t *scf_tmpl_prop_create(scf_handle_t *);
729 void scf_tmpl_prop_destroy(scf_prop_tmpl_t *);
730 void scf_tmpl_prop_reset(scf_prop_tmpl_t *);
731 int scf_tmpl_get_by_prop(scf_pg_tmpl_t *, const char *,
732     scf_prop_tmpl_t *, int);
733 int scf_tmpl_iter_props(scf_pg_tmpl_t *, scf_prop_tmpl_t *, int);
734 #define SCF_PROP_TMPL_FLAG_REQUIRED    0x1

736 ssize_t scf_tmpl_prop_name(const scf_prop_tmpl_t *, char **);
737 int scf_tmpl_prop_type(const scf_prop_tmpl_t *, scf_type_t *);
738 int scf_tmpl_prop_required(const scf_prop_tmpl_t *, uint8_t *);
739 ssize_t scf_tmpl_prop_common_name(const scf_prop_tmpl_t *, const char *,
740     char **);
741 ssize_t scf_tmpl_prop_description(const scf_prop_tmpl_t *, const char *,
742     char **);
743 ssize_t scf_tmpl_prop_units(const scf_prop_tmpl_t *, const char *, char **);
744 int scf_tmpl_prop_cardinality(const scf_prop_tmpl_t *prop, uint64_t *,
745     uint64_t *);
746 int scf_tmpl_prop_internal_seps(const scf_prop_tmpl_t *, scf_values_t *);

748 int scf_tmpl_prop_visibility(const scf_prop_tmpl_t *, uint8_t *);
749 #define SCF_TM_VISIBILITY_HIDDEN      1
750 #define SCF_TM_VISIBILITY_READONLY    2
751 #define SCF_TM_VISIBILITY_READWRITE   3

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

758 int scf_tmpl_value_name_constraints(const scf_prop_tmpl_t *prop,

```

```

759     scf_values_t *vals);
760 void scf_count_ranges_destroy(scf_count_ranges_t *);
761 void scf_int_ranges_destroy(scf_int_ranges_t *);
762 int scf_tmpl_value_count_range_constraints(const scf_prop_tmpl_t *,
763     scf_count_ranges_t *);
764 int scf_tmpl_value_int_range_constraints(const scf_prop_tmpl_t *,
765     scf_int_ranges_t *);
766 int scf_tmpl_value_count_range_choices(const scf_prop_tmpl_t *,
767     scf_count_ranges_t *);
768 int scf_tmpl_value_int_range_choices(const scf_prop_tmpl_t *,
769     scf_int_ranges_t *);
770 int scf_tmpl_value_name_choices(const scf_prop_tmpl_t *prop,
771     scf_values_t *vals);

773 void scf_values_destroy(scf_values_t *);

775 ssize_t scf_tmpl_value_common_name(const scf_prop_tmpl_t *, const char *,
776     const char *, char **);
777 ssize_t scf_tmpl_value_description(const scf_prop_tmpl_t *, const char *,
778     const char *, char **);

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

783 /*
784  * Template validation interfaces
785  */
786 int scf_tmpl_validate_fmri(scf_handle_t *, const char *,
787     const char *, scf_tmpl_errors_t **, int);
788 #define SCF_TMPL_VALIDATE_FLAG_CURRENT 0x1

790 void scf_tmpl_errors_destroy(scf_tmpl_errors_t *errs);
791 scf_tmpl_error_t *scf_tmpl_next_error(scf_tmpl_errors_t *);
792 void scf_tmpl_reset_errors(scf_tmpl_errors_t *errs);
793 int scf_tmpl_strerror(scf_tmpl_error_t *err, char *s, size_t n, int flag);
794 int scf_tmpl_error_source_fmri(const scf_tmpl_error_t *, char **);
795 int scf_tmpl_error_type(const scf_tmpl_error_t *, scf_tmpl_error_type_t *);
796 int scf_tmpl_error_pg_tmpl(const scf_tmpl_error_t *, char **, char **);
797 int scf_tmpl_error_pg(const scf_tmpl_error_t *, char **, char **);
798 int scf_tmpl_error_prop_tmpl(const scf_tmpl_error_t *, char **, char **);
799 int scf_tmpl_error_prop(const scf_tmpl_error_t *, char **, char **);
800 int scf_tmpl_error_value(const scf_tmpl_error_t *, char **);

