HP OpenView Service Desk 4.0

API Programmer's Guide

First Edition

Manufacturing Part Number: N/A

August 2001

Legal Notices

Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Restricted Rights Legend. Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19 (c)(1,2).

The nomenclature of each version of this software (and manuals therefore) has been devised for commercially convenient reasons, and is not intended to denote the degree of originality of any version of the software with respect to any other version. The extent of protection afforded by, and duration of copyright is to be determined entirely independently of this nomenclature.

Trademark Notices

Java™ is a U.S. trademark of Sun Microsystems, Inc.

Microsoft® is a U.S. registered trademark of Microsoft Corporation.

Contents

1. The API

2. API Principles

3. Examples

Preface

The Service Desk **API** is an application programming interface designed to give developers programmatic access to the Service Desk application. By using the API, you can create customized integrations with Service Desk.

NOTE

Service Desk 4.0 comes with a new API: the Web API. This will replace the Service Desk API documented in this guide. The Service Desk API is obsolete: it will not be maintained anymore and it will be discontinued at the same time as Service Desk 3.0. With Service Desk 4.0, the obsolete API and documentation are supplied as a courtesy to assist you in migrating to the new Web API. We strongly recommend that you migrate to the Web API as soon as possible and we discourage any new developments with the deprecated API. The Web API consists of pure-Java interfaces to the Service Desk entities. The Web API has a highly intuitive structure which makes it easy to learn and to use. For more information about the Web API, please refer to the following documents:

- *HP OpenView Service Desk: Release Notes*
- *HP OpenView Service Desk: Web API Programmer's Guide*
- *HP OpenView Service Desk Web API: Javadoc*

This guide contains information about the structure and potential use of the Service Desk API. It is a technical guide designed for use by experienced Java application programmers.

NOTE

You must have knowledge and experience programming with MS Visual J++ to develop programs using the Service Desk API and this guide. The Service Desk API is formed by a combination of Java classes written in MS Visual J++ 6.0.

This guide is organized as follows:

- Chapter 1, “The API,” on page 15 provides an overview of the API, the

Service Desk architecture, and a list of requirements for using the API.

- Chapter 2, “API Principles,” on page 21 briefly describes basic functions available with the API.
- Chapter 3, “Examples,” on page 29 provides explanations and examples of the most common API functions.
- The “Glossary” on page 69 provides a list of terms which may be unfamiliar to you, with definitions.

Revision History

When an edition of a manual is issued with a software release, it has been reviewed and tested and is therefore considered correct at the date of publication. However, errors in the software or documentation that were unknown at the time of release, or important new developments, may necessitate the release of a service pack that includes revised documentation. Revised documentation may also be published on the Internet, see “We Welcome Your Comments!” in this preface for the URL.

A revised edition will display change bars in the left-hand margin to indicate revised text. These change bars will only mark the text that has been edited or inserted since the previous edition or revised edition.

When a revised edition of this document is published, the latest revised edition nullifies all previous editions.

Table 1

Edition and Revision Number	Issue Date	Product Release
First Edition	August 2001	Service Desk 4.0

Related Publications

This section helps you find information that is related to the information in this guide. It gives an overview of the Service Desk documentation and lists other publications you may need to refer to when using this guide.

The Service Desk Documentation

Service Desk provides a selection of books and online help to assist you in using Service Desk and improve your understanding of the underlying concepts. This section illustrates what information is available and where you can find it.

NOTE This section lists the publications provided with Service Desk 4.0. Updates of publications and additional publications may be provided in later service packs. For an overview of the documentation provided in service packs, please refer to the readme file of the latest service pack. The service packs and the latest versions of publications are available on the Internet. See the section “We Welcome Your Comments!” in this preface for the URLs.

- The `Readme.htm` file on the Service Desk CD-ROM contains information that will help you get started with Service Desk. It also contains any last-minute information that became available after the other documentation went to manufacturing.
- The *HP OpenView Service Desk: Release Notes* give a description of the features that Service Desk provides. In addition, they give information that helps you:

  — compare the current software’s features with those available in previous versions of the software;

  — solve known problems.

  The Release Notes are available as a PDF file on the HP OpenView Service Desk 4.0 CD-ROM. The file name is `Release_Notes.pdf`.
- The *HP OpenView Service Desk: User’s Guide* introduces you to the key concepts behind Service Desk. It gives an overview of what you can do with Service Desk and explains typical tasks of different types of Service Desk users. Scenario descriptions are provided as examples of how the described features could be implemented.

The User’s Guide is available as a PDF file on the HP OpenView Service Desk 4.0 CD-ROM. The file name is `User’s_Guide.pdf`.

- The *HP OpenView Service Desk: Supported Platforms List* contains information that helps you determine software requirements. It lists the software versions supported by Hewlett-Packard for Service Desk 4.0.

  The Supported Platforms List is available as a PDF file on the HP OpenView Service Desk 4.0 CD-ROM. The file name is `Supported_Platforms_List.pdf`.

- The *HP OpenView Service Desk: Installation Guide* covers all aspects of installing Service Desk.

  The Installation Guide is available as a PDF file on the HP OpenView Service Desk 4.0 CD-ROM. The file name is `Installation_Guide.pdf`.

- The *HP OpenView Service Desk: Administrator’s Guide* provides information that helps application administrators to set up and maintain the Service Desk application server for client usability.

  The Administrator’s Guide is available as a PDF file on the HP OpenView Service Desk 4.0 CD-ROM. The file name is `Administrator’s_Guide.pdf`.

- The *HP OpenView Service Desk: Data Exchange Administrator’s Guide* explains the underlying concepts of the data exchange process and gives instructions on exporting data from external applications and importing it into Service Desk. The data exchange process includes importing single service events and batches of data.

  The Data Exchange Administrator’s Guide is available as a PDF file on the HP OpenView Service Desk 4.0 CD-ROM. The file name is `Data_Exchange.pdf`.

- The *HP OpenView Service Desk: VantagePoint Operation Integration Administrator’s Guide* explains the integration between Service Desk and VantagePoint for Windows and UNIX®. This guide covers the installation and configuration of the integration and explains how to perform the various tasks available with the integration.

  The VantagePoint Operation Integration Administrator’s Guide is available as a PDF file on the HP OpenView Service Desk 4.0 CD-ROM. The file name is `VPO_Integration_AG.pdf`.

- The *HP OpenView Service Desk: Migration Guide* provides a detailed overview of the migration from ITSM 5.7 to Service Desk 4.0, to include an analysis of the differences in the two applications. Detailed instructions in this guide lead through the installation, configuration and other tasks required for a successful migration.

The Migration Guide is available as a PDF file on the HP OpenView Service Desk 4.0 CD-ROM. The file name is `Migration_Guide.pdf.`

- The *HP OpenView Service Desk: API Programmer's Guide* contains information that will help you create customized integrations with Service Desk. This guide depicts the API structure, and explains some of the basic functions with examples for using the Application Programming Interface (API) provided with Service Desk. The API extends the HP OpenView Service Desk environment by providing independent programmatic access to data-centered functionality in the Service Desk application server environment.

  The API Guide is available as a PDF file on the HP OpenView Service Desk 4.0 CD-ROM. The file name is `API_pg.pdf`.

- The *HP OpenView Service Desk: Web API Programmer's Guide* contains information that will help you create customized integrations with Service Desk using the Service Desk Web API. This API is particularly suited for developing Web applications.

  The Web API Programmer's Guide is available as a PDF file on the HP OpenView Service Desk 4.0 CD-ROM. The file name is `Web_API_pg.pdf`.

- The *HP OpenView Service Desk: Data Dictionary* contains helpful information about the structure of the application.

  The Data Dictionary is available as an HTML file on the HP OpenView Service Desk 4.0 CD-ROM. The file name is `Data_Dictionary.htm`.

- The *HP OpenView Service Desk 4.0 Computer Based Training* (CBT) CD-ROM is intended to assist you in learning about the functionality of HP OpenView Service Desk 4.0 from both a user and a system administrator perspective. The CD-ROM contains demonstration videos and accompanying texts that explain and show how to perform a wide variety of tasks within the application. The CBT also explains the basic concepts of the Service Desk application.

  The *HP OpenView Service Desk 4.0 Computer Based Training* (CBT) CD-ROM will be shipped automatically with the regular Service Desk software. The CBT will be available for shipment shortly after the release of the Service Desk software.

- The online help is an extensive information system providing:
  - — procedural information to help you perform tasks, whether you are a novice or an experienced user;
  - — background and overview information to help you improve your understanding of the underlying concepts and structure of Service Desk;
  - — information about error messages that may appear when working with Service Desk, together with information on solving these errors;
  - — help on help to learn more about the online help.

  The online help is automatically installed as part of the Service Desk application and can be invoked from within Service Desk. See the following section entitled “Using the Online Help” for more information.

Reading PDF Files

You can view and print the PDF files with Adobe® Acrobat® Reader. This software is included on the HP OpenView Service Desk 4.0 CD-ROM. For installation instructions, see the `readme.htm` file on the CD-ROM.

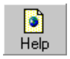
The latest version of Adobe Acrobat Reader is also freely available from Adobe’s Internet site at http://www.adobe.com.

Using the Online Help

You can invoke help from within Service Desk in the following ways:

- To get help for the window or dialog box you are working in, do one of the following:
  - — Press **F1**.
  - — Click the help toolbar button ?.
  - — Choose `Help` from the `Help` menu.
  - — Click the help command button Help in a dialog box.
- To search for help on a specific subject using the table of contents or the index of the help system: choose `Help Contents & Index` from the `Help` menu.

When you are in the help viewer, you can find help on how to use the help system itself by clicking the `Help` toolbar button:



Service Desk also provides *tooltips* and *"What's This?" help* for screen items like buttons, boxes, and menus.

A *tooltip* is a short description of a screen item. To view a tooltip, rest the mouse pointer on the screen item. The tooltip will appear at the position of the mouse pointer.

"What's This?" help is a brief explanation of how to use a screen item. "What's This?" help generally gives more information than tooltips. To view "What's This?" help:

1. First activate the "What's This?" mouse pointer in one of the following ways:

   - Press **Shift+F1**.
   - Click the "What's This?" toolbar button ↖?.
   - Choose `What's This?` from the `Help` menu.
   - In dialog boxes, click the question mark button ? in the title bar.

