Problem

Why can't I import xyz IDL data definition from abc IDL namespace?

Solution

There are a few possibilities:

- The Python CORBA stubs for your IDL interface are not in your *$PYTHONPATH* environment variable
- Another set of Python CORBA stubs are superceding the stubs you are interested in using. There is a very high probability this is the case if the IDL module (i.e., namespace) your IDL interface is defined in is reopened in other ALMA CVS modules **and** you are using < ACS 4.1.4. See ALMASW2002083 in the ALMA SPR system for more details
- You are importing an IDL module which provides definitions used by another IDL module you have already imported. For some reason or another *omniORBPy* restricts users from doing this. Don't worry if this sounds confusing, it is. An example should help:

```
import ACS
import Control #the Control IDL module includes IDL files which utilize the "ACS" IDL module
```

is OK, while there's a strong possibility:

```
import Control
import ACS
```

will throw an exception after the **`import ACS`** statement.

- you've defined **`module xyz`** in IDL where **`xyz`** is a reserved keyword in Python. Look at the IDL->Python CORBA Mapping Specifications available from OMG's website to determine what Python module **`xyz`** gets mapped to.

How you import your IDL interfaces has a great impact on which interfaces are available in your IDL module (namespace). If you have not previously imported the IDL module but you request one of its interfaces, the interface loading code will create the IDL (namespace) module for you and populate it with the information from that **single** interface. Subsequent interface loads will add to this dynamically created IDL (namespace) module. Depending on how many interfaces you load and their dependencies, you may end up with a IDL (namespace) module that contains a small subset of the interfaces.

The fix for this problem is to reload the IDL (namespace) module. This action will force all of the interfaces in the module to be loaded from disk.

To illustrate the fix, here is an example. Suppose the code that is failing in your program looks like this:

```
myModule = __import__("Module", globals(), locals(), ["MyClass"])
myClass = myModule.__dict__.get("MyClass")  # Returns None because "Module" isn't completely loaded
myObject = myClass() # Fails because instance is None
```

You can force ***Module*** to be reloaded by doing the following:

```
myModule = __import__("Module", globals(), locals(), ["MyClass"])
myClass = myModule.__dict__.get("MyClass")  # Returns None because "Module" isn't completely loaded
if myClass is None:
    reload(myModule)  # Reloads "Module" from the file system
    myClass = myModule.__dict__.get("MyClass")  # Returns "MyClass" if it is defined in the "Module"
myObject = myClass() # Works now
```

Related articles

- How can more people do development with ACS on the same machine without disturbing each other?
- Which ports are used by ACS?
- Problems connecting to ACS servers on a remote machine: bad /etc/hosts
- Why does the getComponent method of ZLegacy/ACS.ContainerServices return an object of type None?
- Why are some of my print statements not showing up in the container output section of acscommandcenter?