802 /*
803  * Simplified calls
804  */
805 int smf_enable_instance(const char *, int);
806 int smf_disable_instance(const char *, int);
807 int smf_refresh_instance(const char *);
808 int smf_restart_instance(const char *);
809 int smf_maintain_instance(const char *, int);
810 int smf_degrade_instance(const char *, int);
811 int smf_restore_instance(const char *);
812 char *smf_get_state(const char *);

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

817 scf_simple_prop_t *scf_simple_prop_get(scf_handle_t *, const char *,
818     const char *, const char *);
819 void scf_simple_prop_free(scf_simple_prop_t *);
820 scf_simple_app_props_t *scf_simple_app_props_get(scf_handle_t *, const char *);
821 void scf_simple_app_props_free(scf_simple_app_props_t *);
822 const scf_simple_prop_t *scf_simple_app_props_next(
823     const scf_simple_app_props_t *, scf_simple_prop_t *);
824 const scf_simple_prop_t *scf_simple_app_props_search(

```

```

825     const scf_simple_app_props_t *, const char *, const char *);
826 ssize_t scf_simple_prop_numvalues(const scf_simple_prop_t *);
827 scf_type_t scf_simple_prop_type(const scf_simple_prop_t *);
828 char *scf_simple_prop_name(const scf_simple_prop_t *);
829 char *scf_simple_prop_pname(const scf_simple_prop_t *);
830 uint8_t *scf_simple_prop_next_boolean(scf_simple_prop_t *);
831 uint64_t *scf_simple_prop_next_count(scf_simple_prop_t *);
832 int64_t *scf_simple_prop_next_integer(scf_simple_prop_t *);
833 int64_t *scf_simple_prop_next_time(scf_simple_prop_t *, int32_t *);
834 char *scf_simple_prop_next_astring(scf_simple_prop_t *);
835 char *scf_simple_prop_next_ustring(scf_simple_prop_t *);
836 void *scf_simple_prop_next_opaque(scf_simple_prop_t *, size_t *);
837 void scf_simple_prop_next_reset(scf_simple_prop_t *);

839 /*
840  * smf_state_from_string()
841  * return SCF_STATE_* value for the input
842  * -1 on error. String "all" maps to SCF_STATE_ALL macro
843  */
844 int32_t smf_state_from_string(const char *);

846 /*
847  * smf_state_to_string()
848  * return SCF_STATE_STRING* value for the input
849  * NULL on error.
850  */
851 const char *smf_state_to_string(int32_t);

853 /*
854  * Notification interfaces
855  */
856 int smf_notify_set_params(const char *, nvlist_t *);
857 int smf_notify_get_params(nvlist_t **, nvlist_t *);
858 int smf_notify_del_params(const char *, const char *, int32_t);

860 /*
861  * SMF exit status definitions
862  */
863 #define SMF_EXIT_OK 0
864 #define SMF_EXIT_ERR_FATAL 95
865 #define SMF_EXIT_ERR_CONFIG 96
866 #define SMF_EXIT_MON_DEGRADE 97
867 #define SMF_EXIT_MON_OFFLINE 98
868 #define SMF_EXIT_ERR_NOSMF 99
869 #define SMF_EXIT_ERR_PERM 100

