



Shepherd Hook Specification

J&J Computer Consulting Shepherd Server Publications

Overview

Shepherd Hooks provide a mechanism for calling outside code based on major events within the Shepherd Directory Service. Currently, Shepherd supports hooks for loadObject, addObject, modifyObject, and deleteObject. For implementations that need to know what data is being moved in and out of the directory, hooks are a perfect, thin solution. While a directory service provider could be written to accomplish the same thing, hooks require less work to implement. This document explains both how to configure hooks and how to develop hooks within Shepherd Directory Services (SDS).

Configuring Shepherd Hooks

Shepherd Hooks exist as objects in SDS. An example of an object representing a Shepherd Hook in SDS would look like:

```
dn: cn=sdsrMasterDeleteHook, c=US
cn: sdsrMasterDeleteHook
objectclass: ShepherdHook
hostServer: cn=Master, c=US
hookMethod: deleteObject
api: replicate\sdsrepl.dll!sdsMasterReplicationHook
```

The objectclass "ShepherdHook" identifies to the Shepherd system that the object is in fact a hook. Based on this information, Shepherd looks at the hostServer attribute to determine if the hook should load on the Shepherd system. The hostServer attribute can be multi-valued if the hook should run on more than one Shepherd system in SDS.

If the hook belongs on the server in question, Shepherd then validates the hookMethod attribute. The hookMethod can be one of the following values:

- ◆ deleteObject
- ◆ loadObject
- ◆ addObject
- ◆ modifyObject

The hookMethod is **not** multi-valued. A ShepherdHook object must exist for every hookMethod desired even if the api used is the same for all hookMethods.

The `api` attribute uses the standard Shepherd syntax for an application programming interface (API). That format is:

```
<dll name>!<method name>
```

The `<dll name>` should be the full or relative path to the Dynamic Link Library (DLL) containing the hook API. The Shepherd system requires relative paths specified from the current working directory. In most cases, Shepherd systems start from the installation directory, but they can be configured differently.

The `<method name>` should be the exported method name to call within the specified DLL.

Shepherd will report an error to the console if a Hook fails to load properly.

Developing Shepherd Hooks

A Shepherd Hook exists as an exported method of a DLL with the following signature:

```
int hook( ShepherdHookParameters* hookParam );
```

The parameter passed to the hook is defined as follows:

```
struct ShepherdHookParameters {
    Directory* directory;
    DirectoryObject hookObject;
    int hookMethod;
    DirectorySession adminSession;
    DirectorySession hookSession;
    str engineUsername;
    union {
        struct {
            DirectoryObjectName* objectname;
            DirectoryObject* object;
            BOOL shadow;
        } load;
        struct {
            DirectoryObjectModify* modify;
            BOOL shadow;
        } modify;
        struct {
            DirectoryObjectName* objectname;
        } del;
        struct {
            DirectoryObject* object;
        } add;
    } parameters;
};
```

The directory provides the Shepherd Hook an opportunity to call into SDS to perform actions based on the data provided in the Shepherd Hook. Exercise caution when calling into SDS as doing so could result in an infinite loop if the call invokes the Shepherd Hook recursively.

The hookObject contains the object from SDS that resulted in the Shepherd Hook loading. In most cases, a Shepherd Hook does not need to look at the information in hookObject, but Shepherd provides the information in case a Hook needs it.

The hookMethod allows a Shepherd Hook to identify what action resulted in the hook getting called. Possible values include:

ShepherdHook::LOAD_HOOK
ShepherdHook::DELETE_HOOK
ShepherdHook::ADD_HOOK
ShepherdHook::MODIFY_HOOK

If the Hook handles multiple hookMethods, the Hook should inspect the hookMethod field to determine which struct within the parameters union contains valid data.

The adminSession gives the Hook full access to SDS. The hookSession gives the Hook the same access to SDS that the user that prompted the call to SDS has been granted. A Hook can use one or both of these sessions to call into SDS and perform operations. It depends on the design and purpose of the Hook whether one or both of the DirectorySession objects are used.

The engineUsername gives the Hook the name of the user under which the Shepherd Engine logged onto SDS.

The parameters union allows 4 possible structures. Each structure contains the parameters used to call the hookMethod associated with that structure. For a loadObject hook, the parameters are:

objectname	Distinguished name of the object being loaded.
object	The DirectoryObject into which the object should be loaded.
shadow	true => Load the shadowed data, not the actual Shadow object. false=> Load the actual Shadow object, not the shadowed data.

For a modifyObject hook, the parameters are:

modify	DirectoryObjectModify object containing the data necessary to perform the requested modification.
shadow	true => Turns shadow checking on. Restricts modifies on Shadow objects. false=> Turns shadow checking off. Unrestricted modify of the Shadow object.

For a deleteObject hook, the parameters are:

objectname Distinguished name of the object being deleted.

For an addObject hook, the parameters are:

object The object that is being added to SDS.

Additional information on the classes used in ShepherdHookParameters can be found in the header files.

On return, a Shepherd Hook can return one of the following values:

ShepherdHook::ERROR

ShepherdHook::CONTINUE

ShepherdHook::OK

Shepherd invokes Hooks prior to the operations taking place in SDS. As a result, a Shepherd Hook can prevent an operation from continuing by returning ShepherdHook::ERROR. Both ShepherdHook::CONTINUE and ShepherdHook::OK allow execution of the operation to continue into SDS. ShepherdHook::CONTINUE implies that the Hook did not do any processing, but Shepherd handles both CONTINUE and OK the same.