   The mouse pointer changes to a "What's This?" mouse pointer ↖?.

2. Then click the screen item for which you want information. The "What's This?" help information appears in a pop-up window.

To close the pop-up window, click anywhere on the screen or press any key on your keyboard.

Other Related Publications

In addition to the Service Desk documentation mentioned above, you may want to refer to the following publications when using this guide:

- http://java.sun.com/products/jdk/javadoc/index.html. This link connects to the Javadoc tool home page on the Internet.
- http://msdn.microsoft.com/visualj/default.asp. This is the home Web site for Microsoft®Visual J++.
- http://www.microsoft.com/java/sdk/. This is the site for Microsoft SDK for Java™.

Typographic Conventions

The table below illustrates the typographic conventions used in this guide.

Font	What the Font Represents	Example
Italic	References to book titles Emphasized text	See also the *HP OpenView Service Desk: Installation Guide.* *Do not delete* the System user.
Bold	First-time use of a term that is explained in the glossary	The **service call** is the basis for incident registration.
`Courier`	Menu names Menu commands Button names File names Computer-generated output, such as command lines and program listings	You can adjust the data view with the commands in the `View` menu. Choose `Save` from the menu. Click `Add` to open the Add Service Call dialog box. To start the installation, double-click `setup.htm`. If the system displays the text `C:\>dir a:` `The device is not ready` then check if the disk is placed in the disk drive.
`Courier bold`	User input: text that you must enter in a box or after a command line	If the service call must be solved within 30 minutes, enter **`30`**.
`Courier italic`	Replaceable text: text that you must replace by the text that is appropriate for your situation	Go to the folder *`X`*:\\Setup, where *`X`* is your CD-ROM drive.
Helvetica bold	Keyboard keys A plus sign (**+**) means you must press the first key (**Ctrl** in the example), hold it, and then press the second key (**F1** in the example).	Press **Ctrl+F1**.

We Welcome Your Comments!

Your comments and suggestions help us understand your needs, and better meet them. We are interested in what you think of this manual and invite you to alert us to problems or suggest improvements. You can submit your comments through the Internet, using the HP OpenView Documentation Comments Web site at the following URL:

http://ovweb.external.hp.com/lpe/comm_serv

If you encounter errors that impair your ability to use the product, please contact the HP Response Center or your support representative.

The latest versions of OpenView product manuals, including Service Desk manuals, are available on the HP OpenView Manuals Web site at the following URL:

http://ovweb.external.hp.com/lpe/doc_serv

Software patches and documentation updates that occur after a product release, will be available on the HP OpenView Software Patches Web site at the following URL:

http://support.openview.hp.com/cpe/patches

1 The API

This chapter provides an overview of how the API fits into the Service Desk architecture. It also lists the requirements which must be met to use the Service Desk API effectively.

API Overview

The Service Desk API is a set of Java classes, written in Visual J++. These classes are part of the standard Service Desk software bundle. The API provides a means of integrating with the Service Desk application server independently from the user interface, while ensuring that all authorization and **business rules** are enforced. You can use this API to integrate other management tools with Service Desk. For example, an inventory tool could be integrated so that it automatically updates the Service Desk database when new items or changes in existing items are found by the external inventory tool. The Data Exchange feature included with the Service Desk application was created using this API, for example.

For information on the basic functions performed with the API, see Chapter 2, “API Principles,” on page 21, or Chapter 3, “Examples,” on page 29.

The API Javadoc

A Javadoc document for this API is provided on the Service Desk CD. It consists of a set of hyperlinked HTML files, generated from the API source files, that describe the classes, interfaces, constructors, methods and fields of the API. It also includes the examples described in Chapter 3.

You can find the Javadoc files on the HP OpenView Service Desk 4.0 CD, in the `Doc\Api Javadoc` folder. To open the Javadoc, open the `index.html` file. This file contains links to the other files in the `Doc\API Javadoc` folder.

Deprecated API elements

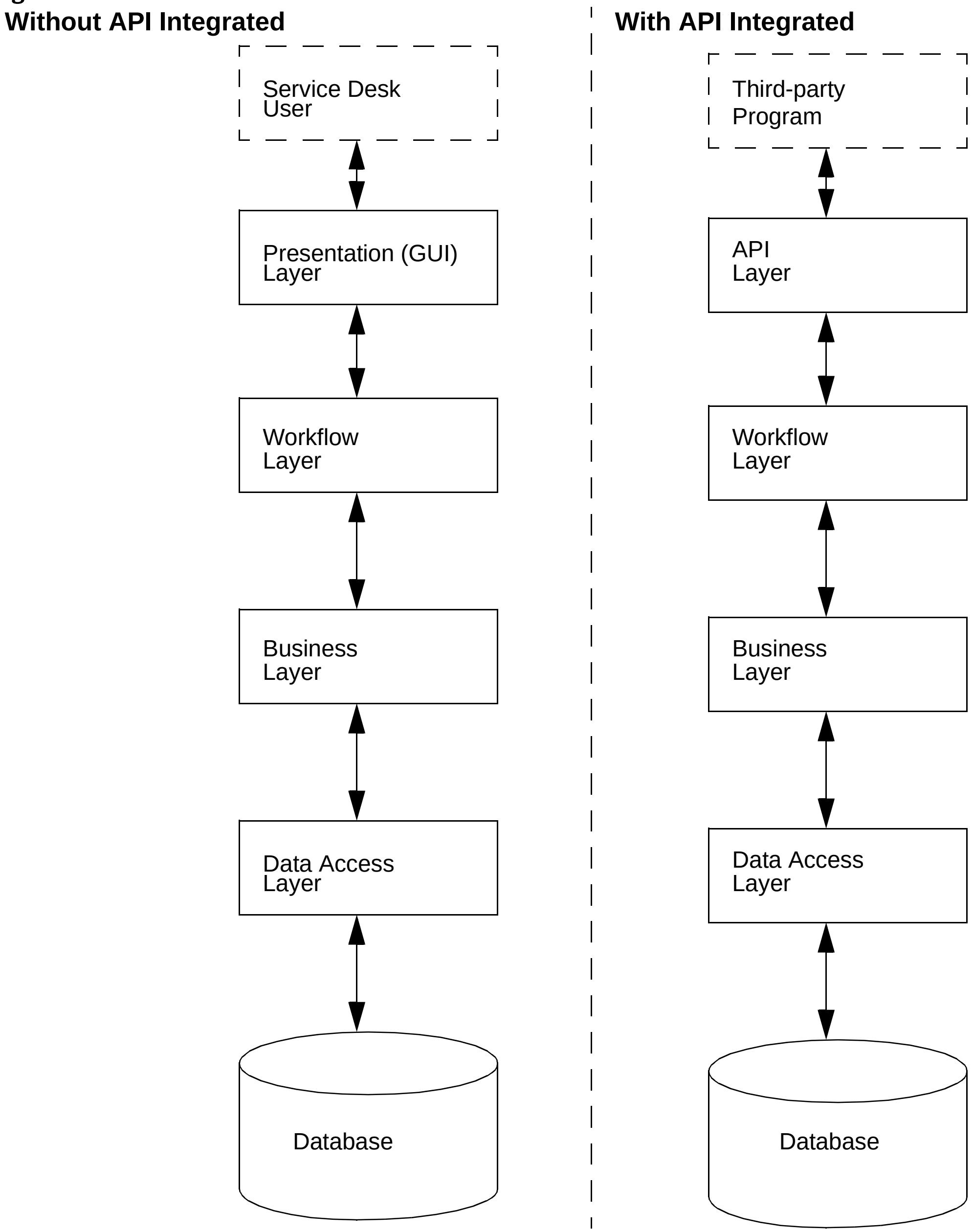
The Javadoc refers to this guide for information about deprecated API elements. The Service Desk API does not provide alternatives for the deprecated elements. Instead of using the Service Desk API, we strongly recommend that you use the Service Desk Web API. The Service Desk API will soon be obsoleted. Please refer to the “Preface” in this guide for more information about this.

Runtime Architecture

The API is tightly integrated with the Service Desk architecture. Figure 1-1 on page 18 shows how the API fits into this architecture. The Service Desk application architecture is made up of four layers, with each layer performing a specific task:

- The **presentation layer** contains the user interface. This layer is in effect replaced by the API.
- The **workflow layer** contains the rules that determine what information is required to complete an operation.
- The **business layer** contains rules that determine how an operation is validated or completed. It communicates with the data access layer.
- The **data access layer** provides access to the data in the database. It communicates between the application's **object model** and the relational representation of what exists in the underlying database.

Figure 1-1 Service Desk Architecture



End users normally communicate with the Service Desk application through the graphical user interface (GUI) referred to as the presentation layer shown on the left in Figure 1-1 on page 18. The API takes the place of the presentation layer, while the role of the user is replaced by a third-party program developed to interact with the API. This is depicted on the right side of Figure 1-1 on page 18. The third-party application provides input to Service Desk while following a number of self-defined rules to ensure the input will be processed appropriately. The API is capable of communicating with the application server over the workflow layer with the same result as if the actions were initiated from the user interface. The difference is that the API reacts to input from another application, causing Service Desk to perform an action.

The API depends heavily on the availability of the Service Desk workflow layer for access to the Service Desk environment. The workflow layer assures that all rules normally applied to actions in the user interface are also applied to the same actions when they are performed by the API. For example, rules exist in the workflow layer to ensure that a service call can never have an end date before the start date. The business and workflow layers work together to make a business object fully functional.

Requirements

The Service Desk API can be run from both the server or client environment. The **classes** that form the API are stored in the `sd_import.zip` file installed with the Service Desk application. The installation procedure adjusts the class path to include the API package by default. Additional requirements for successful use of the API include:

Running API-based programs

Requirements for running API-based programs:

- The `sd_import.zip` file must be referred to in the local class path setting.
- Programs must be run from a machine that is set up as a Service Desk application server. For more information, refer to the *HP OpenView Service Desk Installation Guide* and the *HP OpenView Service Desk: Administrator's Guide*.
- A Service Desk account needs to be created providing access to all applicable areas. It can be a **non-UI account**, see page 71 for more information.