871 #ifdef __cplusplus
872 }
873 #endif

875 #endif /* _LIBSCF_H */

```

new/usr/src/lib/libscf/inc/libscf_priv.h

1

```
*****
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_____

478 /*
479 * The pg_pattern element has two optional attributes that play a part in
480 * selecting the appropriate prefix for the name of the pg_pattern property
481 * group. The two attributes are name and type. The appropriate prefix
482 * encodes the presence or absence of these attributes.
483 *
484 *      SCF_PG_TM_PG_PATTERN_PREFIX      neither attribute
485 *      SCF_PG_TM_PG_PATTERN_N_PREFIX    name only
486 *      SCF_PG_TM_PG_PATTERN_T_PREFIX    type only
487 *      SCF_PG_TM_PG_PATTERN_NT_PREFIX   both name and type
488 */
489 #define SCF_PG_TM_PG_PAT_BASE            "tm_pgpat"
490 #define SCF_PG_TM_PG_PATTERN_PREFIX      ((const char *)SCF_PG_TM_PG_PAT_BASE \
491      " ")
492 #define SCF_PG_TM_PG_PATTERN_N_PREFIX    ((const char *)SCF_PG_TM_PG_PAT_BASE \
493      "n ")
494 #define SCF_PG_TM_PG_PATTERN_T_PREFIX    ((const char *)SCF_PG_TM_PG_PAT_BASE \
495      "t ")
496 #define SCF_PG_TM_PG_PATTERN_NT_PREFIX   ((const char *)SCF_PG_TM_PG_PAT_BASE \
497      "nt ")
498 #define SCF_PG_TM_PROP_PATTERN_PREFIX    ((const char *)"tm_proppat_")

500 /*
501 * Pad character to use when encoding strings for property names.
502 */
503 #define SCF_ENCODE32_PAD                  ('-')

505 /*
506 * Functions for base 32 encoding/decoding
507 */
508 int scf_decode32(const char *, size_t, char *, size_t, size_t *, char);
509 int scf_encode32(const char *, size_t, char *, size_t, size_t *, char);

511 /*
512 * handy functions
513 */
514 /*
515 * _scf_sanitize_locale
516 * Make sure a locale string has only alpha-numeric or '_' characters
517 */
518 void _scf_sanitize_locale(char *);

520 /*
521 * _scf_read_tmpl_prop_type_as_string()
522 * Handy function to get template property type as a string
523 */
524 char *_scf_read_tmpl_prop_type_as_string(const scf_prop_tmpl_t *);
525 /*
526 * _scf_read_single_astring_from_pg()
527 * Given a property group (pg) and a property name (pn), this function
528 * retrieves an astring value from pg/pn.
529 */
530 char *_scf_read_single_astring_from_pg(scf_propertygroup_t *, const char *);

532 /*
533 * scf_instance_delete_prop()
534 * Given instance, property group, and property, delete the property.
535 */
536 int
```

new/usr/src/lib/libscf/inc/libscf_priv.h

2

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

539 /*
540 * Functions to extract boot config information from FMRI_BOOT_CONFIG
541 */
542 void scf_get_boot_config(uint8_t *);
543 void scf_get_boot_config_ovr(uint8_t *);
544 int scf_is_fastboot_default(void);

546 /*
547 * Set value of "config_ovr/fastreboot_default".
548 */
549 int scf_fastreboot_default_set_transient(boolean_t);

551 /*
552 * scf_is_compatible_type()
553 * Return true if the second type is the same type, or a subtype of the
554 * first.
555 */
556 int scf_is_compatible_type(scf_type_t, scf_type_t);

558 /*
559 * Check an array of services and enable any that don't have the
560 * "application/auto_enable" property set to "false", which is
561 * the interface to turn off this behaviour (see PSARC 2004/739).
562 */
563 void _check_services(char **);

565 /*
566 * _scf_handle_create_and_bind()
567 * convenience function that creates and binds a handle
568 */
569 scf_handle_t *_scf_handle_create_and_bind(scf_version_t);

571 /*
572 * _smf_refresh_all_instances()
573 * refresh all instances of a service
574 * return SCF_SUCCESS or SCF_FAILED on _PERMISSION_DENIED, _BACKEND_ACCESS
575 * or _BACKEND_READONLY.
576 */
577 int _smf_refresh_all_instances(scf_service_t *);

579 /*
580 * _scf_get_fma_notify_params()
581 * Specialized function to get fma notification parameters
582 */
583 int _scf_get_fma_notify_params(const char *, nvlist_t *, int);

585 /*
586 * _scf_get_svc_notify_params()
587 * Specialized function to get SMF state transition notification parameters
588 */
589 int _scf_get_svc_notify_params(const char *, nvlist_t *, int32_t, int, int);

591 /*
592 * _scf_notify_get_params()
593 * Specialized function to get notification parameters from a pg into an
594 * nvlist_t
595 */
596 int _scf_notify_get_params(scf_propertygroup_t *, nvlist_t *);

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

new/usr/src/lib/libscf/inc/libscf_priv.h

3

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

