



Hierarchy Developer Guide

Version 5.1

Pub Date 02/09/2006

Siebel Systems, Inc., 2207 Bridgepointe Parkway, San Mateo, CA 94404

Copyright © 2006 Siebel Systems, Inc.

All rights reserved.

Printed in the United States of America

No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way, including but not limited to photocopy, photographic, magnetic, or other record, without the prior agreement and written permission of Siebel Systems, Inc.

Siebel, the Siebel logo, UAN, Universal Application Network, Siebel CRM OnDemand, and other Siebel names referenced herein are trademarks of Siebel Systems, Inc., and may be registered in certain jurisdictions.

Other product names, designations, logos, and symbols may be trademarks or registered trademarks of their respective owners.

PRODUCT MODULES AND OPTIONS. This guide contains descriptions of modules that are optional and for which you may not have purchased a license. Siebel's Sample Database also includes data related to these optional modules. As a result, your software implementation may differ from descriptions in this guide. To find out more about the modules your organization has purchased, see your corporate purchasing agent or your Siebel sales representative.

U.S. GOVERNMENT RESTRICTED RIGHTS. Programs, Ancillary Programs and Documentation, delivered subject to the Department of Defense Federal Acquisition Regulation Supplement, are "commercial computer software" as set forth in DFARS 227.7202, Commercial Computer Software and Commercial Computer Software Documentation, and as such, any use, duplication and disclosure of the Programs, Ancillary Programs and Documentation shall be subject to the restrictions contained in the applicable Siebel license agreement. All other use, duplication and disclosure of the Programs, Ancillary Programs and Documentation by the U.S. Government shall be subject to the applicable Siebel license agreement and the restrictions contained in subsection (c) of FAR 52.227-19, Commercial Computer Software - Restricted Rights (June 1987), or FAR 52.227-14, Rights in Data—General, including Alternate III (June 1987), as applicable. Contractor/licensor is Siebel Systems, Inc., 2207 Bridgepointe Parkway, San Mateo, CA 94404.

Proprietary Information

Siebel Systems, Inc. considers information included in this documentation and in Siebel Online Help to be Confidential Information. Your access to and use of this Confidential Information are subject to the terms and conditions of: (1) the applicable Siebel Systems software license agreement, which has been executed and with which you agree to comply; and (2) the proprietary and restricted rights notices included in this documentation.

Contents

1 Hierarchy Overview

Hierarchy Features and Services 7

Object Management Framework (OMF) Features and Services 8

2 Hierarchy Related Concepts

Basics 9

Hierarchy Roles Based Access Control 10

Hierarchy Type 11

Reporting Periods and Versioning 12

 Reporting Periods 12

 Hierarchy Versioning 12

Hierarchy Life Cycle States 13

 Data Replication 14

Assigned and Unassigned Objects 15

Hierarchy Example 15

3 Hierarchy/OMF Architecture Overview

Hierarchy and OMF Components 19

System Collaborations 22

Working with Reporting 22

4 Basic System Use Cases

Configure Hierarchy Types 25

Create and Modify Hierarchies using APIs 27

 Adding or Removing Entities To or From a Hierarchy 28

 Add, and Remove Users for Hierarchy Access Control (HBAC) 29

Search & Filter Hierarchies 29

 Search and Filter Hierarchies Using IHierarchyManager 29

 Search Nodes within Hierarchy 30

Navigate Hierarchy 30

Search on Link Target Attributes Within Hierarchy 30

Taking the User's Role into Consideration 31

Create Hierarchies 31

Find a List of Hierarchies that a Given User Has Access To 31

Find Top Level Nodes that a User Has Access To for a Given Hierarchy 32

Find Assigned Link Targets or OMF Objects 32

Find Unassigned Link Targets or OMF Objects 33

Managing Business Objects 33

Exchange Hierarchy Data using XML 33

XML Schema 34

Importing Hierarchies 35

Exporting Hierarchies 35

5 Advanced System Use Cases – Extending System

Creating New Type of Business Object to Work with Hierarchy 37

Create new business objects 37

Register Objects With OMF module 37

Making Sure That Your Business Object Is Transactional Aware 39

Making Your OMF Objects Work with Hierarchy 40

Implement Hierarchy Related Interfaces 41

Make Your New OMF Object Searchable Through Hierarchy 47

Configure Searchable Properties 47

Provide support for reporting on the new business object 50

Support XML exchange 51

Working with External SSO System 55

Working with Extended Attributes on Service Agreement Object 58

6 Working with the ETL Process

ETL 59

Load Handler Configuration 60

ETL Loader Interface 61

APIs 61

IAttribute 61

IExpression 61

IFilter 62

IFilteredQuery 62

IHierarchy 63

IHierarchyFolder 67

IHierarchyFolderManager 67

IHierarchyHandle 69

IHierarchyLinkTarget 69

IHierarchyManager 69

IHierarchyNode 73

IHierarchyNodeHandle 77

IHierarchyService 78

IHierarchyType 79

IHierarchyTypeManager 79

IHierarchyUserRef 80

ILinkTargetConfig 80

7 Appendix A Hierarchy XML Exchange Schema

8 Appendix B: Sample Hierarchy XML File

1 Hierarchy Overview

Hierarchy Features and Services

The XMA Hierarchy Manager provides an organizational console for mapping the department and personnel structures of a business customer's entire enterprise. Its web browser interface minimizes training and maximizes productivity. It enables authorized users within a B2B organization to quickly model personnel and departments and grant appropriate access at each level of the structure. It also saves organizational data and structures in an extremely efficient format for rapid searches and queries across the hierarchy.

Hierarchy provides user access control. It also provides a mechanism to glue together data into a tree structure. When you define a hierarchy structure, you can navigate and run reports based on that structure.

Key features of hierarchy include:

- Robust and generic organizational modeling
- Unlimited Levels in the Hierarchy
- Hierarchy-based Access Control (HBAC)
- Work with Object Management Framework (OMF) to allow any type of business object being linked into hierarchy
- Multiple Hierarchies with Associated Type Value
- Versioned Hierarchy Structure Changes
- Advanced Searching and Filtering on Multiple Criteria
- Import/Export of Data and Hierarchy Structures
- Support Analytic Reporting
- Working with ETL Process
- Public APIs to allow additional customization
- Extension Framework to Support Custom Behaviors
- Support configurable hierarchy type with rules
- Provides framework for OLTP and OLAP data synchronization
- Provides flexible transaction management under the Spring framework
- Provides a default Hierarchy UI

Object Management Framework (OMF) Features and Services

OMF provides a set of generic interfaces and functions for you to handle business object creation, registration, lookup and resolving universal resource identification for any kind of business objects. It also provides a reusable catalog of business objects that are used by Hierarchy module. The generic OMF interfaces allows hierarchy to link, remove, find and search objects that might be linked into hierarchy in an uniform way without having to know the specific type of business objects.

The following set of business objects are provided as OOTB features of Billing Analytics:

- Billing Account
- Company
- Service Agreement
- Charge Type
- Service Charge Type
- Service Plan

2 Hierarchy Related Concepts

Basics

- **Hierarchy:** A system organized in the shape of a pyramid, with each row of objects called nodes, linked to objects directly beneath it. A hierarchy contains a root directory at the top of the pyramid and subdirectories below it.
- **Hierarchy Nodes:** Users are assigned to a node in the hierarchy to determine their access to information. Nodes are organized into parent/child relationships in the hierarchy.
- **Element/Business Object:** A generic term for elements represented within hierarchy structures. Business Objects include items such as folders, handsets, users, etc. The Hierarchy Module must be able to link many different types of Business Objects and object types. Each type of object will have different properties and actions associated with it. In order to be linked into a hierarchy, business object needs to implement certain interfaces. Some examples of business object include but not limited to:
 - **Billing Account:** An account from the billing system that generates an invoice or statements.
 - **Service Agreement/Contract:** Could be leaf node of the hierarchy (must have a parent node). A credit card, MTN or a MSISDN are examples of a service agreement or contract.

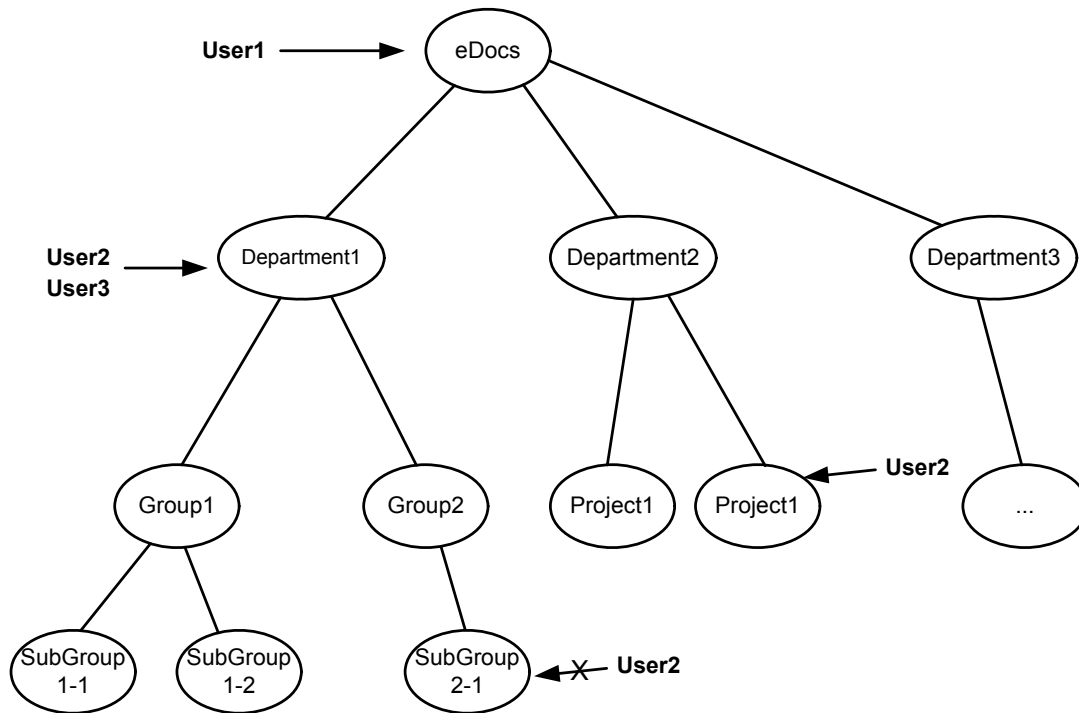
Business objects must implement the OMF object interface.

- **Link Target:** Business objects that are linked into one or more hierarchies. A given link target can only be linked once within one hierarchy, but can be linked to multiple different hierarchies at the same time. To link your business object through a hierarchy, you must implement the link target interface for your OMF object. OOTB, a user is a special element that is NOT a link target.
- **Group/Folder:** A specific link target maintained within the hierarchy that may have business objects as children, such as MSISDNs/MTN and Users along with additional folders.
- **Service Provider:** Company that provides services that a customer has signed contracts for. Verizon Wireless is an example of a telecommunications service provider.
- **Customer:** A Customer of the Service Provider. Example: Siebel is a customer of Verizon Wireless.
- **User:** An enrolled user of the system that has a unique login name, password, and individually assigned and managed permissions that are assigned to a node in the hierarchy
- **Role:** A role is a customer-defined set of default permissions that may be assigned to a user. The roles are permission sets that are customized by the client based on their business requirements. What permissions are mapped to these roles are outside the scope of hierarchy.

- **Permission:** Permission allows a user to view or take action on information they have access to, based on the nodes they are assigned to in the hierarchy. Permissions include view summary, view detail, pay, report, manage hierarchy, assign users, assign permissions, etc.

Hierarchy Roles Based Access Control

- **Hierarchy Based Access Control:** A user is assigned or associated with one of nodes inside hierarchy. Once it is assigned, the user has granted access to all the nodes under the assigned node, as shown in the following diagram:



A user may be assigned to one or multiple nodes in the same hierarchy as long as these nodes are not on the same path to the root node.

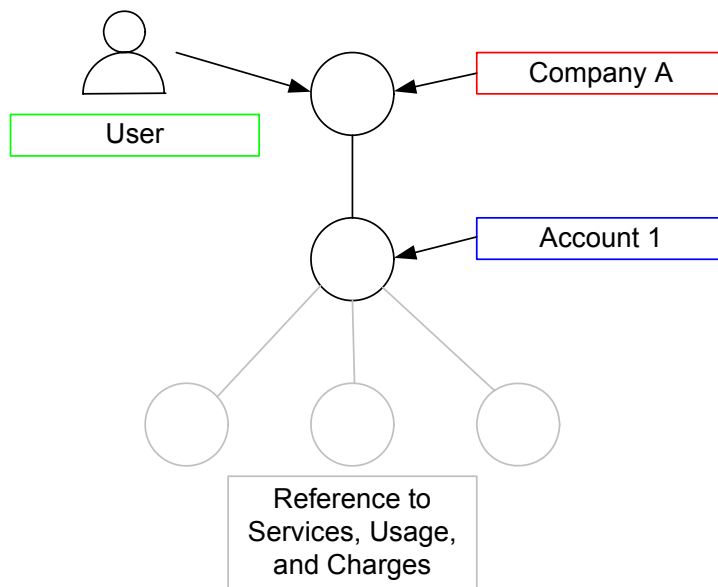
- **Hierarchy Role Based Access Control:** Hierarchy supports the following roles:
 - **Subscriber:** Users with this role can see assigned hierarchies, including all nodes from assigned node down. View assigned hierarchy only. No editing or assignment allowed. Users with this type of role need to be assigned or associated with a node in order to access the hierarchy.
 - **Manager:** User with this role can see assigned hierarchies, including all nodes from assigned node down. They can create/view/edit/delete hierarchies, and assign/un-assign other Managers/Subscribers to/from sub tree. Users with this type of role must be assigned or associated with a node in order to access the hierarchy.

- **System Administrator:** Users with this role can see hierarchies within their company, including all nodes in the hierarchy. They can create/view/edit and delete hierarchies and assign Managers/Subscribers in to three. Users with this type of role do not need to be assigned or associated with any hierarchy tree node in order to see the hierarchy.
- **Customer Service Representative (CSR):** Users with this role can see all hierarchies in the system across companies, starting from root node. They can create/delete hierarchies. OOTB, the hierarchy UI does not support this feature yet. However, the provided API allows you custom implementation of this role.

Hierarchy Type

Each hierarchy has a type associated with it. Multiple hierarchies can share the same type. System can have multiple hierarchy types at any given time. Hierarchy types are configurable during the deployment. As default, Billing Analytics provides types of hierarchy: Billing Hierarchy and Business Hierarchy.

- **Billing Hierarchy:** Based on the inherent structures contained within the invoice data stream. They are created automatically at billing data load time. The structure of the billing hierarchy cannot be modified by the end user. A simple billing hierarchy could look like the following:

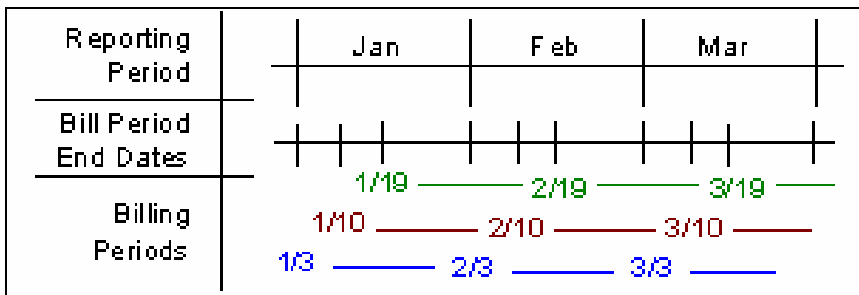


- **Business Hierarchy:** Also referred as organizational hierarchy. Business hierarchies are user defined structures that represent the business' groups and cost centers, they may contain service line usage and charges taken from multiple accounts.

Reporting Periods and Versioning

Reporting Periods

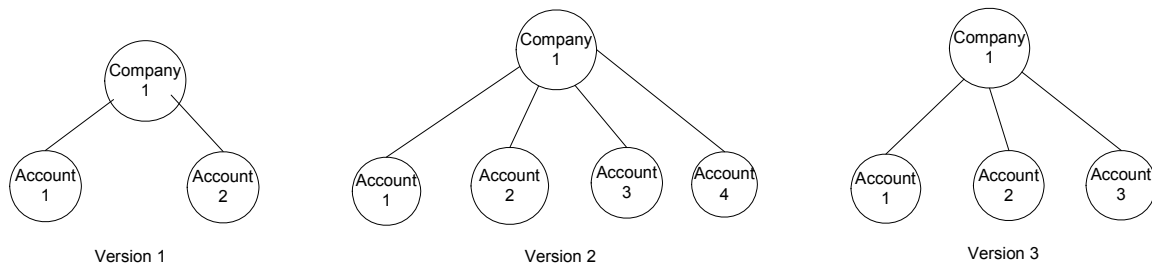
A reporting period defines a time range with a start and end date. The start date and end date do not have to match with billing periods defined by external billing system. Reporting period is the smallest time interval in the system for hierarchy versioning. In other words, changes made to hierarchies within a period will not be tracked or versioned. Only the changes made across different periods will be versioned.



