LDOMs Domain Services API

Revision 1.10

September 10, 2008

Please send comments & queries to:

michael.christensen@sun.com

Table of Contents

1 Introduction

1.1 Background

In a Logical Domain environment the ability to discover whether a guest operating system has various capabilities, and be able to remotely direct it to perform various operations is important. Similarly it is equally important for a guest operating system to be able to discover and communicate with its various support services.

The existing implementation of support for providing these services is kernel-based for domain service providers in the guests and has multiple implementations (e.g. Solaris, LDom Manager, and SP VBSC software) to provide client support. It has been found through experience that additional service providers can be more easily developed if there were a set of user-space interfaces to the domain services functionality. As it is now, each service that wishes to exercise Domain Services functionality must implement a device driver to coordinate the access to these interfaces.

The new user-space interfaces described in this document will be made available through a set of library and device driver. It is expected that developers will only utilize the provided library interfaces and not the device driver directly. The existing kernel interfaces exported by the ds module will not be changed by this project.

The new user-space interfaces, though they are not identical, are designed to co-exist with the existing kernel interfaces, and extend it where necessary.. In particular, it was a design goal to mimic the asynchronous nature of the existing kernel interfaces and carry that over to the user-level interfaces..

1.2 Existing Kernel Domain Services

The existing kernel interfaces for Domain Services provides the following three interfaces:

- Service Registration (ds_cap_init)
- Service Unregistration (ds_cap_fini)
- Send a response message to a received data message (ds_cap_send)

The current implementation of the Domain Service framework primarily consists of domains exporting services and the LDom Manager consuming these services. Service registration on a guest registers the service, and initiates the Domain Services Protocol handshake with the LDom Manager on the Control Domain. If there is an outstanding registration request on the Control Domain for this service, then the request is ack'ed and a registration event callback is invoked.

Similarly, a service unregistration unregisters the service and will cause the unregistration callback to be invoked on those domains which were connected to this service.

When a service provider end of a connection receives data, it will invoke the data callback function provided during initialization. Typically, receipt of a message corresponds to exercising a service provided by the provider and will result in the provider performing the function, e.g. Reboot the domain, or respond with requested data. In either case, a service response message would need to be sent in response to this asynchronous request.

2 User Domain Services Interfaces

This section defines the interfaces supplied by the new User Domain Services library. These interfaces are intended to parallel the definition of the same interfaces that are provided for the kernel-space interfaces, but they are not precisely identical. They also include extensions to support a single callback handler to support multiple domains or even multiple services. It also includes extensions to support a non-asynchronous implementation of the data event handling. The following interfaces are defined:

- Library initialization/finish
- Service Registration
- Service Unregistration
- Handle/Domain/Service/Service Status Lookup
- Send/Receive Message

2.1 Introduction and Overview

The following sections are divided into two sections. The first section defines the data types and structures that are used to interface with the User Domain Services library. The second section defines the available functions that can be called by a user program.

2.2 Data Types and Structures

The basic data types which are defined are:

- Domain Service Handle – ds_hdl_t

  This is an unsigned 64-bit (uint64_t) handle which is used by the library to identify a given service.

- Domain Service Domain Id – ds_domain_hdl_t

  This is an unsigned 64-bit (uint64_t) handle which is used by the library to identify a given domain. This is an arbitrary value given to the Domain Services library by the LDOMs manager and is understood only by the LDOMs manager.

- Domain Service Ids

  Domain Service Ids, or Domain Service Names, are ascii strings which are registered by the various ARC cases that use those Ids. The registry of Domain Service Ids is maintained at:

  http://sac.eng.sun.com/arc/FWARC/Registries/domain_services.txt

- Service Callback argument – ds_cb_arg_t

  This is an void pointer type which can be supplied as an argument to the callback functions.