Developing API-based programs

Requirements for developing API-based programs:

- Programming knowledge of MS Visual J++.
- MS Visual J++ 6.0, or an equivalent development environment. For example, you can compile with the Microsoft Software Development Kit (SDK) and then run it with a Microsoft Jview virtual machine. Jview is installed with Service Desk by default. More information about the Microsoft SDK is available at the following Web site: `http://www.microsoft.com/java/sdk/`
- Reference to the Service Desk `sd_import.zip` delivered with the Service Desk application in the IDE’s class path setting.

NOTE

The Service Desk API is formed by a number of Java classes written in Microsoft Visual J++ 6.0. These classes indirectly use some Microsoft-specific extensions of the Java language. Their inter-operability with non-Microsoft virtual machines and compilers is not guaranteed.

2 API Principles

The API provides users with access to data-related functions in the Service Desk environment. It forms a layer in the software environment with the purpose of assuring optimum communication between the entities on either side:

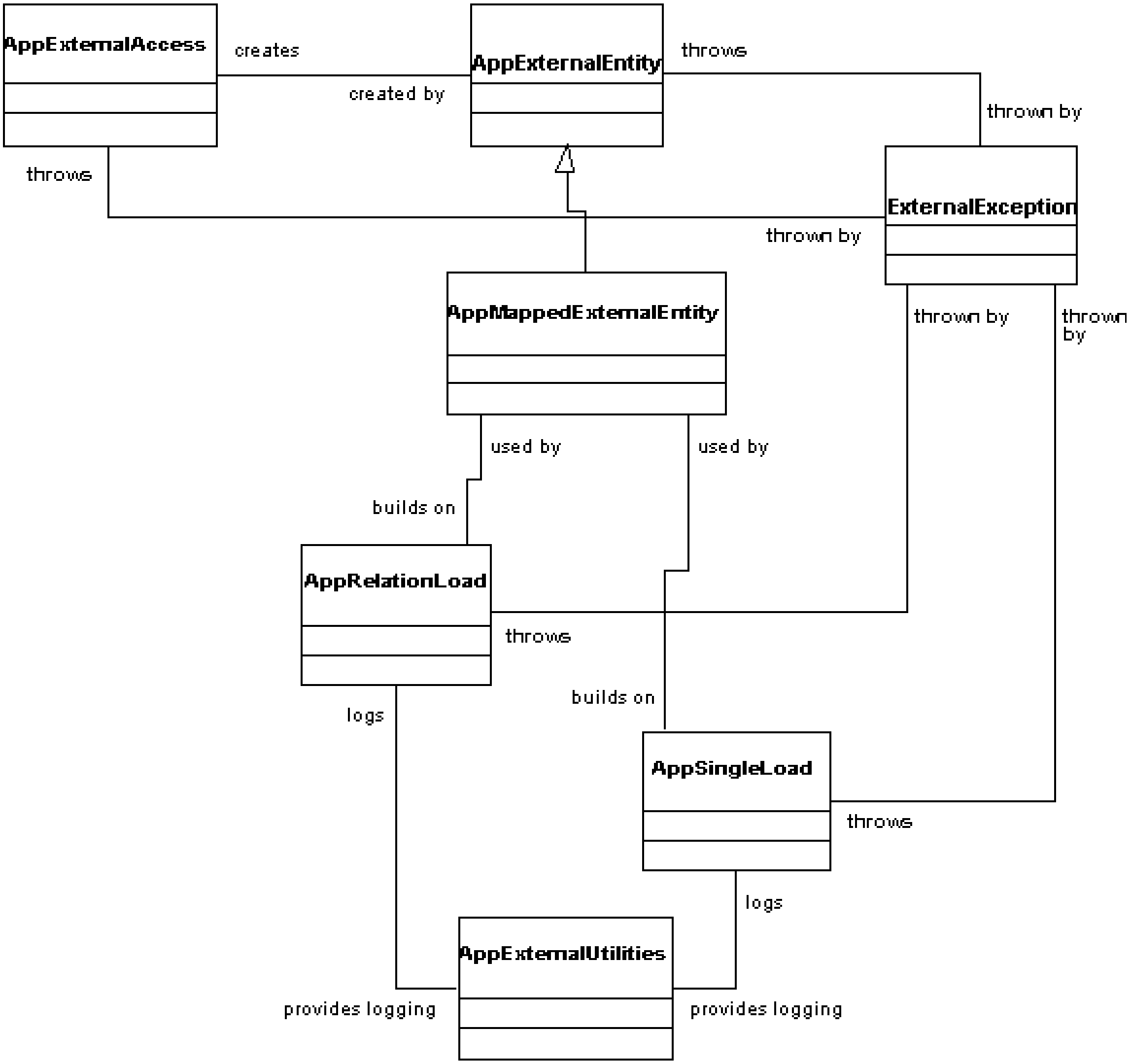
- The third-party program performing actions on Service Desk entities
- The Service Desk application server giving access to Service Desk entities

The Service Desk application is created around a **kernel**, which is defined as a collection of generic reusable sources called ITSM Foundation Classes (IFC). The generic elements that make up the kernel come from all layers of the application. The API is included in the kernel. Because Service Desk-specific functions do not exist in the **IFC**, the IFC can be used for developing other database-dependent applications. The primary classes that form the API are in the `com.hp.ifc.ext` package. You will also need classes from `com.hp.ifc.util.marshal` to communicate with the workflow layer, and specific object classes from `com.hp.ifc.types`.

An additional supporting package within the kernel is `com.hp.ifc.rep.ext`. This package contains the supporting classes that define mapping to external data sources used by the API classes. This package is part of the **repository**, see page 71.

The following object model shows how API classes work together. It is followed by a section about the object model and sections explaining the most commonly used API functions.

Figure 2-1 Main API Functions



The Object Model

The object model describes the objects used by Service Desk. Objects contain attributes and methods. Attributes can be:

- Simple java types (like `int`)
- References to other objects, which can be:
  - Aggregated objects (only belong to this parent object)
  - Associated objects (can have different parent objects)

  Both can be a set of objects.

An object description is sometimes called an entity. Every entity and every object has an ID. The object ID is of type long and sometimes wrapped in class `AppOID`.

The layers of Service Desk use the object model to get information about the objects (mandatory attributes, display information, and so on). So every layer has to know the AppOIDs of the object descriptions.

In the API layer, the class `ITSMExternalEnum` contains the AppOIDs of the object descriptions. Actually, the API layer offers two ways to retrieve information from the object model:

- By Number: class `ITSMExternalEnum`
- By Label: in the mappings of Service Desk, you can define a label for each object description in the object model and you can use this label in the API layer.

In the API layer, you must indicate which entity and which attributes you want to use. The API layer can track attributes that are referenced objects. However, it cannot do this for referenced sets of objects. You must handle each object of the set separately.

For example, the entity ID of the text attribute of an assignment status of a work order:

```
long appOID =
   new long[] {ITSMExternalEnum.WorkorderDefEnum.atAssignment
   ,ITSMExternalEnum.AssignmentDefEnum.atAssignStatus
   ,ITSMExternalEnum.AssignmentStatusDefEnum.atText
   }
```

Principle API Functions

The following sections briefly explain some of the primary API functions. A more detailed explanation together with examples can be found in Chapter 3, “Examples,” on page 29.

Connecting to the Application Server

To obtain access to functions available from the workflow layer, all programmatic API sessions must first log on. API sessions log on using a specialized class, `AppExternalAccess` that also operates as an object server for Service Desk, creating and retrieving objects from the Service Desk environment as needed. For example, retrieving a service call so that it can be modified.

The class `AppExternalAccess` provides access to data managed by the Service Desk application server. This class can be considered a gateway into the workflow layer for all requests going to the application server. Every program that uses the API will be organized around at least one `AppExternalAccess` object to establish a connection with the application server. After successfully logging on, this same object functions as a Service Desk entity server. It takes care of:

- logging on and off from the Service Desk environment;
- activating import mapping settings and presenting those settings;
- creating new Service Desk objects and retrieving existing objects for further processing;
- presenting warnings and error messages created by the Service Desk environment.

For more information, including a detailed example, see “Connecting to the Application Server” on page 31.

Identifying Entities and Attributes

An important part of the integrating with Service Desk involves the manipulation of Service Desk entities and their attributes. A reliable method for identifying entities and attributes must be available for that purpose. The API uses two means of identifying Service Desk entities and attributes for processing:

- By communicating with the application server, creating or modifying Service Desk entities and attributes as necessary. The enumeration class `ITSMExternalEnumeration` identifies the objects and attributes by means of the numeric values assigned to them in the Service Desk repository. This means of identification is also referred to as reference by number. Reference by number provides a high degree of freedom when manipulating Service Desk entities.
- By communicating with the application server using pre-defined import mapping settings. Import mapping requires setup by the Service Desk application administrator prior to actually programming the integration. These settings are labels defined by the user for entities and attributes. Once defined, the labels can be used within the Service Desk environment. Using labels to access data is referred to as reference by label. In order to access the settings, the active `AppExternalAccess` object must be set to use the proper group of import mapping settings.

For more information, including a detailed example, see “Identifying Entities and Attributes” on page 33.

Finding the Proper Entity Instantiation

Once a relevant Service Desk entity is identified, the proper instantiations of that entity must be located. The AppExternalAccess class contains methods for the construction of search criteria, using either the reference by number or the reference by label methods. These criteria can then be used in an AppExternalAccess method that returns a list of identifiers for the objects that comply with the search conditions.

For more information, including a detailed example, see “Finding the Proper Entity Instantiation” on page 38.

Getting an Entity Instantiation

After obtaining the **entity** instantiation identifier, the actual data can be retrieved from the Service Desk application server. AppExternalAccess offers a number of methods for retrieving the data. Methods differ mainly in the way they reference entities and attributes. When reference by number is used, the method returns an `AppExternalEntity` object. When reference by label is used, an `AppMappedExternalEntity` object is returned. `AppMappedExternalEntity` is an extension of the `AppExternalEntity` class.

The AppExternalAccess also offers methods for the creation of new Service Desk objects. When used with the AppMappedExternalEntity class, the template defined in the import mapping settings and its default values are applied immediately.

For more information, including a detailed example, see “Getting an Entity Instantiation” on page 45.

