DOF Interface Design Concepts

Training

Version 7.0

Table of Contents

Chapter 1: Training Overview

The information in this document has been compiled to provide reference material for use in preparing and delivering training on the concepts involved in creating a DOF interface.

This training is not intended to teach someone how to create an interface, step-by-step; its purpose is to introduce what an interface is and what a designer would need to know in order to design one.

Purpose

The purpose of this training is to introduce the DOF Object Model, review DOF operations, and show how a DOF interface is created, from design to publication.

Audience

This training is appropriate for anyone wanting to know more about the DOF Object Model, DOF interfaces, and how DOF interfaces are created. Trainees should already be familiar with the fundamentals of DOF.

Topics

The training will cover the following topics:

- **Introducing the DOF Object Model**
  This chapter examines the DOF Object Model and introduces the concept of the DOF interface.
- **Understanding DOF Operations**
  This chapter covers the basic concepts of DOF operations.
- **Creating a Simple DOF Interface**
  This chapter follows the process of creating a simple interface and examines each interface component.
- **Principles of Effective DOF Interface Design**
  This chapter introduces the basic principles of effective interface design.
- **The DOF Interface Review and Approval Process**
  This chapter describes the interface review and approval process and the role of the OpenDOF Technical Committee.
- **Using the DOF Interface Repository**
  This section introduces the interface repository and provides basic instructions on its use.
- **Glossary**
  This section defines selected terms used in the training.

Chapter 2: Introducing the DOF Object Model

This chapter introduces the basic concepts of the DOF Object Model and how it relates to DOF interfaces.

Elements of the DOF Object Model

The DOF Object Model is made up of four elements:

1. DOF Objects
2. DOF Interfaces
3. Items (elements of an interface)
4. Types

Object

An object is a programming element with properties and behaviors. A DOF object, which is similar to a generic object as used in object-oriented programming, is a construct that includes a state (data) and related behavior (function). However, DOF objects are also collections of functionality, and these DOF objects provide that functionality through DOF interfaces.

A note on usage: Unless specifically noted, the term "object" will refer to a "DOF object," and the term "interface" will refer to a "DOF Interface."

(Unless otherwise noted, "object" as used in this training refers to an DOF object.)

An object can represent both

- Physical devices
- Logical devices

An object provides functionality and is

- Dynamic, which means their capabilities can change
- Defined, which means all the capabilities of an object are discoverable
- Referenced by an object identifier (OID) that is globally unique

Object Identifiers

Every DOF object has an identifier, which is the Object ID (OID). These OIDs must be globally unique, and OIDs used in the DOF Object Model are unique and stable.

Note: Although each DOF object requires an OID, OIDs can be used to identify other, non-DOF objects.

Current OID Class Registrations are maintained in the OID Class Registry.

Object Attributes

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

It's important to understand that 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.

Object State

Object state is the set of all the current values of readable properties on a particular interface. Object state can be stored or passed through a protocol and becomes an object snapshot. A set of object snapshots stored over time becomes an object history. It is also possible to store requested changes for a particular object, which forms an object transaction log.

Modifying and Extending Object Behavior

One benefit of DOF is object refinement. 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. This is an important aspect of the Distributed Service Platform (DSP).

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

The DOF Interface

A DOF interface is an abstract definition, closely related to the idea of an object type. By implementing an interface, an object forms an “is-a” relationship with the interface that governs how the interface (and by extension, the object) reacts to both its environment and any remote management. For example, consider a desk lamp as an object. It has state (on/off) and behavior (it turns the light on and off). The physical switch would be the interface that defines the behavior or functionality of the object. When the switch is up, the light goes on.

Therefore, an interface defines the items of functionality a DOF object must provide; in other words, interfaces are defined functionality. The interface items establish a "contract" between providers and requesters, allowing the providers and requestors to interoperate predictably. You can also consider an interface as representing capabilities and data.

Note the following:

- A DOF object is a collection of functionality, and functionality is defined through DOF interfaces.
- An interface is a collection of items.
- An item is a *property, method, event,* or *exception.*
- Once published, the functionality defined by an interface is fixed; it cannot change.
- The functionality of an interface is registered.
- An interface is referenced by an interface identifier (IID) that is globally unique.

Interface Identifiers

Each interface has a unique identifier (IID). Once the interface is published, the interface identifier, along with the defined functionality, remains fixed. Developers work with the OpenDOF Technical Committee (OpenDOF TC) to register interfaces. IIDs are assigned during registration.

Current interface definitions are maintained in the Interface Registry.

Published Interfaces

Once the interface is published, it remains fixed, which means the functionality doesn’t change. The functionality is also said to be “registered.”

Interface Items

Interface items are the individual pieces of functionality that combine to create the interface’s functionality.

An item is an individual piece of functionality within an interface and is referenced by an item identifier (Item ID) that is unique within the interface.