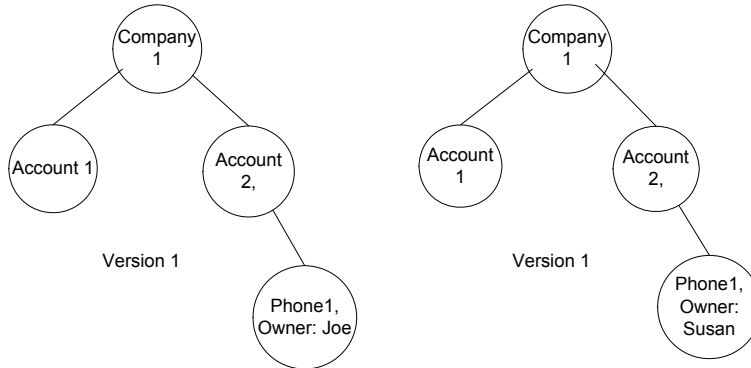
Hierarchy Versioning

Changes made to the hierarchy structure and to the relationship across different reporting periods are versioned. Changes made to user assignments and link target attributes are not versioned. Here are some examples:

Structure changes will be versioned



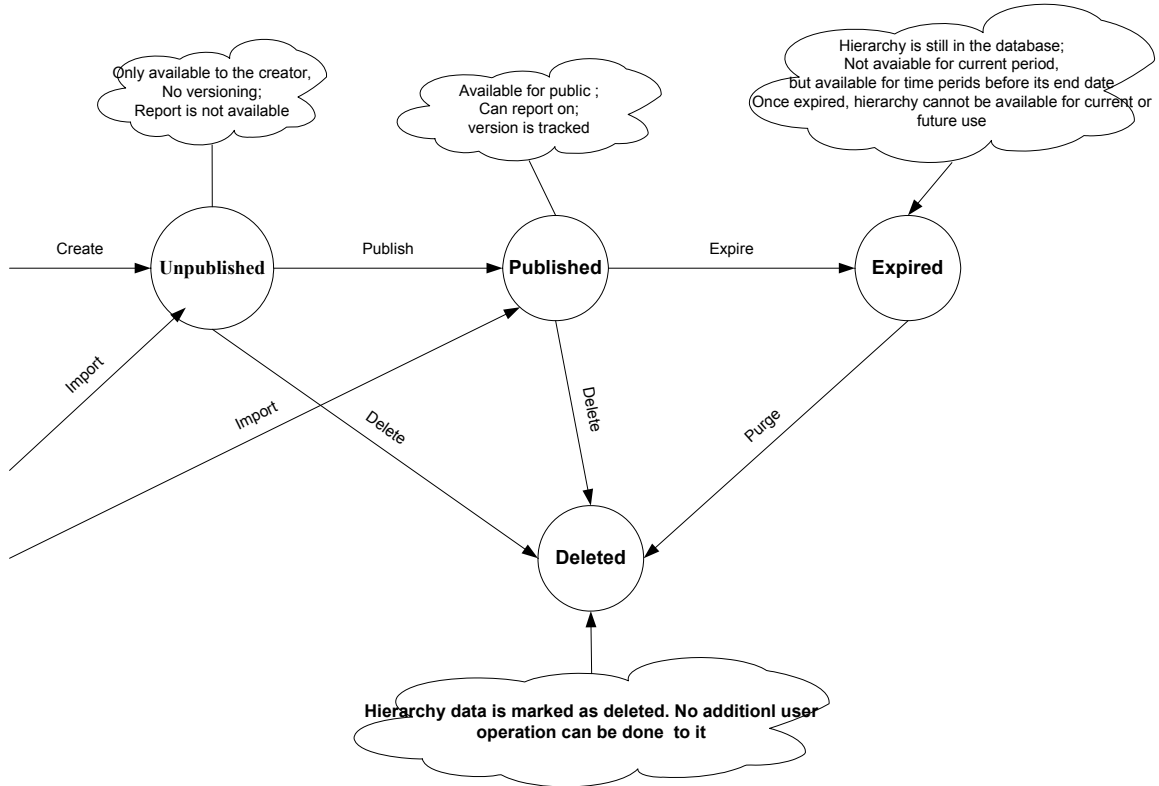
Attributes of link target will not be versioned. For example, the owner of Phone1 changes from "Joe" to "Susan", the information that Joe has owned the Phone1 in the past will be lost.



Hierarchy Life Cycle States

Once a hierarchy is created, it goes through several states before being completely removed from database. The following diagram illustrated the state transition of a hierarchy:

- **Unpublished:** When a hierarchy is first created it goes to unpublished state. Unpublished hierarchies can only be available or accessed by its creator and system administrators. Changes made to the hierarchy are not versioned. Reporting is not available on an unpublished hierarchy.
- **Published:** Once a hierarchy is published, structural changes made to the hierarchy will be versioned. A published hierarchy cannot be unpublished. A published hierarchy is available for public to use, and accessibility is controlled based on user's role and association with hierarchy. Reports can be run on published hierarchies.



- **Expired:** A published hierarchy can be expired by an administrator or a manager user. Once a hierarchy is expired from a given period, hierarchy information from the following period and onwards is removed. However, data for periods prior that expiring period, inclusive, is still available for editing and reporting.
- **Deleted:** A user can delete a hierarchy, which marks the hierarchy as deleted in the database. Deleted hierarchies are not available for the user to access for any periods. A housekeeping batch job should be run to physically remove all marked as deleted hierarchies from database.

Data Replication

Once a hierarchy is published, changes made to that hierarchy will be versioned across reporting periods. When it comes to a new period, a backend scheduled job will be run to replicate the latest hierarchy structure to the current period. Only the references to the relationship will be replicated for each period.

Assigned and Unassigned Objects

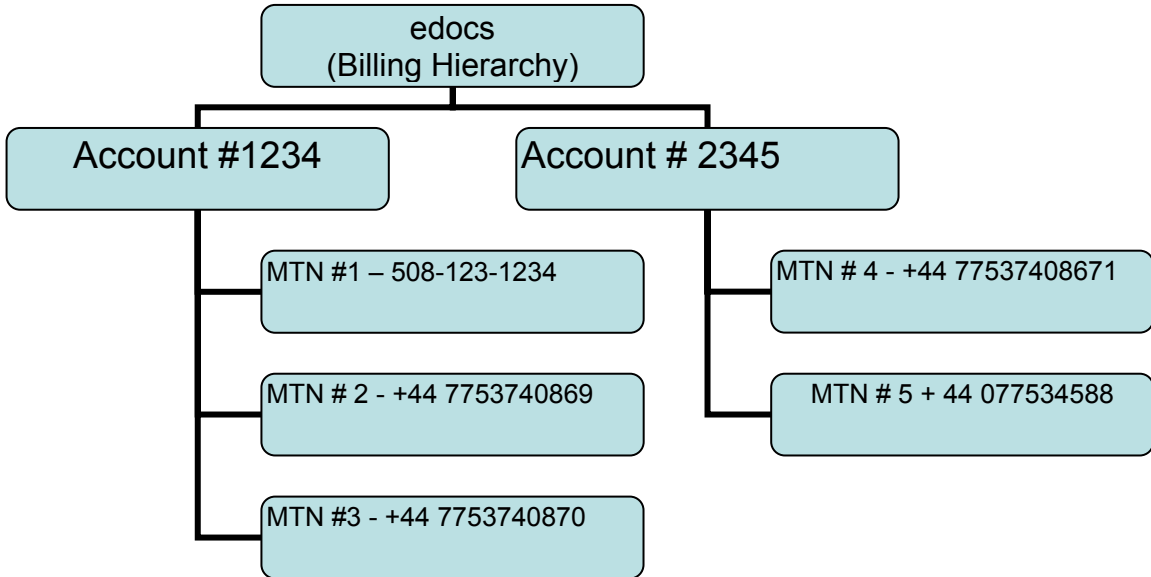
- **Assigned Link Targets/OMF Objects:** Link targets or business objects that have been linked into the current hierarchy or into the sub-tree from the current node down.
- **Un-Assigned Link Targets or OMF Objects:** Link targets or business objects that the current user has been granted access to, but that have not yet been linked into the current hierarchy, or into the sub-tree from the current node down.
- **Assigned Users:** Users of the current company, who have been associated at least once to a node from current node down, or current hierarchy.
- **Unassigned Users:** Users of the current company, who can be associated to the current node, or the root node of the hierarchy.
- **Authorized Users:** Users who have permission to access the current node, which includes users associated with the current node, and those with any ancestor nodes.
- **Unauthorized Users:** Users of the current company that do not have access to the current node.

You should consider the following points when using the hierarchy module in your application:

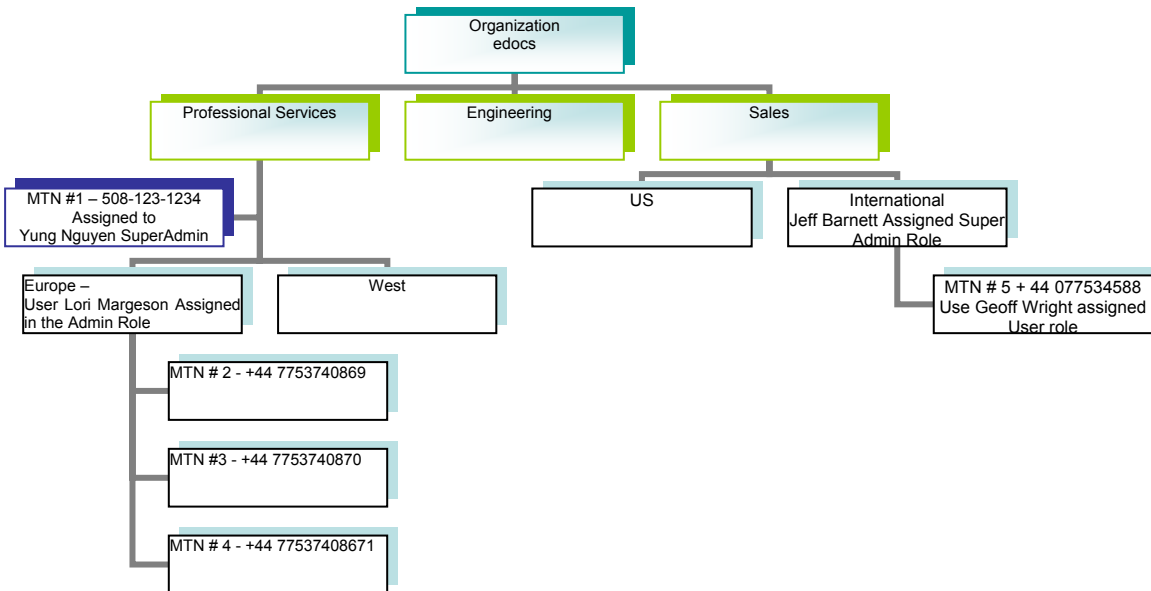
- Hierarchy's tree structure does not support multiple parents.
- Only one unique link target per hierarchy is supported.
- Hierarchy is unique by domain ID (or company ID) and name.
- Each company or domain can only have one billing hierarchy.

Hierarchy Example

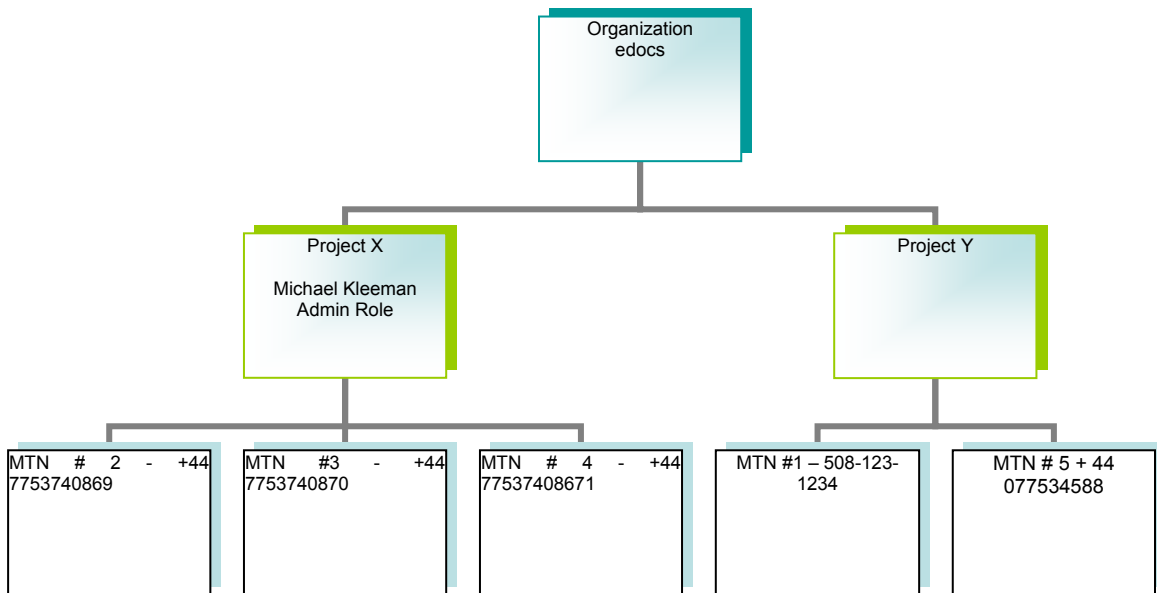
There are two types of hierarchy: the billing system and the organizational structure. Most billing systems cannot accurately model the business structure of an organization, because the two hierarchy structures are usually very different. The following diagram shows how a telecommunication company might model one of their customers in their billing system. In the example, there are two accounts, and each account has several mobile phones. Each account number has an invoice produced for it, and the accounts may be on different billing cycles. For example account # 1234 might be billed on the first of each month, and account # 2345 might be billed on the 15th of each month. The billing system might not know that these two accounts belong to the same company.



Most companies have more complex business structures than the preceding example, which they would like to map to the contracts which they have with the Service Provider. The next example shows how these two accounts might be modeled from a business structure point of view.



This business structure provides much better interaction between the service provider and customer in an online customer self service offering. In this example, both contracts and users are assigned to a business structure. However, in many organizations, one business structure is not sufficient. Month to month, organizations may require a project structure that is very different from the original business structure. The following example illustrates how the same billing hierarchy used in the first business structure can be arranged into a completely different business structure.



Each of these example business structures model how a customer might design a hierarchy to support the functions provided in a customer self service application. Payment, electronic bill viewing, Order Management, Service Management, and reporting are all common features of self service that will utilize the business structures. The business structures provide the scope for the other components to act on. User and business objects can be assigned to each of the business structures. It is important to realize that the hierarchy module itself only provides these business structures and captures user access control. The ability to use this structure to perform analytics, make payments, or other features are the responsibility of the other modules.

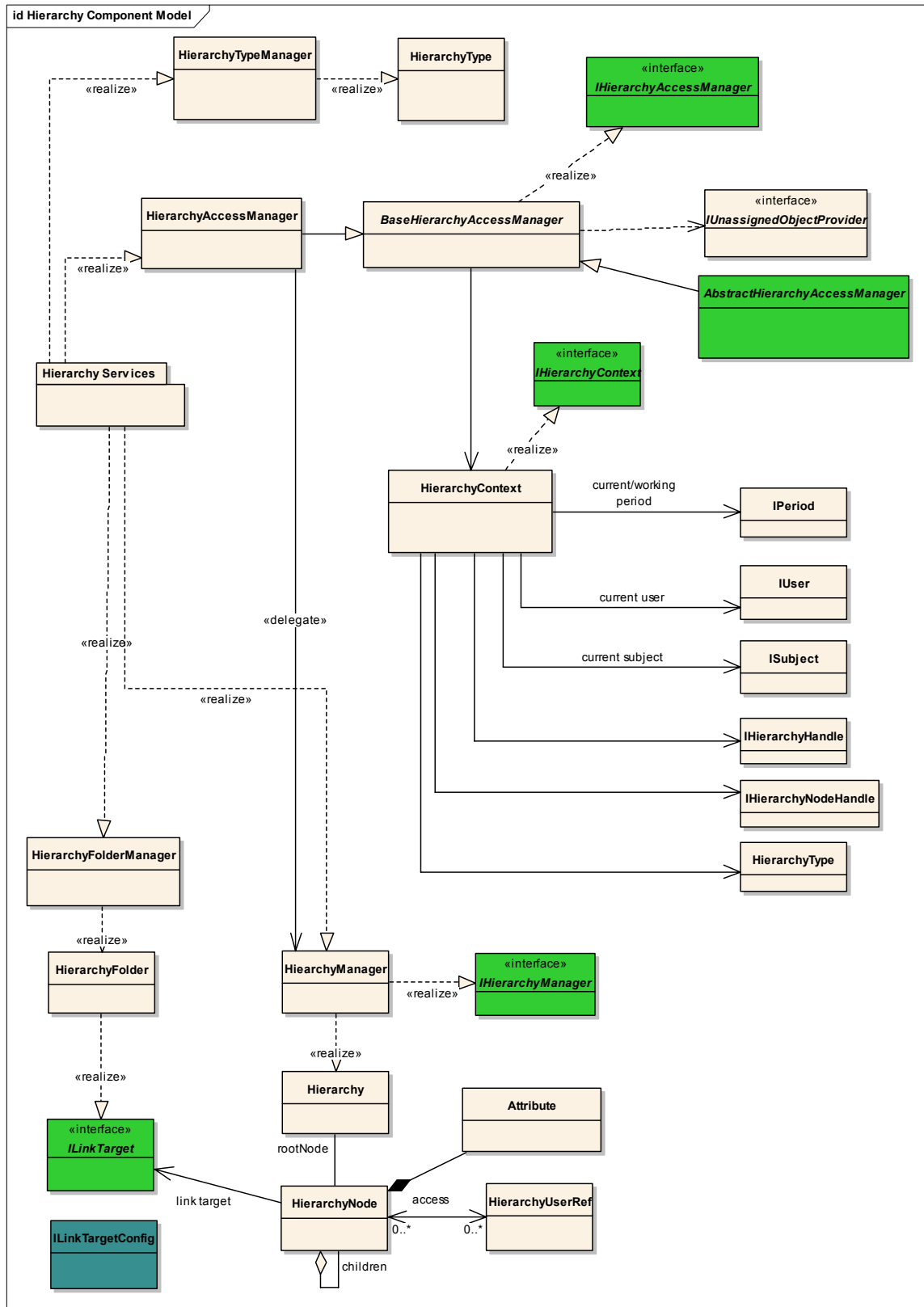
Additionally, the user management and role based access control features are components separate from hierarchy. The assignment and role of a user at a level of hierarchy is done via hierarchy, but management of the user and controlling access based on role is separate.