Getting Attribute Values

After obtaining an AppExternalEntity object wrapping Service Desk data, the current attribute values can be determined. When you use the methods defined in AppExternalEntity class, these values are returned as Java objects. The objects are either standard Java classes, java.lang.String for example, or IFC classes from the com.hp.ifc.types package. When using the methods in AppMappedExternalEntity, a String object is returned for all values.

For more information, including a detailed example, see “Getting and Setting Attribute Values” on page 51.

Setting Attribute Values

The attribute values for an AppExternalEntity object can be also be set. You can either send Java objects directly to the API with AppExternalEntity using the reference by number method or send strings with AppMappedExternalEntity using the reference by label method. Strings are converted into objects and passed to the Service Desk application by the API. With both reference methods, all business rules defined for the object will be applied by the workflow layer.

For more information, including a detailed example see “Getting and Setting Attribute Values” on page 51.

Saving Instantiations

Once the correct data is wrapped in an AppExternalEntity object, it can be saved to the database. This will create a new object in the Service Desk environment, or change an existing one, as existing or new objects are processed. While saving the data, all current business rules applicable for that Service Desk entity will be applied.

For more information, including a detailed example, see “Saving Instantiations” on page 55.

Single Instantiation Processing

The separate actions outlined in the sections above offer maximum control over the functionality realized using the API classes. For integrations limited to relatively simple operations, utility classes are provided. The utility actions are a combination of the actions described in the preceding sections. These are AppSingleLoad, for Service Desk entities, and AppRelationLoad, for relations between Service Desk entities. Both classes use the reference-by-label method. These utilities are capable of:

- adding an object;
- removing an object;
- setting object values;
- adding relations;
- removing relations.

For more information, including a detailed example, see “Single Instantiation Processing” on page 58.

Error Presentation

Exceptions raised within the Service Desk environment usually result in a Microsoft-specific `ComFailException`. These exceptions can be converted to `ExternalException` by the `AppExternalAccess` class. Exceptions are easier to manipulate in the standardized environment provided by the `ExternalException` class.

For more information, including a detailed example, see “Error Presentation” on page 62.

3 Examples

Most example classes explain two methods of referencing Service Desk entities and attributes. Most methods in the example files are defined as static. This has no relevance on the examples, but is intended for easy use within a test class.

Installing the Examples

All example classes provided use the API classes and rely on the contents of the Service Desk demo database. The demo database is installed when you install the evaluation version of Service Desk. When working with the full client version of Service Desk, you will need to select the demo database option during installation.

The examples classes are located in the Javadoc. You can find the Javadoc in the `Doc\API Javadoc` folder on the HP OpenView Service Desk 4.0 CD.

In order for the examples to work, you must complete the following procedure:

Step 1. Compile the example classes, using Microsoft Visual J++ compiler.

Step 2. Add the location of the compiled example classes to the current class path. The examples are located in the com.hp.ifc.ext.example package. For example, if the compiled classes are located in `C:\Program Files\Service Desk\classes\com\hp\ifc\ext\examples`, the class path should list `C:\Program Files \Service Desk\classes` as one of its entries.

Step 3. Run the `CreateStatus` executable. This executable adds some status values to the demo database that are essential when using the examples. A valid Service Desk user name, password, and server will be needed. The syntax is: `CreateStatus` <**`username`**> <**`password`**> <**`server`**>. This program requires the same classpath set for normal Service Desk operations.

Connecting to the Application Server

The AppExternalAccess class provides access to data managed by the Service Desk application server. This class can be considered a gateway into the workflow layer for all requests going to the application server. Every program that uses the API will be organized around at least one AppExternalAccess object to establish a connection with the application server. After successfully logging on, this same object functions as a Service Desk entity server. The Example1 class shows how to make a connection and how to verify that a valid connection exists.

The easiest way to log on is with the AppExternalAccess(String, String, String) constructor. This uses a user name, password, and server as arguments. These credentials can represent any Service Desk account, including a **non-UI account**. The use of this method is shown in the Example1.connect() method. The actual connection is made with the code in line 17:

```
access = new AppExternalAccess(username, password, server);
```

The rest of this method ensures that a message is displayed when a connection is not made.

The connected() method shows how to check whether an AppExternalAccess object contains a valid connection or not. The essential work is being done by calling AppExternalAccess.loggedIn() at line 25. This method returns a boolean value, indicating whether a valid connection exists or not. In the connected() method the boolean value determines what message to print.

Example 3-1 **Connecting to the Application Server**

```
package com.hp.ifc.ext.examples;

import com.hp.ifc.ext.*;

/**
 * Example:
 *   connecting to the HP OpenView Service Desk environment
 *   and testing the connection
 */

public class Example1 {
  private AppExternalAccess access;

  public void connect
    ( String username
    , String password
    , String server
    ) {
    try {
      access =
        new AppExternalAccess(username, password, server);
    }
    catch (ExternalException eEx) {
      System.out.println(eEx.getMessage());
    }
  }

  public void connected() {
    if ((access != null) && (access.loggedIn()))
      System.out.print(“Connected to “);
    else
      System.out.print(“Not connected to “);
    System.out.println
      (“the Service Desk application server.”);
  }

  public void disconnect() {
    if (access != null) access.disconnect();
  }

  public void finish() {
    if (access != null) access.shutdown();
  }

}
```

Identifying Entities and Attributes

An important part of the API programmer's work involves the manipulation of Service Desk entities and their attributes. A reliable method for identifying entities and attributes must be available for that purpose. The `Example2` class shows two ways of doing this: one using reference by number, the other reference by label. This class also provides an example for extracting the labels that are used when referencing by label.

The most reliable way to reference a Service Desk object is the use of the `ITSMExternalEnum` class. This class is a collection of nested classes, each representing a Service Desk entity. These inner classes are named after the entity represented. For example, the class for the person entity is called `PersonDefEnum`. Each inner class consists of a number of data members, one for each attribute. These data members are named after the attributes, with 'at' as a prefix. '`atBirthDate`' thus denotes the `BirthDate` attribute. An `enOid` member is also defined for every inner class. The `enOid` member denotes the numeric value (oid) used to identify the entity. Two simple examples:

`ITSMExternalEnum.IncidentDefEnum.enOid` denotes the incident entity. `ITSMExternalEnum.IncidentDefEnum.atDescription` denotes the `description` attribute for the incident entity.

Not every reference can be made easily with a long value, more complex ways also exist. These occur with the aggregation and referencing connections.

Aggregation is a means of referencing when an object is incorporated in another object. In this situation the attributes of the aggregated object can be considered nested objects. An example would be assignment data for an incident. Assignment, when considered as an object, has attributes such as a reference number or an assignment status. These attributes are referenced using long values, that can be specified using the `ITSMExternalEnum` class. For example:

```
long[] ref = new long[]
    {ITSMExternalEnum.IncidentDefEnum.atAssignment
    ,ITSMExternalEnum.AssignmentDefEnum.atReferenceNumber};
```

denotes the reference number attribute for an incident's assignment.

In other cases an object is not aggregated, but referenced. The referenced and referencing objects have an independent existence, but they are also linked to each other. For example, when a configuration item is mentioned in connection with an incident.

The getEntityAndAttributeLongs() shows how references for an entity and two of its attributes are defined. It returns long values that are referred to by the attributes used. When only indicating a reference, it is enough to give the attribute that references the other object. For example:

ITSMExternalEnum.IncidentDefEnum.atConfigurationItem denotes the reference to a configuration item attribute from the incident entity.

When working with Data Exchange, labels can be defined for Service Desk entities and attributes. These are grouped into import mapping settings (For detailed information on import mapping settings refer to the *HP OpenView Data Exchange Administrator's Guide*). The advantages of using these labels are:

- A tight integration with the labels used for Data Exchange is possible.
- Default values for new objects are defined. These defaults, organized in Service Desk templates, are applied automatically when using reference by label.
- Provides more freedom when choosing label names.

The getEntityAndAttributeNames() shows how references for an entity and two of its attributes are defined. As in Example1, a connection must be made to the Service Desk application server first. Next, the import settings must be chosen, as is done on line 25:

```
access.setSettings("external_event");
```

The getEntityAndAttributeNames () method returns an array of Strings shown in the code. It illustrates how reference by label relies on an external source (like a programmer's memory) to provide the right labels. The methods that use those labels will generate an exception when an invalid label is passed.

The showAvailableSettings() method demonstrates means of retrieving the labels defined for import settings, entities and attributes in the Service Desk environment. The values retrieved are presented in an ordered from, but this is purely meant as an example. The main reason to include this method is to demonstrate the methods that can retrieve these labels.

The important lines of code are:

- `String[] settings=access.getAvailableLoadSettingNames();` (line 41).This method call returns an array of all labels defined for import mapping.
- `String[] items = access.getMappedEntityNames();`(line 49). This line results in array of all labels defined for Service Desk entities within a given set of import mapping settings. For the method call to be effective, the `AppExternalAccess` object must first be set to the right import mapping setting with: `access.setSettings(settings[i]);` (line 47)
- `String[] attrib=access.getMappedAttributeNames(items[j]);` (line 55)Finally, this line produces an array of all labels defined for attributes of a Service Desk entity within an import mapping setting.

To print the available settings in alphabetical order the retrieval labels are organized in nested hashtables. This data structure, or a comparable one, can be used to get an overview of the available labels. Also, this makes it possible to check the validity of a label early on.

Apart from the actual labels, you can also retrieve object IDs. These can be obtained as arrays of long values or as arrays of `AppOID` objects, as described above.