Functionality in an interface is defined by any of four items. These interface “items of functionality” are

- Properties
- Methods
- Events
- Exceptions

A single interface can have multiple items of functionality.

Properties

Interface properties are similar to fields in a Java class. They define one specific type of information about the provider. Properties allow for reading, writing, and subscribing to data related to the current object state. Each property has a property type and defines whether it is readable, writable, or both.

If an interface defines readable properties, a requestor can use a *get* operation to retrieve the current value of the property. The requestor could also use a *subscribe* operation to subscribe to the value of the property and be continually updated whenever the value changes. Lastly, the requestor can use a *set* operation to change the value of writeable properties.

Note the following:

- Properties are similar to fields in a Java class.
- A property defines one specific type of information about the provider.
- If an interface defines properties, a requestor can use a *get* operation to retrieve the current value of the property. (A requestor can also use a *subscribe* operation to subscribe to the value of the property and be continually updated whenever the value changes.)
- Requestors can use a *set* operation to change the value of properties if they are defined as writeable, although the interface can define properties as read-only, in which case set operations are not allowed for the property.

Methods

Methods are functions that the requestor can call remotely through an *invoke* operation and that the provider executes in a prescribed fashion (as defined by the interface). The definition of a method includes method input types and method output types. Methods allow multiple inputs and outputs.

Note the following:

- Interface methods are procedures the requestor can call remotely through an *invoke* operation and that the provider executes in a prescribed fashion.
- Methods may have input parameters and return types.

Events

Events are circumstances that trigger a response. Requestors can use *register* operations to be notified when the circumstances have occurred on the provider. Events allow for registrations and have an event type, which includes a list of return types.

Note the following:

- Events are circumstances that trigger a response.
- Requestors can use *register* operations to be notified when the circumstances have occurred on the provider.
- Events may have output parameters.

Exceptions

Although the DOF API defines a number of different types of exceptions, the interface itself will define exceptions when any of its properties, methods, or events have logical exception cases. Exceptions defined in an interface are called provider exceptions in the API and are represented by the DOFProviderException class.

Item Identifiers

Every item in an interface is assigned an identifier, or item ID, by the designers. This number is unique within a particular interface definition and may be as large as 2^{16}-1; however, because most interfaces have only a handful of items, item identifiers will typically be small. Item IDs are registered along with the interface. Once the interface is published, these item IDs cannot be changed.

Type

The "type" defines the data. For example, a type could be any of the following:

- uint8
- structure
- string
- array
- float

Note: Types are defined for a specific DOF protocol set; for example, one set of types is defined for Object Access Protocol (OAP).

Type definitions also include required metadata, such as

- Related types
- Lengths (arrays, for example)
- Other metadata (based on type)

Type Identifiers

Application developers define the parameters for the item’s metadata 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.

Current Type Identifier Registrations are maintained in the Type Identifier Registry.

Bindings

Both objects and interfaces are uniquely identifiable through fixed identifiers. Bindings are a pair of a particular object identifier and interface identifier. Bindings are what requestors use to interact with providers.

Chapter 3: Understanding DOF Operations

This chapter covers the basic concepts of DOF operations.

DOF Operations

An operation in DOF is the communication between providers and requestors that enables applications to use an item of functionality in a DOF interface. Requestors can initiate operations on properties, methods, or events in a DOF interface.

Note: The "methods" above refer to *DOF interface* methods, not *Java API* methods.

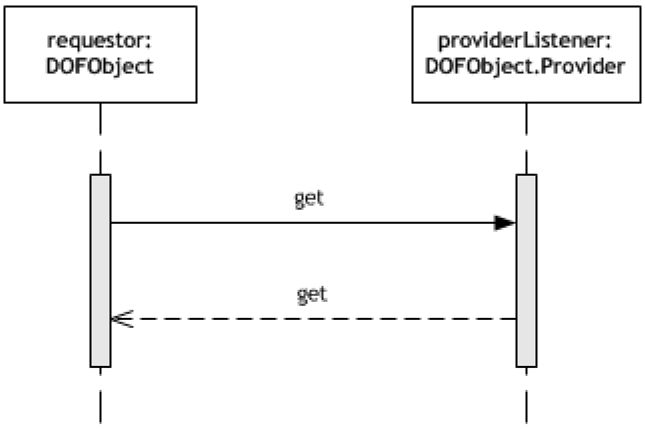
There are five types of operations:

1. *Get*
2. *Subscribe*
3. *Set*
4. *Register*
5. *Invoke*

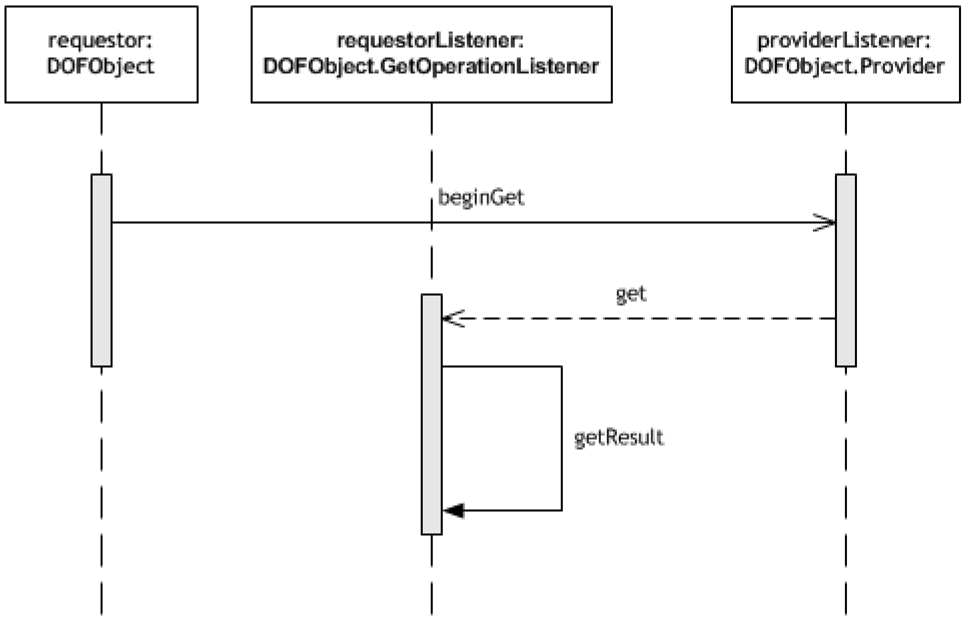
Get Operation

In a *get* operation, a requestor can get the current value of a readable interface property from a provider. *Get* operations can be initiated by the requestor as synchronous operations, using the *DOFObject.get* method, or as asynchronous, using *DOFObject.beginGet*.

The following diagram shows the sequence of method calls in a synchronous *get* operation:



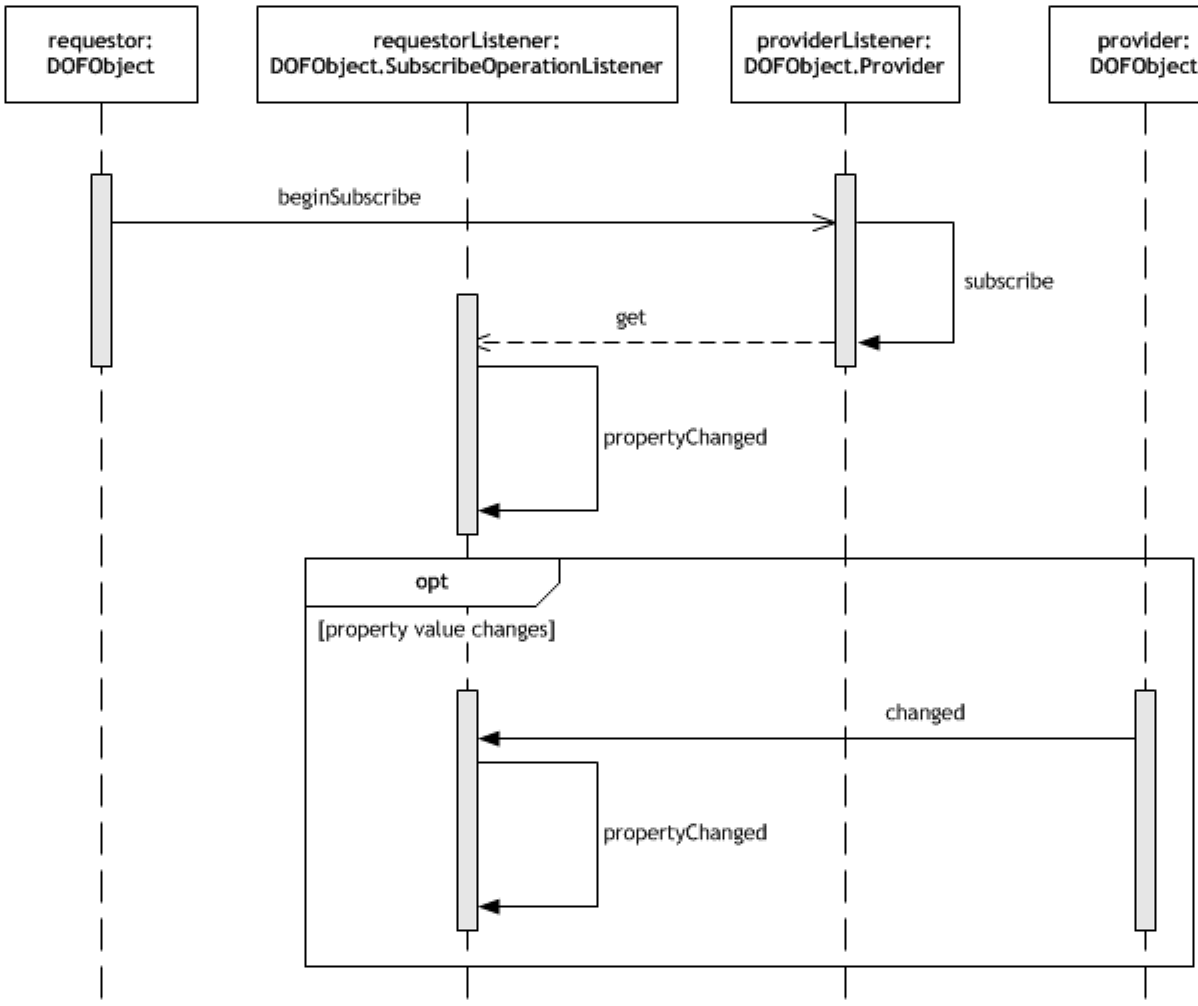
The following diagram shows the sequence of method calls in an asynchronous *get* operation:



Subscribe Operation

In a *subscribe* operation, a requestor subscribes to the value of a provider's interface property, and the provider notifies the requestor each time the value changes. *Subscribe* operations are always asynchronous. They also require interest.

The following diagram shows the sequence of method calls in a subscribe operation:



Set Operation

In a *set* operation, a requestor can set the value of a writeable interface property for a provider. *Set* operations can be initiated by the requestor as synchronous operations, using the *DOFObject.set* method, or as asynchronous, using *DOFObject.beginSet*.

Register Operation

In a *register* operation, a requestor registers for an interface event, and the provider notifies the requestor when circumstances trigger the event. *Register* operations are always asynchronous. They require interest.

Invoke Operation

In an *invoke* operation, a requestor calls an interface method, and the provider executes it and returns results if the method has return values. *Invoke* operations can be initiated by the requestor as synchronous operations, using the *DOFObject.invoke* method, or as asynchronous, using *DOFObject.beginInvoke*.

Synchronous and Asynchronous Operations

The following applies to all operations:

- The requestor initiates the operation.
- The provider always executes the operation and responds.

Synchronous Operations

A synchronous operation is one in which a process responds directly to a request. For example, a requestor could use a *get* operation to obtain the current value of a readable interface property from a provider.

A synchronous operation includes the following characteristics:

- Blocking
- First In/First Out
- Messages are sent and received using the same connection
- Only the first response is received

A synchronous operation requires two methods:

- The requestor uses a DOFObject class method call to initiate the operation.
- The provider uses its corresponding DOFObject.Provider method to respond to the operation.

Asynchronous Operations

An asynchronous operation is one in which a process can respond without a direct request. For example, a register operation can be sent to a provider and the provider will respond only when a specific even occurs, not in direct response to the request.

An asynchronous operation includes the following characteristics:

- Nonblocking
- Multiple results received
- Messages are sent in one form and can be delivered in another

Asynchronous operations adhere to one of the following sequences:

- The provider immediately supplies results that complete the operation.
- Operations are immediately acknowledged by the provider with changes sent to the requestor over time.
- Operations are not complete until the requestor sends a separate call to cancel them.

Single-Transaction Asynchronous Operations

Asynchronous operations that are complete after the initial response require three methods:

1. The requestor uses a *DOFObject* method call to initiate the operation.
2. The provider uses its corresponding *DOFObject.Provider* method to respond to the operation.

3. The requestor uses an operation listener's results handling method to process the results.

Ongoing Asynchronous Operations

Ongoing asynchronous operations have the following characteristics:

- The requestor uses a *DOFObject* method call to initiate the operation.
- The provider uses its corresponding *DOFObject.Provider* method to acknowledge the operation.
- In *subscribe* operations only, the provider uses its *DOFObject.Provider*
- *get* method to send the requestor the initial value of the property.
- *Register* operations do not have a corresponding method.
- The provider uses a *DOFObject* method call to send additional results to the requestor over time.
- The requestor uses an operation listener's results handling method to process all results.
- When finished, the requestor should use the cancel method of the *DOFOperation* interface to unsubscribe or unregister.

Required Operations or Processes

Providers must support all of the following processes to conform to the contract of an DOF interface:

- *Get* and *subscribe* operations if the interface has any "readable" properties
- *Set* operations if the interface has writeable properties
- *Invoke* operations if the interface has methods
- *Register* operations if the interface has events
- Properly thrown exceptions if the interface has exceptions

Chapter 4: Creating a Simple DOF Interface

This chapter describes the process of creating a simple interface and reviews each element.

A Simple Interface Scenario

In this chapter we will examine a simple interface called TimeBasedAlarm, which will define the functionality of a non-recurring alarm that is triggered at a specific time.

