Network Auto-Magic Data Structures & APIs

Version 1.0, 2007-Feb-16

1. Data Structures

1.1 External

1.1.1 Environments

```
typedef struct nwam_env_handle *nwam_env_handle_t;      /* opaque handle */
typedef struct nwam_env_attr {
        uint_t  ea_type;
        char    ea_fmri[MAX_FMRI_LEN]
        char    ea_short_name[MAX_SHORT_NAME_LEN];
        char    *ea_description;
        size_t  ea_desc_len;
        char    *ea_data;
        size_t  ea_data_len;
} nwam_env_attr_t;

enum {
        IP_ADDRESS,
        DOMAIN_NAME,
        ESSID,
        BSSID
} nwam_condition_tag_t;

enum {
        EQUAL,
        NOT_EQUAL,
        IN_RANGE,
        BEGINS,
        ENDS,
        CONTAINS,
        DOESNT_CONTAIN
} nwam_condition_operator_t;

enum {
        AND,
        OR
} nwam_cond_node_operator_t;

typedef struct nwam_condition {
        char                            *nc_name;
        nwam_condition_tag_t            nc_tag;
        nwam_condition_operator_t       nc_operator;
        char                            *nc_operand;
        size_t                          *nc_op_len;
} nwam_condition_t;

typedef struct nwam_cond_node {
        nwam_condition_t                *cn_condition;
        nwam_cond_node_operator_t       cn_operator;
        nwam_cond_node_t                cn_left_child;
```

1.1.2 Network Configuration Profiles (NCPs)

```
typedef struct nwam_ncp_handle *nwam_ncp_handle_t;  /* opaque handle */
typedef struct nwam_ncu_handle *nwam_ncu_handle_t;  /* opaque handle */

enum {
        NWAM_NCU_TYPE_LINK,
        NWAM_NCU_TYPE_IP,
        NWAM_NCU_TYPE_ALL
} nwam_ncu_type_t;

enum {
    NWAM_CLASS_PHYS_LINK,
    NWAM_CLASS_VLAN_LINK,
    NWAM_CLASS_AGGR_LINK,
    NWAM_CLASS_IPTUN_LINK,
    NWAM_CLASS_SIMPLE_IP,
    NWAM_CLASS_IPMP_IP,
    NWAM_CLASS_LOOPBACK_IP,
    NWAM_CLASS_VNI_IP,
    NWAM_CLASS_ALL
} nwam_ncu_class_t;

enum {
    NWAM_NCU_TYPE,                  /* uint32 (#def values) */
    NWAM_NCU_CLASS,                 /* uint32 (#def values) */
    NWAM_NCU_NAME,                  /* string */
    NWAM_NCU_PARENT_NCP,            /* string (name of parent) */
    NWAM_NCU_ACTIVATION,            /* uint32 (#def values) */
    NWAM_NCU_CONTINGENT,            /* string (name of ncu and op) */
    NWAM_NCU_OVER,                  /* string (name of ncu) */
    NWAM_NCU_UNDER,                 /* string (name of ncu) */
    NWAM_LINK_DEVNAME,              /* string */
    NWAM_LINK_LINKID,               /* uint32 */
    NWAM_LINK_MAC_ADDR,             /* uint64 */
    NWAM_LINK_MAC_ADDR_MODE,        /* uint32 (#def values) */
    NWAM_LINK_MTU,                  /* uint32 */
    NWAM_LINK_AUTOPUSH,             /* string */
    NWAM_PHYS_SPEED,                /* uint32 (#def values) */
    NWAM_PHYS_DUPLEX,               /* uint32 (#def values) */
    NWAM_PHYS_JUMBO,                /* boolean */
    NWAM_VLAN_VID,                  /* uint32 */
    NWAM_AGGR_KEY,                  /* uint32 */
    NWAM_AGGR_POLICY,               /* uint32 (#def values) */
    NWAM_AGGR_NPORTS,               /* uint32 */
    NWAM_AGGR_PORTS,                /* string (list of ncus) */
    NWAM_AGGR_LACP_MODE,            /* uint32 (#def values) */
```

1.1.3 External Network Modifiers (ENMs)

```
enum nwam_enm_state { ENABLED, DISABLED, MAINTENANCE };
enum nwam_contingent_type { USER, NCU };
struct nwam_enm_handle {
        const char *fmri;
        const char *name;
        const char *environment;
        enum nwam_contingent_type contingent_type;
        const char *contingency; /* (<op> <ncu>)* NULL means no contingency */
        enum nwam_enm_state state;
};
typedef struct nwam_enm_handle *nwam_enm_handle_t;
```