```
604 #ifdef __cplusplus
```

```
605 }
```

```
_____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
*****
1  \.
2  \. This file and its contents are supplied under the terms of the
3  \. Common Development and Distribution License ("CDDL"), version 1.0.
4  \. You may only use this file in accordance with the terms of version
5  \. 1.0 of the CDDL.
6  \.
7  \. A full copy of the text of the CDDL should have accompanied this
8  \. source. A copy of the CDDL is also available via the Internet at
9  \. http://www.illumos.org/license/CDDL.
10 \.
11 \. Copyright 2015, Richard Lowe.
12 \.
13 .TH "SECURITY-FLAGS" "5" "June 6, 2016"
14 .SH "NAME"
15 \fBsecurity-flags\fR - process security flags
16 .SH "DESCRIPTION"
17 Each process on an illumos system has an associated set of security-flags
18 which describe additional per-process security and exploit mitigation
19 features which are enabled for that process.
20 .P
21 There are four sets of these flags for each process, the effective set
22 (abbreviated \fIE\fR) are the set which currently apply to the process and are
23 immutable. The inheritable set (abbreviated \fII\fR) are the flags which will
24 become effective the next time the process calls one of the \fBexec(2)\fR
25 family of functions, and will be inherited as both the effective and
26 inheritable sets by any child processes. The upper set (abbreviated \fIU\fR)
27 specify the maximal flags that a process can have in its inheritable set. The
28 lower set (abbreviated \fIL\fR) specify the minimal amount of flags that a
29 process must have in its inheritable set. The inheritable set may be changed
30 at any time, subject to permissions and the lower and upper sets.
31 .P
32 To change the security-flags of a process one must have both permissions
33 equivalent to those required to send a signal to the process and have the
34 \fBPRIV_PROC_SECFRAGS\fR privilege.
35 .P
36 Currently available features are:
37
38 .sp
39 .ne 2
40 .na
41 Address Space Layout Randomisation (\fBASLR\fR)
42 .ad
43 .RS 11n
44 The base addresses of the stack, heap and shared library (including
45 \fBld.so\fR) mappings are randomised, the bases of mapped regions other than
46 those using \fBMAP_FIXED\fR are randomised.
47 .P
48 Currently, executable base addresses are \fInot\fR randomised, due to which
49 the mitigation provided by this feature is currently limited.
50 .P
51 This flag may also be enabled by the presence of the \fBBDT_SUNW_ASRL\fR
52 dynamic tag in the \fBdynamic\fR section of the executable file. If this
53 tag has a value of 1, ASLR will be enabled. If the flag has a value of
54 \fB0\fR ASLR will be disabled. If the tag is not present, the value of the
55 ASLR flag will be inherited as normal.
56 .RE
57
58 .sp
59 .ne 2
60 .na
61 Forbid mappings at NULL (\fBFORBIDNULLMAP\fR)

```