Required Information

If you were designing the TimeBasedAlarm interface, you would first have to define the following:

1. A description of the interface items
2. The item identifiers for all of the interface items
3. A means of accessing a DOFInterface instance that represents this interface

Design Considerations

Assume that you wanted to allow a user to set an alarm on a clock and receive an event when the alarm occurred. To do this, the interface would be required to

- Track the current time
- Track the current alarm time
- Allow an alarm to be set
- Allow the alarm to be activated or deactivated
- Send an event when the time was exceeded

Let's say at this point you realize that not all devices that would set or receive an alarm needs a clock. Therefore, you could design a reusable interface that could be used by many different devices by removing the clock.

Consider the following use cases:

- An alarm clock
- A distributed device where the alarm is set by one device but received by another device
- A limited device that includes an alarm but lacks a clock
- A managing program that sets the alarm for many other devices

Interface Items

The TimeBasedAlarm interface has 5 items of “functionality”: 2 properties, 1 method, 1 event, and 1 exception.

Properties

- **AlarmActive.** This is a writeable property that stores a boolean for whether the alarm is set or not. Providers must store the boolean value. The provider must enable requestors to get or subscribe to the value. They must also enable requestors to set the value. A true value means the alarm is on.
- **AlarmTimeValue.** This is a read-only property that stores a date-time value. Providers must store the value and enable requestors to get the value. Providers do not need to implement set for this value since the property is read-only.

Method

SetNewTime. This is a method that allows requestors to set the alarm time. Requestors will send a new alarm time as an input to the method. Providers must change the AlarmTimeValue to the input value in response to this method. In addition, the method must return the boolean for the active value.

Event

AlarmTriggered. Providers must program their applications to trigger this event when the current system time is equal to the AlarmTimeValue and the AlarmActive property is set to true. Providers must also enable requestors to register to the event. The event has no inputs or outputs.

Exception

BadTimeValue. Providers must program their applications to throw this exception when a requestor attempts to set an invalid time value in the SetNewTime method.

Interface Sample Code

```
public class TBAInterface {
public static final DOFType IS_ACTIVE = DOFBoolean.TYPE;
public static final DOFType ALARM_TIME = DOFDateTime.TYPE;

public static final DOFInterface DEF;
public static final DOFInterfaceID IID = DOFInterfaceID.create("[1:{01000034}]");

public static final int PROPERTY_ALARM_ACTIVE_ID = 1;
public static final int PROPERTY_ALARM_TIME_VALUE_ID = 2;
public static final int METHOD_SET_NEW_TIME_ID = 3;
public static final int EVENT_ALARM_TRIGGERED_ID = 4;
public static final int EXCEPTION_BAD_TIME_VALUE_ID = 5;
public static final DOFInterface.Property PROPERTY_ALARM_ACTIVE;
public static final DOFInterface.Property PROPERTY_ALARM_TIME_VALUE;
public static final DOFInterface.Method METHOD_SET_NEW_TIME;
public static final DOFInterface.Event EVENT_ALARM_TRIGGERED;
public static final DOFInterface.Exception EXCEPTION_BAD_TIME_VALUE;

static {
   DEF = new DOFInterface.Builder(IID)
      .addProperty(PROPERTY_ALARM_ACTIVE_ID, true, true, IS_ACTIVE)
      .addProperty(PROPERTY_ALARM_TIME_VALUE_ID, false, true, ALARM_TIME)
```

```
            .addMethod(METHOD_SET_NEW_TIME_ID, new DOFType[] { ALARM_TIME }, new
                     DOFType[] { IS_ACTIVE })
            .addEvent(EVENT_ALARM_TRIGGERED_ID, new DOFType[] {})
            .addException(EXCEPTION_BAD_TIME_VALUE_ID, new DOFType[] {}).build();

         PROPERTY_ALARM_ACTIVE = DEF.getProperty(PROPERTY_ALARM_ACTIVE_ID);
         PROPERTY_ALARM_TIME_VALUE = DEF.getProperty(PROPERTY_ALARM_TIME_VALUE_ID);
         METHOD_SET_NEW_TIME = DEF.getMethod(METHOD_SET_NEW_TIME_ID);
         EVENT_ALARM_TRIGGERED = DEF.getEvent(EVENT_ALARM_TRIGGERED_ID);
         EXCEPTION_BAD_TIME_VALUE = DEF.getException(EXCEPTION_BAD_TIME_VALUE_ID);
         }
      }
```

Chapter 5: Principles of Effective DOF Interface Design

To create an interface, you need to know the fundamentals of object-oriented programming and have an understanding of the DOF Object Model and security considerations.

This chapter presents the following best practices for effective interface design:

- General design principles
- Modularity
- Reuse
- Security considerations

General Design Principles

Since interface design can be difficult, we suggest you either

- Use an existing DOF interface definition
- Redefine existing non-DOF definitions in DOF

If neither of these options are possible or practical, the following design principles should be used:

- Use an interface that is appropriate to the design.
- Reuse interfaces when possible.
- Never partially implement interfaces.
- Think globally.
- Remember that DOF interface definitions include the semantics of the interface.
- Do not reuse interfaces based only on data type.
- Remember that requestors assume behavior, not just data types.

