

# Parsing Properties

The [Properties](#) interface provides several operations for converting properties to and from command-line options.

On this page:

- [Converting Properties to Command-Line Options](#)
- [Converting Command-Line Options to Properties](#)
- [Converting Reserved Command-Line Options to Properties](#)

## Converting Properties to Command-Line Options

The `getCommandLineOptions` operation converts an initialized set of properties into a sequence of equivalent command-line options:

### Slice

```
module Ice {
    local interface Properties {

        StringSeq getCommandLineOptions();

        // ...
    };
};
```

For example, if you have set the `Filesystem.MaxFileSize` property to 1024 and call `getCommandLineOptions`, the setting is returned as the string `--Filesystem.MaxFileSize=1024`. This operation is useful for diagnostic purposes, for example, to dump the setting of all properties to a [logging facility](#), or if you want to fork a new process with the same property settings as the current process.

## Converting Command-Line Options to Properties

The `parseCommandLineOptions` operation examines the passed argument vector for command-line options that have the specified prefix:

### Slice

```
module Ice {
    local interface Properties {

        StringSeq parseCommandLineOptions(string prefix, StringSeq options);

        // ...
    };
};
```

All options having the form `--prefix.Key=Value` are converted to property settings (that is, they initialize the corresponding properties). The operation returns an argument vector containing the options that did not match the prefix.

The value for `prefix` has an implicit trailing period if one is not present. For example, when calling `parseCommandLineOptions` with a prefix value of `"File"`, the option `--File.Owner=root` would match but the option `--Filesystem.MaxFileSize=1024` would not match.

The operation parses command-line options using the same [syntax rules](#) as for properties in a configuration file. However, the user's command shell can cause differences in parsing behavior. Suppose we define the following property in a configuration file:

```
MyApp.Home=C:\Program Files\MyApp
```

The presence of whitespace in the property definition is not an issue in a configuration file but can be an issue on the command line, where the equivalent option is `--MyApp.Home=C:\Program Files\MyApp`. If the user is not careful, the program may receive this as two separate options: `-MyApp.Home=C:\Program` and `Files\MyApp`. In the end, it is the user's responsibility to ensure that the property's complete key and value are contained within a single command-line option.

Because `parseCommandLineOptions` expects a sequence of strings, but C++ programs are used to dealing with `argc` and `argv`, Ice provides two utility functions that convert an `argc/argv` vector into a sequence of strings and vice-versa:

**C++**

```
namespace Ice {

    StringSeq argsToStringSeq(int argc, char* argv[]);

    void stringSeqToArgs(const StringSeq& args, int& argc, char* argv[]);

}
```

You need to use `parseCommandLineOptions` (and the utility functions) if you want to permit application-specific properties to be set from the command line. For example, to allow the `--Filesystem.MaxFileSize` option to be used on the command line, we need to initialize our program as follows:

**C++**

```
int
main(int argc, char* argv[])
{
    // Create an empty property set.
    //
    Ice::PropertiesPtr props = Ice::createProperties();

    // Convert argc/argv to a string sequence.
    //
    Ice::StringSeq args = Ice::argsToStringSeq(argc, argv);

    // Strip out all options beginning with --Filesystem.
    //
    args = props->parseCommandLineOptions("Filesystem", args);

    // args now contains only those options that were not
    // stripped. Any options beginning with --Filesystem have
    // been converted to properties.

    // Convert remaining arguments back to argc/argv vector.
    //
    Ice::stringSeqToArgs(args, argc, argv);

    // Initialize communicator.
    //
    Ice::InitializationData id;
    id.properties = props;
    Ice::CommunicatorPtr ic = Ice::initialize(argc, argv, id);

    // At this point, argc/argv only contain options that
    // set neither an Ice property nor a Filesystem property,
    // so we can parse these options as usual.
    //
    // ...
}
```

Using this code, any options beginning with `--Filesystem` are converted to properties and are available via the property [lookup operations](#) as usual. The call to `initialize` then removes any Ice-specific command-line options so, once the communicator is created, `argc/argv` only contains options and arguments that are not related to setting either a filesystem or an Ice property.

An easier way to achieve the same thing is to use the overload of `Ice::initialize` that accepts a string sequence, instead of an `argc/argv` pair:

#### C++

```
int
main(int argc, char* argv[])
{
    // Create an empty property set.
    //
    Ice::PropertiesPtr props = Ice::createProperties();

    // Convert argc/argv to a string sequence.
    //
    Ice::StringSeq args = Ice::argsToStringSeq(argc, argv);

    // Strip out all options beginning with --Filesystem.
    //
    args = props->parseCommandLineOptions("Filesystem", args);

    // args now contains only those options that were not
    // stripped. Any options beginning with --Filesystem have
    // been converted to properties.

    // Initialize communicator.
    //
    Ice::InitializationData id;
    id.properties = props;
    Ice::CommunicatorPtr ic = Ice::initialize(args, id);

    // At this point, args only contains options that
    // set neither an Ice property nor a Filesystem property,
    // so we can parse these options as usual.
    //
    // ...
}
```

This version of the code avoids having to convert the string sequence back into an `argc/argv` pair before calling `Ice::initialize`.

## Converting Reserved Command-Line Options to Properties

The `parseIceCommandLineOptions` operation behaves like `parseCommandLineOptions`, but removes the [reserved](#) Ice-specific options from the argument vector:

#### Slice

```
module Ice {
    local interface Properties {

        StringSeq parseIceCommandLineOptions(StringSeq options);

        // ...
    };
};
```

This operation is also used internally by the Ice run time to parse Ice-specific options in `initialize`.

#### See Also

- [Properties Overview](#)
- [The Properties Interface](#)
- [Reading Properties](#)

- [Setting Properties](#)
- [Command-Line Parsing and Initialization](#)
- [Logger Facility](#)