Resource Allocation using IceGrid Sessions

IceGrid provides a resource allocation facility that coordinates access to the objects and servers of an IceGrid application. To allocate a resource for exclusive use, a client must first establish a session by authenticating itself with the IceGrid registry or a Glacier2 router, after which the client may reserve objects and servers that the application indicates are allocatable. The client should release the resource when it is no longer needed, otherwise IceGrid reclaims it when the client's session terminates or expires due to inactivity.

An allocatable server offers at least one allocatable object. The server is considered to be allocated when its first allocatable object is claimed, and is not released until all of its allocated objects are released. While the server is allocated by a client, no other clients can allocate its objects.

On this page:

Creating an IceGrid Session

A client must create an IceGrid session before it can allocate objects. If you have configured a Glacier2 router to use IceGrid's session managers, the client's router session satisfies this requirement.

In the absence of Glacier2, an IceGrid client invokes `createSession` or `createSessionFromSecureConnection` on IceGrid's `Registry` interface to create a session:

Slice

```
module IceGrid {
    exception PermissionDeniedException {
        string reason;
    };

    interface Registry {
        Session* createSession(string userId, string password)
            throws PermissionDeniedException;

        Session* createSessionFromSecureConnection()
            throws PermissionDeniedException;

        idempotent int getSessionTimeout();
    };
};
```

The `createSession` operation expects a username and password and returns a session proxy if the client is allowed to create a session. By default, IceGrid does not allow the creation of sessions. You must define the registry property `IceGrid.Registry.PermissionsVerifier` with the proxy of a permissions verifier object to enable session creation with `createSession`.

The `createSessionFromSecureConnection` operation does not require a username and password because it uses the credentials supplied by an SSL connection to authenticate the client. As with `createSession`, you must enable session creation by configuring the proxy of a permissions verifier object so that clients can use `createSessionFromSecureConnection` to create a session. In this case, the property is `IceGrid.Registry.SSLPermissionsVerifier`.

To create a session, the client obtains the registry proxy by converting the well-known proxy string `"IceGrid/Registry"` to a proxy object with the communicator, downcasts the proxy to the `IceGrid::Registry` interface, and invokes on one of the operations. The sample code below demonstrates how to do it in C++; the code will look very similar in other language mappings.

C++

```
Ice::ObjectPrx base = communicator->stringToProxy("IceGrid/Registry");
IceGrid::RegistryPrx registry = IceGrid::RegistryPrx::checkedCast(base);
string username = ...;
string password = ...;
IceGrid::SessionPrx session;
try {
    session = registry->createSession(username, password);
} catch (const IceGrid::PermissionDeniedException & ex) {
    cout << "permission denied:\n" << ex.reason << endl;
}
```

The identity of the registry object may change based on its configuration settings.

After creating the session, the client must keep it alive by periodically invoking its `keepAlive` operation. The session expires if the client does not invoke `keepAlive` within the configured timeout period, which can be obtained by calling the `getSessionTimeout` operation on the `Registry` interface.

If a session times out, or if the client explicitly terminates the session by invoking its `destroy` operation, IceGrid automatically releases all objects allocated using that session.

Controlling Access to IceGrid Sessions

As described above, you must configure the IceGrid registry with the proxy of at least one permissions verifier object to enable session creation:

- `IceGrid.Registry.PermissionsVerifier`
  This property supplies the proxy of an object that implements the interface `Glacier2::PermissionsVerifier`. Defining this property allows clients to create sessions using `createSession`.
- `IceGrid.Registry.SSLPermissionsVerifier`
  This property supplies the proxy of an object that implements the interface `Glacier2::SSLPermissionsVerifier`. Defining this property allows clients to create sessions using `createSessionFromSecureConnection`.

IceGrid supplies built-in permissions verifier objects:

- A null permissions verifier for TCP/IP. This object accepts any username and password and should only be used in a secure environment where no access control is necessary. You select this verifier object by defining the following configuration property:

  ```
  IceGrid.Registry.PermissionsVerifier=<instance-name>/NullPermissionsVerifier
  ```

  Note that you have to substitute the correct instance name for the object identity category.
- A null permissions verifier for SSL, analogous to the one for TCP/IP. You select this verifier object by defining the following configuration property:

  ```
  IceGrid.Registry.SSLPermissionsVerifier=<instance-name>/NullSSLPermissionsVerifier
  ```
- A file-based permissions verifier. This object uses an access control list in a file that contains username-password pairs. The format of the password file is the same as the format of Glacier2 password files. You enable this verifier implementation by defining the configuration property `IceGrid.Registry.CryptPasswords` with the pathname of the password file. Note that this property is ignored if you specify the proxy of a permissions verifier object using `IceGrid.Registry.PermissionsVerifier`.

You can also implement your own permissions verifier object.

Allocating Objects with an IceGrid Session

A client allocates objects using the session proxy returned from `createSession` or `createSessionFromSecureConnection`. The proxy supports the `Session` interface shown below:

Slice

```
module IceGrid {
    exception ObjectNotRegisteredException {
        Ice::Identity id;
    };

    exception AllocationException {
        string reason;
    };

    exception AllocationTimeoutException
        extends AllocationException {
    };

    interface Session extends Glacier2::Session {

        idempotent void keepAlive();

        Object* allocateObjectById(Ice::Identity id)
            throws ObjectNotRegisteredException,
                   AllocationException;

        Object* allocateObjectByType(string type)
            throws AllocationException;

        void releaseObject(Ice::Identity id)
            throws ObjectNotRegisteredException,
                   AllocationException;

        idempotent void setAllocationTimeout(int timeout);
    };
};
```

The client is responsible for keeping the session alive by periodically invoking `keepAlive`, as discussed earlier.