Modularity

DOF encourages modular design because interaction between requestors and providers requires that the objects and interfaces be determined. Engineers should handle the design of the interfaces by factoring functionality into pieces that are related. Each of these steps follows exactly from principles of modular design:

1. Identify the actors and data in the system.
2. Factor the data into related pieces and determine the appropriate access and control required.
3. Assign the functionality and data to the DOF objects that will maintain it. (In the last phase of the design, engineers assign the functionality to providers and objects.)

One of the major benefits of DOF is the flexibility demonstrated in this assignment. An engineer is not required to assign an entire object to a single provider, but may distribute the object (by using different bindings) throughout the system. This allows the provider that has the best access to the required data to provide the binding.

Reuse

A high degree of design sharing is necessary to make the most of DOF functionality. First, it assumes a standard method of factoring the interface functionality. Second, it assumes evaluation of new designs against existing designs in order to identify common interface definitions.

Security

It is important to consider the security the system will require, because security can impact interface design.

DOF utilizes industry-standard, verified security solutions. A product can automatically leverage these solutions by using DOF. Anyone informed of current events is aware of the importance of objectives securing information on networks. Association with a security breach or with the leak of personal information is the last thing that any product wants. This becomes even more important when remote device management is involved because a hacker could gain remote control of a device.

There are already many industry-accepted security standards. However, the requirements of device networking mean that not all of these standards are applicable, particularly when the need is for devices that do not have the resources of a desktop computer or server.

Any DOF security solution must address four critical issues:

- Encryption
- Data Integrity
- Authentication
- Access Control

Using DOF can automatically leverage solutions for these issues in a product. For example, DOF provides centralized and bi-directional access control, including the tools and systems for managing access privileges without requiring the device to manage security by itself. Secure solutions often require changes to existing testing and manufacturing procedures. DOF is no different, but has the benefit that these requirements are well documented and can be understood and planned early in the development process. DOF also includes many different configuration and deployment options that can simplify these requirements.

Security is the term used to cover all means of restricting access to information. Physical security occurs when tangible things like locked doors, pressurized optical fiber, or guard dogs control access. Of more concern to networking are a set of related items, encryption, authentication, data integrity, and access control.

Encryption

Encryption, which includes any process that obscures information by changing it based on some shared key, is possible between authenticated nodes. Encryption protects information (including connections and communication) on the network and ensures that the information is unreadable by intruders.

Data Integrity

Data integrity means protection against modification of information transmitted on the network. This protection can be used in addition to encryption, but encryption is not required. Data integrity prevents intruders from modifying or inserting information.

Two widely used security standards are Secure Sockets Layer (SSL) and the Advanced Encryption Standard (AES).

Authentication

Authentication is used to identify processes, nodes, or people. Systems provide authentication in a number of ways, such as presenting a smart card or scanning a fingerprint. Authentication tells you who is accessing the network.

Access Control

Access control is the process of determining what resources (typically files) a person or proccess can access once authentication has taken place. This allows you to limit an authenticated user to certain parts of the system.

In an DOF system, access control refers to controlling how data is exchanged. It applies not only to requestors (who have their actual access requests checked) but also to providers who must verify that they have permission to provide a binding.

Access control is limited in its granularity to that of a binding, although the system provides control over different actions on the binding. This has an impact on the factoring of interfaces, because items in the same interface have the same access control.

Device Identification and Secrets

Security builds on authentication, which, in turn, builds on identification. To be valid, this identification must be as specific as possible. This usually means unique identification of each secure provider and requestor.

In addition to unique identification, authentication requires a shared secret or token. Some process (manufacture or installation) must assign and share these secrets with the system components that need them.

Physical Security of the Device

Authentication is only as good as the identifiers and secrets that it is based on. This means that secrets that are not protected sufficiently will lose much of their benefit.

One of the difficult aspects of device security is the protection of the device itself. In many cases, the party that has an interest in its security does not physically control the device, for example, a sensor inside a customer's home. In addition, measures should be taken to secure a device if there's a chance a customer might tamper with it.

Because attacks on security are becoming increasingly sophisticated, it might appear that there is no completely secure system. However, a good device design will include an analysis of the security requirements and how to protect sensitive information.

Network Security

A device that is on a shared network is always open to attack from other devices on the network. These attacks can be simple or complicated; they can be local, remote, or even distributed. Therefore, DOF designs should incorporate appropriate protection against network-based attacks. The amount of protection will depend on the importance of the device and the information that it provides. For example, a security sensor for a home alarm system may not require encryption, but it may require a high-level of availability. In this case, if an attacker could take the sensor off-line, it could be a serious breach of security for the whole house.

In general, the design of the system should always address not just the security of the information, but also the security of the system itself, which includes the connections and networks that it uses.

Partitioning

DOF security can be used to partition communication, even when the communication takes place on the same connection. This partitioning extends throughout DOF gateways, and it is a powerful way to share resources while preserving security.

