Asynchronous Dynamic Invocation and Dispatch in C-Sharp

Previous Next

This page describes the asynchronous C# mapping for the `ice_invoke` proxy function and the `Blobject` class.

On this page:

Calling `ice_invoke` Asynchronously in C# with Tasks

The asynchronous mapping for `ice_invoke` resembles that of the static AMI mapping. The return values and the parameters `operation`, `mode`, and `inParams` have the same semantics as for the synchronous version of `ice_invoke`.

The `ice_invokeAsync` method has the following signature:

C#

```
System.Threading.Tasks.Task<Ice.Object_Ice_invokeResult>
ice_invokeAsync(string operation,
                Ice.OperationMode mode,
                byte[] inParams,
                Ice.OptionalContext context = new Ice.OptionalContext(),
                System.IProgress<bool> progress = null,
                System.Threading.CancellationToken cancel = new System.Threading.CancellationToken());
```

The method sends (or queues) an invocation of the given operation and does not block the calling thread. It returns a `Task` that you can use in a number of ways, including blocking to obtain the result, configuring a continuation to be executed when the result becomes available, and polling to check the status of the request. Refer to the static AMI mapping for more information on the `context`, `progress` and `cancel` arguments.

The `ice_invokeAsync` signature is consistent with the AMI mapping of operations that return multiple values, therefore it uses a structure as its result type:

C#

```
namespace Ice
{
    public struct Object_Ice_invokeResult
    {
        public Object_Ice_invokeResult(bool returnValue, byte[] outEncaps);
        public bool returnValue;
        public byte[] outEncaps;
    }
}
```

User exceptions are handled differently than for static asynchronous invocations. Calling `ice_invokeAsync` can raise Ice run-time exceptions but never raises user exceptions. Instead, the `returnValue` member of `Object_Ice_invokeResult` indicates whether the operation completed successfully (true) or raised a user exception (false). If `returnValue` is true, the byte sequence in `outEncaps` contains an encapsulation of the results; otherwise, the byte sequence in `outEncaps` contains an encapsulation of the user exception.

Back to Top ^

Calling `ice_invoke` Asynchronously in C# with AsyncResult

The AsyncResult API is deprecated and provided only for backward compatibility. New applications should use the Task API.

The asynchronous mapping for `ice_invoke` resembles that of the static AMI mapping. Multiple overloadings are provided to support the use of callbacks and request contexts. The return value and the parameters `operation`, `mode`, and `inParams` have the same semantics as for the synchronous version of `ice_invoke`.

Basic Asynchronous Mapping for `ice_invoke` in C#

The basic mapping is shown below:

C#

```
Ice.AsyncResult<Ice.Callback_Object_ice_invoke>
begin_ice_invoke(
    string operation,
    Ice.OperationMode mode,
    byte[] inParams);

Ice.AsyncResult<Callback_Object_ice_invoke>
begin_ice_invoke(
    string operation,
    Ice.OperationMode mode,
    byte[] inParams,
    Dictionary<string, string> context);

bool end_ice_invoke(out byte[] outParams, AsyncResult r);
```

User exceptions are handled differently than for static asynchronous invocations. Calling `end_ice_invoke` can raise Ice run-time exceptions but never raises user exceptions. Instead, the boolean return value of `end_ice_invoke` indicates whether the operation completed successfully (true) or raised a user exception (false). If the return value is true, the byte sequence contains an encapsulation of the results; otherwise, the byte sequence contains an encapsulation of the user exception.

Back to Top ^

Generic Asynchronous Callback Mapping for `ice_invoke` in C#

The generic callback API is also available:

C#

```
Ice.AsyncResult begin_ice_invoke(
    string operation,
    Ice.OperationMode mode,
    byte[] inParams,
    Ice.AsyncCallback cb,
    object cookie);

Ice.AsyncResult begin_ice_invoke(
    string operation,
    Ice.OperationMode mode,
    byte[] inParams,
    Dictionary<string, string> context,
    Ice.AsyncCallback cb,
    object cookie);
```

Refer to the static AMI mapping for a callback example.

Back to Top ^

Type-Safe Asynchronous Callback Mapping for `ice_invoke` in C#

For the type-safe callback API, you register callbacks on the `AsyncResult` object just as in the static AMI mapping:

C#

```
public class MyCallback
{
    public void responseCB(bool ret, byte[] results)
    {
        if(ret)
        {
            System.Console.Out.WriteLine("Success");
        }
        else
        {
            System.Console.Out.WriteLine("User exception");
        }
    }

    public void failureCB(Ice.Exception ex)
    {
        System.Console.Err.WriteLine("Exception is: " + ex);
    }
}

...

Ice.AsyncResult<Ice.Callback_Object_ice_invoke> r = proxy.begin_ice_invoke(...);
MyCallback cb = new MyCallback();
r.whenCompleted(cb.responseCB, cb.failureCB);
```

The caller invokes `whenCompleted` on the `AsyncResult` object and supplies delegates to handle response and failure. The response delegate must match the signature of `Ice.Callback_Object_ice_invoke`:

C#

```
public delegate void Callback_Object_ice_invoke(bool ret__, byte[] outParams);
```

Back to Top ^

Subclassing `BlobjectAsync` in C#

`BlobjectAsync` is the name of the asynchronous counterpart to `Blobject`:

C#

```
public abstract class BlobjectAsync : Ice.ObjectImpl
{
    public abstract void ice_invoke_async(
        AMD_Object_ice_invoke cb,
        byte[] inParams,
        Current current);
}
```

To implement asynchronous dynamic dispatch, a server must subclass `BlobjectAsync` and override `ice_invoke_async`.

The first argument to the servant's member function is a callback object of type `Ice.AMD_Object_ice_invoke`, shown here:

C#

```
namespace Ice
{
    public interface AMD_Object_ice_invoke
    {
        void ice_response(bool ok, byte[] outParams);
        void ice_exception(System.Exception ex);
    }
}
```

Upon a successful invocation, the servant must invoke `ice_response` on the callback object, passing `true` as the first argument and encoding the encapsulated operation results into `outParams`. To report a user exception, the servant invokes `ice_response` with `false` as the first argument and the encapsulated form of the exception in `outParams`.

The servant may optionally raise a user exception directly and the Ice run time will marshal it for you.

Back to Top ^

See Also

- Asynchronous Method Invocation (AMI) in C-Sharp
- Request Contexts
- Dynamic Invocation and Dispatch Overview

Previous Next