The `allocateObjectById` operation allocates and returns the proxy for the allocatable object with the given identity. If no allocatable object with the given identity is registered, the client receives `ObjectNotRegisteredException`. If the object cannot be allocated, the client receives `AllocationException`. An allocation attempt can fail for the following reasons:

- the object is already allocated by the session
- the object is allocated by another session and did not become available during the configured allocation timeout period
- the session was destroyed.

The `allocateObjectByType` operation allocates and returns a proxy for an allocatable object registered with the given type. If more than one allocatable object is registered with the given type, the registry selects one at random. The client receives `AllocationException` if no objects with the given type could be allocated. An allocation attempt can fail for the following reasons:

- no objects are registered with the given type
- all objects with the given type are already allocated (either by this session or other sessions) and none became available during the configured allocation timeout period
- the session was destroyed.

The `releaseObject` operation releases an object allocated by the session. The client receives `ObjectNotRegisteredException` if no allocatable object is registered with the given identity and `AllocationException` if the object is not allocated by the session. Upon session destruction, IceGrid automatically releases all allocated objects.

The `setAllocationTimeout` operation configures the timeout used by the allocation operations. If no allocatable objects are available when the client invokes `allocateObjectById` or `allocateObjectByType`, IceGrid waits for the specified timeout period for an allocatable object to become available. If the timeout expires, the client receives `AllocationTimeoutException`.

Allocating Servers with an IceGrid Session

A client does not need to explicitly allocate a server. If a server is allocatable, IceGrid implicitly allocates it to the first client that claims one of the server's allocatable objects. Likewise, IceGrid releases the server when all of its allocatable objects are released.

Server allocation is useful in two situations:

- Only allocatable servers can use the `session` activation mode, in which the server is activated on demand when allocated by a client and deactivated upon release.
- An allocatable server can be secured with IceSSL or Glacier2 so that its objects can only be invoked by the client that allocated it.

Security Considerations for Allocated Resources

IceGrid's resource allocation facility allows clients to coordinate access to objects and servers but does not place any restrictions on client invocations to allocated objects; any client that has a proxy for an allocated object could conceivably invoke an operation on it. IceGrid assumes that clients are cooperating with each other and respecting allocation semantics.

To prevent unauthorized clients from invoking operations on an allocated object or server, you can use IceSSL or Glacier2:

- Using IceSSL, you can secure access to a server or a particular object adapter with the properties `IceSSL.TrustOnly.Server` or `IceSSL.TrustOnly.Server.AdapterName`. For example, if you configure a server with the session activation mode, you can set one of the `IceSSL.TrustOnly` properties to the `${session.id}` variable, which is substituted with the session ID when the server is activated for the session. If the IceGrid session was created from a secure connection, the session ID will be the distinguished name associated with the secure connection, which effectively restricts access to the server or one of its adapters to the client that established the session with IceGrid.

- With Glacier2, you can secure access to an allocated object or the object adapters of an allocated server with the Glacier2 filtering mechanism. By default, IceGrid sessions created with a Glacier2 router are automatically given access to allocated objects, allocatable objects, certain well-known objects, and the object adapters of allocated servers.

Deploying Allocatable Resources

Allocatable objects are registered using a descriptor that is similar to well-known object descriptors. Allocatable objects cannot be replicated and therefore can only be specified within an object adapter descriptor.

Servers can be specified as allocatable by setting the server descriptor's `allocatable` attribute.

As an example, the following application defines an allocatable server and an allocatable object:

XML

```
<icegrid>
    <application name="Ripper">
        <node name="Node1">
            <server id="EncoderServer"
                exe="/opt/ripper/bin/server"
                activation="on-demand"
                allocatable="true">
                <adapter name="EncoderAdapter" id="EncoderAdapter" endpoints="tcp">
                    <allocatable identity="EncoderFactory" type="::Ripper::MP3EncoderFactory"/>
                </adapter>
            </server>
        </node>
    </application>
</icegrid>
```

Using Resource Allocation in the Ripper Application

We can use the allocation facility in our MP3 encoder factory to coordinate access to the MP3 encoder factories. First we need to modify the descriptors to define an allocatable object:

XML

```
<icegrid>
    <application name="Ripper">
        <server-template id="EncoderServerTemplate">
            <parameter name="index"/>
            <server id="EncoderServer${index}"
                exe="/opt/ripper/bin/server"
                activation="on-demand">
                <adapter name="EncoderAdapter"  endpoints="tcp">
                    <allocatable identity="EncoderFactory${index}"
                        type="::Ripper::MP3EncoderFactory"/>
                </adapter>
            </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>
```

Next, the client needs to create a session and allocate a factory:

C++

```
Ice::ObjectPrx obj = session->allocateObjectByType(Ripper::MP3EncoderFactory::ice_staticId());
try {
    Ripper::MP3EncoderPrx encoder = factory->createEncoder();
    // Use the encoder to encode a file ...
}
catch (const Ice::LocalException & ex) {
    // There was a problem with the encoding, we catch the
    // exception to make sure we release the factory.
}
session->releaseObject(obj->ice_getIdentity());
```

It is important to release an allocated object when it is no longer needed so that other clients may use it. If you forget to release an object, it remains allocated until the session is destroyed.

See Also

- Getting Started with Glacier2
- IceSSL
- Well-Known Registry Objects
- Securing a Glacier2 Router
- Object Descriptor Element
- Allocatable Descriptor Element
- IceGrid Properties
- IceSSL Properties