The Makefile is provided with the module generated from getTemplateForDirectory. In the case of Python server implementation, you need to configure the compilation and installation of a Python Package:

Makefile for Python

```
PY_PACKAGES:=<PackageName>
```

The first thing needed to implement a server, is to import the required modules and classes:

Python Imports

```
#Client stubs and definitions, such as structs, enums, etc.
import <Module>
#Skeleton infrastructure for server implementation
import <Module>__POA

#Base component implementation
from Acspy.Servants.ACSComponent import ACSComponent
#Services provided by the container to the component
from Acspy.Servants.ContainerServices import ContainerServices
#Basic component lifecycle (initialize, execute, cleanUp and aboutToAbort methods)
from Acspy.Servants.ComponentLifecycle import ComponentLifecycle

#Error definitions for catching exceptions
import ServiceErr
import <Interface>Err

#Error definitions for creating and raising exceptions
import ServiceErrImpl
import <Interface>ErrImpl
```

Besides this, there's the need to create the class using the mentioned infrastructure provided by some of the imported classes:

Python Server Class Definition

```
class <Name>(<Module>__POA.<InterfaceName>, ACSComponent, ContainerServices, ComponentLifecycle):
    def __init__(self):
        ACSComponent.__init__(self)
        ContainerServices.__init__(self)
        self._logger = self.getLogger()
        ... #Custom implementation for the server constructor
```

To access a struct, enum, typedef or definition in a module or in an interface, simply do the following:

Python Types Usage

```
#From IDL <Module>::<EnumName>::<VALUE>
<Module>.<VALUE>

#From IDL <Module>::<Interface>::<Enumname>::<VALUE>
<Module>.<Interface>.<VALUE>
```

To retrieve a component to interact with, simply do the following:

Python Component Interaction

```
#By Name
comp = self.getComponent("<Name>")

#By Interface. Must be at least one component configured as default!
comp = self.getDefaultComponent("IDL:alma/<Module>/<Interface>:1.0")

#Release Components
self.releaseComponent(comp.name)
```

For logging, as you saw in the class constructor, there's a facility provided by the component. To use it, simply do as follows:

Python Logger

```
logger = self.getLogger()
logger.logTrace("...")
logger.logDebug("...")
logger.logInfo("...")
logger.logWarning("...")
logger.logError("...")
```

For catching and raising exceptions:

Python Error Handling

```
#For catching exceptions
except <Interface>Err.<ExceptionName>Ex as e:

#For raising exceptions
raise <Interface>ErrImpl.<ExceptionName>ExImpl().get<ExceptionName>Ex()

#For raising exceptions with parameters
err = <Interface>ErrImpl.<ExceptionName>ExImpl()
err.set<ParamName>(<Value>)
raise err.get<ExceptionName>Ex()

#For logging an error message from the exceptions
err = <Interface>ErrImpl.<ExceptionName>ExImpl()
err.log()
raise err.get<ExceptionName>Ex()
```

Data types mapping:

To check how data types are mapped between IDL and implementation click here.