3

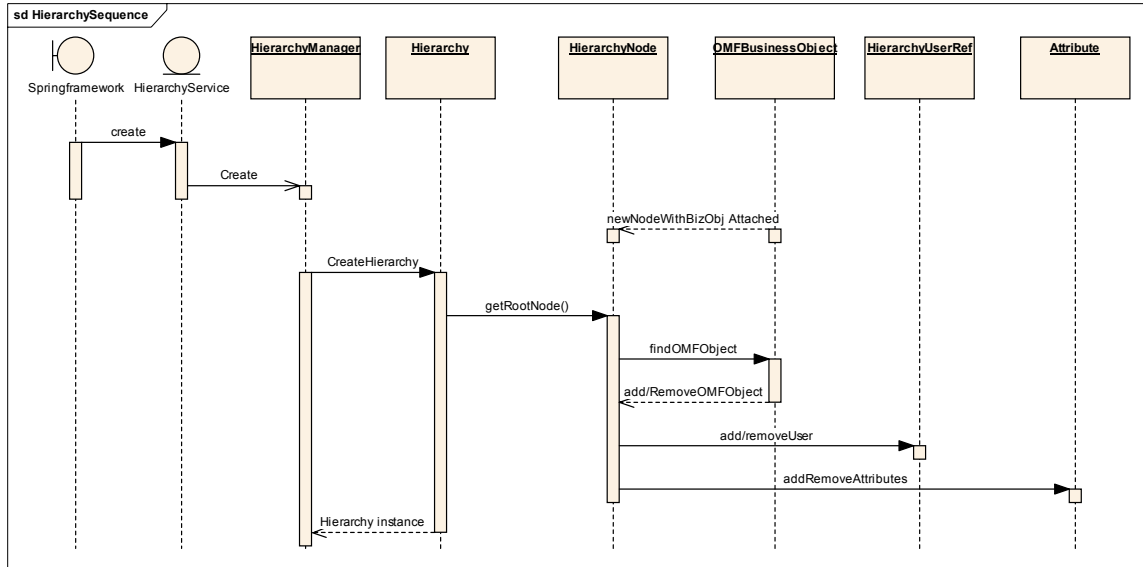
Hierarchy/OMF Architecture Overview

Hierarchy and OMF Components

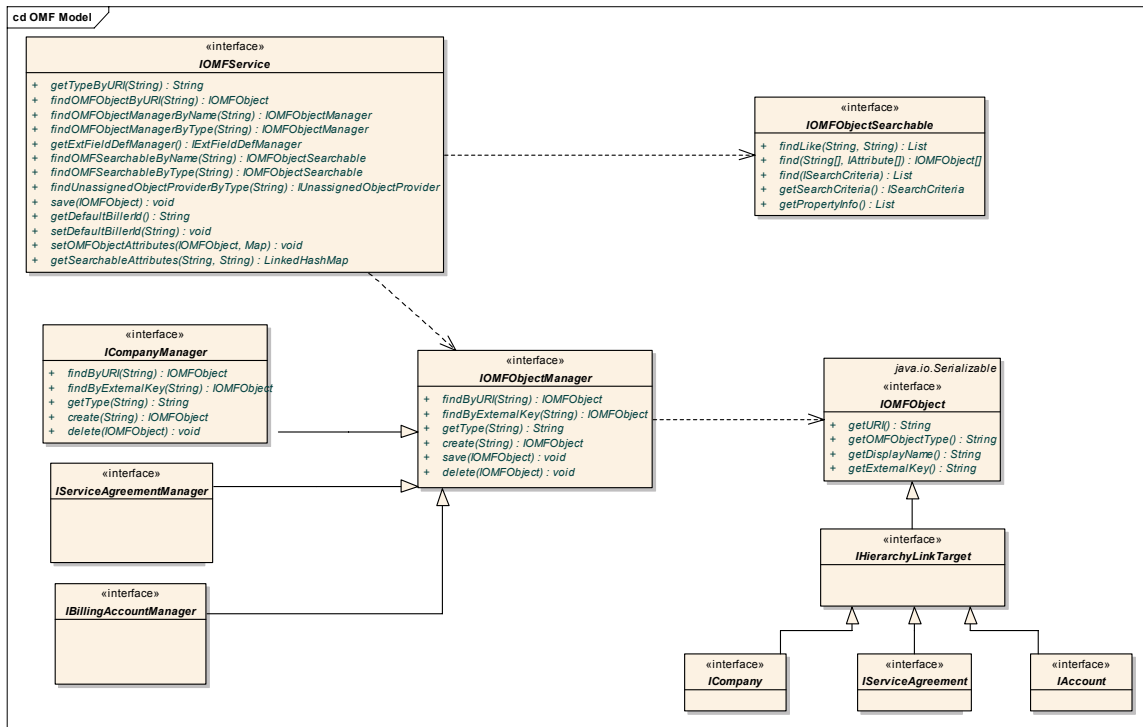
The hierarchy module provides a set of API and services to allow applications to model arbitrary hierarchical relationships among any kind of business objects. The core module design and its new extension framework make it well suited in tightly coupled systems or a federated database.



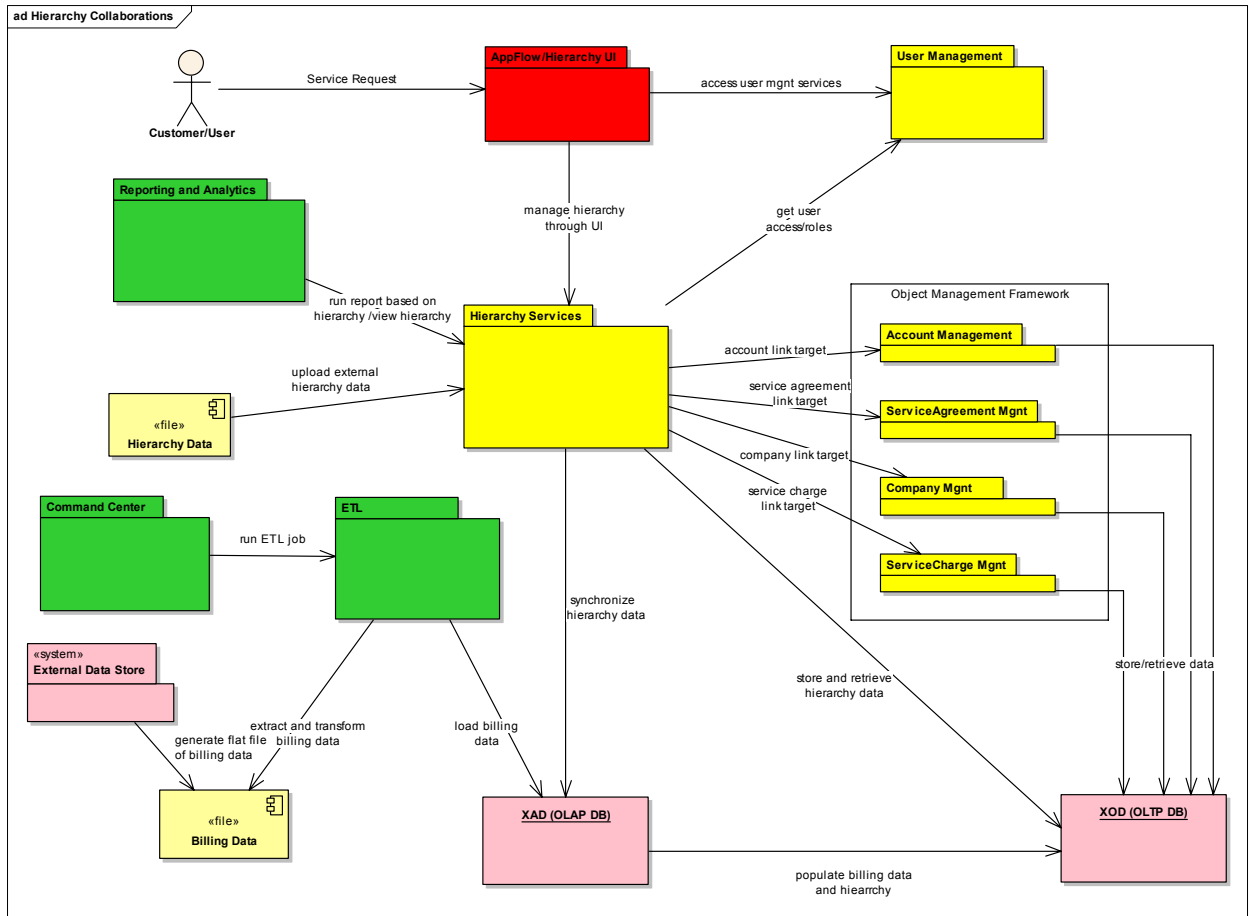
The following interaction diagram shows a typical activity flow within the hierarchy module.



The following diagram shows the key interfaces from OMF component.



System Collaborations

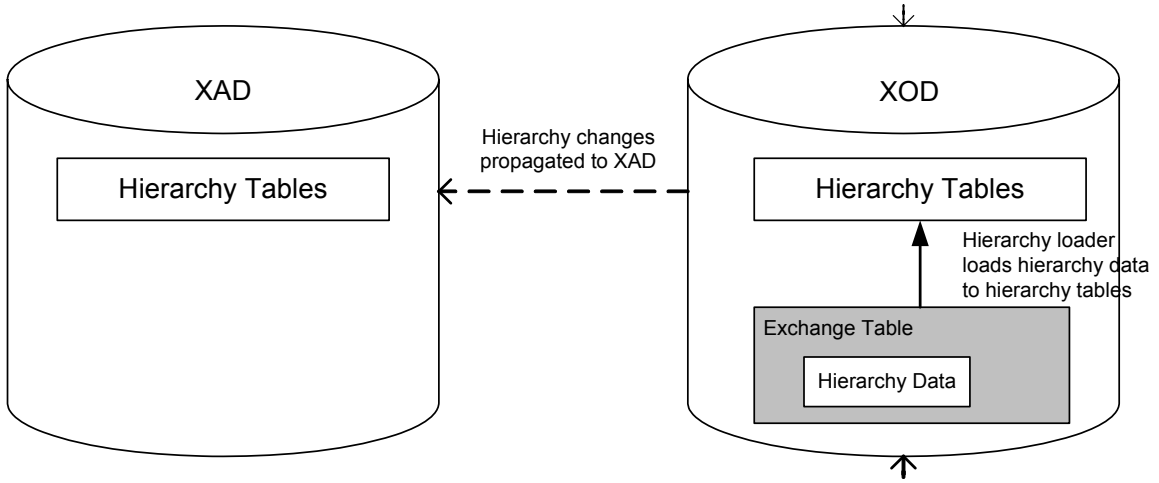


Working with Reporting

In order to run report based on hierarchy structures, changes made to hierarchies must be synchronized over to OLAP. In addition to hierarchy tree structure, each OMF object linked into hierarchy must be synchronized as well. Each OMF object has implemented an OLAP handler to handle their specific synchronization logic for being linked into a hierarchy.

Changes made to the OLTP hierarchies are categorized into different types of events. An event handler is called directly by the hierarchy API code to process the corresponding event. As a result, the same changes are then be propagated into the OLAP side of the hierarchy tables. The changes made in OLTP and OLAP are bounded into a single transaction to guarantee the data integrity between the OLTP and OLAP databases. Distributed database transaction management is used to achieve this.

For each OMF object that needs to be linked into hierarchy, you must implement the `ILinkTargetEventHandler` interface defined in the `com.edocs.common.api.hierarchy.connector` package.



4 Basic System Use Cases

This chapter describes some basic developer use cases that are useful when deploying Billing Analytics applications.

Configure Hierarchy Types

The Hierarchy module supports configurable hierarchy types. Developers can change the properties of OOTB hierarchy types, and add additional hierarchy types when necessary. Each hierarchy type can have name, code, description and allowable link targets for that type. Code in a hierarchy type uniquely identifies the type object. The `IHierarchyTypeManager` interface can be used to find all hierarchy types and the valid link targets for each type.

The following aspects of hierarchy type are configurable:

- Number of hierarchy types supported by the system
- Name of each hierarchy type
- Valid link target types each hierarchy type allows
- Valid children types for each link target allowed in a hierarchy type
- Some special behavior for hierarchy importer and exporter

By default, the hierarchy module supports two types of hierarchy: Billing and Business. The default type configuration is shown in the file:

Siebel\CBA\estatement\xma\config\modules\hierarchy\ Hierarchy.cfg.xma.xml

```
<bean id="HierarchyConfig" class="com.edocs.common.hierarchy.core.HierarchyConfig" singleton="true">
  <property name="types">
    <list>
      <ref bean="BusinessHierarchyType"/>
      <ref bean="BillingHierarchyType"/>
    </list>
  </property>
</bean>
```

The preceding bean entries define the two hierarchy types that will be supported: "BusinessHierarchyType" and "BillingHierarchyType". To add new additional types, simply add a new bean into the "types" list. Let's look at the hierarchy type bean next.

The BusinessHierarchyType is configured in the following example:

```

1      <bean id="BusinessHierarchyType" class="com.edocs.common.hierarchy.core.HierarchyTypeConfig">
2      <property name="name"><value>hierarchy.type.Business</value></property>
3      <property name="code"><value>BUSINESS</value></property>
4      <property name="description"><value>Business hierarchy</value></property>
5      <property name="validLinkTargets">
6      <list>
7      <ref bean="HierarchyFolderConfig"/>
8      <ref bean="CompanyConfig"/>
9      <ref bean="ServiceAgreementConfig"/>
10     <ref bean="ServiceChargeConfig"/>
11     </list>
12 </property>
13 <property name="linkTargetRelationship">
14 <list>
15 <!-- if any of the link target doesn't defined below, then by default it can have any type of valid -->
16 <!-- link target defined above as it children -->
17 <!-- Service agreement is a leaf node -->
18 <bean class="com.edocs.common.hierarchy.core.LinkTargetRelationship">
19 <property name="parent"><ref bean="ServiceAgreementConfig"/></property>
20 </bean>
21 <!-- Company can have folder, and account as it children -->
22 <bean class="com.edocs.common.hierarchy.core.LinkTargetRelationship">
23 <property name="parent"><ref bean="CompanyConfig"/></property>
24 <property name="validChildren">
25 <list><ref bean="HierarchyFolderConfig"/></list>
26 </property>
27 </bean>
28 </list>
29 </property>
30 <property name="hierarchyExchangeConfiguration">
31 <bean class="com.edocs.common.hierarchy.connector.exchange.HierarchyExchangeConfiguration">
32 <property name="updateHierarchyIdentity"><value>FALSE</value></property>
33 <property name="ignoreUserAccess"><value>FALSE</value></property>
34 </bean>
35 </property>
36 </bean>

```

Notes about example:

- “name” -- Specifies the resource bundle key for displaying the name of this hierarchy type.
- “code” -- Unique string value to identify business hierarchy type. We recommend that you do not change this value.
- “validLinkTarget” – A list of valid link target object can be linked into this type of hierarchy. For more information, see `HierarchyFolderConfig` and `CompanyConfig`.
- “linkTargetRelationship” – Specifies the rule by which a link target can be linked in as a child node. The list includes all link targets that might have restricted their list of link targets when they linked into hierarchies.

Indicates that for Service Agreement objects in a business type of hierarchy, no valid link target can be linked in as its child. In other words, Service Agreement should be the leaf node of business hierarchy.

Indicates that in Business Hierarchy, Company objects can have folders as its child node.

- “hierarchyExchangeConfiguration”— Specifies certain override-able behaviors when dealing with hierarchy import and export.

The following file specifies the default Billing Hierarchy type:

```

<bean id="BillingHierarchyType" class="com.edocs.common.hierarchy.core.HierarchyTypeConfig">
  <property name="name"><value>hierarchy.type.Billing</value></property>
  <property name="code"><value>BILLING</value></property>
  <property name="description"><value>Billing hierarchy</value></property>
  <property name="validLinkTargets">
    <list>
      <ref bean="HierarchyFolderConfig"/>
      <ref bean="AccountConfig"/>
      <ref bean="CompanyConfig"/>
      <ref bean="ServiceAgreementConfig"/>
    </list>
  </property>
  <property name="linkTargetRelationship">
    <list>
      <!-- if any of the link target doesn't defined below, then by default it can have any type of valid -->
      <!-- link target defined above as it children -->
      <!-- Account can only have service as it children -->
      <bean class="com.edocs.common.hierarchy.core.LinkTargetRelationship">
        <property name="parent"><ref bean="AccountConfig"/></property>
        <property name="validChildren">
          <list>
            <ref bean="ServiceAgreementConfig"/>
          </list>
        </property>
      </bean>
      <!-- Service agreement is a leaf node -->
      <bean class="com.edocs.common.hierarchy.core.LinkTargetRelationship">
        <property name="parent"><ref bean="ServiceAgreementConfig"/></property>
      </bean>
      <!-- Company can have folder, and account as it children -->
      <bean class="com.edocs.common.hierarchy.core.LinkTargetRelationship">
        <property name="parent"><ref bean="CompanyConfig"/></property>
        <property name="validChildren">
          <list>
            <ref bean="HierarchyFolderConfig"/>
            <ref bean="AccountConfig"/>
          </list>
        </property>
      </bean>
    </list>
  </property>
  <property name="hierarchyExchangeConfiguration">
    <bean class="com.edocs.common.hierarchy.connector.exchange.HierarchyExchangeConfiguration">
      <property name="updateHierarchyIdentity"><value>FALSE</value></property>
      <property name="ignoreUserAccess"><value>FALSE</value></property>
    </bean>
  </property>
</bean>

```

Create and Modify Hierarchies using APIs

This section describes how to use APIs to create and manage hierarchies.

To create a new hierarchy

```

LookupService lookup = LookupServiceFactory.getInstance();
IHierarchyService hierarchyServices =
  (IHierarchyService)lookup.getModule("hierarchy");
IHierarchyManager hm = hierarchyServices.createHierarchyManager(hierarchy_creator);
IHierarchy hierarchy =
  hm.createHierarchy(companyid, hierarchyname, hierarchy_type);
...

```

To create hierarchy from an existing hierarchy

```
Hm.createHierarchyFromNode(companyIdStr, hierarchyNameStr,
                           hierarchyType,
                           fromNode, preserveUserAccess);
```

Adding or Removing Entities To or From a Hierarchy

Any OMF Business Objects that implements IHierarchyLinkTarget can be added into Hierarchy:

```
public interface IOMFObject {
    public String getDisplayName();
    public String getExternalKey();
    public String getOMFObjectType();
    public String getURI();
}

public interface IHierarchyLinkTarget extends IOMFObject {
    public String getLinkTargetId();
    public String getLinkTargetName();
    public boolean isEditable();
    public boolean isEditable();
}
```

To add a link target to Hierarchy

```
IServiceAgreementManager samgr =
IOMFService.findOMFObjectManagerByName("edx:omf:serviceAgreement");
IHierarchyLinkTarget linkSrv =
(IHierarchyLinkTarget)samgr.find("5088002000", "ACCT001");
IHierarchyNode hNode = hierarchy.getRoot();
IHierarchyNode srvNode = hNode.addLinkTarget(linkSrv);
```

To add a folder to Hierarchy

```
IHierarchyFolderManager fMgr = hierarchyService.createHierarchyFolderManager();
IHierarchyFolder hFolder = fMgr.create("HR", "Human Resource", "This is HR folder");
hFolder.addAttribute(hierarchyService.createAttribute("Phone: ", "508-123-8700"));
IHierarchyNode fNode = rootNode.addLinkTarget(hFolder);
}
```

To remove a business object from hierarchy

```
srvNode.remove();
```

To move a business object from one node to another parent node

```
IHierarchyNode newParentNode = ...
srvNode.move(newParentNode);
```

Add, and Remove Users for Hierarchy Access Control (HBAC)

To add/remove user access

```
IHierarchyNode.addUserAccess(String userId)
IHierarchy.addUserAccess(IOMFObject linkTarget, String userId)
IHierarchyNode.removeUserAccess(String userId)
```

To retrieve authorized users

```
IHierarchyNode.getUsers()
IHierarchyNode.getAllAuthorizedUsers()
```

Search & Filter Hierarchies

The hierarchy module service provides a search API, which supports finding hierarchies and entries within a hierarchy or on single or multiple attributes of link targets.

