# The ServantLocator Interface

A servant locator has the following interface:

**Slice**

```
module Ice {
    local interface ServantLocator {
        ["UserException"]
        Object locate(Current curr, out LocalObject cookie);

        ["UserException"]
        void finished(Current curr, Object servant, LocalObject cookie);

        void deactivate(string category);
    };
};
```

Note that `ServantLocator` is a local interface. To create an actual implementation of a servant locator, you must define a class that is derived from `Ice::ServantLocator` and provide implementations of the `locate`, `finished`, and `deactivate` operations. The Ice run time invokes the operations on your derived class as follows:

- `locate`

  Whenever a request arrives for which no entry exists in the active servant map (ASM), the Ice run time calls `locate` and supplies the `Current` object for the request. The implementation of `locate` (which you provide as part of the derived class) is supposed to return a servant that can process the incoming request. Your implementation of `locate` can behave in three possible ways:

  Unknown macro: 'ol'

  You can also throw user exceptions from `locate`. If the user exception is in the corresponding operation's exception specification, that user exception is returned to the client. User exceptions thrown by `locate` that are not listed in the exception specification of the corresponding operation are also returned to the client, and then thrown in the client as `UnknownUserException`. Non-Ice exceptions are returned to the client as `UnknownException`.

  The `cookie` out-parameter to `locate` allows you to return a local object to the object adapter. The object adapter does not care about the contents of that object (and it is legal to return a null cookie). Instead, the Ice run time passes whatever cookie you return from `locate` back to you when it calls `finished`. This allows you to pass an arbitrary amount of state from `locate` to the corresponding call to `finished`.

- `finished`

  If a call to `locate` has returned a servant to the Ice run time, the Ice run time dispatches the incoming request to the servant. Once the request is complete (that is, the operation being invoked has completed), the Ice run time calls `finished`, passing the servant whose operation has completed, the `Current` object for the request, and the cookie that was initially created by `locate`. This means that every call to `locate` is balanced by a corresponding call to `finished` (provided that `locate` actually returned a servant).

  If you throw an exception from `finished`, the Ice run time propagates the thrown exception back to the client. As for `locate`, you can throw user exceptions from `finished`. If a user exception is in the corresponding operation's exception specification, that user exception is returned to the client. User exceptions that are not in the corresponding operation's exception specification are also returned to the client, and then thrown by the client as `UnknownUserException`.

  `finished` can also throw run-time exceptions. However, only `ObjectNotExistException`, `OperationNotExistException`, and `FacetNotExistException` are propagated without change to the client; other run-time exceptions are returned to the client as `UnknownLocalException`.

  Non-Ice exceptions thrown from `finished` are returned to the client as `UnknownException`.

  If both the operation implementation and `finished` throw a user exception, the exception thrown by `finished` overrides the exception thrown by the operation.

- `deactivate`

  The `deactivate` operation allows a servant locator to clean up once it is no longer needed. (For example, the locator might close a database connection.) The Ice run time passes the category of the servant locator being deactivated to the `deactivate` operation.

  The run time calls `deactivate` when destroying the object adapter to which the servant locator is attached. More precisely, `deactivate` is called when you call `destroy` on the object adapter, or when you call `destroy` on the communicator (which implicitly calls `destroy` on the object adapter).

Once the run time has called `deactivate`, it is guaranteed that no further calls to `locate` or `finished` can happen, that is, `deactivate` is called exactly once, after all operations dispatched via this servant locator have completed.

This also explains why `deactivate` is not called as part of `ObjectAdapter::deactivate`: `ObjectAdapter::deactivate` initiates deactivation and returns immediately, so it cannot call `ServantLocator::deactivate` directly, because there might still be outstanding requests dispatched via this servant locator that have to complete first — in turn, this would mean that either `ObjectAdapter::deactivate` could block (which it must not do) or that a call to `ServantLocator::deactivate` could be followed by one or more calls to `finished` (which must not happen either).

It is important to realize that the Ice run time does not "remember" the servant that is returned by a particular call to `locate`. Instead, the Ice run time simply dispatches an incoming request to the servant returned by `locate` and, once the request is complete, calls `finished`. In particular, if two requests for the same servant arrive more or less simultaneously, the Ice run time calls `locate` and `finished` once for each request. In other words, `locate` establishes the association between an object identity and a servant; that association is valid only for a single request and is never used by the Ice run time to dispatch a different request.

## See Also

- The Active Servant Map
- The Current Object
- Run-Time Exceptions
- Object Adapter States