Understanding the DOF Object Model

The DOF Object Model was created to simplify the implementation and programming tasks associated with the OpenDOF Project and its Object Access Libraries (OALs). The DOF Object Model describes and defines the technological foundation of the OpenDOF Project. It shows how DOF Objects, DOF Interfaces, and other related elements work together to create scalable and reliable network services based on a system of providers and requestors in peer-to-peer relationships.

> **Note**: "DOF" is an acronym for "Distributed Object Framework," which is the underlying technology of the OpenDOF Project. "Distributed" refers to the way objects and related functionality are distributed on a network; "Object" denotes the DOF Object Model (explained here); and "Framework" means that this is a complete solution that can be integrated into your application.

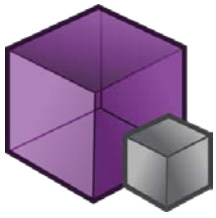
The DOF Object Model and Object Oriented Programming:

If you're already familiar with the concept of Objects and Object Oriented Programming (OOP), you'll recognize many of the terms used to describe the DOF Object Model. However, even though many of the OpenDOF concepts and terms are similar to OOP, there are differences. For example, a DOF Object is not the same as an object in Java. Therefore, whenever a term is used that differs from its common definition, a distinction will be made and the appropriate definition given.

The DOF Object Model is made up of several primary elements. Those elements are

- DOF Objects
- DOF Interfaces
- DOF Items
- DOF Types

DOF Objects



A DOF Object, like a Java object, is a programming construct that has state (data) and behavior (function) and can represent both physical and logical devices. For example, a DOF Object could

represent a light switch. A light switch has “state,” that is, it can be either on or off; and it has behavior, which is the action of flipping the switch or changing its state.

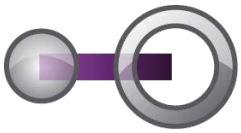
DOF Objects are also collections of functionality. (This “functionality” is defined by DOF Interfaces, which we’ll discuss later.) And as a collection of functionality, the DOF Object is

- Dynamic, which means its capabilities can change
- Defined, which means the capabilities of the object are discoverable

More About DOF Objects

- **Object State**: The current set of all values or readable properties that are defined by the interface. Once those values are stored, it becomes an object snapshot. Object snapshots saved over time are known as an object transaction log.
- **Object Attribute**: Objects can be qualified by attributes. These attributes allow application developers to define relationships and grouping. OIDs are extremely flexible. Any unique identifier can be converted to an OID.
- **Object Discovery**: There is a difference between node discovery and object discovery. The ability to discover a set of objects that are available on a network is separate from discovering the set of nodes on the network. After all, a node can represent multiple objects. DOF makes a sharp distinction between node discovery and object discovery.
- **All DOF objects should be discoverable**, although they do not have to use a standardized procedure. This means that applications may choose to use a common method, but that they can also provide application-specific methods.

The DOF Interface



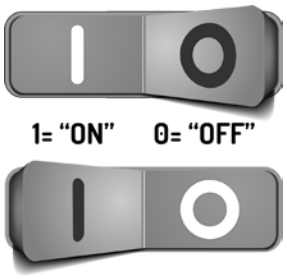
A DOF Interface defines the functionality that a DOF Object must provide, and this is done by defining the properties, functions, events, and exceptions that will be used by the DOF Objects. Furthermore, these definitions establish a “contract” between DOF Objects, specifying how those objects will interact. You could also say that by implementing an interface, the DOF Object forms an “is-a” relationship with the interface—*the object* is a *light switch*—and this governs how the interface and the object react to their environment and any remote management.

Let’s look more closely at our example of a DOF Object that represents a light switch. We said that the DOF Interface defines four elements which determine the behavior of the object: properties, functions, events, and exceptions. We could define those elements as follows:

- Property: The light is “on” or “off.”

- Functions: The switch can be turned “on” or “off,” and the “state” of the switch can be determined.
- Event: If the switch is “on,” send a message.
- Exception: If something unexpected happens, throw an exception. For example, if you turn the switch off, and the light stays on.

Next, let’s take our example and make it more general, more abstract. Instead of a light switch, let’s say our DOF Object—whatever it is—has a state of 1 or 0, and the function is to change the state. Now our DOF Object can represent any number of things because it is the DOF Interface that defines what the data means and how the object behaves.



Interface Definitions

Now that you have a basic understanding of DOF Interfaces, let’s get into the details.

Interface Property

A specific piece of information and if/how it can be modified

An Interface Property is similar to a field in a Java class: it defines one specific type of information. Functions can then be used to read a property, write to a property (change it), or monitor the value of the property. In addition, the interface determines whether the property is readable, writable, or both.

