

IceBox Properties

On this page:

- [IceBox.InheritProperties](#)
- [IceBox.InstanceName](#)
- [IceBox.LoadOrder](#)
- [IceBox.PrintServicesReady](#)
- [IceBox.Service.name](#)
- [IceBox.ServiceManager.AdapterProperty](#)
- [IceBox.UseSharedCommunicator.name](#)

IceBox.InheritProperties

Synopsis

```
IceBox.InheritProperties=num
```

Description

If *num* is set to a value larger than zero, each service [inherits the configuration properties](#) of the IceBox server's communicator. If not defined, the default value is zero.

IceBox.InstanceName

Synopsis

```
IceBox.InstanceName=name
```

Description

Specifies an [alternate identity category](#) for the IceBox service manager object. If defined, the identity of the object becomes *name/ServiceManager*. If not specified, the default identity category is IceBox.

IceBox.LoadOrder

Synopsis

```
IceBox.LoadOrder=names
```

Description

Determines the [order](#) in which services are loaded. The service manager loads the services in the order they appear in *names*, where each service name is separated by a comma or white space. Any services not mentioned in *names* are loaded afterward, in an undefined order.

IceBox.PrintServicesReady

Synopsis

```
IceBox.PrintServicesReady=token
```

Description

If this property is set to a value greater than zero, the service manager prints "*token* ready" on standard output once initialization of all the services is complete. This is useful for scripts that need to wait until all services are ready to be used.

IceBox.Service.*name*

Synopsis

```
IceBox.Service.name=entry_point[,version] [args]
```

Description

Defines a [service](#) to be loaded during IceBox initialization. Any arguments that follow the entry point are examined; those matching the `--name.*=value` pattern are interpreted as property definitions and appear in the property set of the communicator that is passed to the service `start` method, and all remaining arguments are passed to the `start` method in the `args` parameter.

Platform Notes

C++

The value of `entry_point` has the following form:

```
basename[,version]:function
```

The `basename` and optional `version` components are used to construct the name of a DLL or shared library. If no version is supplied, the version is the empty string. The `function` component is the name of a function with extern C linkage. For example, the entry point `IceStormService,34:createIceStorm` implies a shared library name of `libIceStormService.so.34` on Unix and `IceStormService34.dll` on Windows. Furthermore, if IceBox is built on Windows with debugging, a `d` is automatically appended to the version (e.g., `IceStormService34d.dll`).

The function must be declared with extern C linkage and have the following signature:

C++

```
IceBox::Service* function(Ice::CommunicatorPtr c);
```

Note that the function must return a pointer and not a smart pointer. The Ice core deallocates the object when it unloads the library. The communicator instance passed to this function is the server's communicator, which is not the same as the communicator passed to the service's `start` method.

Java

The value of `entry_point` is the name of a class that must implement the `IceBox.Service` interface. The class must provide at least one of the constructors shown in the example below:

Java

```
public class MyService implements IceBox.Service
{
    public MyService(Ice.Communicator serverCommunicator);
    public MyService();

    // ...
}
```

The constructor taking an `Ice.Communicator` argument is invoked if present, otherwise the default constructor is invoked.

.NET

The value of `entry_point` has the form `assembly:class`. The `assembly` can be a partially or fully qualified assembly name, such as `myplugin, Version=0.0.0.0,Culture=neutral`, or an assembly DLL name such as `myplugin.dll`.



You *must* use a fully qualified assembly name to load a service from an assembly in the Global Assembly Cache.

The specified class must implement the `IceBox.Service` interface and provide at least one of the constructors shown in the example below:

C#

```
public class MyService : IceBox.Service
{
    public MyService(Ice.Communicator serverCommunicator);
    public MyService();

    // ...
}
```

The constructor taking an `Ice.Communicator` argument is invoked if present, otherwise the default constructor is invoked.

IceBox.ServiceManager.AdapterProperty

Synopsis

```
IceBox.ServiceManager.AdapterProperty=value
```

Description

IceBox uses the adapter name `IceBox.ServiceManager` for its object adapter. Therefore, [adapter properties](#) can be used to configure the IceBox object adapter.

IceBox.UseSharedCommunicator.name

Synopsis

```
IceBox.UseSharedCommunicator.name=num
```

Description

If *num* is set to a value larger than zero, the service manager supplies the service with the given *name* a communicator that might be shared by other services. If the [#IceBox.InheritProperties](#) property is also defined, the shared communicator inherits the properties of the IceBox server. If not defined, the default value is zero.