```

62 .ad
63 .RS 11n
64 Mappings with an address of 0 are forbidden, and return EINVAL rather than
65 being honored.
66 .RE
67
68 .sp
69 .ne 2
70 .na
71 Make the userspace stack non-executable (\fBNOEXECSTACK\fR)
72 .ad
73 .RS 11n
74 The stack will be mapped without executable permission, and attempts to
75 execute it will fault.
76 .RE
77
78 System default security-flags are configured via properties on the
79 \fBsvc:/system/process-security\fR service, which contains a boolean property
80 per-flag in the \fBdefault\fR, \fBblower\fR and \fBbupper\fR, property groups.
81 The value indicates the setting of the flag, flags with no value take their
82 defaults. For example, to enable ASLR by default you would execute the
83 following commands:
84 For example, to enable ASLR by default you would execute the following
85 commands:
86 .sp
87 .in +2
88 .nf
89 # svccfg -s svc:/system/process-security setprop default/aslr = true
90 .fi
91 .in -2
92 .sp
93 .P
94 To restore the setting to the defaults you would execute:
95 .sp
96 .in +2
97 .nf
98 # svccfg -s svc:/system/process-security delpropvalue default/aslr true
99 .fi
100 .in -2
101 .sp
102 .P
103 #endif /* ! codereview */
104 .P
105 This can be done by any user with the \fBSolaris.smf.value.process-security\fR
106 authorization.
107 .P
108 Since security-flags are strictly inherited, this will not take effect until
109 the system or zone is next booted.
110
111 .SH "SEE ALSO"
112 .BR psecflags (1),
113 .BR svccfg (1M),
114 .BR brk (2),
115 .BR exec (2),
116 .BR mmap (2),
117 .BR mmapobj (2),
118 .BR privileges (5),
119 .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
*****
1 \" te
2 .\" Copyright (c) 2009, Sun Microsystems, Inc. All Rights Reserved.
3 .\" The contents of this file are subject to the terms of the Common Development
4 .\" See the License for the specific language governing permissions and limitat
5 .\" the fields enclosed by brackets \"[]\" replaced with your own identifying info
6 .TH SMF_METHOD 5 \"June 6, 2016\"
7 .SH NAME
8 smf_method \- service management framework conventions for methods
9 .SH DESCRIPTION
10 .LP
11 The class of services managed by \fBsvc.startd\fR(1M) in the service management
12 framework, \fBsmf\fR(5), consists of applications that fit a simple
13 \fBfork\fR(2)-\fBexec\fR(2) model. The \fBsvc.startd\fR(1M) master daemon and
14 other restarters support the \fBfork\fR(2)-\fBexec\fR(2) model, potentially
15 with additional capabilities. The \fBsvc.startd\fR(1M) daemon and other
16 restarters require that the methods which activate, manipulate, or examine a
17 service instance follow the conventions described in this manual page.
18 .SS \"Invocation form\"
19 .LP
20 The form of a method invocation is not dictated by convention. In some cases, a
21 method invocation might consist of the direct invocation of the daemon or other
22 binary executable that provides the service. For cases in which an executable
23 script or other mediating executable is used, the convention recommends the
24 form:
25 .sp
26 .in +2
27 .nf
28 /path/to/method_executable abbr_method_name
29 .fi
30 .in -2
31
32 .sp
33 .LP
34 The \fIabbr_method_name\fR used for the recommended form is a supported method
35 such as \fBstart\fR or \fBstop\fR. The set of methods supported by a restarter
36 is given on the related restarter page. The \fBsvc.startd\fR(1M) daemon
37 supports \fBstart\fR, \fBstop\fR, and \fBrefresh\fR methods.
38 .sp
39 .LP
40 A restarter might define other kinds of methods beyond those referenced in this
41 page. The conventions surrounding such extensions are defined by the restarter
42 and might not be identical to those given here.
43 .SS \"Environment Variables\"
44 .LP
45 The restarter provides four environment variables to the method that determine
46 the context in which the method is invoked.
47 .sp
48 .ne 2
49 .na
50 \fB\FBSMF_FMRI\fR
51 .ad
52 .sp .6
53 .RS 4n
54 The service fault management resource identifier (FMRI) of the instance for
55 which the method is invoked.
56 .RE
57
58 .sp
59 .ne 2
60 .na
61 \fB\FBSMF_METHOD\fR

```