Consider, for example, a single enterprise server that hosts services from a variety of different providers in different partitions. By configuring security and access control, managers can assign devices throughout the system to these partitions and grant access to the services in those partitions.

Chapter 6: The DOF Interface Review and Approval Process

This chapter explains the DOF interface review and approval process.

Note: The DOF Interface Repository discussed in this chapter is not yet available.

Design Resources

The OpenDOF Technical Committee was created to provide assistance, direction, and oversight in working with DOF interfaces.

OpenDOF Technical Committee

The OpenDOF Technical Committee oversees the technical and architectural aspects of DOF. Specifically, the committee is responsible for the following:

- Maintaining the registries of all interface identifiers.
  Since the identifiers must be unique, the OpenDOF Technical Committee maintains the registries to ensure there are no duplications.
- Providing a web-based repository of all DOF interfaces.
  This repository includes design references and other interface information. Designers can search this repository for related interfaces. The committee must add the new DOF interfaces to the repository before they can be used in final products.
- Providing design guidance and training.
  System designers may contact the OpenDOF Technical Committee with questions about their design. Product designers can have their designs reviewed by the committee to insure that they conform to DOF guidelines. The committee can also provide training on object-oriented design principles, including interface design.
- Managing the review process of proposed DOF interface definitions.
- Recommending reviewed interface definitions.
- Monitoring other network-based API standards (interfaces) for inclusion in DOF (for example, interface definitions coming from other technologies like ZigBee).

The DOF Interface Review Process

The process of designing, creating, and publishing a DOF interface is as follows:

1. The design engineers who will be creating the interface receive training in DOF principles and architecture.
2. Once trained, the design engineers are given access to the DOF Interface Repository.
3. The design engineers create a high-level logical design of the application. (The data has been identified and assigned to objects.)

4. The design engineers create the interface and associated metafiles and place them into the DOF Interface Repository. (The associated metafiles can be written in languages other than English; however, an English translation needs to be included.)
5. The design engineers send a request to the OpenDOF Technical Committee to review the interface. This is done by sending an email to dof-tc@opendof.org. Designers should attach all related documents pertaining to the system design and requirements.
6. The primary reviewer checks whether the metafiles include sufficient descriptions. If the descriptions are insufficient, the primary reviewer can reject the design and return it to the design engineers to provide more detail.
7. Once the metafiles are approved, the reviewers (including the primary reviewer) review the interface definitions, consider the recommended sizes of the IIDs (if needed), and create a report.
8. The primary reviewer gives the review results and recommended sizes of the IIDs to the OpenDOF Technical Committee.
9. The committee checks the review results and provides feedback to the designers.
10. A request is made for an interface identifier.
11. The interface is reviewed again.
12. The OpenDOF Technical Committee assigns the final interface identifier.
13. A request is made to publish the interface.
14. The committee publishes the interface and makes it generally available to DOF developers.

Chapter 7: Using the DOF Interface Repository

The DOF Interface Repository is an online resource that provides access to the interfaces. Developers working with DOF can use the repository to

- View published interfaces.
- Search for published interfaces.
- Share information with other developers about interfaces that are being designed.
- Design new interfaces.

The DOF Interface Repository also contains information for developers and answers to frequently asked questions.

Note: The DOF Interface Repository discussed in this chapter is not yet available.

Chapter 8: Terms Used in the Training

The following terms are used in this document:

access control: The process of determining and defining the extent of access once authentication has taken place.

actor: Something that interacts with an application or system; in general, an actor could be a person, machine, or another system or subsystem. In DOF, actors are requestors, providers, and objects.

asynchronous operations: An asynchronous operation is one in which a process can respond without a direct request. For example, a *register* operation can be sent to a provider and the provider will respond only when a specific even occurs, not in direct response to the request. An asynchronous operation has non-blocking characteristics and can provide multiple results. Asynchronous messages sent in one form and can be delivered in another. Any method call that starts with the word “begin” is asynchronous.

authentication: The means of confirming the identification of who or what is accessing the system, including processes, nodes, and people. Confirming a password is an example of authentication.

binding: The combination of an OID and an IID that identifies a provider on the network and the functionality it provides. Requesters use bindings to interact with providers. **Note:** A provider can be the source of one or more bindings.

contract: A contract defines the relationship between a requestor and a provider; in DOF, this is the interface. A contract also specifies the meaning of the data.

data integrity: The protection against modifying information transmitted on the network. Two widely used standards are Secure Sockets Layer (SSL) and the Advanced Encryption Standard (AES).

DOF Object Access Libraries (OALs): The OALs provide a general API for developers to write software that deals with the DOF protocols and objects on the network. The DOF application programming interface (API) currently includes versions of the OAL in Java, C, and C#.

DOF object model: Describes the relationship and interactions between DOF objects, DOF interfaces, items, and types.

