Object Identity

On this page:

The `Ice::Identity` Type

Each Ice object has an object identity defined as follows:

Slice

```
module Ice {
    struct Identity {
        string name;
        string category;
    };
};
```

As you can see, an object identity consists of a pair of strings, a `name` and a `category`. The complete object identity is the combination of `name` and `category`, that is, for two identities to be equal, both `name` and `category` must be the same. The `category` member is usually the empty string, unless you are using servant locators, default servants or callbacks with Glacier2.

If `name` is an empty string, `category` must be the empty string as well. (An identity with an empty `name` and a non-empty `category` is illegal.) If a proxy contains an identity in which `name` is empty, Ice interprets that proxy as a null proxy.

Object identities can be represented as strings; the category part appears first and is followed by the name; the two components are separated by a `/` character, for example:

```
Factory/File
```

In this example, `Factory` is the category, and `File` is the name. If the `name` or `category` member themselves contain a `/` character, the stringified representation escapes the `/` character with a `\`, for example:

```
Factories\/Factory/Node\/File
```

In this example, the category is `Factories/Factory` and the name is `Node/File`.

Syntax for Stringified Identities

You rarely need to write identities as strings because, typically, your code will be using the identity helper functions `identityToString` and `stringToIdentity`, or simply deal with proxies instead of identities. However, on occasion, you will need to use stringified identities in configuration files. If the identities happen to contain meta-characters (such as a slash or backslash), or characters outside the printable ASCII range, these characters must be escaped in the stringified representation. Here are rules that the Ice run time applies when parsing a stringified identity:

1. The parser scans the stringified identity for an unescaped slash character (`/`). If such a slash character can be found, the substrings to the left and right of the slash are parsed as the `category` and `name` members of the identity, respectively; if no such slash character can be found, the entire string is parsed as the `name` member of the identity, and the `category` member is the empty string.
2. Each of the `category` (if present) and `name` substrings is parsed according to the following rules:
    - All characters in the string must be in the ASCII range 32 (space) to 126 (~); characters outside this range cause the parse to fail.
    - Any character that is not part of an escape sequence is treated as that character.
    - The parser recognizes the following escape sequences and replaces them with their equivalent character:
      `\\` (backslash)
      `\'` (single quote)
      `\"` (double quote)
      `\b` (space)
      `\f` (form feed)

\n (new line)
\r (carriage return)
\t (tab)

- An escape sequence of the form \o, \oo, or \ooo (where *o* is a digit in the range 0 to 7) is replaced with the ASCII character with the corresponding octal value. Parsing for octal digits allows for at most three consecutive digits, so the string \0763 is interpreted as the character with octal value 76 (>) followed by the character 3. Parsing for octal digits terminates as soon as it encounters a character that is not in the range 0 to 7, so \7x is the character with octal value 7 (bell) followed by the character x. Octal escape sequences must be in the range 0 to 255 (octal 000 to 377); escape sequences outside this range cause a parsing error. For example, \539 is an illegal escape sequence.
- If a character follows a backslash, but is not part of a recognized escape sequence, the backslash is ignored, so \x is the character .

Identity Helper Functions

To make conversion of identities to and from strings easier, Ice provides functions to convert an `Identity` to and from a native string, using the string format described in the preceding paragraph.

In C++, these helper functions are in the Ice namespace:

C++

```
namespace Ice
{
    std::string identityToString(const Identity&);
    Identity    stringToIdentity(const std::string&);
}
```

For Java, the utility functions are in the `Ice.Util` class and are defined as:

Java

```
package Ice;

public final class Util {
    public static String   identityToString(Identity id);
    public static Identity stringToIdentity(String s);
}
```

For C#, the utility functions are in the `Ice.Util` class and are defined as:

C#

```
namespace Ice
{
    public sealed class Util
    {
        public static string   identityToString(Identity id);
        public static Identity stringToIdentity(string s);
    }
}
```

The Python functions are in the `Ice` module:

Python

```
def identityToString(ident)
def stringToIdentity(str)
```

The PHP functions are in the `Ice` namespace:

PHP

```
function identityToString($str)
function stringToIdentity($ident)
```

The Ruby functions are in the `Ice` module:

Ruby

```
def Ice.identityToString(str)
def Ice.stringToIdentity(ident)
```

The local interface Communicator provides two equivalent operations, `identityToString` and `stringToIdentity`:

Slice

```
module Ice
{
local interface Communicator {
   Identity stringToIdentity(string str);
   string   identityToString(Identity ident);
};
};
```

In Ice 3.6.1 and prior releases, Ice provides these helper functions in C++, Objective-C, Ruby, PHP and Python only on the Communicator interface; the freestanding functions were accidentally left out.

See Also

- Servant Activation and Deactivation
- Servant Locators
- Default Servants
- C++ Strings and Character Encoding
- Glacier2