IceBox Properties

On this page:

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`* [*`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. Whitespace separates the arguments, and any arguments that contain whitespace must be enclosed in quotes.

Platform Notes

C++

The value of *`entry_point`* has the following form:

`path`[`,`*`version`*]`:`*`function`*

The *`path`* 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,35:createIceStorm` implies a shared library name of `libIceStormService.so.35` on Unix and `IceStormService35.dll` on Windows. Furthermore, if IceBox is built on Windows with debugging, a `d` is automatically appended to the version (e.g., `IceStormService35d.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.

The *`path`* component may optionally contain a relative or absolute path name, indicated by the presence of a path separator (`/` or `\`). In this case, the last component of the path is used to construct the name of the shared library or DLL. Consider this example:

```
IceBox.Service.IceStorm=./IceStormService,35:createIceStorm
```

The use of a relative path means the Ice run time will look in the current working directory for `libIceStormService.so.35` on Unix or `IceStormService35.dll` on Windows.

If the *`path`* component contains spaces, the entire entry point must be enclosed in quotes:

```
IceBox.Service.IceStorm="C:\Program Files\ZeroC\Ice-3.5b\bin\IceStormService,35:createIceStorm"
```

If the *`path`* component does not include a leading path name, Ice delegates to the operating system to locate the shared library or DLL, which typically means that the plug-in can reside in any of the directories in your shared library or DLL search path.

Java

The value of *`entry_point`* has the following form:

[*`path`*`:`]*`class`*

The *`class`* component must be the name of a class that implements the `IceBox.Service` interface and provides 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.

If *`path`* is specified, it may be the path name of a JAR file or class directory, as shown below:

```
IceBox.Service.MyService=MyService.jar:MyServiceImpl
IceBox.Service.MyOtherService=/classes:MyOtherServiceImpl
```

If *`path`* contains spaces, it must be enclosed in quotes:

```
IceBox.Service.MyService="factory classes.jar":MyServiceImpl
```

IceBox uses a single class loader to load all services having the same value for *`path`*.

If *`class`* is specified without a path, IceBox attempts to load the class using class loaders in a well-defined order.

.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`, and may optionally include a leading relative or absolute path name.

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.

If you specify a relative path name in the entry point, the assembly is located relative to the program's current working directory:

```
IceBox.Service.MyService=..\MyService.dll:MyServiceImpl
```

Enclose the assembly's path name in quotes if it contains spaces:

```
IceBox.Service.MyService="C:\Program Files\MyService\MyService.dll:MyServiceImpl"
```

Finally, if the assembly uses a leading path name, be sure to include the `.dll` extension.

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 *name* with 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.