Search and Filter Hierarchies Using IHierarchyManager

■ **Retrieve Hierarchies for a given user, and by hierarchy type using**

```
IHierarchy[] getHierarchies(IUser user)
IHierarchy[] getHierarchies(IUser user, IHierarchyType hierType)
IHierarchy[] getHierarchiesForUser(String userId)
IHierarchy[] getHierarchiesForUser(String userId, IHierarchyType type)
```

■ **Retrieve Hierarchies by company, and by hierarchy type**

```
IHierarchy[] getHierarchiesForDomain(String domainId)
IHierarchy[] getHierarchiesForDomain(String domainId,
                                     HierarchyType hierType)
```

■ **Locate a Hierarchy**

```
IHierarchy findHierarchy(String hName, String domainId)
```

■ **Find Containing Hierarchy for an Object**

```
IHierarchy[] findHierarchyForLinkTargetURI(IHierarchyType type, String linkTargetURI)
```

Search Nodes within Hierarchy

You can search for hierarchy nodes across all hierarchies to which you have access, within a single hierarchy, or within a sub-tree of a hierarchy. Based on the type of search, you can use methods defined in `IHierarchyManger`, `IHierarchy`, or `IHierarchyNode` to conduct your search.

■ Find Root Node for User within a Hierarchy

```
IHierarchyNode[] IHierarchy.findRootNodeForUser(IUser)
IHierarchyNode[] IHierarchy.findRootNodeForUser(String uid)
```

■ Find Node within a Hierarchy, or a Section of Hierarchy

■ By Link Target URI

```
IHierarchyNode[] findNodeByLinkTargetURI(String linkTargetURI)
IHierarchyNode[] findNodeByLinkTargetURI(String URI, IPeriod period)
```

■ By Link Target Type

```
findNodeByLinkTargetType(String linkTargetType)
```

■ By Link Target Id

```
findNodeByLinkTargetId(string linkTargetId)
```

Navigate Hierarchy

■ Get Root Node for Hierarchy

```
IHierarchy.getRoot()
```

■ For a Node, get Its Parent, or Children

```
IHierarchyNode.getParent()
or
IHierarchyNode.getChildren()
```

Search on Link Target Attributes Within Hierarchy

To search on an attribute of a link target within hierarchy, search capability must be configured in the XML configuration files, and the link target object must implement `IOMFSearchable` interface. Once the attributes are set for search, you can use the following methods to find objects based on their attribute values.

```
IServiceAgreementManager saManager =
IOMFService.getOMFManagerByName(ServiceAgreementManager.getClassName());
ISearchCriteria criteria = saManager.getSearchCriteria();
criteria.add(criteria.equals("subscriberName", "John Willson")).add(
criteria.isNull("extAttr1"));
IHierarchyNode foundNodes =
rootNode.findNodeByLinkTargetTypeAndCriteria(saManager.getType(), criteria);
```

Taking the User's Role into Consideration

In addition to the APIs that provide a variety of methods that retrieve information based on the system requirements, the Hierarchy module also provides a set of class that takes the user's role and permissions into consideration.

Create Hierarchies

```

IHierarchyContext hierarchyContext = new HierarchyContext();
//hierarchyContext can be set through web appflow layer

hierarchyContext.setUser(user);
hierarchyContext.setSubject(subject);
...
LookupService lookup = LookupServiceFactory.getInstance();
IHierarchyService hierarchyServices =
(IHierarchyService)lookup.getModule("hierarchy");
IHierarchyAccessManager haMgr =
hierarchyServices.createHierarchyAccessManager(hierarchyContext);
IHierarchy hierarchy =
    haMgr.createHierarchy (hierarchyname, hierarchy_type, hierarchyDesc);
...

```

Use `IHierarchyAccessManager` to create a hierarchy, based on the current user or subject information stored in the `HierarchyContext`. Note that the method acts differently depending on the user role:

- **User is a Subscriber:** `HierarchyAccessException` is thrown, because the subscriber is not allowed to create hierarchy.
- **User is a Manager:** A new hierarchy is created with a domain id of the user's company id, and the user is assigned to the root node of the tree.
- **User is a System Administrator:** A new hierarchy is created with a domain id of the user's company id.

Find a List of Hierarchies that a Given User Has Access To

The following code retrieves a list of hierarchies based on the user type:

```

IHierarchy[ ] hierarchies = haMgr.getHierarchies(hierarchyType);

```

Depending on the user role set in the current context, the method returns the following if the user is:

- **Subscriber or Manager:** A list of hierarchies that the subscriber has been given access explicitly to some nodes in the tree.
- **System Administrator:** All hierarchies that belong to the company of the user.

Find Top Level Nodes that a User Has Access To for a Given Hierarchy

The following code retrieves a list of the top level nodes for the specified user type:

```
IHierarchyNode[] rootNodes = haMgr.getRootNodes(hierarchy, period);
or
IHierarchyNode[] rootNodes = haMgr.getRootNodes();
```

Since hierarchy structure is versioned for each period, the root node may vary depending on which period you are currently looking at. When the hierarchy and period are not passed in as parameters, the current hierarchy and current period stored in `hierarchyContext` are used. Again, depending on the user role set in the current context, the method returns the following:

- **Subscriber or Manager:** One or more hierarchy nodes that the current user has been given access explicitly. For example, the nodes that the current user is assigned to.
- **System Administrator:** The root node of the hierarchy.

Find Assigned Link Targets or OMF Objects

```
IHierarchyNodeObjWrapper[] linkTargetNodes =
haMgr.findAssignedLinkTargets(hierarchy, linkTargetType, searchCriteria);

or

IHierarchyNodeObjWrapper[] linkTargetNodes =
haMgr.findAssignedLinkTargets(hierNode, linkTargetType, searchCriteria);
```

This method finds link targets of a given type within the hierarchy. The `searchCriteria` is used as a filter to apply on link target objects themselves. The returned `IHierarchyNodeObjWrapper` is a wrapper class, which contains the handle to the `HierarchyNode` object and the link target object itself. The returned list returns information about all the objects you need, as well as the corresponding hierarchy node that the object is linked with.

The search takes user's role into consideration. In other words, it only returns objects that qualify the search criteria and are accessible to the current user.

```
IOMFObject[] omfObjects =
haMgr.findAssignedOMFs(hierarchy, omfType, searchCriteria);

or

IOMFObject[] omfObjects =
haMgr.findAssignedOMFs(hierNode, omfType, searchCriteria);
```

This method finds OMF objects of given type within the hierarchy. The `searchCriteria` is used as a filter to apply to the OMF objects. It returns a list of business objects without node information. The search takes user's role into consideration as well, and returns objects that qualify the search criteria and are accessible to the current user.

Find Unassigned Link Targets or OMF Objects

```
IOMFObject[] unassignedOMF =
    findUnassignedOMFObjects(hierarchy, omfType, searchCriteria);
```

This method returns a list of OMF objects that: qualify the search criteria, the current user has access to, and have not been linked into the current hierarchy yet.

To determine the list of objects that the current user has access to typically involves interaction with other hierarchies, or even to another data rule set by the system to determine what objects are available for current user to access.

As an OOTB Billing Analytics offering, Billing hierarchy is used as a master access control hierarchy. When searching for an unassigned account or unassigned service agreement, Hierarchy can return a list of accounts or service agreements that are granted access to the current user but have not been assigned to the current hierarchy.

The `IUnassignedObjectProvider` interface can implement specific rules for retrieve unassigned objects based on the application business logic. Sometimes, it could be necessary to talk to an external system for the access information.

Managing Business Objects

The Object Management Framework has adopted a set of patterns for object creation, modification, and retrieval. Within the catalog of business objects provided by OOTB, each type of business object has its own Manager class, Object class, and DAO (Data Access Object) class. The manager class deals with a business object's lifecycle, the object class is the data object and may contain business logic, and DAO class deals with database persistence.

As the standard product offering from OMF, the following business objects and their services are provided: Billing Account, Company, Service Agreement, Charge Type, Service Charge Type, and Service Plan.

When using the OMF framework, the following terms are heavily used:

- **Business Object Type:** A string uniquely identifies the type of business object in the system.
- **Business Object URI:** A string uniquely identifies the business object instance in the system. The URI consists of the type and the id of that object within the type.

Exchange Hierarchy Data using XML

The Hierarchy module supports both XML import and XML export. Since objects linked into a hierarchy can be of any type, it is important for the Importer and Exporter to support flexible XML formatting. This allows you to specify information that you would like to get processed through the XML importer and exporter. Both the XML importer and exporter share the same XML format.

XML Schema

The hierarchy importer is flexible enough to import hierarchy relationships as well as link target content. The application developer can decide whether the importer and exporter will deal with only relationships or will also deal with link target content.

To meet these requirements, the hierarchy module provides two XML schema definition files:

- `Common-hierarchy-interchange-1.0.xsd` specifies the format that is core to the hierarchy module. We suggest that application developers use the file as is. The schema found can be in the appendix of this document. The actual file is in the installed directory of `C:\Siebel\CBA\estatement\config\xml`.
- `Instance-hierarchy-interchange-1.0.xsd` is used to customize the hierarchy structure. Application developers can add one or more sections that describe application-specific business objects (link targets) in the import or export xml file.

`Instance-hierarchy-interchange-1.0.xsd` is referenced by `common-hierarchy-interchange-1.0.xsd` schema file. So developers only need reference `common-hierarchy-interchange-1.0.xsd` in their XML import or export files:

Both the XML importer and exporter share the same xml schema.

Importing Hierarchies

To import the hierarchy XML, you can use the following method through `IHierarchyAccessManager`:

```
HierarchyExchangeResult importXML(hierarchyInputStream, startPeriod, endPeriod);
```

A hierarchy XML file can contain one or multiple hierarchy structures. The XML file only specifies the structure relationship of the hierarchy and user association, but does not specify any time period information. The period information is provided as additional parameters passed in from higher layer application code. When importing for multiple periods, the hierarchy structure will be replicated for each period. If the hierarchy has changes between different periods, it either must be imported separately (one for each period), or it must be modified for the period once they are imported.

You can import a hierarchy either in a published or unpublished state. If the hierarchy already exists in a published state, then any subsequent imported structures must be imported in a published state.

When calling the preceding method where `startPeriod` and `endPeriod` are provided, the hierarchy will be imported and published for the periods within the specified range. If the periods are null, the hierarchy will be in an unpublished state.

If a hierarchy exists for a given period, the hierarchy structure will be overwritten. If the hierarchy does not exist, the hierarchy structure will be created.

Exporting Hierarchies

To Export a hierarchy to an XML file, use the following method of `IHierarchyAccessManager`:

```
InputStream exportXMLAsInputStream(hierarchy, newHierarchyNameStr);
```

When exporting a hierarchy, the period specified in the `HierarchyContext` is used to get the correct version of the hierarchy to export. Also, based on the current user's role, only the content that is available to view for the current user is exported, which may not be the complete hierarchy stored in the database.

5

Advanced System Use Cases – Extending System

This section discusses various extension frameworks and shows how to extend the current OOTB capabilities for client solutions.

Creating New Type of Business Object to Work with Hierarchy

Hierarchy modules allow any kind of business object to be linked through hierarchy, as long as that business object implements a set of interfaces, and registers with hierarchy properly. In addition to the set of business objects provided with the Billing Analytics product OOTB, developers can extend the system by creating their own business objects and linking the new types of business objects into hierarchy.

This section describes the steps on how to create a new business object and make it work with the hierarchy module.

Create new business objects

To create a new type of business object, write your business object java classes and manage your business logic and persistent store, potentially using hibernate. We suggest you have a manager class, business object class and Data Access Object class for your new type of business object.

- 1 Create a new (or extend an existing) business object by implementing `IOMFObject.java`.
- 2 Create a new object manager by implement `IOMFObjectManager.java`.
- 3 Configure hibernate-mappings for your new business objects.

Register Objects With OMF module

In order to use the OMF service, you must register your business objects with the OMF framework. To do so, add another `OMFObjectManagerConfig` bean into the following section of the `omf.xma.xml` file, located in `xma/config/omf` directory:

Section for registering your business object manager by adding

```

<bean id="OMFService" class="com.edocs.common.omf.OMFService" singleton="true">
  <property name="registeredOMFangers">
    <list>
      <bean class="com.edocs.common.omf.OMFObjectManagerConfig">
        <property
name="name"><value>chargeTypeManager</value></property>
name="type"><value>edx:omf:chargeType:</value></property>
        <property name="implementation">
          <bean
class="com.edocs.common.omf.chargeType.ChargeTypeManager" singleton="true"/>
        </property>
      </bean>
      <bean class="com.edocs.common.omf.OMFObjectManagerConfig">
        <property
name="name"><value>companyManager</value></property>
name="type"><value>edx:omf:company:</value></property>
        <property name="implementation"><ref
bean="companyManager"/></property>
      </bean>
      <bean class="com.edocs.common.omf.OMFObjectManagerConfig">
        <property
name="name"><value>accountManager</value></property>
name="type"><value>edx:amf:account:</value></property>
        <property name="implementation">
          <bean
class="com.edocs.domain.telco.amf.defaultimpl.BillingAccountManager"
singleton="true"/>
        </property>
      </bean>
      <bean class="com.edocs.common.omf.OMFObjectManagerConfig">
        <property
name="name"><value>serviceAgreementManager</value></property>
name="type"><value>edx:omf:serviceagreement:</value></property>
        <property name="implementation">
          <bean
class="com.edocs.common.omf.serviceagreement.ServiceAgreementManager"
singleton="true"/>
        </property>
      </bean>
      <bean class="com.edocs.common.omf.OMFObjectManagerConfig">
        <property
name="name"><value>serviceChargeManager</value></property>
name="type"><value>edx:omf:servicecharge:</value></property>
        <property name="implementation">
          <bean
class="com.edocs.common.omf.servicecharge.ServiceChargeManager"
singleton="true"/>
        </property>
      </bean>
    </list>
  </property>
  <property name="defaultBillerId"><value>1</value></property>
</bean>

```

where

- "name" – Specifies the name of the manager, which can be any string you choose.
- "type" – A unique string that identifies the type of business object in the system. This value must be the same as the one returned from `IOMFObject.getOMFObjectType()` and `IOMFObjectManager.getType()`.
- "implementation" – The full class name of your object manager class. Typically a singleton class.

Once you register your business object with OMFService, you can do something similar for your business objects:

To find your business object manager class

```
IServiceAgreementManager samgr =
IOMFService.findOMFObjectManagerByType("edx:omf:serviceAgreement:");
```

Or

```
IServiceAgreementManager samgr =
IOMFService.findOMFObjectManagerByName(serviceAgreementManager);
```

To find the corresponding business object instance for given object URI

```
IServiceCharge sc = IOMFService.findOMFObjectByURI(scURIStrng);
```

Making Sure That Your Business Object Is Transactional Aware

To ensure that your OMF object supports flexible transaction management, each object's DAO layer is wrapped in either the Spring framework's transaction proxy, or it is managed through the spring hibernate transaction interceptor. Here are some examples:

```
<bean id="ServiceAgreementDimDaoTarget"
class="com.edocs.common.omf.serviceagreement.olap.ServiceAgreementDimDao"
singleton="true">
<property name="sessionFactory">
<ref bean="OLAPSessionFactory"/></property>
</bean>
<bean id="ServiceAgreementDaoTarget"
class="com.edocs.common.omf.serviceagreement.ServiceAgreementDao"
singleton="true">
<property name="sessionFactory"><ref
bean="OMFSessionFactory"/></property>
<property name="enableOLAPSync"><value>true</value></property>
<property name="serviceDimDao"><ref
bean="ServiceAgreementDimDaoTarget"/>
</property>
</bean>
<bean id="ServiceAgreementDao"
class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean"
singleton="true">
<property name="proxyTargetClass"><value>true</value></property>
<property name="transactionManager"><ref
bean="TransactionManager"/></property>
<property name="target"><ref
bean="ServiceAgreementDaoTarget"/></property>
<property name="transactionAttributes">
<props>
<prop
key="find*">PROPAGATION_REQUIRED,readOnly</prop>
<prop key="get*">PROPAGATION_REQUIRED,readOnly</prop>
<prop key="*">PROPAGATION_REQUIRED,-
HierarchyException,-DataStoreException</prop>
</props>
</property>
</bean>
```

Where OMFSessionFactory and TransactionManager beans are defined in omf.xma.xml file for OMF module objects as the following:

Note that you need to add your new hbm.xml file into the <list> section:

```

<bean id="OMFSessionFactory"
class="org.springframework.orm.hibernate.LocalSessionFactoryBean">
<property name="dataSource"><ref bean="myDataSource"/></property>
<property name="mappingResources">
<list>
<value>com/edocs/common/omf/charge/charge.type.hbm.xml</value>
<value>com/edocs/common/omf/company/company.hbm.xml</value>
<value>com/edocs/common/omf/company/companyprofile.hbm.xml</value>
<value>com/edocs/common/omf/costcenter/costcenter.hbm.xml</value>
<value>com/edocs/common/omf/period/period.hbm.xml</value>
<value>com/edocs/common/omf/flexfieldmap/extfielddef.hbm.xml</value>
<value>com/edocs/common/omf/service/service.hbm.xml</value>
<value>com/edocs/common/omf/serviceagreement/serviceagreement.hbm.xml</va
Tue>
<value>com/edocs/common/omf/servicecharge/servicecharge.hbm.xml</value>
<value>com/edocs/common/omf/plan/plan.hbm.xml</value>
</list>
</property>
<property name="hibernateProperties"><ref
bean="defaultHibernateProps"/></property>
</bean>

```

To use the JTA transaction manger

Specify the TransactionManager using JtaTransactionManager. For example:

```

<!-- Use JTA Transaction Manager for multiple data sources)-->
<bean id="TransactionManager"
class="org.springframework.transaction.jta.JtaTransactionManager">
<property name="transactionManagerName">
<value>javax.transaction.TransactionManager</value>
</property>
<property
name="userTransactionName"><value>javax.transaction.UserTransaction</value>
</property>
</bean>

```

To use the hibernate transaction manager

Specify the TransactionManager using HibernateTransactionManager. For example:

```

<!-- Use Hibernate manage transaction manager for single data source -->
<bean id="TransactionManager"
class="org.springframework.orm.hibernate.HibernateTransactionManager"
singleton="true" lazy-init="default" autowire="default" dependency-
check="default">
<property name="sessionFactory"><ref
bean="OMFSessionFactory"/></property>
</bean>

```

You should use one transaction manager for all your modules. This is because Billing Analytics works with the OLAP database using the JTA transaction manager, and the JTA transaction manager is used by all modules and components. Use the hibernate transaction manager to unit test your code.

