# Object Adapter Replication

As an implementation of an Ice location service, IceGrid supports object adapter replication. An application defines its replica groups and their participating object adapters using descriptors, and IceGrid generates the server configurations automatically.

On this page:

## Deploying a Replica Group

The descriptor that defines a replica group can optionally declare well-known objects as well as configure the group to determine its behavior during locate requests. Consider this example:

**XML**

```xml
<icegrid>
    <application name="ReplicaApp">
        <replica-group id="ReplicatedAdapter">
            <object identity="TheObject" type="::Demo::ObjectType"/>
        </replica-group>
        <node name="Node">
            <server id="ReplicaServer" activation="on-demand" exe="/opt/replica/bin/server">
                <adapter name="TheAdapter" endpoints="default" replica-group="ReplicatedAdapter"/>
            </server>
        </node>
    </application>
</icegrid>
```

The adapter's descriptor declares itself to be a member of the replica group `ReplicatedAdapter`, which must have been previously created by a replica group descriptor.

The replica group `ReplicatedAdapter` declares a well-known object so that an indirect proxy of the form `TheObject` is equivalent to the indirect proxy `TheObject@ReplicatedAdapter`. Since this trivial example defines only one adapter in the replica group, the proxy `TheObject` is also equivalent to `TheObject@TheAdapter`.

## Replica Group Membership

An object adapter participates in a replica group by specifying the group's ID in the adapter's `ReplicaGroupId` configuration property. Identifying the replica group in the IceGrid descriptor for an object adapter causes the node to include the equivalent `ReplicaGroupId` property in the configuration file it generates for the server.

By default, the IceGrid registry requires the membership of a replica group to be statically defined. When you create a descriptor for an object adapter that identifies a replica group, the registry adds that adapter to the group's list of valid members. During an adapter's activation, when it describes its endpoints to the registry, an adapter that also claims membership in a replica group is validated against the registry's internal list.

In a properly configured IceGrid application, this activity occurs without incident, but there are situations in which validation can fail. For example, adapter activation fails if an adapter's ID is changed without notifying the registry, such as by manually modifying the server configuration file that was generated by a node.

It is also possible for activation to fail when the IceGrid registry is being used solely as a location service, in which case descriptors have not been created and therefore the registry has no advance knowledge of the replica groups or their members. In this situation, adapter activation causes the server to receive `NotRegisteredException` unless the registry is configured to allow dynamic registration, which you can do by defining the following property:

```
IceGrid.Registry.DynamicRegistration=1
```

With this configuration, a replica group is created implicitly as soon as an adapter declares membership in it, and any adapter is allowed to participate.

Copyright © 2017, ZeroC, Inc.

The use of dynamic registration often leads to the accumulation of obsolete replica groups and adapters in the registry. The IceGrid administration tools allow you to inspect and clean up the registry's state.

## Using Replica Groups in the Ripper Application

Replication is a perfect fit for the ripper application. The collection of encoder factory objects should be treated as a single logical object, and replication makes that possible.

### Adding a Replica Group to the Ripper Deployment

Adding a replica group descriptor to our application is very straightforward:

**XML**

```
<icegrid>
    <application name="Ripper">
        <replica-group id="EncoderAdapters">
            <object identity="EncoderFactory" type="::Ripper::MP3EncoderFactory"/>
        </replica-group>
        <server-template id="EncoderServerTemplate">
            <parameter name="index"/>
            <parameter name="exepath" default="/opt/ripper/bin/server"/>
            <server id="EncoderServer${index}" exe="${exepath}" activation="on-demand">
                <adapter name="EncoderAdapter" replica-group="EncoderAdapters"
                    endpoints="tcp"/>
            </server>
        </server-template>
        <node name="Node1">
            <server-instance template="EncoderServerTemplate" index="1"/>
        </node>
        <node name="Node2">
            <server-instance template="EncoderServerTemplate" index="2"/>
        </node>
    </application>
</icegrid>
```

The new descriptor adds the replica group called `EncoderAdapters` and registers a well-known object with the identity `EncoderFactory`. The adapter descriptor in the server template has been changed to declare its membership in the replica group.

### Using a Replica Group in the Ripper Client

In comparison to the examples that demonstrated querying for well-known objects, the new version of our client has become much simpler:

```
Ice::ObjectPrx obj = communicator->stringToProxy("EncoderFactory");
Ripper::MP3EncoderFactoryPrx factory = Ripper::MP3EncoderFactoryPrx::checkedCast(obj);
Ripper::MP3EncoderPrx encoder = factory->createEncoder();
```

The client no longer needs to use the `IceGrid::Query` interface, but simply creates a proxy for a well-known object and lets the Ice run time transparently interact with the location service. In response to a locate request for `EncoderFactory`, the registry returns a proxy containing the endpoints of both object adapters. The Ice run time in the client selects one of the endpoints at random, meaning we have now lost some functionality compared to the prior example in which system load was considered when selecting an endpoint. We will learn how to rectify this situation in our discussion of load balancing.

#### See Also

- Terminology
- Replica-Group Descriptor Element
- Object Descriptor Element
- Well-Known Objects
- Load Balancing

Copyright © 2017, ZeroC, Inc.