IceBox Administration

An IceBox server internally creates an object called the service manager that is responsible for loading and initializing the configured services. You can optionally expose this object to remote clients, such as the IceBox and IceGrid administrative utilities, so that they can execute certain administrative tasks.

On this page:

IceBox Administrative Slice Interfaces

The Slice definitions shown below comprise the IceBox administrative interface:

Slice

```
module IceBox {
exception AlreadyStartedException {};
exception AlreadyStoppedException {};
exception NoSuchServiceException {};

interface ServiceObserver {
    void servicesStarted(Ice::StringSeq services);
    void servicesStopped(Ice::StringSeq services);
};

interface ServiceManager {
    idempotent Ice::SliceChecksumDict getSliceChecksums();
    void startService(string service)
        throws AlreadyStartedException, NoSuchServiceException;
    void stopService(string service)
        throws AlreadyStoppedException, NoSuchServiceException;
    void addObserver(ServiceObserver* observer)
    void shutdown();
};
};
```

The IceBox `ServiceManager` Interface

The `ServiceManager` interface provides access to the service manager object of an IceBox server. It defines the following operations:

- `getSliceChecksums`
  Returns a dictionary of checksums that allows a client to verify that it is using the same Slice definitions as the server.

- `startService`
  Starts a pre-configured service that is currently inactive. This operation cannot be used to add new services at run time, nor will it cause an inactive service's implementation to be reloaded. If no matching service is found, the operation raises `NoSuchServiceException`. If the service is already active, the operation raises `AlreadyStartedException`.

- `stopService`
  Stops an active service but does not unload its implementation. The operation raises `NoSuchServiceException` if no matching service is found, and `AlreadyStoppedException` if the service is stopped at the time `stopService` is invoked.

- `addObserver`
  Adds an observer that is called when IceBox services are started or stopped. The service manager ignores operations that supply a null proxy, or a proxy that has already been registered.

- `shutdown`
  Terminates the services and shuts down the IceBox server.

The IceBox `ServiceObserver` Interface

An administrative client that is interested in receiving callbacks when IceBox services are started or stopped must implement the `ServiceObserver` interface and register the callback object's proxy with the service manager using its `addObserver` operation. The `ServiceObserver` interface defines two operations:

- `servicesStarted`
  Invoked immediately upon registration to supply the current list of active services, and thereafter each time a service is started.

- `servicesStopped`
  Invoked whenever a service is stopped, and when the IceBox server is shutting down.

The IceBox server unregisters an observer if the invocation of either operation causes an exception.

Our discussion of IceGrid includes an example that demonstrates how to register a `ServiceObserver` callback with an IceBox server deployed with IceGrid.

Enabling the Service Manager

IceBox's administrative functionality is disabled by default. You can enable it in two ways:

1. Define endpoints for the `IceBox.ServiceManager` object adapter.
2. Satisfy the prerequisites for enabling the Ice administrative facility.

For example, the following configuration property enables the `IceBox.ServiceManager` object adapter:

```
IceBox.ServiceManager.Endpoints=tcp -h 127.0.0.1 -p 10000
```

Similarly, the Ice administrative facility requires that endpoints be defined for the `Ice.Admin` object adapter with the property `Ice.Admin.Endpoints`. Note that the `Ice.Admin` object adapter is enabled automatically in an IceBox server that is deployed by IceGrid.

Regardless of which object adapter(s) you choose to enable, exposing the service manager makes an IceBox server vulnerable to denial-of-service attacks from malicious clients. Consequently, you should choose the endpoints and transports carefully.

IceBox Object Identities

Although an IceBox server has only one service manager object, the object is accessible via two different identities depending on how the administrative functionality was enabled.

`IceBox.ServiceManager` Object Adapter

When this object adapter is enabled, the service manager object has the default identity `IceBox/ServiceManager`. If an application requires the use of multiple IceBox servers, it is a good idea to assign unique identities to their service manager objects by configuring the servers with different values for the `IceBox.InstanceName` property, as shown in the following example:

```
IceBox.InstanceName=IceBox1
```

This property changes the category of the object's identity, which becomes `IceBox1/ServiceManager`. A corresponding change must be made in the configuration of administrative clients.