Making Your OMF Objects Work with Hierarchy

The hierarchy module allows any kind of business object to be linked through hierarchy, as long as that business object implements a set of interfaces and registers with hierarchy properly.

For the standard product, the following types of business objects are ready to be used as hierarchy link targets: Account, Company, Service, ServiceAgreement, ChargeType, ServiceCharge, CostCenter, and Folder.

In addition to linking the business object to the hierarchy, the new types of business objects can participate in various hierarchy related processes, such as ETL loader, hierarchy XML exchange, and OLTP to OLAP synchronization through a framework.

Once you have created your business object, to link your object through hierarchy, implement a list of hierarchy related interfaces, and then tell the hierarchy module where to find these implementations by configuring them the in hierarchy/hierarchy.cfg.xma.xml file.

Implement Hierarchy Related Interfaces

This section lists the Hierarchy Interfaces.

IHierarchyLinkTarget

To link your business object through hierarchy, first implement the IHierarchyLinkTarget interface.

```
public interface IHierarchyLinkTarget extends IOMFObject
{
    /**
     * This is the link target identifier. This id is decided by the actually
     link target implementation.
     * For example, the link target id for IAccount is account number, for
     IService is account number
     * and service number. Note, this id may only unique identify this object
     in a special domain. For example,
     * the account number is guaranteed unique only for the accounts for a
     particular billing system.
     * @return link target id string
     */
    public String getLinkIdTargetId();

    /**
     * This is the link target name. This sometimes can be the same as the link
     target Id.
     * But usually this is a short, descriptive name that can stand for the
     link target id,
     * but looks nicer on the UI.
     * @return link target name.
     */
    public String getLinkIdTargetName();

    /**
     * Returns true if this object has one or more properties that are
     modifiable.
     * @return true if this object is modifiable through hierarchy.
     */
    public boolean isEditable();

    /**
     * Returns true if the specified property is modifiable through hierarchy.
     * @param propertyName - the name of a property.
     * @return true if property is modifiable.
     */
    public boolean isEditable(String propertyName);
}
```

The values returned from the getLinkIdTargetId() and getLinkIdTargetName() methods will be persisted with the Hierarchy tree node, and may be used to display and quickly search the object inside the tree. The isEditable() method indicates whether your object can be edited through the hierarchy tree. Method isEditable(String) specifies which attributes of the object can be edited if the object is editable.

IHierarchyXMLExchangeHandler

Based on the schema provided by each application, the Hierarchy module allows application developers to provide their own handlers to handle both import and export of the business objects. This gives provides flexibility in deciding how the hierarchy importer should be used in an application.

If you would like to interchange (import or export) your hierarchy data structure with another part of the system or with an external system through XML, you must implement this interface.

```
package com.edocs.common.api.hierarchy.connector;

public interface IHierarchyXMLExchangeHandler
{
    /**
     * Processes the information passed through a domObject to see if
     * it can be added to the Hierarchy.
     * @param domObject XML DOM representation of the OMF object
     * @param period that will be used to validate link target
     * @return An Instance of {@link ImportLinkTargetValidationStatus} to
     indicate what action should be taken by the XMLContentHandler
     */
    public ILinkTargetImportResult validateLinkTarget(Document domObject,
    IPeriod period);
    /**
     * Processes the information passed through a domObject. Either find
     * the specified link target object or create a new link target with
     * the supplied information.
     * @param domObject XML DOM representation of the OMF object
     * @return created or found link target (OMF object)
     */
    public IHierarchyLinkTarget importLinkTarget(Document domObject)
    throws HierarchyException;
    /**
     * used for delta hierarchy import processing
     * @param domObject object representing the link target.
     * @param isNew boolean indicating whether the hierarchy to be imported is
     a new.
     * @return a converted link target object.
     * @throws HierarchyException
     */
    public IHierarchyLinkTarget importDeltaLinkTarget(Document domObject,
    boolean isNew) throws HierarchyException;
    /**
     * Converts the link target (OMF object) into XML.
     * @param linkTarget the link target object to be exported.
     * @return XML representation of the OMF object.
     */
    public String exportLinkTarget(IHierarchyLinkTarget linkTarget)
    throws HierarchyException;
    /**
     * Initializes this exchange handler with the xmlTag to use.
     * @param xmlTag the tag string for given type of link target class.
     */
    public void initialise(String xmlTag);
}

```

By implementing this interface, application developers can decide the rule for valid objects: whether the given business object should be created by the hierarchy importer, if the object does not already exist in the system.

ILinkTargetEventHandler

If you need to run reports against your link target through hierarchy, and your link target data are stored in OLAP, then you must implement this handler. When the hierarchy is changed on the OLTP side, the hierarchy OLAP synchronizer invokes your handler to synchronize hierarchy and the relationship with your link target on the OLAP side.

```
package com.edocs.common.api.hierarchy.connector;

public interface ILinkTargetEventHandler
{
    /**
     * Hook that is called when adding a new link target.
     * @param event the event object
     * @throws DataStoreException thrown if database errors occurred.
     */
    public void processAddEvent(IEvent event) throws DataStoreException;

    /**
     * Hook that is called when adding a node to a period.
     * @param event the event object
     */
    public void processNodePeriodAddEvent(IEvent event);

    /**
     * Hook called when removing a link target from hierarchy tree.
     * @param event the event object that contains information about the event.
     * @throws DataStoreException thrown if database errors occurred.
     */
    public void processRemoveEvent(IEvent event) throws DataStoreException;

    /**
     * Hook called when removing a hierarchy.
     * @param event the event object that contains information about the event.
     */
    public void processRemoveHierarchyEvent(IEvent event);

    /**
     * Hook called when a node is removed from a period.
     * @param event the event object that contains information about the event.
     */
    public void processNodePeriodRemoveEvent(IEvent event);

    /**
     * Hook called for moving a node to a different node.
     * @param event the event object that contains information about the event.
     */
    public void processMoveEvent(IEvent event);

    /**
     * Hook called when publishing hierarchy. Implementing class needs to
    override
        this method
        * to deal with any link target related operation.
        * @param event the event object that contains information about the event.
        */
    public void processPublishEvent(IEvent event);

    /**
     * Hook called when expiring a hierarchy.
     * @param event the event object that contains information about the event.
     */
    public void processExpireHierarchy(IEvent event);
}
```

Configure Link Target

Register link targets for new business objects with the hierarchy module by adding LinkTargetConfig beans to hierarchy.cfg.xma.xml.

The hierarchy module provides a framework that handles a variety of hierarchy-related operations, including object definition, creation, lookup, resolving a linkage, XML exchange, and OLTP to OLAP synchronization. Link targets are highly configurable so that application-specific business objects can provide their own implementations as needed.

Not all handlers are required for all business objects. Application developers can decide which handlers to implement based on application requirements.

Here are some examples in `hierarchy.cfg.xma.xml`:

```
<bean id="ServiceAgreementConfig"
class="com.edocs.common.hierarchy.core.LinkTargetConfig" singleton="true">
  <property
name="targetType"><value>edx:omf:serviceagreement:</value></property>
  <property name="targetTypeName"><value>Service
Agreement</value></property>
  <property name="displayName"><value>getLinkTargetName</value></property>
  <property name="xmlTag"><value>ServiceAgreement</value></property>
  <property name="storedHierXRef"><value>>false</value></property>
  <property name="xmlExchangeHandler">
    <bean class="com.edocs.common.omf.serviceagreement.ServiceAgreementXMLExch
angeHandler"/>
  </property>
  <property name="linkTargetEventHandlers">
    <list>
      <bean
class="com.edocs.common.hierarchy.connector.olap.OLAPServiceAgreementHandler"/>
    </list>
  </property>
</bean>
```

Where:

- "targetType" - This type string uniquely identifies the type of the business object in question. This value must be the same value as `IOMFManager.getOMFObjectType()` and `IOMFService.getTypeByURI()`.
- "targetTypeName" - The displayable string representing the type of object, such as "Service Agreement" for serviceAgreement objects.
- "displayName" – When the object is linked into a hierarchy, which method to call on the object to display the object's name. If this value is specified, the specified method is called. If not, then `getLinkTargetName()` is used.
- "xmlTag" - Tag name used in the XML exchange (import/export) file. (See the `IHierarchyXMLExchangeHandler` on page 42 for more information).
- "xmlExchangeHandler" - The implementation class of `IHierarchyXMLExchangeHandler` to use for XML import and export.
- "linkTargetEventHandler" - The implementation class of `ILinkTargetEventHandler` to use for object-specific OLTP to OLAP synchronization. It can support a list of handlers to synchronize to multiple destinations.

```

<bean id="ChargeTypeConfig"
class="com.edocs.common.hierarchy.core.LinkTargetConfig" singleton="true">
  <property name="targetType"><value>edx:omf:chargeType:</value></property>
  <property name="targetTypeName"><value>Charge</value></property>
  <property name="displayName"><value>getLinkTargetName</value></property>
  <!-- my list of customize handlers to handle charge type. -->
  <property name="linkTargetAddEventHandlers">
    <list>
      <bean
class="com.edocs.common.hierarchy.connector.olap.OLAPChargeTypeHandler"/>
    </list>
  </property>
</bean>

<bean id="HierarchyFolderConfig"
class="com.edocs.common.hierarchy.core.LinkTargetConfig" singleton="true">
  <property name="targetType"><value>edx:hierarchy:folder:</value></property>
  <property name="targetTypeName"><value>Hierarchy Folder</value></property>
  <property name="displayName"><value>getLinkTargetName</value></property>
  <property name="xmlTag"><value>Folder</value></property>
  <property name="xmlExchangeHandler">
    <bean
class="com.edocs.common.hierarchy.connector.exchange.FolderXMLExchangeHandler"/>
  </property>
</bean>

<bean id="CompanyConfig" class="com.edocs.common.hierarchy.core.LinkTargetConfig"
singleton="true">
  <property name="targetType"><value>edx:omf:company:</value></property>
  <property name="targetTypeName"><value>Company</value></property>
  <property name="displayName"><value>getLinkTargetName</value></property>
  <property name="xmlTag"><value>Company</value></property>
  <property name="xmlExchangeHandler">
    <bean
class="com.edocs.common.omf.company.CompanyXMLExchangeHandler">
      <constructor-arg index="0">
        <bean
class="com.edocs.common.xma.api.LookupBeanFactoryBean">
          <property name="beanUri">
            <value>edx:platform://modules/omf?id=companyManager</value>
          </property>
        </bean>
      </constructor-arg>
    </bean>
  </property>
</bean>

<bean id="AccountConfig"
class="com.edocs.common.hierarchy.core.LinkTargetConfig" singleton="true">
  <property
name="targetType"><value>edx:amf:billingaccount:</value></property>
  <property
name="targetTypeName"><value>hierarchy.element.class.Account</value></property>
  <property
name="displayName"><value>getLinkTargetName</value></property>
  <property name="xmlTag"><value>Account</value></property>
  <property name="xmlExchangeHandler">
    <bean
class="com.edocs.domain.telco.amf.defaultimpl.BillingAccountXMLExchangeHandler"
>
      <constructor-arg index="0">
        <bean
class="com.edocs.domain.telco.amf.defaultimpl.BillingAccountManager"
singleton="true"/>
      </constructor-arg>
    </bean>
  </property>
  <property name="linkTargetEventHandlers">
    <list>
      <bean
class="com.edocs.common.hierarchy.connector.olap.OLAPAccountHandler"/>
    </list>
  </property>
</bean>

```

```

<bean id="ServiceChargeConfig"
class="com.edocs.common.hierarchy.core.LinkTargetConfig" singleton="true">
  <property
name="targetType"><value>edx:omf:servicecharge:</value></property>
  <property
name="targetTypeName"><value>hierarchy.element.class.ServiceCharge</value></pro
perty>
  <property
name="displayName"><value>getLinkTargetName</value></property>
  <property name="xmlTag"><value>ServiceCharge</value></property>
  <property name="xmlExchangeHandler">
    <bean
class="com.edocs.common.omf.servicecharge.ServiceChargeXMLExchangeHandler"/>
  </property>
  <!-- my list of customize handlers to handle charge type. -->
  <property name="linkTargetEventHandlers">
    <list>
      <bean
class="com.edocs.common.hierarchy.connector.olap.OLAPServiceChargeHandler"/>
    </list>
  </property>
</bean>

```

Configure the Link Target into a Hierarchy Type

Add the new config bean for the new link target to the hierarchy type config where the business object should be linked. See the following example from hierarchy.cfg.xma.xml:

```

<bean id="BusinessHierarchyType"
class="com.edocs.common.hierarchy.core.HierarchyTypeConfig">
  <property name="name"><value>hierarchy.type.Business</value></property>
  <property name="code"><value>BUSINESS</value></property>
  <property name="description"><value>Business hierarchy</value></property>
  <property name="validLinkTargets">
    <list>
      <ref bean="HierarchyFolderConfig"/>
      <ref bean="CompanyConfig"/>
      <ref bean="ServiceAgreementConfig"/>
      <ref bean="ServiceChargeConfig"/>
    </list>
  </property>
  <property name="linkTargetRelationship">
    <list>
      <!-- if any of the link target doesn't defined below, then by
default it can have any type of valid -->
      <!-- link target defined above as it children -->
      <!-- Service agreement is a leaf node -->
      <bean
class="com.edocs.common.hierarchy.core.LinkTargetRelationship">
        <property name="parent"><ref
bean="ServiceAgreementConfig"/></property>
      </bean>
      <!-- Company can have folder, and account as it children -->
      <bean
class="com.edocs.common.hierarchy.core.LinkTargetRelationship">
        <property name="parent"><ref bean="CompanyConfig"/></property>
        <property name="validChildren">
          <list><ref bean="HierarchyFolderConfig"/></list>
        </property>
      </bean>
    </list>
  </property>
</bean>

```

Make Your New OMF Object Searchable Through Hierarchy

Through XML configuration, developers can specify which attributes can be used as part of search criteria for a given business object. Based on those attributes, developers can construct flexible search criteria to conduct different searches. If the related information is spread out among several java classes, the searchable attributes can be specified through object references. For a predefined business object, you can add or remove searchable attributes through XML configuration as well.

The following sections list the steps required to support search capability for your OMF object.

Define Search Criteria

First, define where the search should start when you search for an attribute. Typically, this is a top-level class. For example: search service number on service agreement object. To enable service agreement object support search criteria:

```
<bean id="serviceAgreementSearchCriteria"
      class="com.edocs.common.omf.search.SearchCriteriaSupport"
      singleton="false">
  <property name="searchMetaData">
    <ref local="serviceAgreementSearchMeta"/>
  </property>
</bean>
```

ServiceAgreementSerachMeta is explained in the next section.

When a class is a subordinate class, the search for the specified bean class is always done through its parent class. For example, if you would like to search for a user's address, which is stored in the UserProfile class (not the user class), the search criteria support is specified on user class, not on UserProfile class.

Configure Searchable Properties

Three types of property are available for search: simple property, references to other objects, and extended attribute of the business object. Through object reference, your search can be conducted across multiple objects. You can specify your own search configuration in one or multiple files.

Specify the search Meta data bean to specify which attributes can be searched by. See the following example configuration:

In ServiceAgreement class, use the SimpleProperty bean class to indicate that simple property "name" and "description" can be searched.

```

    <bean id="serviceAgreementSearchMeta"
        class="com.edocs.common.omf.search.SearchMetaDataSupport"
        singleton="true">
        <property name="beanClassName">
<value>com.edocs.common.omf.serviceagreement.ServiceAgreement</value>
        </property>
        <property name="propertyList">
            <list>
                <bean class="com.edocs.common.omf.search.SimpleProperty">
                    <property name="name">
                        <value>name</value>
                    </property>
                </bean>
                <bean class="com.edocs.common.omf.search.SimpleProperty">
                    <property name="name">
                        <value> subscriberName</value>
                    </property>
                </bean>
                <bean class="com.edocs.common.omf.search.SimpleProperty">
                    <property name="name">
                        <value> extAttr1</value>
                    </property>
                </bean>
            </list>
        </property>
    </bean>

```

In the ServiceCharge class, use the EntityProperty bean class to indicate that serviceAgreementId is an object reference to serviceAgreement. Through the reference of serviceAgreementId, the searchable attributes on ServiceAgreement are available for search on ServiceCharge class.

```

<bean id="serviceChargeSearchMeta"
    class="com.edocs.common.omf.search.SearchMetaDataSupport"
    singleton="true">
    <property name="beanClassName">
        <value>com.edocs.common.omf.servicecharge.ServiceCharge</value>
    </property>
    <property name="propertyList">
        <list>
            <bean class="com.edocs.common.omf.search.EntityProperty">
                <property name="name"><value>serviceAgreementId</value>
            </property>
            <property name="searchMetaData"><ref
                local="serviceAgreementSearchMeta"/></property>
        </list>
    </property>
    <bean class="com.edocs.common.omf.search.EntityProperty">
        <property
            name="name"><value>chargeTypeId</value></property>
        <property name="searchMetaData"><ref
            local="chargeTypeSearchMeta"/></property>
    </bean>
    </list>
</property>
</bean>

```

In the BillingAccount class, MapProperty indicates that "attr1" and "attr2" are extended attributes that can be used in search criteria.


```

    <bean id="accountSearchMeta"
        class="com.edocs.common.omf.search.SearchMetaDataSupport"
        singleton="true">
        <property name="beanClassName">
<value>com.edocs.domain.telco.amf.defaultimpl.BillingAccount</value>
        </property>
        <property name="propertyList">
<list>
        <bean class="com.edocs.common.omf.search.SimpleProperty">
            <property name="name"><value>Name</value></property>
        </bean>
        <bean class="com.edocs.common.omf.search.SimpleProperty">
            <property name="name"><value>Description</value></property>
        </bean>
        <bean class="com.edocs.common.omf.search.SimpleProperty">
            <property name="name"><value>companyName</value></property>
        </bean>
        <bean class="com.edocs.common.omf.search.SimpleProperty">
            <property name="name"><value>contactName</value></property>
        </bean>
        <bean class="com.edocs.common.omf.search.MapProperty">
            <property
name="name"><value>attributes</value></property>
            <property
name="key"><value>attr1</value></property>
        </bean>
        <bean
class="com.edocs.common.omf.search.MapProperty">
            <property
name="name"><value>attributes</value></property>
            <property
name="key"><value>attr2</value></property>
        </bean>
        </list>
    </property>
</bean>

```

Implement Search Interfaces

Implement `IOMFObjectSearchable` to tell which attributes are available for search. The search component reads the preceding configuration file to get a list of attributes that can be used as searchable attributes.

```

public interface IOMFObjectSearchable
{
    /**
     * Finds IOMFObjects that match the provided attribute value.
     * A "%" character is interpreted as the like wildcard.
     * So for instance, "foo%" will match all strings that
     * start with "foo". If there is no "%" character present
     * in the value then an equals match is performed.
     * @param criteria - the search criteria
     * @return list of IOMFObject objects
     */
    public List findLike(String attributeName, String value);

    /**
     * Finds IOMFObjects that satisfy the specified search criteria.
     * @param criteria - the search criteria
     * @return list of IOMFObject objects
     */
    public List find(ISearchCriteria criteria);

    /**
     * Returns a new ISearchCriteria object.
     * @return ISearchCriteria object.
     */
    public ISearchCriteria getSearchCriteria();

    /**
     * Gets a list of IPropertyInfo - one for each searchable property.
     *The IPropertyInfo
     * object specifies the name and display name of a searchable property.
     * @return List of IPropertyInfo objects.
     */
    public List getPropertyInfo();
}

```

Construct Search Criteria

Use ISearchCriteria and IPredicate to build search criteria.