1.1.4 Notifications

```
enum nwam_events {
        NWAM_LINK_CREATE = 0x1,
        NWAM_LINK_DESTROY = 0x2,
        NWAM_LINK_UP = 0x4,
        NWAM_LINK_DOWN = 0x8,
        NWAM_LINK_INFO = 0x10,
        NWAM_IF_CREATE = 0x11,
        NWAM_IF_DESTROY = 0x12,
        NWAM_IF_UP = 0x14,
        NWAM_IF_DOWN = 0x18,
        NWAM_IF_INFO = 0x20,
};

struct nwam_query_msg {
        enum nwam_query_opcode { WIFILIST };
        union {
                struct {
                        char *essids; /*
                                       * separated by "\0".
                                       * terminated by "\0\0".
                                       */
                } wifilist;
        } request;
}
```

1.2 Internal

```
typedef struct {
    enum { LINK_CREATE, LINK_DESTROY, LINK_UP, LINK_DOWN, WIRELESS_NET,
            IF_INFO, IF_UP, IF_DOWN, SIGNAL, STATE_CHANGE } type;
    union {
            struct {
                    char *name;
            } link; /* create, destroy, up, down */

            struct {
                    char *essid;
                    uint8_t *bssid;
            } wireless_net;

            struct {
                    char *interface;
                    /* info */
            } interface; /* info, up, down */

            struct {
                    int sig;
            } signal;

            /* state change is just boolean indicator */
} nwam_event_t;
```

```
struct ncp;
struct ncu_link;
struct ncu_ip;
struct ncu_link_phys;
struct ncu_link_aggr;
struct ncu_link_vlan;
struct ncu_link_iptun;

struct ncu {
    struct ncp *parent;
    nwam_link_class_t type;
    enum { ALWAYS, NEVER, CONTINGENT, USER, MANUAL } activation;
    struct ncu *contingency;

    union {
            struct ncu_link link;
            struct ncu_ip ip;
    } value;
};

struct ncu_link {
    enum { PHYS, IPTUN, AGGR, VLAN } class;
    char *name;
    char *devname;
    int linkid;
    int mtu;
    char **autopush;
    int numparents, numchildren;
    struct ncu *parents;
    struct ncu *children;

    union {
            struct ncu_link_phys phys;
            struct ncu_link_aggr aggr;
            struct ncu_link_vlan vlan;
            struct ncu_link_iptun iptun;
    } link;
};

struct ncu_link_phys {
    enum { 802_3, 802_11 } type;
};

struct ncu_link_aggr {
    int key;
```

2. APIs

2.1 External

2.1.1 Environments

- Handles

```
/*
 * General usage model:
 * 1. handle = nwam_env_handle_init();
 * 2. (use the handle more generally as noted below)
 * 3. nwam_env_handle_fini(handle);
 */
nwam_env_handle_t nwam_env_handle_init(void);
void nwam_env_handle_fini(nwam_env_handle_t handle);
```

- Associating handles

```
/*
 * Associate a handle with an Environment of the given name by calling
 * _create() for a new Environment, or _find() for an existing one.
 */
int nwam_env_create(const char *name, nwam_env_handle_t handle);
int nwam_env_find(const char *name, nwam_env_handle_t handle);
```

- Other major functions

```
int nwam_env_destroy(nwam_env_handle_t handle);
int nwam_env_activate(nwam_env_handle_t handle);
```

- Getting and setting names

```
/*
 * To find out what name a given handle represents, use _get();
 * to change that name, use _set().
 */
int nwam_env_name_get(nwam_env_handle_t handle, char *name, size_t namelen)
int nwam_env_name_set(nwam_env_handle_t handle, char *name);
```

- Walking external list of attributes

```
/*
 * Arbitrary services may have attributes which they want updated when an
 * Environment changes.  This presents the problem that if you want to spec
 * an Environment, you don't know what all the attributes are to prompt a
 * user about.  To solve this problem, we provide a walker function which c
 * walk the list of services and call back to the named function, which can
 * then be used to build up the list of attributes.
 *
 * Usage example:
 * static int
 * mywalker(void *handle, nwam_env_attr_t *attr)
 * {
 *    int result;
 *
 *    query_user(attr);
 *    result = nwam_env_insert(handle, attr)
 * }
 * nwam_env_walk(mywalker, &handle);
 */
int nwam_env_walk(int (*func)(void *, nwam_env_attr_t *), void*);
```

- Inserting, deleting, & updating attributes in an Environment

```
int nwam_env_insert(nwam_env_handle_t handle, nwam_env_attr_t *attr);
int nwam_env_delete(nwam_env_handle_t handle, nwam_env_attr_t *attr);
int nwam_env_update(nwam_env_handle_t handle, nwam_env_attr_t *attr);
```