DOF (Distributed Object Framework): enables remote capabilities through networking systems. DOF allows many different products, using many different standards, to work together and share information across many different networks.

encryption: Any process that obscures information to those without access by changing it based on a shared key. Encryption ensures that information is unreadable by intruders.

event: Circumstances that trigger a response. Requestors can use *register* operations to request notification if a particular event occurs on the provider. Events may have output parameters. An event is an interface item.

exception (interface): An error condition that interrupts the normal flow of an operation, specifically a logical exception case involving the properties, methods, or events defined by an interface. Exceptions defined in an interface are called provider exceptions in the API and are represented by the *DOFProviderException* class. Exception handling is the process of responding to exceptions. An exception is an interface item.

interface or DOF interface: Defines a set of functionality that a DOF object must provide. An interface consists of properties, methods, events, exceptions, and defined behaviors that govern how the interface (and by extension, the object) reacts to both its environment and any remote management. DOF interfaces must have an assigned interface identifiers (IID) that is globally unique.

interface definition: The approved specification of an interface that has been published and registered. The interface definition is maintained by the OpenDOF Technical Committee.

interface identifier (IID): The reference identification given to an interface. Interface identifiers are globally unique. They are assigned by the OpenDOF Technical Committee.

interface item: An individual piece of functionality within an interface. An interface item can be a property, method, event, or exception. An interface item is referenced by an item identifier (Item ID).

item identifier (Item ID): The reference identification given to an item within an interface. The Item ID may be as large as 2^{16}-1; however, because most interfaces have only a handful of items, item IDs will typically be small. Item IDs must be unique within the interface.

item, see interface item.

method: A function the requestor can call remotely through an invoke operation, which the provider executes in a prescribed fashion (as defined by the DOF interface). A method may have input parameters and return types. A method is an interface item.

node: A physical or logical networking device that communicates with other nodes. In DOF, a node can be a provider, requestor, or proxy.

object attributes: Used to qualify objects. Attributes allow application developers can use attributes to define relationships and grouping. Attributes augment OIDs, which makes OIDs more flexible.

object history: A set of object snapshots stored over time.

object identifier (OID): The reference identification given to objects. OIDs are globally unique. **Note:** although each DOF object requires an OID, OIDs can be used to identify other programming objects as well. OIDs used in the DOF Object Model are stable. Any unique identifier can be converted to an OID.

object snapshot: The state of an object that is stored or passed through a protocol.

object state: The current set of values (readable properties) on a particular interface.

object transaction log: A history of requested changes for a particular object.

object or DOF object: A programming element with properties and behaviors; an object can represent physical or logical devices; they are dynamic and defined. An object can be anything that provides the functionality defined in an interface. All objects are providers and must have an assigned object identifier (OID). In this document, all "objects" are DOF objects.

object-oriented programming (OOP): Any programming language that uses objects as the basis of functionality instead of executing a sequence of commands.

operation or DOF operation: The communication between providers and requestors that enables applications to use an item of functionality in an DOF interface. Operations are controlled through a specific set of API calls (methods or functions) provided by the OAL. The most common operations are as follows:

- *Get*: Allows a requestor to get the current value of a readable interface property from a provider. *Get* operations can be initiated by the requestor as synchronous or asynchronous operation.
- *Subscribe*: Allows a requestor to be notified every time there's a change to the value of a provider's interface property. *Subscribe* operations are always asynchronous and require interest.
- *Set*: Allows a requestor to set the value of a writeable interface property for a provider. Set operations can be initiated by the requestor as a synchronous or asynchronous operation.
- *Register*: Allows a requestor to be notified when an event is triggered for the provider. Register operations are always asynchronous and require interest.
- *Invoke*: Allows a requestor to call an interface method that the provider executes and then returns results (if the method has return values).

property: The definition of a specific type of data, including whether the data is readable, writable, or both. Properties are similar to fields in a Java class. They allow reading, writing, and subscribing to data related to the current object state. A property is an interface item.

provider: 1. A node on an DOF network that provides functionality defined in one or more DOF interfaces. 2. A device that sends (provides) "state" data in response to a requestor.

proxy: A node that forwards operations from a provider to a requestor. A proxy routes operations but doesn't process them.

interface, published or registered: Once an interface is published, it is fixed, which means the functionality doesn't change. Once the interface is published, it remains static. The functionality is then "registered."

requestor: A node on a DOF network that requests functionality from a provider.

reuse in interface design: A design strategy that emphasizes the use of existing programming elements or repurposing existing functionality instead of creating something completely new that may be redundant to some extent.

synchronous operations: An operation with the following characteristics: blocking call, or first in/first out. Synchronous messages are sent and received using the same connection; only the first response is received.

OpenDOF Technical Committee: The committee responsible for the technical and architectural oversight of DOF.

type identifier: The reference identification given to types. Each type requires a unique identifier that must be registered.

type: Defines the data used in an interface item.