```

62 .ad
63 .sp .6
64 .RS 4n
65 The full name of the method being invoked, such as \fBstart\fR or \fBstop\fR.
66 .RE
67
68 .sp
69 .ne 2
70 .na
71 \fB\FBSMF_RESTARTER\fR
72 .ad
73 .sp .6
74 .RS 4n
75 The service FMRI of the restarter that invokes the method
76 .RE
77
78 .sp
79 .ne 2
80 .na
81 \fB\FBSMF_ZONE_NAME\fR
82 .ad
83 .sp .6
84 .RS 4n
85 The name of the zone in which the method is running. This can also be obtained
86 by using the \fBzonename\fR(1) command.
87 .RE
88
89 .sp
90 .LP
91 These variables should be removed from the environment prior to the invocation
92 of any persistent process by the method. A convenience shell function,
93 \fBsmf_clear_env\fR, is given for service authors who use Bourne-compatible
94 shell scripting to compose service methods in the include file described below.
95 .sp
96 .LP
97 The method context can cause other environment variables to be set as described
98 below.
99 .SS \"Method Definition\"
100 .LP
101 A method is defined minimally by three properties in a propertygroup of type
102 \fBmethod\fR.
103 .sp
104 .LP
105 These properties are:
106 .sp
107 .ne 2
108 .na
109 \fBexec (\fIastring\fR)\fR
110 .ad
111 .RS 27n
112 Method executable string.
113 .RE
114
115 .sp
116 .ne 2
117 .na
118 \fBtimeout_seconds (\fIcount\fR)\fR
119 .ad
120 .RS 27n
121 Number of seconds before method times out. See the \fBTimeouts\fR section for
122 more detail.
123 .RE
124
125 .sp
126 .ne 2
127 .na

```

```

128 \fBtype (\fIastring\fR)\fR
129 .ad
130 .RS 27n
131 Method type. Currently always set to \fBmethod\fR.
132 .RE

134 .sp
135 .LP
136 A Method Context can be defined to further refine the execution environment of
137 the method. See the \fBMethod Context\fR section for more information.
138 .SS "Method Tokens"
139 .LP
140 When defined in the \fBexec\fR string of the method by the restarter
141 \fBsvc.startd\fR, a set of tokens are parsed and expanded with appropriate
142 value. Other restarters might not support method tokens. The delegated
143 restarter for inet services, \fBinetd\fR(1M), does not support the following
144 method expansions.
145 .sp
146 .ne 2
147 .na
148 \fB\fB%\fR\fR
149 .ad
150 .sp .6
151 .RS 4n
152 %
153 .RE

155 .sp
156 .ne 2
157 .na
158 \fB\fB%\fR\fR
159 .ad
160 .sp .6
161 .RS 4n
162 Name of the restarter, such as \fBsvc.startd\fR
163 .RE

165 .sp
166 .ne 2
167 .na
168 \fB\fB%\fR\fR
169 .ad
170 .sp .6
171 .RS 4n
172 The full name of the method being invoked, such as \fBstart\fR or \fBstop\fR.
173 .RE

175 .sp
176 .ne 2
177 .na
178 \fB\fB%\fR\fR
179 .ad
180 .sp .6
181 .RS 4n
182 Name of the service
183 .RE

185 .sp
186 .ne 2
187 .na
188 \fB\fB%\fR\fR
189 .ad
190 .sp .6
191 .RS 4n
192 Name of the instance
193 .RE

```

```

195 .sp
196 .ne 2
197 .na
198 \fB\fB%\fR\fR
199 .ad
200 .sp .6
201 .RS 4n
202 FMRI of the instance
203 .RE

205 .sp
206 .ne 2
207 .na
208 \fB\fB%\fR\fR
209 .ad
210 .sp .6
211 .RS 4n
212 Value(s) of a property. The \fBprop\fR might be a property FMRI, a property
213 group name and a property name separated by a \fB/\fR, or a property name in
214 the \fBapplication\fR property group. These values can be followed by a \fB,\fR
215 (comma) or \fB:\fR (colon). If present, the separators are used to separate
216 multiple values. If absent, a space is used. The following shell metacharacters
217 encountered in string values are quoted with a \ (backslash):
218 .sp
219 .in +2
220 .nf
221 ; & ( ) | ^ < > newline space tab \ " '
222 .fi
223 .in -2

225 An invalid expansion constitutes method failure.
226 .RE

228 .sp
229 .LP
230 Two explicit tokens can be used in the place of method commands.
231 .sp
232 .ne 2
233 .na
234 \fB\fB:kill [-signal]\fR\fR
235 .ad
236 .sp .6
237 .RS 4n
238 Sends the specified signal, which is \fBSIGTERM\fR by default, to all processes
239 in the primary instance contract. Always returns \fBSMF_EXIT_OK\fR. This token
240 should be used to replace common \fBkill\fR invocations.
241 .RE