ISearchCriteria.toQueryString() converts the search criteria to a Hibernate SQL string, calls the hibernate API to do the final query, and returns the result set. For example:

```

IServiceAgreementManager saManager =
IOFMService.getOMFManagerByName(ServiceAgreementManager.getClassName());
ISearchCriteria criteria = saManager.getSearchCriteria();
criteria.add(criteria.equals("subscriberName", "John Willson" ));add(
criteria.isNull( "extAttr1"));
IHierarchyNode foundNodes =
rootNode.findNodeByLinkTargetTypeAndCriteria(saManager.getType(), criteria);

```

Provide support for reporting on the new business object

You must implement an event handler to run reports against your business objects through the hierarchy link target, because the link target data is stored in OLAP. When hierarchy is changed in OLTP, either through the UI or a batch job, an event handler can be called directly by the hierarchy API to process these events. Changes can be categorized by event type.

The same mechanism can be used to propagate to the reporting (OLAP) database. The changes made to OLTP and OLAP are bound in a single transaction to guarantee data integrity between OLTP and OLAP databases. Distributed database transaction management achieves this.

How to write your own synchronization handler

If you create a new business object and register it as a link target with hierarchy, you may also want to write your own OLAP handler, so that when your business object is linked into hierarchy, the same relationship can be propagated to the OLAP reporting table.

The way you construct the relationship between business object and hierarchy cross reference table determines how much control you have over the relationship. The general pattern is that each hierarchy node has a corresponding entry in `edx_rpt_hierarchy_xref_dim` table.

The node and concrete link target type relationship is modeled through the hierarchy and link target workspace table. You can choose whether each link target has a node entry in `edx_rpt_hierarchy_xref_dim` or not. The alternative is to build a workspace table with just the link targets that are parent nodes of your object.

If you do not want to add a node entry to your link target object, make sure that you return `isStoredFlat() = true` in your link target event handler. Then, the node entry is not added to the link target `edx_rpt_hierarchy_xref_dim` table, and your objects and parent objects are stored flat in the workspace table. This may speed up the query.

To write your own OLAP link target handler, implement interface `ILinkTargetEventHandler`.

Once you finished your OLAP handler, you need to register it with hierarchy through hierarchy link target config in `hierarchy.cfg.xma.xml` file.

```
<bean id="ServiceAgreementConfig"
class="com.edocs.common.hierarchy.core.LinkTargetConfig" singleton="true">
  <property
name="targetType"><value>edx:omf:serviceagreement:</value></property>
  <property name="targetTypeName"><value>Service
Agreement</value></property>
  <property name="displayName"><value>getLinkTargetName</value></property>
  <property name="xmlTag"><value>ServiceAgreement</value></property>
  <property name="storedHierXref"><value>>false</value></property>
  <property name="xmlExchangeHandler">
    <bean class="com.edocs.common.omf.serviceagreement.ServiceAgreementXMLExch
angeHandler"/>
  </property>
  <property name="linkTargetEventHandlers">
    <list>
      <bean
class="com.edocs.common.hierarchy.connector.olap.OLAPServiceAgreementHandler"/>
    </list>
  </property>
</bean>
```

Support XML exchange

Configure XML Schema for Your Business Object

The hierarchy importer is flexible enough to import hierarchy relationships as well as link target content. The application developer can decide whether the importer and exporter will deal with only relationships or will also deal with link target content.

To meet these requirements, the hierarchy module provides two XML schema definition files:

- **Common-hierarchy-interchange-1.0.xsd** specifies the format that is core to the hierarchy module. Application developers should use this file as is (do not change it).

- **Instance-hierarchy-interchange-1.0.xsd** customizes the hierarchy structure. Application developers can add one or more sections to describe application-specific business objects (link targets) in the import or export XML file.

Instance-hierarchy-interchange-1.0.xsd is referenced in the common-hierarchy-interchange-1.0.xsd schema file. So in preparing an XML data file, developers only need to reference common-hierarchy-interchange-1.0.xsd as shown in the following example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ListOfHierarchies xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="common-hierarchy-interchange-1.0.xsd">
```

Link target can be defined in instance-hierarchy-interchange.1.0.xsd, such as:

```
<xs:element name="CostCenter">
<xs:complexType>
<xs:attribute name="name" type="xs:string"/>
<xs:attribute name="fiscalCode" type="xs:string"/>
<xs:attribute name="manager" type="xs:string"/>
</xs:complexType>
</xs:element>
```

In the XML data file, a node that represents a cost object can be expressed as the following example:

```
<Node>
<BusinessObject>
<CostCenter name="PS", fiscalCode="10001" manager="Joe
Smith"/>
</BusinessObject>
<CanBeAccessedBy>
<userId>jsmith</userId>
<userId>jane</userId>
</CanBeAccessedBy>
</Node>
```

Write your own Link Target Exchange Handler

The following example shows how to define your own link target exchange handler:

```

package com.edocs.common.api.hierarchy.connector;

public interface IHierarchyXMLExchangeHandler
{
    /**
     * Processes the information passed through a domObject to see if
     * it can be added to the Hierarchy.
     * @param domObject XML DOM representation of the OMF object
     * @param period that will be used to validate link target
     * @return An Instance of {@link ImportLinkTargetValidationStatus} to
     indicate
     * what action should be taken by the XMLContentHandler
     */
    public ILinkTargetImportResult validateLinkTarget(Document domObject,
    IPeriod period);
    /**
     * Processes the information passed through a domObject. Either find
     * the specified link target object or create a new link target with
     * the supplied information.
     * @param domObject XML DOM representation of the OMF object
     * @return created or found link target (OMF object)
     */
    public IHierarchyLinkTarget importLinkTarget(Document domObject)
    throws HierarchyException;
    /**
     * Used for delta hierarchy import processing
     * @param domObject object representing the link target.
     * @param isNew boolean indicating whether the hierarchy to be imported is
     a new.
     * @return a converted link target object.
     * @throws HierarchyException
     */
    public IHierarchyLinkTarget importDeltaLinkTarget(Document domObject,
    boolean isNew) throws HierarchyException;
    /**
     * Converts the link target (OMF object) into XML.
     * @param linkTarget the link target object to be exported.
     * @return XML representation of the OMF object.
     */
    public String exportLinkTarget(IHierarchyLinkTarget linkTarget)
    throws HierarchyException;
    /**
     * Initializes this exchange handler with the xmlTag to use.
     * @param xmlTag the tag string for given type of link target class.
     */
    public void initialise(String xmlTag);
}

```

By implementing this interface, application developers can decide the rule for valid objects; whether the given business object should be created by the hierarchy importer when the object does not already exist in the system.

The `ValidateLinkTarget()` method is the first one called on your `XMLExchangeHandler` during the process of importing. You can put your own logic in the validation. For example, if the object you are handling does not exist in the system when the hierarchy is imported, you can decide whether to stop the importer from continuing, or just skip over the object and continue to import the rest of hierarchy.

`ILinkTargetImportResult` returns validation results for each link target being imported. It contains the validation result and some other context information.

```

package com.edocs.common.api.hierarchy.connector.exchange;
public interface ILinkTargetImportResult
{
    /**
     * Returns the link target element tag.
     * @return the link target element tag.
     */
    public String getTag();

    /**
     * Returns the status of import.
     * @return the status of import.
     */
    public ImportLinkTargetValidationStatus getStatus();

    /**
     * Returns a list of attribute errors, if any.
     * @return a list of attribute errors, if any.
     */
    public List getAttributeErrors();

    /**
     * Returns the identifier.
     * @return the identifier.
     */
    String getIdentifier();
}

```

Where ImportLinkTargetValidationStatus contains the result of link target validation with the following possible values:

- PASS – Indicates that validation passed.
- WARN_SKIP -- Indicates that there is a warning message. The link target to be imported must be skipped. For example, will not be linked into the hierarchy.
- WARN_INCLUDE – Indicates that there is a warning message. However, the link target to be imported will be included in the hierarchy tree.
- FAIL_END – Indicates that there is an error. However, the importer process can continue with a failure message at the end.
- FAIL_NOW – Indicates that there is a fatal error and the importer process must end immediately.

By returning different status code, your validation method can tell the importer what to do next.

Transactions in Hierarchy Importer

Operations that create data in the system using the importer are bounded into a single database transaction. Changes made to the database are not committed until the importer commits the changes.

Commit size and approach can be configured by modifying HierarchyImporter.xma.xml.

```

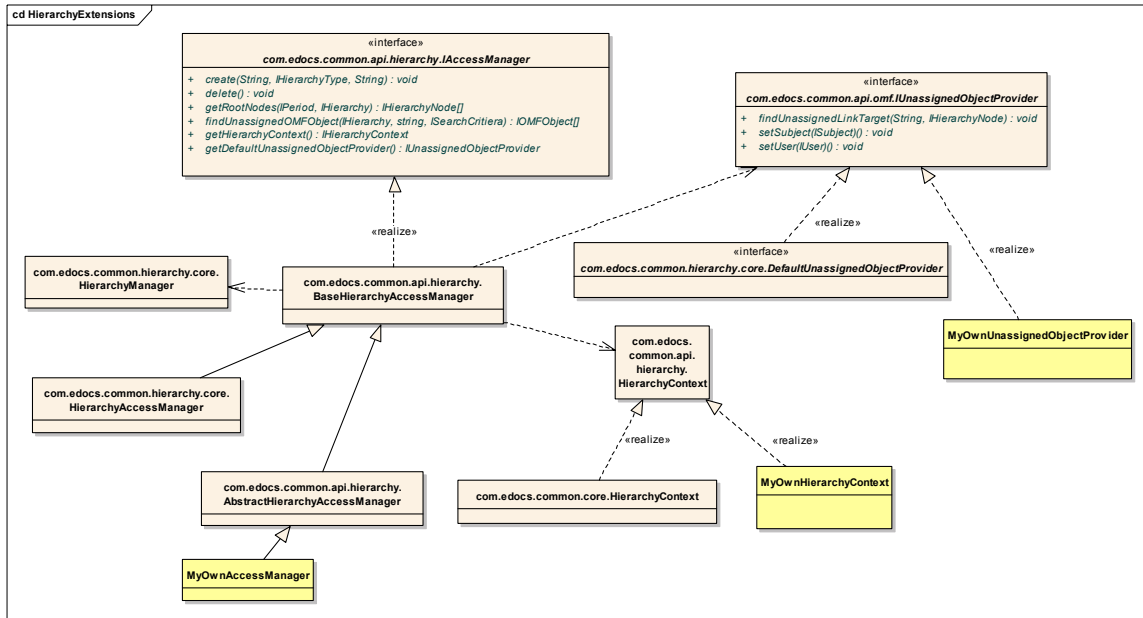
<bean id="HierarchyXMLContentHandler"
class="com.edocs.common.hierarchy.connector.exchange.HierarchyXMLContentHandler"
" singleton="false">
  <constructor-arg index="0">
    <ref bean="IHierarchyManager"/>
  </constructor-arg>
  <constructor-arg index="1">
    <ref bean="IHierarchyTypeManager"/>
  </constructor-arg>
  <constructor-arg index="2">
    <ref bean="TransactionManager"/>
  </constructor-arg>
  <constructor-arg index="3">
    <ref bean="IHierarchyPeriodValidationManager"/>
  </constructor-arg>
  <constructor-arg index="4">
    <bean
class="com.edocs.common.xma.api.LookupBeanFactoryBean">
      <property name="beanUri">
        <value>edx:platform://modules/omf?id=periodManager</value>
      </property>
    </bean>
  </constructor-arg>
  <property name="committedBatchSize"><value>200</value><!-- 200
records --></property>
  <property name="batchCommitEnable"><value>true</value></property>
  <property name="transactionHoldingTime"><value>5</value><!--
holding the transaction for 5 minutes max --></property>
</bean>

```

Working with External SSO System

In some client systems, data for accessing billing information is stored in an external system. While replicating information into Billing Analytics is possible for some deployments, for others Billing Analytics must work with an external access control system to provide hierarchical reporting functionality.

The following set of interfaces and base classes are provided as part of the set of extension points:



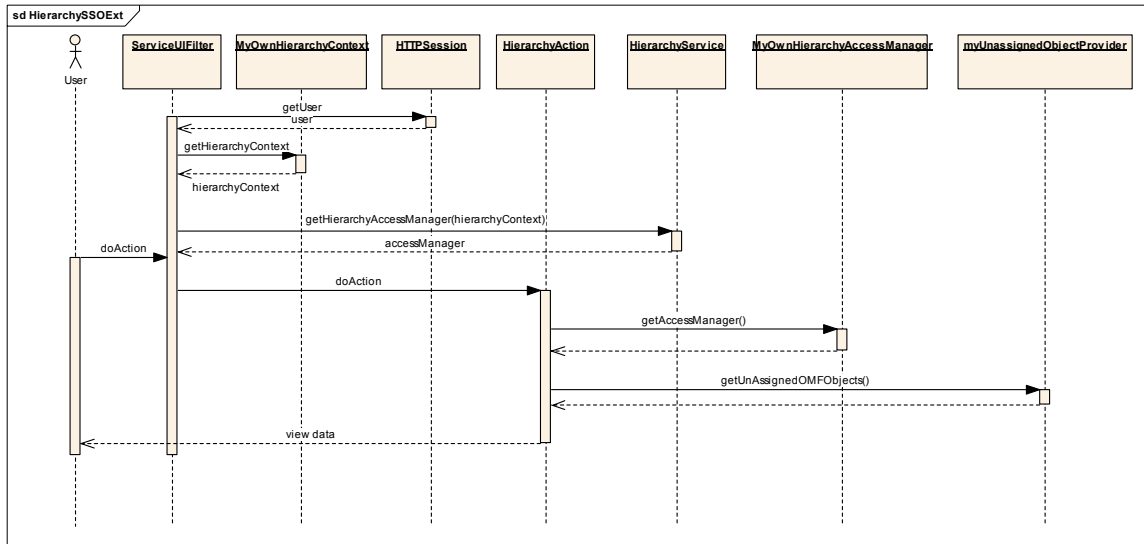
In the preceding diagram, class `myOwnHierarchyContext`, `myOwnAccessManager`, and `myOwnUnassignedObjectProvider` are places where you can add custom logic for access control related integration.

In the `myOwnHierarchyContext` class, you can store any information you gathered from external systems: http session and/or internal tables.

The `myOwnAccessManager` class is based on the information you stored to determine what level of access control to give, which delegates to `IHierarchyManager` to actually do the hierarchy related work.

The `myOwnUnassignedObjectProvider` class can be any class you create, as long as it provides the correct list of unassigned objects for a given user and hierarchy position. This could include code that queries an external system, or uses PL/SQL to retrieve a list of objects based on your business rules.

The following diagram shows a possible interaction sequence:



Working with Extended Attributes on Service Agreement Object

As one of the OOTB features, the `ServiceAgreement` object provides five flexible string attributes for developer use. These attributes can represent any properties. The label of these extended attributes can be different for each company. For example, in `companyA`, the `extendedAttribute1` field keeps the License Number attribute, which is related to the `serviceAgreement`. In `companyB`, the same `extendedAttribute1` fields for different service Agreement objects can keep service agreement owner's nick-name. Billing Analytics internally maintains a list of attribute name mappings for each company to support extended attribute label mapping. The map can be used to render extended attribute names in the UI.

`ExtFieldDefManager` is responsible for managing and finding the mapped entries. Details about using this feature can be found in java doc for the `com.edocs.common.api.omf.flexfieldmap` package.

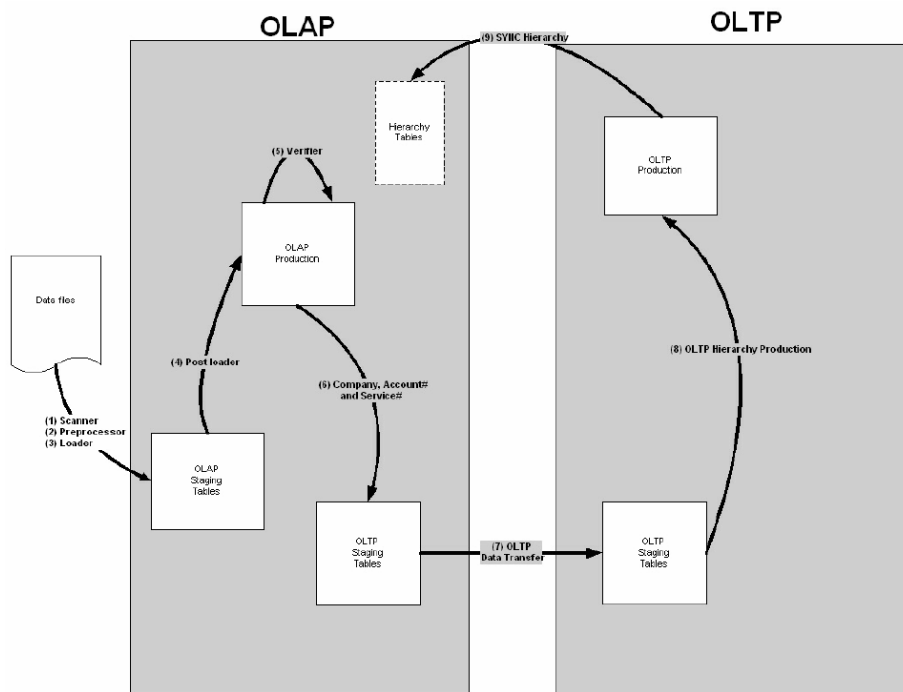
6 Working with the ETL Process

ETL

During ETL processing, accounts, service agreements (for example, MTNs or MSISDNs) and other business object dimension data and fact data are loaded into the OLAP database. In addition, an ETL exchange table (EDX_RPT_ETL_XCHANGE), which captures information about accounts, service agreements, companies and billing hierarchies, is populated and used to load corresponding production tables in OLTP database at a later time.

An ETL controller job is run to call different module loaders to load module specific data from the exchange table and other OLAP tables into OLTP production tables.

The following diagram illustrates this process.



Among account and service loaders, which populate account and service agreement tables to OLTP production (step 8), Hierarchy ETL loader reads hierarchy information from EDX_RPT_ETL_XCHANGE table and populates Hierarchy tables using hierarchy module APIs. While the changes are made to hierarchy in OLTP tables, the same changes are pushed into OLAP hierarchy tables synchronously.

Load Handler Configuration

OLTPProductionLoader is part of the ETL process. It accepts a list of loaders that implement the `com.edocs.common.api.omf.IETLLoader` interface. This job picks up each loader and runs the load method on each load in the specified order. The list of loads can be configured in the file located at `C:\Siebel\CBA\estatement\config\etl\ProductionLoader.xml` as shown in the following example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Loaders>
<loader id="OLTPProductionLoader" class="com.edocs.common.etl.loader" >
  <displayName id="displayName" value="oltpProductionLoader" />
  <table id="statusTable" value="edx_etl_xchange_status" />
  <dataSource id="oltp" uri="edx.databasePool" />
  <dataSource id="olap" uri="edx.report.databasePool" />
  <handlers>
    <handler id="amf" name="amfLoaders"
class="com.edocs.tasks.oltpProductionLoader.AmfLoader" loaderType="1"
exception="Error Loading AMF" loadMethod="load"/>
    <handler id="service" name="serviceLoader"
class="com.edocs.tasks.oltpProductionLoader.ServiceLoader" loaderType="2"
exception="Error Loading Services" loadMethod="load"/>
    <handler id="cmf" name="cmfLoader"
class="com.edocs.tasks.oltpProductionLoader.CmfLoader" loaderType="3"
exception="Error Loading Companies" loadMethod="load"/>
    <handler id="hierarchy" name="hierarchyLoader"
class="com.edocs.common.hierarchy.connector.etl.ETLHierarchyLoader2" loaderType="4"
exception="Error Loading Hierarchy" loadMethod="load"/>
    <handler id="omf" name="omfLoader"
class="com.edocs.tasks.oltpProductionLoader.OmfLoader" loaderType="5"
exception="Error Loading Omf" loadMethod="load"/>
    <handler id="period" name="periodLoader"
class="com.edocs.tasks.oltpProductionLoader.PeriodLoader" loaderType="6"
exception="Error Loading Peiods" loadMethod="load"/>
  </handlers>
</loader>
</Loaders>
```

The ETL task calls the `load(userId, loaderContext)` method on each registered load handler object. The handler reads data from OLAP, processes it, and populates the data into OLTP production tables.