2.2.1 Version Structure

This structure is used to specify supported versions in the Capability structure and also is passed to the registration callback function to indicate which negotiated version is being supported by the service. Please see FWARC/2006/055 for more information on the version negotiation.

```
typedef struct ds_ver {
    uint16_t        major;
    uint16_t        minor;
} ds_ver_t;
```

2.2.2 Capability Structure

A DS capability is specified by a service provider using a unique service identifier string. Along with this identifier is the list of versions of the capability that the service provider supports. The Capability structure is passed as an argument to the Service Registration functions.

```
typedef struct ds_capability {
    char            *svc_id; /* service identifier */
    ds_ver_t        *vers;  /* list of supported versions */
    uint_t          nvers;  /* number of supported versions */
} ds_capability_t;
```

2.2.3 Callback Ops Structure

The Callback Ops structure specifies the user-supplied callback functions that will be invoked during the registration, unregistration and received data events. Also, the user supplied callback argument, cb_arg, is passed as an argument to each callback function in the arg argument.

```
typedef struct ds_ops {
    void (*ds_reg_cb)(ds_hdl_t hdl, ds_cb_arg_t arg, ds_ver_t *ver,
        ds_domain_hdl_t dhdl);
    void (*ds_unreg_cb)(ds_hdl_t hdl, ds_cb_arg_t arg);
    void (*ds_data_cb)(ds_hdl_t hdl, ds_cb_arg_t arg, void *buf,
        size_t buflen);
    ds_cb_arg_t cb_arg;
} ds_ops_t;
```

The ds_reg_cb() callback is invoked when the DS framework has successfully completed version negotiation with the remote endpoint for the capability. It provides the service provider / client with the negotiated version, domain name and a handle to use when performing various operations. There is no guarantee that multiple register/unregisters of the same service on the same domain will have the same handle identifier. The domain name argument is a DS specific name, it may not correspond to the actual LDOMs domain name specified during its creation

The ds_unreg_cb() callback is invoked when the DS framework detects an event that causes the registered capability to become unavailable. This includes an explicit unregister message, a failure in the underlying communication transport, etc. Any such event invalidates the service handle that was received from the register callback. The handle that is passed back as an argument is a

stale handle and no operations may be done using it. It may only be used to differentiate between different instances.

The ds_data_cb() callback is invoked whenever there is an incoming data message for the service provider / client to process. It provides the contents of the message along with the message length.

2.3 User Domain Services Library functions

2.3.1 Initialize and Finish Library Usage

There are two functions that initialize and finish library usage. A call to the initialization function is not necessary as it is automatically called in the registration functions, ds_svc_reg and ds_clnt_reg. In normal usage, calling the finish function is also unnecessary, as resources are released automatically upon process exit. However, in the situation where a loadable module is using the DS library, it is necessary to explicitly call the finish function in order to release resources and perform cleanup so that should the module using the DS library be reloaded, it may use the DS library.

The DS library initialization function is:

```
extern int ds_init(void);
```

Possible error return values are:

- ENOENT – can't open DS driver
- EACCES – no permission to DS driver

```
extern void ds_fini(void);
```

There are no error return values from this function.

2.3.2 Service Registration

There are two Service Registration functions. One registers a Service Provider while the other registers an interest in that Service, a Service Client or Consumer.

```
extern int ds_svc_reg(ds_capability_t *cap, ds_ops_t *ops);
```

This function registers a service provider. Registration of the service initiates the handshake with other domain(s) to announce service availability. The Capability structure is defined in 2.2.2. Callback events are defined in 2.2.3.

```
extern int ds_clnt_reg(ds_capability_t *cap, ds_ops_t *ops);
```

This function registers interest in a service from a specific domain. When that service is registered, the register callback is invoked. When that service is unregistered, the unregister callback is invoked. When data is received, the receive data callback is invoked.

NOTE: If the ds_data_cb function is NULL, it is assumed that the caller wants to use a polling interface by calling ds_recv_msg in place of the asynchronous callback. If the ds_data_cb function is not NULL, any call to ds_recv_msg will return an error.

Possible error return values are:

- EFAULT – an address argument is an invalid address
- EINVAL – an argument is invalid or out of range
- EALREADY – the service is already registered
- EBUSY – the complementary service is already active

2.3.3 Service Unregistration

Typically explicit unregistration of a service is not needed, as various events, such as the closing of the communication channel, death of the user process, etc. will cause the unregistration to happen automatically. However, this function is supplied in case a manual unregistration is required. Note that you may only perform this operation on a handle registered by the current process.

```
extern int ds_unreg_hdl(ds_hdl_t hdl);
```

Unregisters either a service provider or an interest as a client in the service indicated by the supplied handle.

Possible error return values are:

- ENXIO – unknown handle
- EACCES – invalid handle access

An alternate way of unregistering all handles for a given service, given the service name and service type is:

```
extern void ds_unreg_svc(char *service, boolean_t is_client);
```

This may be useful in the situation where a service has been registered, yet no registration callbacks have yet been performed. No errors are returned for this function.