- Getting the internal list of attributes

```
/*
 * This function is for getting the list of attributes in a completed
 * Environment: each iterative call returns the next attribute.
 */
int nwam_env_attr_get(nwam_env_handle_t handle, nwam_env_attr_t *attr);
```

- Conditions

```
/*
 * Fill in a condition, then create a node out of it.
 */
int nwam_env_cond_create(nwam_condition_t *cond, nwam_cond_node_t **result)

/*
 * Join two nodes together, the result being the parent, and each original
 * node being the children on either side.
 */
int nwam_env_cond_join(nwam_cond_node_t *node1, nwam_cond_node_t *node2,
                       nwam_cond_node_t **result);

/*
 * Prune a node out of a tree, with the result being that the remaining
 * "sibling" of the node in question will "move up" one level.
 */
int nwam_env_cond_prune(nwam_cond_node_t *node, nwam_condition_t *cond,
                        nwam_cond_node_t **result);

/*
 * Once the consumer has built its tree properly, insert it into the
 * Environment (or delete it therefrom).
 */
int nwam_env_cond_ins(nwam_env_handle_t handle, nwam_cond_node_t *node);
int nwam_env_cond_del(nwam_env_handle_t handle, nwam_cond_node_t *node);
```

- policy engine
  - temporary suspension (time_t timeout)
  - immediate reinstatement

2.1.2 NCPs

2.1.2.1 Operations on NCPs

An NCP is a collection of NCUs that make up a complete view of the layer 2/3 network configuration on a system. Multiple NCPs may be created, and the user may request (via the NWAM GUI or CLI) that a new NCP be activated.

```
/*
 * Create/destroy an NCP handle.
 *
 * General usage model:
 * 1. handle = nwam_ncu_handle_init();
 * 2. (use the handle to create/modify an NCP)
 * 3. nwam_ncu_handle_fini(handle);
 */
nwam_ncp_handle_t nwam_ncp_handle_init(void);
void nwam_ncp_handle_fini(nwam_env_handle_t ncp);

/*
 * Link the handle to an NCP; either a newly created one by calling
 * _create(), or an existing one by calling _find().
 */
int nwam_ncp_create(const char *name, nwam_ncp_handle_t ncp);
int nwam_ncp_find(const char *name, nwam_ncp_handle_t ncp);

/*
 * To find out what name a given handle represents, use _get();
 * to change that name, use _set().
 */
int nwam_ncp_name_get(nwam_ncp_handle_t ncp, char *name, size_t namelen);
int nwam_ncp_name_set(nwam_ncp_handle_t ncp, char *name);

/*
 * Destroy an NCP (remove it from persistant storage).  This does
 * *not* destroy the handle; nwam_ncp_handle_fini() must be called
 * separately to do that.
 */
int nwam_ncp_destroy(nwam_ncp_handle_t ncp);

/*
 * Walk the list of NCUs that make up an NCP.  The NCP is identified
 * by the handle passed in; for each NCU, the callback function is
 * called, with an NCU handle and arg as parameters.
 */
int nwam_ncp_walk(nwam_ncp_handle_t ncp,
        int (*fn)(nwam_ncu_handle_t ncu, void *), void *arg);

/*
 * Activate an NCP.
 */
int nwam_ncp_activate(const char *name);
```

2.1.2.2 Operations on NCUs

```
/*
 * Create/destroy an NCU handle.
 */
nwam_ncu_handle_t nwam_ncu_handle_init(void);
void nwam_ncu_handle_fini(nwam_ncu_handle_t ncu);

/*
 * Link the handle to an NCU; either a newly created one by calling
 * _create(), or an existing one by calling _find().  An NCU is
 * identified by the tuple {name, type, class}.  Additionally, an
 * existing NCU must be associated with an NCP (identified by an
 * NCP handle).
 */
int nwam_ncu_create(const char *name, nwam_ncu_type_t type,
        nwam_ncu_class_t class, nwam_ncu_handle_t ncu);
int nwam_ncu_find(nwam_ncp_handle_t ncp, const char *ncu_name,
        nwam_ncu_type_t type, nwam_ncu_class_t class, nwam_ncu_handle_t ncu);

/*
 * Destroy an NCU (remove it from persistant storage).  This does
 * *not* destroy the handle; nwam_ncu_handle_fini() must be called
 * separately to do that.
 */
int nwam_ncu_destroy(nwam_ncu_handle_t ncu);

/*
 * Get/set a particular property value in an NCU.
 */
int nwam_ncu_prop_get(nwam_ncu_handle_t ncu, nwam_ncu_proptype_t prop,
        nwam_ncu_datatype_t *type, void *value, size_t vallen);
int nwam_ncu_prop_set(nwam_ncu_handle_t ncu, nwam_ncu_proptype_t prop,
        nwam_ncu_datatype_t type, void *value);

/*
 * To find out what name a given handle represents, use _get();
 * to change that name, use _set().
 */
int nwam_ncu_name_get(nwam_ncu_handle_t ncu, char *name, size_t namelen);
int nwam_ncu_name_set(nwam_ncu_handle_t ncu, char *name);
```