ETL Loader Interface

An ETL loader interface is defined as follows: Each configured loader implements the interface and moves the data from exchange tables to OLTP production tables. The following example shows that interface:

```
Package com.edocs.common.api.omf
public interface IETLLoader {
    /**
     * Moves data from OLTP staging tables to OLTP production tables.
     * @param user the etl user
     * @param loadcontext uniquely identifies data feed
     * @return true if the load process succeed, false otherwise.
     */
    public boolean load(String user, long loadcontext) throws SQLException;
}
```

APIs

IAttribute

This interface represents an attribute of a hierarchy node. Each attribute is essentially a name-value pair. A node can have a list of attributes associated with it. Each attribute is identified by its name and value and so, it's possible to have multiple nodes which have the same name but different values. For example, {"fruit", "apple"}, {"fruit", "orange"} and {"fruit", "pear"}.

Method	Description
String getName()	Gets the name of the attribute
java.lang.Object getValue()	Gets the value of the attribute.
void setName(java.lang.String name)	Sets the name of the attribute
void setValue(java.lang.Object value)	Sets the value of the attribute.

IExpression

This interface represents different filter operations you can have on a query, especially for searching name-value pairs such as hierarchy node attributes. Use `IHierarchyService.createExpression()` to get an instance of this object.

Method	Description
IFilter and(IFilter c1, IFilter c2)	Create a filter which does the AND logic operation on the two passed-in filters.
IFilter between(java.lang.String propertyName, java.lang.Object lo, java.lang.Object hi)	Create a filter which does the BETWEEN operation.

Method	Description
<code>IFilter eq(java.lang.String propertyName, java.lang.Object value)</code>	Create a filter which does the EQUAL logic operation.
<code>IFilter ge(java.lang.String propertyName, java.lang.Object value)</code>	Create a filter which does the GREATER THAN OR EQUAL (\geq) operation.
<code>IFilter gt(java.lang.String propertyName, java.lang.Object value)</code>	Create a filter which does the GREATER THAN ($>$) operation.
<code>IFilter in(java.lang.String propertyName, java.util.Collection values)</code>	Create a filter which does the IN operation.
<code>IFilter isNotNull(java.lang.String propertyName)</code>	Create a filter which does the IS NOT NULL operation.
<code>IFilter isNull(java.lang.String propertyName)</code>	Create a filter which does the IS NULL operation.
<code>IFilter le(java.lang.String propertyName, java.lang.Object value)</code>	Create a filter which does the LESS THAN OR EQUAL (\leq) operation.
<code>IFilter like(java.lang.String propertyName, java.lang.Object value)</code>	Create a filter which does the SQL LIKE operation.
<code>IFilter lt(java.lang.String propertyName, java.lang.Object value)</code>	Create a filter which does the LESS THAN ($<$) operation.
<code>IFilter not(IFilter expression)</code>	Create a filter which does the NOT operation on the passed in filter.
<code>IFilter or(IFilter c1, IFilter c2)</code>	Create a filter which does the OR logic operation on the two passed-in filters.
<code>IFilter sql(java.lang.String sql)</code>	Create a filter using the passed in SQL expression.

IFilter

This interface represents a Query Filter, which can be used as a filter to the query result.

Method	Description
<code>java.lang.String toSQLString()</code>	The string representation of this filter.

IFilteredQuery

This interface provides methods to access a query which, when executed, returns a list of objects that can be filtered to match the specified filter class.

Method	Description
<code>IFilteredQuery add(IFilter filter)</code>	Adds the filter to be apply when execute the <code>list()</code> .
<code>java.lang.Class getFilteredClass()</code>	Returns the filtered class
<code>java.util.List getFilters()</code>	Returns a list of filter inserted by the <code>add()</code> .
<code>java.util.List list()</code>	Execute the query and return a list of java object for class set by <code>setFilteredClass()</code>
<code>void setFilteredClass(java.lang.Class class)</code>	Sets the class for filtering.

IHierarchy

This interface represents a contract of a hierarchy. Use it to remove, update, and publish a hierarchy, or access and modify the attributes or tree nodes of a hierarchy. Each hierarchy has a root node and may include many levels of child nodes. A hierarchy only maintains the parent-child relationship among nodes. The nodes themselves are not business objects; they have no business meaning. The business meaning is expressed through the link target of each node.

Hierarchy has three states:

- **UNPUBLISHED** -- When a hierarchy is first created, it is in unpublished state. Unpublished hierarchies can only be available to or accessed by its creator and system administrators. Changes made to an unpublished hierarchy are not versioned. When an unpublished hierarchy is removed by an end user, the hierarchy is physically removed from database.
- **PUBLISHED** -- Once a hierarchy is published, any changes made to the hierarchy structure will be versioned based on periods defined in the system. A published hierarchy cannot be unpublished again.
- **EXPIRED** -- Once a hierarchy is expired, it becomes not available for periods which are later than the expiration period. However, the hierarchy is still available for the periods prior to expiration period, and the user can still edit the hierarchy for the older version.

Each hierarchy also has a list of attributes(properties) associated with. These attributes include:

- **id**: internal unique id to identify this hierarchy.
- **version**: version of the hierarchy.
- **hierarchy type**: the type of the hierarchy.
- **company id**: the id of the company where this hierarchy belongs to.
- **name**: the name of the hierarchy. It is unique within the specific company.
- **display name**: a more descriptive name for the hierarchy.
- **description**: the description of the hierarchy.
- **created date**: the date when the hierarchy is created.

- created by: the id of the user who creates the hierarchy.
- modified date: the date when the hierarchy is last modified.
- modified by: the id of the user who last modified the hierarchy.
- deleted date: the date when the hierarchy is marked as deleted.
- publish date: the date when the hierarchy is published.
- expire date: the date when the hierarchy is expired.
- expire period: the period starting which the hierarchy becomes expired. Get an instance of IHierarchy through IHierarchyManager.

Method	Description
<code>void addUserAccess(IOMFObject omfObject, java.lang.String userId)</code>	Gives user access to a targeted node.
<code>void expire()</code>	Expires the hierarchy starting from current working period.
<code>IHierarchyNode[] findAllNodeForUserOfType(java.lang.String userId, java.lang.String omfObjectType)</code>	Returns all nodes which the user has access to and whose link target types are of the specified type.
<code>IHierarchyNode[] findFolderNode(java.lang.String folderName)</code>	Deprecated.
<code>IHierarchyNode findFolderNodeByLinkTargetId(java.lang.String linkTargetId)</code>	Deprecated.
<code>IHierarchyNode[] findNodeByLinkTargetId(java.lang.String linkTargetId)</code>	Finds all nodes from a specific hierarchy that match a link target Id.
<code>IHierarchyNode[] findNodeByLinkTargetType(java.lang.String linkTargetType)</code>	Find all nodes in this hierarchy whose link target types match the one specified by linkTargetType.
<code>IHierarchyNode[] findNodeByLinkTargetTypeAndDisplayName(java.lang.String linkTargetType, java.lang.String linkTargetDisplayName)</code>	Finds all nodes in this hierarchy whose link target types are the specified linkTargetType and whose link target names are the specified linkTargetDisplayName.
<code>IHierarchyNode findNodeByLinkTargetTypeAndLinkTargetId(java.lang.String linkTargetType, java.lang.String linkTargetId)</code>	Finds all nodes in this hierarchy that match a link target type and link target Id.
<code>IHierarchyNode findNodeByLinkTargetURI(java.lang.String linkTargetURI)</code>	Finds a node in a hierarchy whose link target URL matches the specified linkTargetURI.
<code>IHierarchyNode[] findNodeByLinkTargetURI(java.lang.String[] linkTargetURIs)</code>	Finds an array of nodes in the hierarchy whose link target URIs match one of the URIs specified by linkTargetURIs.
<code>IHierarchyNode[] findRootNodeForUser(java.lang.String</code>	Finds a list of root hierarchy nodes which the

Method	Description
userID)	user has access to.
IHierarchyNode[] findRootNodes(IUser user)	Finds a list of top level node for the user based on user's role.
java.util.List findUsers(java.lang.String userID, HierSearchType searchType, IAttribute[] attributes)	Deprecated. since hierarchy version2.0
java.util.List findUsers(java.lang.String userID, HierSearchType searchType, java.lang.String attrName, java.lang.Object attrValue)	Finds user objects (Assigned, Unassigned, Authorized or Unauthorized as specified in HierSearchType) that match the specified attribute value within the hierarchy.
IUser getActor()	Return the user who is currently operating on the hierarchy.
java.util.List getAvailablePeriods()	Gets a list of periods this hierarchy exists in.
java.util.List getAvailablePeriods(java.lang.String userID)	Gets all periods within which that specified user can have access to the hierarchy.
java.lang.String getCompanyID()	Gets unique id of the company which this hierarchy belongs to.
java.util.Date getCreatedAt()	Deprecated. In 2.0, gets the date when the hierarchy was created.
java.util.Date getCreatedate()	Get the hierarchy creation date.
java.lang.String getCreatedBy()	Gets user id who created the hierarchy.
java.util.Date getDeletedAt()	Deprecated. In 2.0, returns the date when the hierarchy was marked as deleted.
java.lang.String getDescription()	Gets the description of the hierarchy.
java.lang.String getDisplayName()	Gets the display name of the hierarchy.
java.util.Date getExpiredate()	Gets the hierarchy expiration date.
IPeriod getExpiredPeriod()	Gets the period when the hierarchy is expired, or good through.
IHierarchyHandle getHandle()	Gets the hierarchy handle for this hierarchy: you can serialize this handle into a persistent store and then use it to restore the hierarchy.
java.lang.Long getID()	Gets the internal ID used to identify this hierarchy.
java.util.List getLinkTargetTypes()	Gets all valid link targets type string.
java.util.Date getModifiedAt()	Deprecated. In 2.0 Gets the date when the hierarchy was modified last time.

Method	Description
<code>java.lang.String getModifiedBy()</code>	Gets the login name of user who modified the hierarchy last time.
<code>java.util.Date getModifydate()</code>	Gets the hierarchy last modified date.
<code>java.lang.String getName()</code>	Get the name of the hierarchy
<code>java.util.Date getPublishdate()</code>	Gets hierarchy publish date.
<code>IHierarchyNode getRoot()</code>	Returns the root node of this hierarchy.
<code>IHierarchyType getType()</code>	Gets the type of the hierarchy.
<code>java.lang.Long getVersion()</code>	Gets the versioning number of this object stored in the database.
<code>IPeriod getworkingPeriod()</code>	Gets the current working period.
<code>boolean isActivated()</code>	The opposite of the <code>isDeleted()</code> .
<code>boolean isDeleted()</code>	Checks whether this hierarchy has been marked as deleted.
<code>boolean isPublished()</code>	Checks whether the hierarchy has been published.
<code>void publish()</code>	Publishes the hierarchy for the current period.
<code>void publish(IPeriod startPeriod, IPeriod endPeriod)</code>	Publishes the hierarchy from <code>startPeriod</code> to <code>endPeriod</code> , inclusive.
<code>void purge()</code>	Remove the hierarchy from database.
<code>void remove()</code>	Removes the hierarchy.
<code>void removeAccessForUser(java.lang.String userId)</code>	Removes a user's access to this hierarchy, which actually means all the nodes in this hierarchy.
<code>void removeUserAccess(IHierarchyLinkTarget linkTarget, java.lang.String userId)</code>	Removes user's access to the node whose link target is parameter <code>linkTarget</code> .
<code>void replicateData(IPeriod startPeriod, IPeriod endPeriod)</code>	Replicates current hierarchy structure for new periods specified between starting and end periods, inclusive.
<code>void setActor(IUser user)</code>	Sets the actor.
<code>void setDescription(java.lang.String description)</code>	Sets the description of the hierarchy.
<code>void setDisplayName(java.lang.String displayName)</code>	Sets the display name of the hierarchy.
<code>void setName(java.lang.String name)</code>	Sets the name of the hierarchy.

Method	Description
<code>void setToLastestAvailablePeriod()</code>	Sets the working period to the latest available period.
<code>void setWorkingPeriod(IPeriod workingPeriod)</code>	Sets a new working period.
<code>void update()</code>	Update this hierarchy with the new attributes and or properties.

IHierarchyFolder

This interface provides a protocol to access, update, or add attributes to a hierarchy folder.

Method	Description
<code>void addAttribute(IAttribute attribute)</code>	Adds additional IAttribute object to the folder.
<code>java.util.Set getAttributes()</code>	Gets a set of attributes added through addAttribute() method.
<code>java.lang.String getDescription()</code>	Gets description of the folder.
<code>java.lang.String getDisplayName()</code>	Returns display name of the folder.
<code>java.lang.String getLinkTargetName()</code>	Gets link target name.
<code>java.lang.String getName()</code>	Deprecated. please use getDisplayName()
<code>void setDescription(java.lang.String description)</code>	Sets description.
<code>void setDisplayName(java.lang.String displayName)</code>	Sets display name.
<code>void setLinkTargetName(java.lang.String name)</code>	Sets folder name.
<code>void setName(java.lang.String name)</code>	Deprecated. please use setDisplayName()
<code>void update()</code>	Updates the folder object into database.

IHierarchyFolderManager

This interface provides a protocol to access the hierarchy folder manager.

Method	Description
<code>IHierarchyFolder createFolder(java.lang.String folderName, java.lang.String description)</code>	Creates a new folder object by passing in name and description.

Method	Description
<code>IHierarchyFolder createFolder(java.lang.String folderExternalKey, java.lang.String folderName, java.lang.String description)</code>	Creates a new folder object by passing in unique string key, name and description.

IHierarchyHandle

Like an EJBHandler for EJB object, this interface provides a handler to a hierarchy object, serializes it to a secondary storage, and then uses it to restore the hierarchy object.

Method	Description
<code>java.lang.String getDisplayName()</code>	Display Name for the hierarchy (this can be used to display a human readable description of the Hierarchy).
<code>IHierarchy getHierarchy()</code>	Get the hierarchy corresponding to this handle.
<code>java.lang.String getHierarchyId()</code>	Returns a string identifier for the hierarchy (this is usually a Long value but may change depending on implementation).
<code>java.lang.Long getVersion()</code>	The version number of hierarchy represented by this handle, this is the db version number and NOT related to versioned hierarchies.

IHierarchyLinkTarget

This interface represents an IOMFObject, which resolves to a link target. Any object that wishes to be linked into hierarchy must implement this interface.

Method	Description
<code>java.lang.String getLinkTargetId()</code>	This is the link target identifier.
<code>java.lang.String getLinkTargetName()</code>	This is the link target name.
<code>boolean isEditable()</code>	Returns true if this object has one or more properties that are modifiable.
<code>boolean isEditable(java.lang.String propertyName)</code>	Returns true if the specified property is modifiable.