2.3.4 Handle Lookup

This function is intended to provide access to currently registered handles for a given service. This could be used to poll for connection registration/unregistration events, as supposed to using the register/unregister callbacks.

A client service may have multiple service providers, and therefore multiple handles. Though the framework will enforce that there cannot be multiple service providers on the same domain, there may be multiple service client handles from the same service registration on one domain. But, two processes may not register the same client service.

```
extern int ds_hdl_lookup(char *service, boolean_t is_client,
    ds_hdl_t *hdlp, uint_t maxhdls, uint_t *nhdlsp);
```

Given a service name and client type, the function returns all of the handles that are currently registered to that service.

This interface allows the caller to obtain the handles are currently registered to a service, without using the asynchronous registration callback mechanism. The *hdlp* argument will return an array of handles, the maximum number being the *maxhdls* argument and the number of handles returned will be in *nhdlsp*

argument.

Possible error return values are:

- EFAULT – invalid address argument
- EINVAL – invalid argument
- EBADF – no prior prior registration call

2.3.5 Domain Lookup

This function is intended to provide a service provider with the ability to get a unique domain identifier that a particular service handle is connected to. Given a handle, the fuction returns its associated domain identifier.

```
extern int ds_domain_lookup(ds_hdl_t hdl, ds_domain_hdl_t *dhdlp);
```

Possible error return values are:

- EFAULT – invalid address argument
- EINVAL – invalid argument
- ENXIO – unknown handle
- EBADF – no prior registration call

2.3.6 Handle Is Ready

Although a handle or handles may be returned by ds_hdl_lookup, the handle(s) does not correspond to a link over which data transfers can be done until the communication handshake is completed. If no registration or unregistration callback function was specified, the caller may need to determine if a returned handle is actually ready to perform a transmit (*ds_send_msg*) or a receive (*ds_recv_msg*) operation. The following interface allows a caller to do that check:

```
extern int ds_isready(ds_hdl_t hdl, boolean_t *is_ready);
```

The returned value is_ready is TRUE (1) if the handle is ready for data transfers, FALSE (0) otherwise.

NOTE: If the caller is not using the registration/unregistration callbacks, it must handle the situation where a service is unregistered and then reregistered. In this case, the caller will get a different handle from the original handle it was using, so it must not only check to see if the handle is ready, but also whether it is still valid. If it is not valid, i.e. ds_isready returns an error, then the caller must obtain a new handle value for this service by performing ds_hdl_lookup, prior to doing any other operation.

Possible error return values are:

- EFAULT – invalid address argument
- EINVAL – invalid argument
- ENXIO – unknown handle
- EBADF – no prior registration call

2.3.7 Send Message

Send data to the appropriate service provider or client indicated by the handle. The sender will block until the message has been sent. The underlying protocol does not guarantee that multiple calls to ds_send_msg by the same thread will result in the data showing up at the receiver in the same order as it was sent. If multiple messages needs to be sent by the caller, it is up to the sender and receiver will need to implement a higher level protocol to ensure ordering.

```
extern int ds_send_msg(ds_hdl_t hdl, void *buf,
   size_t buflen);
```

Possible error return values are:

- EFAULT – invalid address argument
- EINVAL – invalid argument
- ENXIO – unknown handle
- EBADF – no prior registration call
- EACCES – invalid handle access

2.3.8 Receive Message

Receive data from the appropriate service provider or client indicated by the provided handle. The receiver will block until a message has been received. The *buflen* argument is the maximum message that may be received. The *msglenp* argument is a pointer to the actual received message size.

```
extern int ds_recv_msg(ds_hdl_t hdl, void *buf,
    size_t buflen, size_t *msglenp);
```

If *buflen* is smaller than the message currently pending, then ds_recv_msg will return EFBIG error and the returned message size (*msglenp*) will be the size of the message. Note that if *buflen* is zero, ds_recv_msg will return without blocking and return the size of the pending message in the returned message size.

Possible error return values are:

- EFAULT – invalid address argument
- EINVAL – invalid argument
- ENXIO – unknown handle
- EFBIG – pending message is bigger than buflen argument
- EBADF – no prior registration call
- EACCES – invalid handle access

3 References

- FWARC/2006/055 Domain Services Specification
- FWARC/2008/563 - Virtual Domain Service MD nodes and misc. properties