Example 3-2 Identifying Entities and Attributes

```
package com.hp.ifc.ext.examples;

import java.util.*;
import com.hp.ifc.ext.*;
import com.hp.ifc.rep.ext.*;

/**
 * Example: referencing entities and attributes
 */

public class Example2 {

  public static long[] getEntityAndAttributeLongs() {
    return new long[]
      { ITSMExternalEnum.IncidentDefEnum.enOid
      , ITSMExternalEnum.IncidentDefEnum.atStatus
      , ITSMExternalEnum.StatusIncidentDefEnum.atText
      };
  }
```

```
public static String[] getEntityAndAttributeNames
  ( String username
  , String password
  , String server
  ) {
  AppExternalAccess access =
    new AppExternalAccess(username, password, server);
  access.setSettings(“external_event”);
  AppExternalEntityInfo ai =
    access.getExternalEntityInfo(“incident”);
  access.disconnect();
  return ai.getMappedAttributeNames();
}

public static void showAvailableSettings
  ( String username
  , String password
  , String server
  ) {
  AppExternalAccess access =
    new AppExternalAccess(username, password, server);

  // get all Import Mappings
  String[] settings= access.getAvailableLoadSettingNames();
  Hashtable hSets = new Hashtable();
  int iSet = settings.length;

  // get items per import mapping
  for (int i = 0; i < iSet; i++) {
    access.setSettings(settings[i]);
    Hashtable hItems = new Hashtable();
    String[] items = access.getMappedEntityNames();
    int iLen = items.length;
    // get attributes per item
    // add attributes to inner hashtable
    for (int j = 0; j < iLen; j++) {
      String[] attribs =
        access.getMappedAttributeNames(items[j]);
      hItems.put(items[j], attribs);
    }
    // add items to outer hashtable
    hSets.put(settings[i], hItems);
  }
  access.disconnect();
```

```
    // print hashtables
    StringBuffer sOut = new StringBuffer();
    Enumeration eIKeys = hSets.keys();
    Enumeration eIVals = hSets.elements();
    while (eIKeys.hasMoreElements()) {
      String sSet = (String) eIKeys.nextElement();
      sOut.append(“Import Mapping “ + sSet + “\r\n\r\n”);
      Hashtable hItems = (Hashtable) eIVals.nextElement();
      Enumeration eEKeys = hItems.keys();
      Enumeration eEVals = hItems.elements();
      while (eEKeys.hasMoreElements()) {
        String sEnt = (String) eEKeys.nextElement();
        sOut.append(“\tMapped Entity: “ + sEnt + “\r\n”);
        sOut.append(“\tAttributes:\r\n”);
        String[] aAttrs = (String[]) eEVals.nextElement();
        int iAttr = aAttrs.length;
        for (int k = 0; k < iAttr; k++) {
          sOut.append(“\t\t” + aAttrs[k] + “\r\n”);
        }
        sOut.append(“\r\n”);
      }
      sOut.append(“\r\n\r\n”);
    }
    System.out.println(sOut.toString());
  }
```

Finding the Proper Entity Instantiation

The Example3 class shows how to obtain a list of Service Desk entities that comply to a given search criteria. This is important when checking whether a given entity exists, or when retrieving a list of entities with certain characteristics.

In this example, three new classes are introduced. The most important one is the AppCriterium class, used to pass search criteria to the method that performs the actual search. In the construction of an AppCriterium object, the AppWhereOperatorEnum class is used to indicate the logical operators to use when combining AppCriterium objects. Apart from this, the AppAttributeSelection class is introduced. This class is used to specify the attributes that should be exposed on an entity. These classes belong to the Service Desk IFC and reside in the com.hp.ifc.util.marshal package. This package is imported on line 4 of the Example3 class.

Both methods used in this example produce an array of long values that represent the relevant object IDs. Relevance is determined here by the search criteria defined in the method. Also, the Service Desk entity must be defined for a search action.

An illustration of this process is given in the findIncidentByLabel() method. As the name indicates, this method uses reference by label. The construction of the actual list of object IDs is done by the listMappedEntitiesAsLong() method defined in the AppExternalAccess class (line 58). This method uses two arguments:

- The label for the Service Desk entity within the current import mapping settings.
- An array of AppCriterium objects defining the search criteria.

The single AppCriterium object used in this example is created in a call to another AppExternalAccess method, createEqualSearchCondition() on line 49. This is a utility method for the creation of a simple search condition. As arguments it takes:

- an indicator for the logical operator used in the combination of subsequent AppCriterium objects. This is a value from AppWhereOperatorEnum and can be either crtAnd (value 1) or crtOr (value 2). The first AppCriterium object in an array is normally given the and operator;

- the label for the Service Desk entity;
- the label for the Service Desk attribute involved;
- the attribute value to use in the search. This must always be a String object when working with reference by label.

The findIncidentByNumber () method shows the same operation performed using reference by number. The actual list is constructed by the find () method in line 32. The Service Desk attribute is indicated by the use of the ITSMExternalEnum class, in this case. In addition to the AppCriterium objects, the AppAttributeSelection object created on line 19, is passed to the find () method. This object serves to explicitly specify the attributes to retrieve. When only building a list of object oids, no attributes are necessary since the object ID is always retrieved by the API. It is sufficient to use the default selection.

NOTE The Service Desk application server automatically adds the object ID, and lockseq-value attributes used for Service Desk’s internal administration of objects retrieved and their status (that is, whether they are updated or not, and if changes have taken place since retrieval or not) when returning a selection. The workflow layer also adds attributes necessary for performing business rules. This prevents failures during the execution of business rules.

The creation of the AppCriterium object (line 22) also uses reference by number, diminishing the number of arguments to three. Though the actual search value is a String object, it is important to know that the class for the object passed here must be the same as the class defined for the Service Desk attribute referenced.

A number of methods are available in the AppExternalAccess class for both ways of listing Service Desk attributes. The methods differ in the arguments they take: long values, AppOID objects, String objects, and the results they return: arrays of long values, or AppOID objects.

The number of methods available for the creation of AppCriterium objects is even more abundant. Five different ways exist for passing arguments into methods. Three different types of search criteria that can be used are described below:

- Equality searches

- Range searches
- Free-format searches

Equality and range searches are made accessible with `createEqualSearchCondition()`and `createRangeSearchCondtion()` methods. Refer to the Javadoc for more information.

The free-format search methods are more complicated. Again an `AppCriterium` object is being constructed, but some additional arguments can be used. Two extensions make these methods more flexible; the addition of an operator for the interpretation of search values, and the use of a boolean to obtain the negation of a search condition. Together the free-format search methods offer much greater freedom in the construction of tailor-made search conditions.

As for the interpretation of the values specified, this is directed by labels defined in the `com.hp.ifc.util.marshal.AppCriteriumOperatorEnum` class. The available values are listed in the following table. An additional boolean argument can be used to reverse the selection condition. For example; instead of objects that have a value equivalent to yesterday in an attribute, one can specify those that have any value except yesterday. Note that some of these negations are already provided by labels:

Table 3-1 AppCriteriumOperationEnum Values

Label	Int value	# values
opEqual	0	1
opNotEqual	1	1
opGreaterThan	2	1
opLessThan	3	1
OpGreaterThanOrEqualTo	4	1
opLessThanOrEqualTo	5	1
OpBetween	6	2
opNotBetween	7	2
opContains	8	1

Table 3-1 **AppCriteriumOperationEnum Values**

Label	Int value	# values
opNotContains	9	1
opEmpty	11	0
opNotEmpty	12	0
opYesterday	14	0
opToday	15	0
opTomorrow	16	0
opLast7Days	17	0
opNext7Days	18	0
opLastWeek	19	0
opThisWeek	20	0
opNextWeek	21	0
opLastMonth	22	0
opThisMonth	23	0
opNextMonth	24	0
opStartsWith	25	1

Example 3-3 Finding the Proper Entity Instantiation

```
package com.hp.ifc.ext.examples;

import com.hp.ifc.ext.*;
import com.hp.ifc.util.marshal.*;

/**
 * Example: listing Service Desk objects
 */

public class Example3 {

  public static long[] findIncidentByNumber
  ( String username
  , String password
  , String server
  ) {
  AppExternalAccess access  =
    new AppExternalAccess(username, password, server);

  AppAttributeSelection sel =
    new AppAttributeSelection();

  AppCriterium[] crits =
    new AppCriterium[]
      { access.createEqualSearchCondition
        ( AppWhereOperatorEnum.crtAnd
        , ITSMExternalEnum.IncidentDefEnum.atDescription
        , “Server 02 booted”
        )
      };

  long[] Incs =
    access.find
      (ITSMExternalEnum.IncidentDefEnum.enOid, sel, crits);

  access.disconnect();
  return Incs;
}

public static long[] findIncidentByLabel
  ( String username
  , String password
  , String server
  ) {
  AppExternalAccess access =
    new AppExternalAccess(username, password, server);

  access.setSettings(“external_event”);

  AppCriterium[] crits =
    new AppCriterium[]
      { access.createEqualSearchCondition
        ( AppWhereOperatorEnum.crtAnd
        , “incident”
        , “description”
        , “Server 02 booted”
        )
      };
    long[] Incs =
      access.listMappedEntitiesAsLong(“incident”, crits);

    access.disconnect();
    return Incs;
  }
```

More about Finding Entity Instantiations

After you have told the API layer what kind of entity you want to use, you probably want to retrieve an instantiation of that entity type. This can be done with a call to find:

```
find(entity, selection, where);
```

After some processing, this command is translated into an SQL query and it can be clarifying to keep this mind:

SELECT ... FROM ... WHERE ...

In the API layer, the AppAttributeSelection class describes the SELECT part and the AppCriterium class describes the WHERE part.

Every object has a default selection. Sometimes you have to extend that default selection:

```
AppAttributeSelection asWo = new AppAttributeSelection();
asWo.putValue(ITSMExternalEnum.WorkorderDefEnum.atActualCost);
```

If an attribute references an object, the returned value of that attribute would be the AppOID of the referenced object. However, the object itself is returned when you extend the selection of the attribute with a subselection. The following example selects the default selection of the assignment object of a work order:

```
AppAttributeSelection subAsAs = new AppAttributeSelection();
AppAttributeSelection asWo    = new AppAttributeSelection();
asWo.putValue
  ITSMExternalEnum.WorkorderDefEnum.atAssignment, subAsAs  );
```

Sometimes a default selection already contains some subselections. Probably, you still have to extend the default selection and the default subselections in order to retrieve the desired attributes.

The following is an example of the WHERE part. This example retrieves all work orders assigned to “Janssen, John” that have the status “open”:

```
 AppCriterium[] where =
    new AppCriterium[]
      { access.createEqualSearchCondition
          ( AppWhereOperatorEnum.crtAnd
          , new long []
              { ITSMExternalEnum.WorkorderDefEnum.atAssignment
              ,
ITSMExternalEnum.AssignmentDefEnum.atAssigneePerson
              , ITSMExternalEnum.PersonDefEnum.atName
              }
          , “Janssen, John”
          )
      , access.createEqualSearchCondition
          ( AppWhereOperatorEnum.crtAnd
          , new long []
              { ITSMExternalEnum.WorkorderDefEnum.atStatus
             , ITSMExternalEnum.AssignmentStatusDefEnum.atText
              }
          , “Open”
          )
      }
```

Getting an Entity Instantiation

Example4 shows how to get a Service Desk object into the API programming environment. A large part of this example is copied from the example3 class, which begins the process for retrieving Service Desk objects.