243 .sp
244 .ne 2
245 .na
246 \fB\fB:true\fR\fR
247 .ad
248 .sp .6
249 .RS 4n
250 Always returns \fBSMF_EXIT_OK\fR. This token should be used for methods that
251 are required by the restarter but which are unnecessary for the particular
252 service implementation.
253 .RE

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

```

```

260 .LP
261 The following exit status codes are defined in \fB<libscf.h>\fR and in the
262 shell support file.
263 .sp
265 .sp
266 .TS
267 l l l
268 l l l .
269 \fB$EXIT_OK\fR          \fB$EXIT_OK\fR T{
270 Method exited, performing its operation successfully.
271 T}
272 \fB$EXIT_ERR_FATAL\fR   \fB$EXIT_ERR_FATAL\fR T{
273 Method failed fatally and is unrecoverable without administrative intervention.
274 T}
275 \fB$EXIT_ERR_CONFIG\fR   \fB$EXIT_ERR_CONFIG\fR T{
276 Unrecoverable configuration error. A common condition that returns this exit sta
277 T}
278 \fB$EXIT_ERR_NOSMF\fR     \fB$EXIT_ERR_NOSMF\fR T{
279 Method has been mistakenly invoked outside the \fB$SMF(5)\fR facility. Services t
280 T}
281 \fB$EXIT_ERR_PERM\fR      \fB$EXIT_ERR_PERM\fR T{
282 Method requires a form of permission such as file access, privilege, authorizati
283 T}
284 \fB$EXIT_ERR_OTHER\fR     \fB$EXIT_ERR_OTHER\fR T{
285 Any non-zero exit status from a method is treated as an unknown error. A series
286 T}
287 .TE
289 .sp
290 .LP
291 Use of a precise exit code allows the responsible restarter to categorize an
292 error response as likely to be intermittent and worth pursuing restart or
293 permanent and request administrative intervention.
294 .SS "Timeouts"
295 .LP
296 Each method can have an independent timeout, given in seconds. The choice of a
297 particular timeout should be based on site expectations for detecting a method
298 failure due to non-responsiveness. Sites with replicated filesystems or other
299 failover resources can elect to lengthen method timeouts from the default.
300 Sites with no remote resources can elect to shorten the timeouts. Method
301 timeout is specified by the \fB$timeout_seconds\fR property.
302 .sp
303 .LP
304 If you specify \fB$timeout_seconds\fR for a method, it declares to the
305 restarter that there is no timeout for the service. This setting is not
306 preferred, but is available for services that absolutely require it.
307 .sp
308 .LP
309 \fB$-1 timeout_seconds\fR is also accepted, but is a deprecated specification.
310 .SS "Shell Programming Support"
311 .LP
312 A set of environment variables that define the above exit status values is
313 provided with convenience shell functions in the file
314 \fB/lib/svc/share/smf_include.sh\fR. This file is a Bourne shell script
315 suitable for inclusion via the source operator in any Bourne-compatible shell.
316 .sp
317 .LP
318 To assist in the composition of scripts that can serve as SMF methods as well
319 as \fB/etc/init.d\fR scripts, the \fB$smf_present()\fR shell function is
320 provided. If the \fB$SMF(5)\fR facility is not available, \fB$smf_present()\fR
321 returns a non-zero exit status.
322 .sp
323 .LP
324 One possible structure for such a script follows:
325 .sp

```