Interface Method

A rule of expected behavior

An Interface Method defines how a DOF Object can “behave.” This definition also includes the method’s input and output types. (Methods can allow multiple inputs and outputs.)

Interface Event

When something expected happens that requires a response.

An Interface Event defines the circumstances that can trigger a response. Events may have output parameters, which includes a list of return types.

Interface Exception

How the Interface responds to the unexpected.

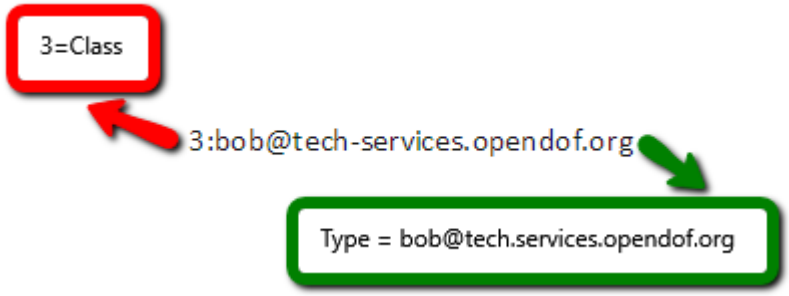
An Interface Exception defines the error conditions that can interrupt the normal flow of an operation; specifically, it is a logical exception case involving the interface items, which are its properties, methods, and events.

Interface Identifiers

In the DOF Object Model, every element is identified as follows:

- **DOF Objects are recognized by an Object Identifier or OID**
  OIDs used in the DOF Object Model are globally unique and stable.

  Note: Although each DOF object requires an OID, OIDs can be used to identify other, non-DOF objects; this is done by identifying the OID Class. For example, is the OID you wish to use an email address? Then it would fall under OID Class 3 in the Class Registry. If you wanted to use a MAC address, that would be Class 2. The Classes allow you to build a globally unique identifier that can be referenced by any DOF application with the correct information. (OID Class Registrations are maintained in a Class Registry.)

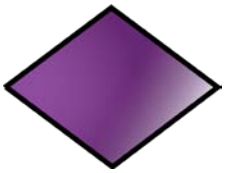


- **DOF Interfaces are recognized by an Interface Identifier or IID**
  Each DOF Interface has a unique Interface Identifier (IID). Once a DOF Interface is published, the IID, along with the interface's defined functionality, remains fixed.
- **Interface Items are recognized by an Item ID**
  Remember, an Interface item is a property, method, events and exceptions. These are not globally unique, but are unique to the Interface. If you are already using Item 1, you cannot create another Item 1 within the same interface, so you move onto the next available item.

The DOF Item

A DOF Item is another way to refer to the definitions in a DOF Interface; in other words, these DOF Items are individual “pieces” of functionality within an Interface (i.e., properties, methods, events, and exceptions). Interface Items are central to a DOF Interface. Most programmers are familiar with these terms and concepts. However, the difference with DOF Interface items is the Item ID. This is an identifier that is unique within the Interface itself. For example, Item ID 1 may refer to a Property, Item ID 2 may refer to a Method, and so on. Both Item IDs will be contained within a single Interface with its own globally unique IID.

The DOF Type



Types define the data. For example, a type could be uint8, structure, string, array, float, and so on. Types are predefined for specific protocols and will define the required metadata. Application developers define the parameters for the item’s metadata (related types, lengths, arrays, etc.) and assign an identifier for each type. Each type requires a unique identifier that must be registered. However, since identifiers are built into the libraries, developers will rarely see them.

Type Identifiers are listed in the Type Identifier Registry.

The DOF Object Model in Review

- A DOF Object has state and behavior.
- A DOF Object is a collection of functionality, and functionality is defined through DOF interfaces.
- A DOF Interface is a collection of Interface Items, which are properties, methods, events, and exceptions.
- An Interface property defines the object’s data.
- An Interface function defines the object’s behavior.

- An Interface event defines the circumstances that trigger a response.
- An Interface exception defines error conditions.

Modifying and Extending Object Behavior

One advantage of working with DOF is the ability to use object refinement and object augmentation. Object refinement allows a specified provider to override the default provider of an object and provide different implementations for some of the same interfaces. This allows, among other things, for systems to resolve potential issues in the implementation of an interface without changing the original provider of the interface.

Developers can increase the functionality of an object through object augmentation. In object augmentation, a provider can add new interfaces to an object that the original provider does not support.

Together, object refinement and object augmentation allow you to create extremely powerful distributed solutions.

More information about object refinement and augmentation is available, but is not within the scope of this document.