2.1.2.3 Usage examples

Create an NCU

```
nwam_ncu_handle_t ncu;
ncu = nwam_ncu_handle_init();
nwam_ncu_create("net0", NWAM_NCU_TYPE_LINK, NWAM_NCU_CLASS_PHYS, ncu);
<series of calls to nwam_ncu_prop_set>
<add ncu to ncp>
nwam_ncu_handle_fini(ncu);
```

Create and populate a new NCP

```
nwam_ncp_handle_t ncpa;
nwam_ncu_handle_t ncu1, ncu2;
ncpa = nwam_ncp_handle_init();
nwam_ncp_create("ncp_a", ncpa);
<create ncu1, ncu2>
nwam_ncu_insert(ncpa, ncu1);
nwam_ncu_insert(ncpa, ncu2);
<fini ncu handles>
nwam_ncp_handle_fini(ncpa);
```

Find and remove an NCU from its NCP

```
nwam_ncp_handle_t ncpa;
nwam_ncu_handle_t ncu1;
ncpa = nwam_ncp_handle_init();
ncu1 = nwam_ncu_handle_init();
nwam_ncp_find("ncp_1", ncpa);
nwam_ncu_find(ncpa, "net1", NWAM_NCU_TYPE_IP, NWAM_NCU_CLASS_ANY, ncu1);
nwam_ncu_delete(ncpa, ncu1);
nwam_ncu_destroy(ncu1);
nwam_ncu_handle_fini(ncu1);
nwam_ncp_handle_fini(ncpa);
```

2.1.3 ENMs

- Create

```
/*
 * fmri and handle must be non NULL, environment, name, and contingency can
 * NULL.  name is set to fmri if it is NULL.
 * fmri - the service in SMF to start/stop
 * name - the thing that the user refers to this by
 * environment - the Environment that is applied when this is started
 * contingent_type - when this is started
 * contingency - a string describing an NCU contingency if it exists
 * handle - return value, memory owned by library
 */
int nwam_enm_create(const char *fmri, const char *name, const char *environ
  enum nwam_contingent_type contingent_type, const char *contingency,
  nwam_enm_handle_t *handle);
```

- Replace

```
/*
 * Remove the old one and replace it with the new one.  When done memory
 * for old is destroyed.
 */
int nwam_enm_replace(const nwam_enm_handle_t old, nwam_enm_handle_t new);
```

- Destroy

```
/*
 * Destroy storage for ENM.
 */
int nwam_enm_destroy(nwam_enm_handle_t handle);
```

- Store

```
/*
 * These two functions implement a store for ENM information.
 */
int nwam_enm_store(nwam_enm_handle_t handle);
int nwam_enm_find(const char *fmri, nwam_enm_handle_t *handle);
```

- Registration

```
/*
 * Register these with NWAM.
 */
int nwam_enm_register(nwam_enm_handle_t handle);
int nwam_enm_deregister(const char *fmri);
```

- Start/Stop

```
/*
 * These functions send a message to nwam to start or stop this ENM.  These
 * should normally only be used if the contingent_type is USER.
 */
int nwam_enm_start(const char *fmri);
int nwam_enm_stop(const char *fmri);
```

2.1.4 Notifications

- Arm

```
/*
 * This takes a bitmask of nwam_events that you are interested on for all
 * NCUs.
 */
int nwam_arm(unsigned int *events);
```

- Notify

```
/*
 * Upon return active_ncus is a list of NCUs of size num_active_ncus togeth
 * with an array of bit-field mapped events in active_events.
 *
 * event_data is filled with event specific data (possible multiple chunks
 * of data per ncu linked together).
 */
int nwam_notify(nwam_ncu_handle_t *active_ncus, unsigned int *active_events
        void **event_data, int *num_active_ncus, int unused_flags);
```

- Query

```
/*
 * This allows an external program to register to manage NWAM's user
 * interactions.  Only one external program can register.  The door
 * passed in is used to call back into the external program from nwam.
 *
 * A struct nwam_query_msg is passed as the door data_ptr.
 * flags is currently undefined.
 */
int nwam_arm_query(void *door, int flags);
```