Customizing the MATLAB Mapping

You can customize the code that the Slice-to-MATLAB compiler produces by annotating your Slice definitions with metadata. This page describes how metadata influences the generated MATLAB code.

On this page:

MATLAB Packages

By default, the scope of a Slice definition determines the package of its mapped MATLAB construct. A Slice type defined in a module hierarchy is mapped to a type residing in the equivalent MATLAB package.

There are times when applications require greater control over the packaging of generated MATLAB classes. For instance, a company may have software development guidelines that require all MATLAB classes to reside in a designated package. One way to satisfy this requirement is to modify the Slice module hierarchy so that the generated code uses the required package by default. In the example below, we have enclosed the original definition of `Workflow::Document` in the modules `com::acme` so that the compiler will create the class in the `com.acme` package:

Slice

```
module com
{
    module acme
    {
        module Workflow
        {
            class Document
            {
                // ...
            }
        }
    }
}
```

There are two problems with this workaround:

1. It incorporates the requirements of an implementation language into the application's interface specification.
2. Developers using other languages, such as C++, are also affected.

The Slice-to-MATLAB compiler provides a better way to control the packages of generated code through the use of global metadata. The example above can be converted as follows:

Slice

```
[["matlab:package:com.acme"]]
module Workflow
{
    class Document
    {
        // ...
    }
}
```

The global metadata directive `matlab:package:com.acme` instructs the compiler to generate all of the classes resulting from definitions in this Slice file into the MATLAB package `com.acme`. The net effect is the same: the class for `Document` is generated in the package `com.acme.Workflow`. However, we have addressed the two shortcomings of the first solution by reducing our impact on the interface specification: the Slice-to-MATLAB compiler recognizes the package metadata directive and modifies its actions accordingly, whereas the compilers for other language mappings simply ignore it.

Back to Top ^

Package Configuration Properties

Using global metadata to alter the default package of generated classes has ramifications for the Ice run time when unmarshaling exceptions and concrete class types. The Ice run time dynamically loads generated classes by translating their Slice type IDs into MATLAB class names. For example, the Ice run time translates the Slice type ID `::Workflow::Document` into the class name `Workflow.Document`.

However, when the generated classes are placed in a user-specified package, the Ice run time can no longer rely on the direct translation of a Slice type ID into a MATLAB class name, and therefore requires additional configuration so that it can successfully locate the generated classes. Two configuration properties are supported:

- `Ice.Package.`*`Module`*`=`*`package`*
  Associates a top-level Slice module with the package in which it was generated.

  > Only top-level module names are allowed; the semantics of global metadata prevent a nested module from being generated into a different package than its enclosing module.

- `Ice.Default.Package=package`
  Specifies a default package to use if other attempts to load a class have failed.

The behavior of the Ice run time when unmarshaling an exception or concrete class is described below:

1. Translate the Slice type ID into a MATLAB class name and attempt to load the class.
2. If that fails, extract the top-level module from the type ID and check for an `Ice.Package` property with a matching module name. If found, prepend the specified package to the class name and try to load the class again.
3. If that fails, check for the presence of `Ice.Default.Package`. If found, prepend the specified package to the class name and try to load the class again.
4. If the class still cannot be loaded, the instance may be sliced.

Continuing our example from the previous section, we can define the following property:

```
Ice.Package.Workflow=com.acme
```

Alternatively, we could achieve the same result with this property:

```
Ice.Default.Package=com.acme
```

Back to Top ^

See Also

- Metadata
- Ice.Default.*
- Miscellaneous Ice.* Properties