```

326 .in +2
327 .nf
328 if smf_present; then
329     # Shell code to run application as managed service
330     ....
332     smf_clear_env
333 else
334     # Shell code to run application as /etc/init.d script
335     ....
336 fi
337 .fi
338 .in -2
340 .sp
341 .LP
342 This example shows the use of both convenience functions that are provided.
343 .SS "Method Context"
344 .LP
345 The service management facility offers a common mechanism set the context in
346 which the \fBfork(2)\fR-\fBexec(2)\fR model services execute.
347 .sp
348 .LP
349 The desired method context should be provided by the service developer. All
350 service instances should run with the lowest level of privileges possible to
351 limit potential security compromises.
352 .sp
353 .LP
354 A method context can contain the following properties:
355 .sp
356 .ne 2
357 .na
358 \fB$BUSE_PROFILE\fR
359 .ad
360 .sp .6
361 .RS 4n
362 A boolean that specifies whether the profile should be used instead of the
363 \fB$BUSER\fR, \fB$BGROUP\fR, \fB$BPRIVILEGES\fR, and \fB$BLIMIT_PRIVILEGES\fR
364 properties.
365 .RE
367 .sp
368 .ne 2
369 .na
370 \fB$BENVIRONMENT\fR
371 .ad
372 .sp .6
373 .RS 4n
374 Environment variables to insert into the environment of the method, in the form
375 of a number of \fB$BNAME=value\fR strings.
376 .RE
378 .sp
379 .ne 2
380 .na
381 \fB$BPROFILE\fR
382 .ad
383 .sp .6
384 .RS 4n
385 The name of an RBAC (role-based access control) profile which, along with the
386 method executable, identifies an entry in \fB$BEXEC_ATTR\fR(4).
387 .RE
389 .sp
390 .ne 2
391 .na

```

```

392 \fB\fBuser\fR\fR
393 .ad
394 .sp .6
395 .RS 4n
396 The user ID in numeric or text form.
397 .RE

399 .sp
400 .ne 2
401 .na
402 \fB\fBgroup\fR\fR
403 .ad
404 .sp .6
405 .RS 4n
406 The group ID in numeric or text form.
407 .RE

409 .sp
410 .ne 2
411 .na
412 \fB\fBsupp_groups\fR\fR
413 .ad
414 .sp .6
415 .RS 4n
416 An optional string that specifies the supplemental group memberships by ID, in
417 numeric or text form.
418 .RE

420 .sp
421 .ne 2
422 .na
423 \fB\fBprivileges\fR\fR
424 .ad
425 .sp .6
426 .RS 4n
427 An optional string specifying the privilege set as defined in
428 \fBprivileges\fR(5).
429 .RE

431 .sp
432 .ne 2
433 .na
434 \fB\fBlimit_privileges\fR\fR
435 .ad
436 .sp .6
437 .RS 4n
438 An optional string specifying the limit privilege set as defined in
439 \fBprivileges\fR(5).
440 .RE

442 .sp
443 .ne 2
444 .na
445 \fB\fBworking_directory\fR\fR
446 .ad
447 .sp .6
448 .RS 4n
449 The home directory from which to launch the method. \fB:home\fR can be used as
450 a token to indicate the home directory of the user whose \fBuid\fR is used to
451 launch the method. If the property is unset, \fB:home\fR is used.
452 .RE

454 .sp
455 .ne 2
456 .na
457 \fB\fBsecurity_flags\fR\fR

```

```

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

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

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

499 .sp
500 .ne 2
501 .na
502 \fB\fBresource_pool\fR\fR
503 .ad
504 .sp .6
505 .RS 4n
506 The resource pool name on which to launch the method. \fB:default\fR can be
507 used as a token to indicate the pool specified in the \fBproject\fR(4) entry
508 given in the \fBproject\fR attribute above.
509 .RE

511 .sp
512 .LP
513 The method context can be set for the entire service instance by specifying a
514 \fBmethod_context\fR property group for the service or instance. A method might
515 override the instance method context by providing the method context properties
516 on the method property group.
517 .sp
518 .LP
519 Invalid method context settings always lead to failure of the method, with the
520 exception of invalid environment variables that issue warnings.
521 .sp

```

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

537 .sp
538 .ne 2
539 .na
540 \fBFile descriptors\fR
541 .ad
542 .sp .6
543 .RS 4n
544 File descriptor \fB0\fR is \fB/dev/null\fR. File descriptors \fB1\fR and
545 \fB2\fR are recommended to be a per-service log file.
546 .RE

548 .SH FILES
549 .ne 2
550 .na
551 \fB\fB/lib/svc/share/smf_include.sh\fR\fR
552 .ad
553 .sp .6
554 .RS 4n
555 Definitions of exit status values.
556 .RE

558 .sp
559 .ne 2
560 .na
561 \fB\fB/usr/include/libscf.h\fR\fR
562 .ad
563 .sp .6
564 .RS 4n
565 Definitions of exit status codes.
566 .RE

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