Ice Administrative Facility

When this facility is enabled, the service manager is added as a facet of the server's `admin` object. As a result, the identity of the service manager is the same as that of the `admin` object, and the name of its facet is `IceBox.ServiceManager`. The identity of the `admin` object uses either a UUID or a statically-configured value for its category, and the value `admin` for its name. For example, consider the following property definitions:

```
Ice.Admin.Endpoints=tcp -h 127.0.0.1 -p 10001
Ice.Admin.InstanceName=IceBox
```

In this case, the identity of the `admin` object is `IceBox/admin`.

IceBox also registers a `Properties` facet for each of its services so that the configuration properties of a service can be inspected remotely. The facet name is constructed as follows:

```
IceBox.Service.name.Properties
```

The value *name* represents the service name.

IceBox Administrative Client Configuration

A client requiring administrative access to the service manager can create a proxy using the endpoints configured for the service manager.

Using the `IceBox.ServiceManager` Object Adapter

To access the service manager via the `IceBox.ServiceManager` object adapter, the proxy should use the default identity `IceBox/ServiceManager` unless the server has changed the category using the `IceBox.InstanceName` property.

Using the Ice Administrative Facility

To access the service manager via the administrative facility, the client must first obtain (or be able to construct) a proxy for the `admin` object. The default identity of the `admin` object uses a UUID for its category, which means the client cannot predict the identity and therefore will be unable to construct the proxy itself. If the IceBox server is deployed with IceGrid, the client can use the technique described in our discussion of IceGrid to access its `admin` object.

In the absence of IceGrid, the IceBox server should set the `Ice.Admin.InstanceName` property if remote administration is required. In so doing, the identity of the `admin` object becomes well-known, and a client can construct the proxy on its own. For example, let us assume that the IceBox server defines the following property:

```
Ice.Admin.InstanceName=IceBox
```

A client can define the proxy for the `admin` object in a configuration property as follows:

```
ServiceManager.Proxy=IceBox/admin -f IceBox.ServiceManager -h 127.0.0.1 -p 10001
```

The proxy option `-f IceBox.ServiceManager` specifies the name of the service manager's administrative facet.

IceBox Administrative Utility

IceBox includes C++ and Java implementations of an administrative utility. The utilities have the same usage:

```
Usage: iceboxadmin [options] [command...]
Options:
-h, --help           Show this message.
-v, --version        Display the Ice version.

Commands:
start SERVICE        Start a service.
stop SERVICE         Stop a service.
shutdown             Shutdown the server.
```

The C++ utility is named `iceboxadmin`, while the Java utility is represented by the class `IceBox.Admin`.

The `start` command is equivalent to invoking `startService` on the service manager interface. Its purpose is to start a pre-configured service; it cannot be used to add new services at run time. Note that this command does not cause the service's implementation to be reloaded.

Similarly, the `stop` command stops the requested service but does not cause the IceBox server to unload the service's implementation.

The `shutdown` command stops all active services and shuts down the IceBox server.

The C++ and Java utilities obtain the service manager's proxy from the property `IceBoxAdmin.ServiceManager.Proxy`, therefore this proxy must be defined in the program's configuration file or on the command line, and the proxy's contents of depend on the server's configuration. If the IceBox server is deployed with IceGrid, we recommend using the IceGrid administrative utilities instead, which provide equivalent commands for administering an IceBox server. Otherwise, the proxy should have the endpoints and identity configured for the server.

See Also

- Slice Checksums
- Administrative Facility
- The admin Object
- The Properties Facet
- icegridadmin Command Line Tool
- IceGrid and the Administrative Facility
- IceBox Properties
- IceBoxAdmin Properties
- Ice Administrative Properties