The getIncidentByNumber() method shows the retrieval of a Service Desk incident, using reference by number. This method is similar to the findIncidentByNumber() method in example3, with a minor extension. After obtaining the list of relevant object IDs, the Service Desk object for each value is retrieved and wrapped in an AppExternalEntity object. This is done in line 40:

```
Ents[i]=access.open(ITSMExternalEnum.IncidentDefEnum.enOid,
sel, Incs[i]);
```

Arguments for this method are:

- An identifier for the Service Desk entity.
- An AppAttributeSelection object (see “Finding the Proper Entity Instantiation” on page 41).
- The object ID for the Service Desk object.

The return type for this method is the AppExternalEntity class. This is the API class for manipulating Service Desk objects using reference by number.

The getIncidentByLabel() method shows how to retrieve a Service Desk incident using labels. This is again very much like the findIncidentByLabel() method in Example3. The essential operation is performed on line 74:

```
Ents[i]=access.openMappedEntity("incident", Incs[i]);
```

This is a simplified variation of the open() method explained earlier. This method does not take an AppAttributeSelection object as an argument, because the API always retrieves all attributes that are provided with a label, when working with reference by label.

The openMappedEntity() method returns an AppMappedExternalEntity object. This is an extension of the AppExternalEntity object, aimed specifically at the manipulation of Service Desk objects using reference by label.

Example 3-4 Getting an Entity Instantiation

```
package com.hp.ifc.ext.examples;

import com.hp.ifc.ext.*;
import com.hp.ifc.util.marshal.*;

/**
 * Example: retrieval of Service Desk objects
 */

public class Example4 {

  public static AppExternalEntity[] getIncidentByNumber
    ( String username
    , String password
    , String server
    ) {
    AppExternalAccess access  =
      new AppExternalAccess(username, password, server);

    AppCriterium[] crits =
      new AppCriterium[]
        { access.createEqualSearchCondition
          ( AppWhereOperatorEnum.crtAnd
          , ITSMExternalEnum.IncidentDefEnum.atDescription
          , “Server 02 booted”
          )
        };
    AppAttributeSelection sel =
      new AppAttributeSelection();
    // extending the default selection
    sel.putValue
      (ITSMExternalEnum.IncidentDefEnum.atDescription);
    long[] Incs =
      access.find
        (ITSMExternalEnum.IncidentDefEnum.enOid, sel, crits);

    AppExternalEntity[] Ents =
      new AppExternalEntity[Incs.length];
    for (int i = 0; i < Incs.length; i++) {
      Ents[i] =
        access.open
          ( ITSMExternalEnum.IncidentDefEnum.enOid
          , sel
          , Incs[i]
        );
    }
    access.disconnect();
    return Ents;
  }

  public static AppMappedExternalEntity[] getIncidentByLabel
    ( String username
    , String password
    , String server
    ) {
    AppExternalAccess access =
      new AppExternalAccess(username, password, server);
    access.setSettings(“external_event”);

    AppCriterium[] crits =
      new AppCriterium[]
        { access.createEqualSearchCondition
          ( AppWhereOperatorEnum.crtAnd
          , “incident”
          , “description”
          , “Server 02 booted”
        )
      };
    long[] Incs =
      access.listMappedEntitiesAsLong(“incident”, crits);

    AppMappedExternalEntity[] Ents =
      new AppMappedExternalEntity[Incs.length];
    for (int i = 0; i < Incs.length; i++) {
      Ents[i] =
        access.openMappedEntity(“incident”, Incs[i]);
    }
    access.disconnect();
    return Ents;
  }

  public static void showDefaultSelections
    ( String username
    , String password
    , String server
    ) {
    AppExternalAccess access  =
      new AppExternalAccess(username, password, server);
    AppCriterium[] crits =
      new AppCriterium[]
```

```
        { access.createEqualSearchCondition
          ( AppWhereOperatorEnum.crtAnd
          , ITSMExternalEnum.IncidentDefEnum.atDescription
          , “Server 02 booted”
          )
        };
    AppAttributeSelection sel =
      new AppAttributeSelection();

    System.out.println
      (“Initial selection: “ + access.showSelection(sel));
    long[] Incs =
      access.find
        (ITSMExternalEnum.IncidentDefEnum.enOid, sel, crits);
    System.out.println
      (“Default selection of find: “ +
access.showSelection(sel));

    if (Incs.length>0) {
      sel = new AppAttributeSelection();
      access.open
       (ITSMExternalEnum.IncidentDefEnum.enOid, sel, Incs[0]);
      System.out.println
        (“Default selection of open: “ +
access.showSelection(sel));
    }
    access.disconnect();
  }
```

Creating New Objects

Example5 shows how new Service Desk objects are created by means of the AppExternalAccess class.

The makeIncidentByNumber() method shows how to create Service Desk objects using reference by number. A quick glance at the actual method call on create() (line 24), makes clear that it is used in much the same way as the open() method used in Example4. The only difference is that an object ID is not passed into this method. The create() method returns an AppExternalEntity object.

The same applies to the createMappedEntity() method used in makeIncidentByNumber(), showing reference by label. The createMappedEntity() method called on line 41, compares to the openMappedEntity() as does the create() method to the open() method. The createMappedEntity() method returns an AppMappedExternalEntity object.

Example 3-5 Creating New Objects

```
package com.hp.ifc.ext.examples;

import com.hp.ifc.ext.*;
import com.hp.ifc.util.marshal.*;

/**
 * Example: creating of Service Desk objects
 */

public class Example5 {

  public static AppExternalEntity makeIncidentByNumber
    ( String username
    , String password
    , String server
    ) {
    try {
      AppExternalAccess access =
        new AppExternalAccess(username, password, server);
      AppAttributeSelection sel =
        new AppAttributeSelection();
      sel.putValue
        (ITSMExternalEnum.IncidentDefEnum.atDescription);
      return
        access.create
          (ITSMExternalEnum.IncidentDefEnum.enOid, sel);
    }
    catch (ExternalException exc) {
      System.out.println(exc.getMessage());
      return null;
    }
  }

  public static AppMappedExternalEntity makeIncidentByLabel
    ( String username
    , String password
    , String server
    ) {
    try {
      AppExternalAccess access =
        new AppExternalAccess(username, password, server);
      access.setSettings(“external_event”);
      return access.createMappedEntity(“incident”);
    }
    catch (ExternalException exc) {
      System.out.println(exc.getMessage());
      return null;
    }
  }
```

Getting and Setting Attribute Values

The Example6 class shows the creation of a new Service Desk object, and the elementary manipulation of one of the object's attributes.

The first example method, changeIncidentByNumber(), as always applies to reference by number. After creating a new incident object, this method shows the contents of the description attribute. This will be empty (null) since we are working with a new incident.

On line 33, a value is assigned to the description. This is a done by calling the setValue() method on the AppExternalEntity object that represents the incident. The changeIncidentByNumber() shows the attribute's contents, that are now set to "An example".

In this example a String object is passed into setValue(), this method expects to receive an appropriate object type. The Java types you are most likely to encounter are listed in Table 3-2. You will need to determine the type of Service Desk attribute to be manipulated in order to use the table. The determineAttribueType() method can be used to determine this, when needed.

The changeIncidentByLabel() method shows the same sequence of events using reference by label. Apart from the differences in object creation that are already familiar from Example5, it will appear that the description attribute already has a value the first time it is shown. This is caused by the application of default values from the template, assigned to the incident entity defined in the import settings.

Setting the attribute value on line 66 differs little from the approach used in changeIncidentByNumber(). Apart from using reference by label the setValue() method defined for AppMappedExternalEntity always takes a String object for the value.

Table 3-2 **Java Types**

Service Desk Attribute Type	Java Data Type
Boolean (Yes/No)	Boolean
Currency (money)	Double

Table 3-2 **Java Types**

Service Desk Attribute Type	Java Data Type
Date (without time)	Com.ms.wfc.app.Time or Double representing the number of hundred nanosecond units elapsed since January 1, 100AD 12:00 midnight.
Datetime (timestamp)	Com.ms.wfc.app.Time or Double representing the number of hundred nanosecond units elapsed since January 1, 100AD 12:00 midnight.
Description (length 80)	String
Double	Double
Duration	Double representing number of minutes.
Email	String
EntityReference	AppOID
Gender	AppOID (0=male, 1=female)
Integer	Integer
Long	Long
Longtext (memo)	String
Name (length 50)	String
Searchcode (length 50)	String (Searchcode must be all capitals, cannot contain any spaces or the characters: ’.’ , ’*’ , ’_’ , ’%’ and cannot start with any numeric characters.
Shortext (length 40)	String
Telephone	String
Text (length 225)	String
Text64kB	String

Example 3-6 Getting and Setting Attribute Values

```
package com.hp.ifc.ext.examples;

import com.hp.ifc.ext.*;
import com.hp.ifc.util.marshal.*;

/**
 * Example: manipulating Service Desk objects
 */

public class Example6 {

  public static void changeIncidentByNumber
    ( String username
    , String password
    , String server
    ) {
    Object s;
    try {
      AppExternalAccess access =
        new AppExternalAccess(username, password, server);
      AppAttributeSelection sel =
        new AppAttributeSelection();
      sel.putValue
        (ITSMExternalEnum.IncidentDefEnum.atDescription);
      AppExternalEntity oEnt =
        access.create
          (ITSMExternalEnum.IncidentDefEnum.enOid, sel);

      s = oEnt.getValue
          (ITSMExternalEnum.IncidentDefEnum.atDescription);
      System.out.println(“Description is “ + s);
      oEnt.setValue
        ( ITSMExternalEnum.IncidentDefEnum.atDescription
        , “An example”
        );
      s = oEnt.getValue
          (ITSMExternalEnum.IncidentDefEnum.atDescription);
      System.out.println(“Description is set to “ + s);
      access.disconnect();
    }
    catch (ExternalException exc) {
      System.out.println(exc.getMessage());
    }
  }
  public static void changeIncidentByLabel
    ( String username
    , String password
    , String server
    ) {
    Object s;
    try {
      AppExternalAccess access =
        new AppExternalAccess(username, password, server);
      access.setSettings(“external_event”);
      AppMappedExternalEntity oMEnt =
        access.createMappedEntity(“incident”);
      s=oMEnt.getValue(“description”);
      System.out.println(“Description is “ + s);
      oMEnt.setValue(“description”, “Another example”);
      s=oMEnt.getValue(“description”);
      System.out.println(“Description is set to “ + s);
      access.disconnect();
    }
    catch (ExternalException exc) {
      System.out.println(exc.getMessage());
    }
  }

  public static void showAttributeType
    ( String username
    , String password
    , String server
    ) {
    try {
      AppExternalAccess access =
        new AppExternalAccess(username, password, server);
      String s =
        access.showAttributeType
          (ITSMExternalEnum.IncidentDefEnum.atDescription);
      System.out.println(“Type of incident description is “ +
s);
      access.disconnect();
    }
    catch (ExternalException exc) {
      System.out.println(exc.getMessage());
    }
  }
```

Saving Instantiations

Example7 shows how to save an object to the Service Desk environment. In doing so, the new or changed object is made available for use by other Service Desk users. This example in fact is a minor extension of the previous example.

In both methods of the Example7 class the object is stored by a call to the save() method on the AppExternalEntity class. In saveIncidentByLabel(), the AppMappedExternalEntity class is used, but the save() method is inherited from AppExternalEntity. The calls can be found on lines 45and 64. This method saves the Service Desk object wrapped in the AppExternalEntity object to the Service Desk database.

The saveIncidentByNumber() method can produce an error message. Since only some attributes are set, the Service Desk application server can generate an exception with the text "you must fill in the <attribute> box". This demonstrates that it is important to know which attributes must be filled, because if these are not set, a new object will not be saved.

The saveIncidentByLabel() method doesn't report an error, because the mandatory attributes are filled from the template defined by the import settings. This demonstrates a big advantage of working with the import settings: it is possible to define default values that make sense. One can even define several sets for a single Service Desk entity, that will than be applied for different labels. Of course, using a template will only prevent messages caused by empty attributes when all mandatory attributes have received a template value.

The AppExternalEntity class has two methods that are more or less analogous to the save() method:

- delete()
  This method will remove the Service Desk object being processed (as far as business rules allow this). This clearly does not work on newly created objects.
- rollback()
  This method will revert all changes on the Service Desk object being processed, since it was created or retrieved from Service Desk. If rollback() was applied previously, rollback() will revert all

changes applied since the previous call on `rollback()`.

Example 3-7 Saving Instantiations

```
package com.hp.ifc.ext.examples;

import com.hp.ifc.ext.*;
import com.hp.ifc.util.marshal.*;

/**
 * Example: manipulating and saving Service Desk objects
 */

public class Example7 {

  public static void saveIncidentByNumber
    ( String username
    , String password
    , String server
    ) {
    try {
      AppExternalAccess access =
        new AppExternalAccess(username, password, server);
      AppAttributeSelection sel  =
        new AppAttributeSelection();
      AppAttributeSelection subSel =
        new AppAttributeSelection();
      sel.putValue
        (ITSMExternalEnum.IncidentDefEnum.atDescription);
      sel.putValue
        (ITSMExternalEnum.IncidentDefEnum.atInformation);
      sel.putValue
        (ITSMExternalEnum.IncidentDefEnum.atStatus, subSel);

      AppExternalEntity oEnt =
        access.create
          (ITSMExternalEnum.IncidentDefEnum.enOid, sel);
      oEnt.setValue
          ( ITSMExternalEnum.IncidentDefEnum.atDescription
          , “An example”
          );
      oEnt.setValue
          ( ITSMExternalEnum.IncidentDefEnum.atInformation
          , “An example”
          );
      oEnt.setValue
    ( ITSMExternalEnum.IncidentDefEnum.atStatus
    , “Registered”
    );
  oEnt.save();
  access.disconnect();
  System.out.println(“New incident is saved”);
 }
 catch (ExternalException exc) {
  System.out.println(exc.getMessage());
 }
}

public static void saveIncidentByLabel
 ( String username
 , String password
 , String server
 ) {
 try {
  AppExternalAccess access =
   new AppExternalAccess(username, password, server);
  access.setSettings(“external_event”);
  AppMappedExternalEntity oMEnt =
  access.createMappedEntity(“incident”);
  oMEnt.setValue(“description”, “Another example”);
  oMEnt.save();
  access.disconnect();
  System.out.println(“New incident is saved”);
 }
 catch (ExternalException exc) {
  System.out.println(exc.getMessage());
 }
}
```

Single Instantiation Processing

Example8 shows a simplified way of working with the Service Desk API. By using two utility classes you simplify the tasks involved in opening, creating, and saving App(Mapped)ExternalEntities and setting their values. A single method call can create a Service Desk object, or a relation between two Service Desk objects. The one condition is that import settings must be defined for these objects, because the methods use reference by label.

The relateObjects() method shows the creation of two Service Desk objects and a relation between these two objects. The actual objects are created using the AppSingleLoad.processEntity() method (lines 32 and 37), the relation is created by the AppRelationLoad.processRelation() method (line 50).

To get into the AppSingleLoad class first, this has several overloaded versions of the processEntity() method. The one used here is the simplest, that takes four arguments:

- The label for the Service Desk entity within the current import mapping.
- An array of labels for the Service Desk attributes whose values are to be set.
  This is created, and filled on line 30. It is the API programmer's responsibility to include all attribute labels that are defined as key attributes for the entity, otherwise an exception will be thrown.
- An array of String objects, specifying the values to which these attributes will be set.
  This second array is created and filled on line 31. Labels and values are combined according to their positions in the arrays.
- A boolean value, indicating whether the Service Desk object defined must be saved (false) or removed (true).

A number of prominent Service Desk entities are provided with a Source ID. This attribute is intended for the storage of object identifiers from external systems. This is illustrated here by the use of the NNM_ID label, this is used to store an external NNM ID field in the Source ID attribute. This use of the Source ID field is strongly recommended.

Lines 35 through 36 show the re-use of previously defined Java objects to process a second configuration item.

The AppSingleLoad class has a number of processEntity() methods. The ones not shown here take the following additional arguments:

- The import setting name, relieving us from the need to set that attribute (can be important when different import settings are used in one program).
- The import setting name, plus a Service Desk user name, password, and server.

The AppRelationLoad class has just one method, processRelation() (lines 50, 51). This method takes five arguments:

- The label for the first Service Desk entity.
- An array with key attributes and values, as created on line 42 through 44. These are used for identifying the first Service Desk object.
- A label describing the kind of Service Desk relation.
- The label for the second Service Desk entity.
- An array with key attributes and their values, as created on line 46 through 48. These are used for the identification of the second Service Desk object.

It is the API programmer's responsibility to include all key attributes. If all key attributes are not included, an exception will be thrown, because the object cannot be identified.

Example 3-8 Single Instantiation Processing

```
package com.hp.ifc.ext.examples;

import com.hp.ifc.ext.*;
import com.hp.ifc.util.marshal.*;

/**
 * Example: simplified manipulation of objects and relations
 */

public class Example8 {

  public static void relateObjects
    ( String username
    , String password
    , String server
    ) {
    try {
      String nextNumber  = “0008”;
      String segmentId   = “NNM_Segment_”  + nextNumber;
      String segmentName = “Test_Segment_” + nextNumber;
      String networkId   = “NNM_Network_”  + nextNumber;
      String networkName = “Test_Network_” + nextNumber;
      String[] aAttr, aVals;

      AppExternalAccess access =
        new AppExternalAccess(username, password, server);
      access.setSettings(“nnm6_import”);

      AppSingleLoad load = new AppSingleLoad(access);

      aAttr = new String[] {“NNM_ID”, “NAME”};
      aVals = new String[] {segmentId, segmentName};
      load.processEntity(“SEGMENT”, aAttr, aVals, false);
      System.out.println(“Saved segment “+ segmentId);

      aAttr = new String[] {“NNM_ID”, “NAME”};
      aVals = new String[] {networkId, networkName};
      load.processEntity(“NETWORK”, aAttr, aVals, false);
      System.out.println(“Saved network “+ networkId);

      AppRelationLoad relate = new AppRelationLoad(access);

      String[][] aParentKey = new String[2][1];
      aParentKey[0][0] = “NNM_ID”;
      aParentKey[1][0] = segmentId;

      String[][] aChildKey = new String[2][1];
      aChildKey[0][0] = “NNM_ID”;
      aChildKey[1][0] = networkId;

      relate.processRelation
        ( “SEGMENT”, aParentKey, “Parent”
        , “NETWORK”, aChildKey
        );
      System.out.println
        (“Saved relation: “ + segmentId + “ - “ + networkId);
      access.disconnect();
    }
    catch (ExternalException exc) {
      System.out.println(exc.getMessage());
```

```
    }
}
```

Error Presentation

Example9 demonstrates how Java exceptions, thrown by the Service Desk API classes, can be caught and displayed. This has already been shown in a number of other classes, but the Example9 class is more detailed.

TheService Desk application server normally throws a `ComFailException` when something unexpected happens. Since these exceptions are Microsoft-specific, the Service Desk API throws them again as `ExternalExceptions`. These are generic `Exception` objects, that also have a severity attribute. This is used to distinguish between errors, warnings and information messages. The severity values are defined in `com.hp.ifc.rep.AppSeverityEnum`, as imported in `Example9` on line 5.

The `AppExternalAccess` class has a method `getMessage()` that re-throws any exception passed to it as an ExternalException. Any Exception caught by the API classes is re-thrown by this method, so these are all caught with an exception handler on `ExternalExceptions`. If the exception passed is a `ComFailException`, its severity will be taken over. All other exceptions will be assigned severity `svCritical`. Since other kinds of exceptions can always be thrown, it is a good idea to catch all exceptions, this example doesn't do this.

The `Example9` class contains an exception handler on both member methods. This shows how context-specific error messages are generated from an `ExternalException` (lines 31-37 and 56-61 using the `switch` statement). The exception handlers record the `External Exception's` severity (lines 32 and 57), and use this to determine a prefix for the error message.

This example can be expanded using different ways of displaying or logging the external exceptions thrown. In doing so, error reporting can be brought in line with the API programmer's organizational standards.

Example 3-9 **Error Presentation**

```
package com.hp.ifc.ext.examples;

import com.hp.ifc.ext.*;
import com.hp.ifc.util.marshal.*;
import com.hp.ifc.rep.AppSeverityEnum;
```

```
/**
 * Example: extended error reporting
 */

public class Example9 {

  public static void saveIncidentByNumber
    ( String username
    , String password
    , String server
    ) {
    try {
      AppExternalAccess access =
        new AppExternalAccess(username, password, server);
      AppAttributeSelection sel =
        new AppAttributeSelection();
      sel.putValue
        (ITSMExternalEnum.IncidentDefEnum.atDescription);
      AppExternalEntity oEnt =
        access.create
          (ITSMExternalEnum.IncidentDefEnum.enOid, sel);
      oEnt.save();
      access.disconnect();
      System.out.println(“New incident is saved”);
    }
    catch (ExternalException exc) {
      String s = “INFO”;
      switch(exc.getSeverity()) {
        case AppSeverityEnum.svCritical: s = “ERROR”;  break;
        case AppSeverityEnum.svWarning: s = “WARNING”; break;
      }
      System.out.println(s + “: “ + exc.getMessage());
    }
  }

  public static void saveIncidentByLabel
    ( String username
    , String password
    , String server
    ) {
    try {
      AppExternalAccess access =
        new AppExternalAccess(username, password, server);
      access.setSettings(“external_event”);
      AppMappedExternalEntity oMEnt =
        access.createMappedEntity(“incident”);
      oMEnt.setValue(“description”, “dummy description”);
      oMEnt.setValue(“status”, null);
      oMEnt.save();
      access.disconnect();
      System.out.println(“New incident is saved”);
    }
    catch (ExternalException exc) {
      String s = “INFO”;
      switch(exc.getSeverity()) {
        case AppSeverityEnum.svCritical: s = “ERROR”;  break;
        case AppSeverityEnum.svWarning: s = “WARNING”; break;
      }
      System.out.println(s + “: “ + exc.getMessage());
    }
  }
```

Handling a Reference to a Set of Entities

When an attribute references a set, each item of the set must be handled separately. This can be done as follows:

```
...

  // get the set of attached items
  long[] oidAttachedItems =
    eWo.getSet
      (new long[]
          { ITSMExternalEnum.WorkorderDefEnum.atAttachment
          , ITSMExternalEnum.AttachmentDefEnum.atAttachedItems
          }
       );

    // open each attached item using the object id
    for (int j = 0; j < oidAttachedItems.length; j++) {
      AppExternalEntity  eAttItem =
        access.open
          ( ITSMExternalEnum.AttachedItemDefEnum.enOid
          , new AppAttributeSelection();
          , oidAttachedItem[j]
          );

      ...
    }
  }

  ..
```

The following example demonstrates how you can handle a reference to a set of entities.

Example 3-10 **Reference to a Set of Entities**

```
package com.hp.ifc.ext.examples;

import com.hp.ifc.ext.*;
import com.hp.ifc.types.*;
import com.hp.ifc.util.marshal.*;

/**
 * Example: handling entity sets
 */
```

```
public class Example10 {

  public static void getCiChildren
    ( String username
    , String password
    , String server
    ) {

    // login
    AppExternalAccess access =
      new AppExternalAccess(username, password, server);

    // Set the search condition, use the default selection and
find the
    // object ID of the parent CI (the object id is of type
long, it is
    // sometimes wrapped in class AppOID)
    AppCriterium[] whereParent =
      new AppCriterium[]
        { access.createEqualSearchCondition
          ( AppWhereOperatorEnum.crtAnd
          ,
ITSMExternalEnum.ConfigurationItemDefEnum.atSearchcode
          , “WAN01”
          )
        };
    AppAttributeSelection selParent = new
AppAttributeSelection();
    long[] oidParents =
        access.find
          ( ITSMExternalEnum.ConfigurationItemDefEnum.enOid
          , selParent
          , whereParent
          );

    // extend the default selection and open each found parent
object id
    AppAttributeSelection subSelChildCis = new
AppAttributeSelection();
    selParent.putValue
      (
ITSMExternalEnum.ConfigurationItemDefEnum.atChildConfiguration
Items
      , subSelChildCis
      );
    for (int i = 0; i < oidParents.length; i++) {
      AppExternalEntity eParent = access.open
          ( ITSMExternalEnum.ConfigurationItemDefEnum.enOid
          , selParent
          , oidParents[i]
          );
      // get the set of ci components
      long[] oidCiComponents =
        eParent.getSet

(ITSMExternalEnum.ConfigurationItemDefEnum.atChildConfiguratio
nItems);
      // open each ci component of the set
      for (int j = 0; j < oidCiComponents.length; j++) {
        AppAttributeSelection selCiComp = new
AppAttributeSelection();
        AppExternalEntity eCiComponent =
          access.open
            ( ITSMExternalEnum.CiComponentDefEnum.enOid
            , selCiComp
            , oidCiComponents[j]
            );

        // get the object ID of the child
        //
getValue(ITSMExternalEnum.CiComponentDefEnum.atCiChild)
        //   returns an object instead of an object id because
        //   ITSMExternalEnum.CiComponentDefEnum.atCiChild has
a
        //   default subselection
        Object oOidChild =
          eCiComponent.getValue
            ( new long[]
               { ITSMExternalEnum.CiComponentDefEnum.atCiChild
                ,
ITSMExternalEnum.ConfigurationItemDefEnum.atObjectId
                }
            );
        long oidChild = ((AppOID) oOidChild).longValue();

        // open the child
        AppAttributeSelection selChild = new
AppAttributeSelection();
        selChild.putValue

(ITSMExternalEnum.ConfigurationItemDefEnum.atSearchcode);
```

```
        AppExternalEntity eChild =
          access.open
            ( ITSMExternalEnum.ConfigurationItemDefEnum.enOid
            , selChild
            , oidChild
         );

        // get the searchcode
        Object oSearch =
          eChild.getValue(

ITSMExternalEnum.ConfigurationItemDefEnum.atSearchcode);
        System.out.println(“Component searchcode is “ +
oSearch);
      }
    }
    access.disconnect();
    return;
  }
```

Glossary

A

API Application programming interface. An interface that enables programmatic access to an application.

attribute A characteristic or property associated with a system, network, or other item. OpenView items represent system, network, and personnel resources by modeling and providing information on the attributes (properties and state) of an object. An attribute has a name and a value. An attribute is a place holder in which a specific value is held to provide information about the state of the item. For example, incident description, or employee name.

B

business layer The business layer contains rules that determine how an operation is validated or completed. It communicates with the data access layer and the workflow layer. The business layer works on the database side to find, load, delete, or save data and then present only the data asked for to the workflow layer.

business rules A set of hooks that overrule the standard behavior defined for an entity in the repository and inherited from the super class. The hooks are collected in a class that extends AppEntity.

C

configuration item An item belonging to the technical infrastructure of an organization. A configuration item may consist of other configuration items, and may be part of other configuration items. For example, a PC, an application program, a network, a work space with desks and chairs. Configuration items are stored in the application database.

class An encapsulated collection of data and methods to operate on the data. A class may be instantiated to produce an object that is an instance of the class. The class defines how the objects should be accessed by other objects, whether the object is public, and under what circumstances it can be created.

D

data access layer The data access layer provides access to the data in the database. It communicates between the application's object model and the relational representation of what exists in the underlying database.

data mapping model Describes how entities and their attributes in the object model are mapped to the database. The application server reads it when started. It is used by the business layer to persist the data. The data mapping model is persisted in the ifc_tables and the ifc_columns.

E

encapsulate An object-oriented programming technique that makes an object's data private or protected (that is: hidden) and allows programmers to access and manipulate that data only through method calls. Done well, encapsulation reduces bugs and promotes reusability and modularity of classes. This technique is also known as data hiding.

entity An entity is a logical collection of attributes. It is the basic item in the object model of Service Desk. In the object model, the entity is the top item.

I

IDL Interface definition language. Used to describe CORBA object interfaces. It describes the interfaces that client objects call and object implementations provide.

IFC ITSM foundation classes. These classes together form the base of the Service Desk application, called the kernel. Service Desk-specific features have not been incorporated in the IFC, making it possible to develop other database-based applications with it.

instance An object that exposes a particular interface. An object is an instance of an interface if it provides the operations, signatures, and semantics specified by that interface. An object is an instance of an implementation if its behavior is provided by that implementation.

interface A specification, written in IDL, of the operations and attributes that an object provides.

K

kernel See IFC

N

non-UI account Service Desk accounts granting users access to the Service Desk application but not the user interface. Non-UI accounts can be created for users who will not be using the Service Desk interfaces but need access for other purposes. The number of named user accounts is usually limited by the licensing agreement, while non-UI accounts are not.

O

object An encapsulated software unit consisting of both state (data) and behavior (code). Objects have attributes, methods, and events. Attributes make up the data that describes an object. Methods are the functions an object can perform. Events are the functions an object can perform in response to another event. In some object models an object is an instance of a class as specified in some object-modeling languages.

object model A model of the application's entities, attributes, and their relations.

P

presentation layer The presentation layer contains the user interfaces. Information from the presentation layer is passed to the workflow layer.

R

repository The repository is a collection of data that governs the behavior of the application. The repository contains information about: the object model, data mapping model, user-interface configuration, role information, accounts, folders, labels, and messages.

U

user interface configuration

Includes the data form definitions, data view definitions, and user-specific definitions.

W

workflow layer The workflow layer consists of classes on the client. It operates directly under the presentation layer and is responsible for handling the data flow between the business layer and the presentation layer (or the API). If data is changed by the presentation layer or the API the workflow layer check that the data is logically correct.

wrapper A type of glueware that is used to attach other software components together. A wrapper may encapsulate a single system, often a data source, to make it usable in some new way that the unwrapped system was not. Wrappers can be used to expose all or some of the functionality of the thing they are wrapping, and present a simplified or standard interface to make a component more available.

Index

Index