IHierarchyManager

This interface provides a contract for managing hierarchy-related operations like create, update, delete, or search. Obtain an instance of IHierarchyManager through IHierarchyService:

- `LookupService lookup: LookupServiceFactory.getInstance()`
- `IHierarchyService hierarchyService: (IHierarchyService) lookup.getModule("hierarchy")`

■ IHierarchyManager hierarchyManager: hierarchyService.createHierarchyManager(anIUser)

Method	Description
<code>void addUserAccess(IOMFObject omfObject, java.lang.String userId)</code>	Allows user to access nodes whose link targets are the same as the given object in all hierarchies.
<code>void addUserAccess(java.lang.String userId, IHierarchyNode[] nodes)</code>	Associates a user to multiple nodes.
<code>IHierarchy createHierarchyFromNode(java.lang.String companyId, java.lang.String hierarchyName, IHierarchyType hierarchyType, IHierarchyNode nodeToCopy, boolean preserveUserAccess)</code>	Creates a hierarchy with a new copy of a tree base on the passed in node.
<code>IHierarchy createHierarchyFromNode(java.lang.String companyId, java.lang.String hierarchyName, IHierarchyType hierarchyType, java.util.Set nodesToCopy, boolean preserveUserAccess)</code>	Creates a hierarchy with a new copy of a tree base on the passed in node.
<code>IHierarchy createHierarchy(java.lang.String domain, java.lang.String hierarchyName, IHierarchyType hierarchyType)</code>	Creates a hierarchy and persist it to the DB.
<code>HierarchyExchangeResult exportXML(IHierarchy hierarchy, java.lang.String newHierarchyName)</code>	Deprecated. Exports to XML file a specified hierarchy from the point of view of the currently logged in user.
<code>java.io.InputStream exportXMLAsInputStream(IHierarchy hierarchy, java.lang.String newHierarchyName)</code>	Exports to XML file a specified hierarchy from the point of view of the currently logged in user.
<code>IHierarchy findHierarchy(java.lang.String companyId, java.lang.String hierName)</code>	Finds the hierarchy with corresponding company id and hierarchy name.
<code>IHierarchy[] findHierarchyForLinkTargetURI(IHierarchyType hierarchyType, java.lang.String linkTargetURI)</code>	Finds the hierarchies with the specific hierarchy type and containing the specific link target.
<code>IHierarchy[] findHierarchyForLinkTargetURI(IHierarchyType hierarchyType, java.lang.String linkTargetURI, IPeriod period)</code>	Finds the hierarchies with the given hierarchy type and containing the given link target for the given period.
<code>IHierarchyNode[] findNodeByLinkTargetId(java.lang.String linkTargetId)</code>	Finds all nodes in all hierarchies that match a link target id.
<code>IHierarchyNode[] findNodeByLinkTargetId(java.lang.String linkTargetId, IPeriod period)</code>	Finds all nodes in all hierarchies that match a link target id in specific period.
<code>IHierarchyNode[] findNodeByLinkTargetType(java.lang.String linkTargetType)</code>	Finds all nodes in all hierarchies that match a link target type.
<code>IHierarchyNode[] findNodeByLinkTargetType(java.lang.String linkTargetType, IPeriod period)</code>	Finds all nodes in all hierarchies that match a link target type in specific period.

Method	Description
<code>linkTargetType, IPeriod period)</code>	target type in specific period.
<code>IHierarchyNode[] findNodeByLinkTargetTypeAndDisplayName(java.lang.String linkTargetType, java.lang.String linkTargetDisplayName)</code>	Finds all nodes in all hierarchies that match a link target type and link target display name.
<code>IHierarchyNode[] findNodeByLinkTargetTypeAndDisplayName(java.lang.String linkTargetType, java.lang.String linkTargetDisplayName, IPeriod period)</code>	Finds all nodes in all hierarchies in the specific period that match the given link target type and link target display name.
<code>IHierarchyNode[] findNodeByLinkTargetTypeAndLinkTargetId(java.lang.String linkTargetType, java.lang.String linkTargetId)</code>	Finds all nodes in all hierarchies that match a link target type and link target id.
<code>IHierarchyNode[] findNodeByLinkTargetTypeAndLinkTargetId(java.lang.String linkTargetType, java.lang.String linkTargetId, IPeriod period)</code>	Finds all nodes in all hierarchies that match a link target type and link target id in the specified period.
<code>IHierarchyNode[] findNodeByLinkTargetURI(java.lang.String uri)</code>	Finds all nodes that have the matched URI from all hierarchies.
<code>IHierarchyNode[] findNodeByLinkTargetURI(java.lang.String uri, IPeriod period)</code>	Finds all nodes with link targets having matched URI from all hierarchies for the given period.
<code>IHierarchyNode[] findRootNodeForUser(java.lang.String userId)</code>	Finds all top level nodes the user has access to across all hierarchies.
<code>IHierarchyNode[] findRootNodeForUser(java.lang.String userId, IPeriod period)</code>	Finds all top level nodes the user has access to across all hierarchies for a given period.
<code>IUser getActor()</code>	Gets the current user who is managing this hierarchy.
<code>IHierarchy[] getDeletedHierarchies()</code>	Gets all hierarchies marked as deleted.
<code>IHierarchy[] getExpirableHierarchyCreatedBy(IHierarchyType hierarchyType, java.lang.String userId)</code>	Finds all hierarchies of specific type that can be expired by the given user.
<code>IHierarchy[] getExpirableHierarchyForCompany(IHierarchyType hierarchyType, java.lang.String domainId)</code>	Finds all hierarchies of specific type that can be expired for the current domain(company).
<code>IHierarchy[] getHierarchies()</code>	Gets all hierarchies currently existed in the database.
<code>IHierarchy[] getHierarchies(IHierarchyType hierarchyType)</code>	Gets all hierarchies for a given hierarchy type.
<code>IHierarchy[] getHierarchies(IUser user, IHierarchyType type)</code>	Gets all hierarchies of a certain type the specified user has access to.

Method	Description
<code>IHierarchy[] getHierarchiesCreatedByUserOfType(java.lang. String userId, IHierarchyType hierarchyType)</code>	Finds all hierarchies of specific type that are created by the given user.
<code>IHierarchy[] getHierarchiesForDomain(java.lang.String domainName)</code>	Returns all the hierarchies for a company.
<code>IHierarchy[] getHierarchiesForDomain(java.lang.String domainName, IHierarchyType hierarchyType)</code>	Gets all hierarchies for a company with specified hierarchy type.
<code>IHierarchy[] getHierarchiesForUser(java.lang.String userId)</code>	Gets all hierarchies the user has access to.
<code>IHierarchy[] getHierarchiesForUser(java.lang.String userId, IHierarchyType hierarchyType)</code>	Gets all hierarchies of given type for a given user.
<code>IHierarchy getHierarchy(java.lang.Long hierarchyID)</code>	Returns a hierarchy with an internal id that is useful on the UI, where hierarchy is saved in the session instead of in the hierarchy object.
<code>IHierarchyNode getHierarchyNode(java.lang.Long hierarchyNodeId)</code>	Returns a hierarchy node with an internal id (database key).
<code>HierarchyExchangeResult importXML(java.io.InputStream hierarchyFile)</code>	Imports a hierarchy from an XML file into the system.
<code>HierarchyExchangeResult importXML(java.io.InputStream hierarchyFile, IPeriod startPeriod, IPeriod endPeriod)</code>	Imports a hierarchy from an XML file into the system.
<code>void purgeHierarchy(IHierarchy hierarchy)</code>	Permanently deletes hierarchy from database.
<code>void purgeHierarchy(java.lang.String hierarchyDomain, java.lang.String hierarchyName)</code>	Permanently deletes the inactive hierarchy from persistent storage.
<code>void removeAllAccessForUser(java.lang.String userExternalID)</code>	Removes this user's access from all hierarchies.
<code>void removeAllHierarchyForCompany(java.lang.Str ing companyId)</code>	Removes all the hierarchies for a company.
<code>void setActor(IUser actor)</code>	Sets the current user who is managing this hierarchy.
<code>void validateXML(java.io.InputStream hierarchyFile)</code>	Validates hierarchy XML file based on XML schema provided in common-hierarchy-interchange-1.0.xsd and instance-hierarchy-interchange-1.0.xsd.

IHierarchyNode

This interface provides a contract for a hierarchy node. It expresses the parent-child relationship of a hierarchy, and has no direct business meaning. A node will only have business meaning after it is linked to a business object. The list of business objects include IAccount, IService, ICharge.

This object has the following properties:

- **id:** The internal id used to identify this hierarchy node.
- **alias:** The alias given to this node.
- **isContainer:** Whether or not this node is a container. Only container nodes can have children.
- **createdAt:** The date when the node was created.
- **createdBy:** The user id who created this node.
- **deletedAt:** The date when this node was marked as deleted.
- **description:** The description of the node.
- **linkTargetId:** The id of the link target of the node. For example, account number for IAccount.
- **linkTargetName:** The name of the link target of the node.
- **linkTargetType:** The type of the link target of the node.
- **linkTargetURI:** The URI of the link target of the node.

Method	Description
<code>void addAttribute(IAttribute attribute)</code>	Adds an attribute to the attribute list of the node.
<code>void addAttributes(java.util.Collection attributes)</code>	Adds a collection of attributes to the attribute list of the node.
<code>IHierarchyNode addLinkTarget(IHierarchyLinkTarget child)</code>	Creates a new node whose link target is as specified and add that node as the child node of this node object.
<code>void addUserAccess(java.lang.String userId)</code>	Gives a user access to this node.
<code>IHierarchyNode[] findByOMFTypeAndAttributes(java.lang.String omfType, IAttribute[] attributes)</code>	Deprecated. Please use <code>#findNodeByLinkTargetTypeAndCriteria(String, com.edocs.common.api.omf.search.ISearchCriteria)</code> instead Finds all nodes that are the descendants of this node whose link target types match the specified OMF object type and node attributes match the ones in the specified attributes.
<code>IHierarchyNode[] findNodeByLinkIdTargetId(java.lang.String linkTargetId)</code>	Finds all nodes that match the specified link target id, beginning at this node and searching recursively to the child nodes.

Method	Description
<code>IHierarchyNode[] findNodeByLinkTargetType(java.lang.String linkTargetType)</code>	Finds all nodes that are the descendants of this node whose link target types match the specified link target type.
<code>IHierarchyNode[] findNodeByLinkTargetTypeAndCriteria(java.lang.String linkTargetType, ISearchCriteria criteria)</code>	Finds all nodes that are the descendants of this node whose link target type matches the specified link target type and link target object match the specified criteria.
<code>IHierarchyNode[] findNodeByLinkTargetTypeAndDisplayName(java.lang.String linkTargetType, java.lang.String linkTargetDisplayName)</code>	Finds all nodes that are the descendants of this node whose link target types match the specified link target type and link target display names match the specified link target display name.
<code>IHierarchyNode findNodeByLinkTargetURI(java.lang.String linkTargetURI)</code>	Finds a node that matches the specified link target URI, beginning at this node and searching recursively to the child nodes.
<code>IOMFObject[] findOMFObject(java.lang.String userId, java.lang.String elementType, HierSearchType status, IAttribute[] attributes)</code>	Deprecated. Please use <code>IHierarchyAccessManager#findAssignedOMFObjects(com.edocs.common.api.hierarchy.IHierarchyNode, String, com.edocs.common.api.omf.search.ISearchCriteria)</code> or <code>IHierarchyAccessManager#findUnassignedOMFObjects(com.edocs.common.api.hierarchy.IHierarchy, String, com.edocs.common.api.omf.search.ISearchCriteria)</code>
<code>IHierarchyNodeObjwrapper[] findOMFObject(java.lang.String userId, java.lang.String elementType, HierSearchType status, java.lang.String attrName, java.lang.Object attrValue)</code>	Deprecated. Please use <code>IHierarchyAccessManager#findAssignedOMFObjects(com.edocs.common.api.hierarchy.IHierarchyNode, String, com.edocs.common.api.omf.search.ISearchCriteria)</code> or <code>IHierarchyAccessManager#findUnassignedOMFObjects(com.edocs.common.api.hierarchy.IHierarchy, String, com.edocs.common.api.omf.search.ISearchCriteria)</code>
<code>java.util.List findUsers(HierSearchType status, IAttribute[] attributes)</code>	Deprecated in 2.0
<code>java.util.List findUsers(HierSearchType searchType, java.lang.String attrName, java.lang.Object attrValue)</code>	Finds users (Assigned, Unassigned, Authorized or Unauthorized as specified in HierSearchType) that match the specified attribute name and value within the sub tree starting from the current node.
<code>java.lang.String getAlias()</code>	Each node can be given an alias.

Method	Description
<code>java.util.Set getAllAuthorizedUsers()</code>	Gets all the users who have access to this node.
<code>java.util.Collection getAllUsers()</code>	Returns a collection of users having access to this node and its parent nodes.
<code>java.lang.Object getAttribute(java.lang.String attributeName)</code>	Gets the value of the attribute for a given name.
<code>java.util.Set getAttributes()</code>	Gets all the attributes for this node.
<code>java.util.Set getAttributes(java.lang.String attributeName)</code>	Gets all attributes of the node whose names match the specified attribute name.
<code>java.util.Set getAuthorizedUsers()</code>	Gets all the users who are directly assigned to access this node.
<code>java.util.Set getChildren()</code>	Gets all the immediate children nodes of this node.
<code>java.util.Set getChildren(boolean initializeHasChildren)</code>	Get all the immediate children nodes of this node.
<code>java.util.Set getChildren(boolean initializeHasChildren, IHierarchyNode startFromNode, int fetchSize)</code>	Returns a list of IHierarchyNode objects that represent as child nodes of the current node.
<code>java.util.Set getChildren(boolean initializeHasChildren, int startPosition, int fetchSize)</code>	Returns a list of IHierarchyNode objects that represent as child nodes of the current node.
<code>java.util.Set getChildrenOfType(java.lang.String omfObjectType)</code>	Returns a set of immediate children whose link target types match the specified type.
<code>java.util.Set getChildrenOfType(java.lang.String omfObjectType, java.util.Collection attributes)</code>	Returns a set of immediate children whose link target types match the specified type and whose node attributes match those specified.
<code>java.util.Date getCreatedAt()</code>	Gets the date when the node was created.
<code>java.lang.String getCreatedBy()</code>	Gets the user id the node was created.
<code>java.util.Date getDeletedAt()</code>	Gets the date when the node was marked as deleted.
<code>java.lang.String getDescription()</code>	Gets the description property of this node.
<code>java.util.Date getExpiredate()</code>	Gets the date when the node is removed.
<code>IHierarchyNodeHandle getHandle()</code>	Gets the hierarchy node handle that can be serialized and passed across the net.
<code>IHierarchy getHierarchy()</code>	Gets the current hierarchy object which this node belongs to.
<code>java.lang.String getHierarchyName()</code>	Gets the name of the hierarchy this node is associated with.

Method	Description
<code>java.lang.Long getID()</code>	Returns the internal node id.
<code>IHierarchyLinkTarget getLinkTarget()</code>	Gets the link target object of this nodes
<code>java.lang.String getLinkTargetExtKey()</code>	Gets link target external key.
<code>java.lang.String getLinkTargetID()</code>	Gets the id of the link target of this node.
<code>java.lang.String getLinkTargetName()</code>	Gets the name of the link target of this node.
<code>java.lang.String getLinkTargetType()</code>	Gets the type of the link target for this node.
<code>java.lang.String getLinkTargetURI()</code>	Gets the URI of the link target of this node.
<code>java.lang.String getOmfoObjectType()</code>	Deprecated. use <code>{#getLinkTargetType}</code> instead.
<code>java.lang.String getOmfoObjectURI()</code>	Gets the URI of the link target of this node.
<code>IHierarchyNode getParent()</code>	Gets the parent node of this node.
<code>java.lang.String getPath()</code>	This is the absolute path from the root node to this node.
<code>int getScrollPosition()</code>	Gets the position and focus of current child nodes.
<code>java.util.Collection getUsers()</code>	Returns a collection of IUser objects assigned to this node.
<code>java.util.List getUsersAssignedToNodeOrAnyAncestor()</code>	Gets all the users who have access to this node or any of its ancestor nodes.
<code>java.util.List getUsersAssignedToNodeOrAnyDescendent()</code>	Gets all the users who have access to this node or any of its descendant(child) nodes.
<code>boolean hasChildren()</code>	Use this method to check if the node has any children without loading the entire children node.
<code>boolean hasUser()</code>	Indicates whether the node has any users associated with.
<code>boolean isDeleted()</code>	Deprecated. in 2.0, with versioning it doesn't make sense to have a deleted node. Checks whether this node has been marked as deleted.
<code>boolean isFolder()</code>	Checks whether this node is a folder.
<code>boolean isRoot()</code>	Checks whether this node is a root node.
<code>void move(IHierarchyNode destination)</code>	Moves this node to a different parent in the same hierarchy or between two different hierarchies.
<code>void moveUser(IHierarchyNode destination, java.lang.String userid)</code>	Re-associate the given user from current node to the destination node.
<code>void remove()</code>	Marks this hierarchy node and its children as deleted.

Method	Description
<code>void removeAllAttributes()</code>	Removes all attributes for this node.
<code>IAttribute removeAttribute(java.lang.String name, java.lang.String value)</code>	Removes the attribute with specified name and value from this node.
<code>void removeAttributes(java.lang.String name)</code>	Removes all attributes whose names are the same as the specified name.
<code>void removeUserAccess(java.lang.String userId)</code>	Removes user access from node.
<code>void setAlias(java.lang.String alias)</code>	Gives this node an alias.
<code>void setDescription(java.lang.String description)</code>	Sets the description property of this node.
<code>void setLinkTargetName(java.lang.String name)</code>	Sets the name of the link target of this node.
<code>void update()</code>	Persists any changes made to the node.
<code>void update(IHierarchyLinkTarget linkTarget)</code>	Updates the current node value using the information from given link target.

IHierarchyNodeHandle

This interface provides a protocol to access a hierarchy node. `IHierarchyNodeHandle` is a light version of a hierarchy node. The information stored here can be used to retrieve `IHierarchyNode` when full information about the node is required. The handle class has been used by higher layer, such as UI service layer to build UI element of the tree.

Method	Description
<code>IHierarchyNode getHierarchyNode()</code>	Get the hierarchy node corresponding to this handle.
<code>java.lang.String getLinkTargetName()</code>	Gets link target name for displaying hierarchy node.
<code>java.lang.String getNodeid()</code>	Returns string identifier for the hierarchy node (this is usually a Long value but may change depending on implementation).
<code>java.lang.Long getParentId()</code>	Gets unique identifier of the parent node.
<code>java.lang.String getParentLinkTargetName()</code>	Gets the link target name for parent node.
<code>java.lang.String getPath()</code>	Gets the path this node holds in the hierarchy

Method	Description
<code>java.lang.Long getVersion()</code>	The version number of hierarchy node represented by this handle, this is the db version number and NOT related to versioned hierarchies

IHierarchyService

This interface provides a contract for creating a hierarchy service that provides an entry-point to the hierarchy module. Implement this interface to create important Hierarchy components such as IHierarchyManager.

Use XMA to obtain an instance of IHierarchyService as follows:

- 1 `LookupService lookup: LookupServiceFactory.getInstance();`
- 2 `IHierarchyService hierarchyService: (IHierarchyService) lookup.getModule("hierarchy");`
- 3 Use this interface to create hierarchy managers and filter queries on hierarchies.

Method	Description
<code>IAttribute createAttribute(java.lang.String name, java.lang.Object value)</code>	Create an instance of IAttribute, which represents a Hierarchy node attribute.
<code>IExpression createExpression()</code>	Create an instance of IExpression.
<code>IFilteredQuery createFilteredQueryForIHierarchy()</code>	Create an instance of IFilteredQuery to manage hierarchy query-related operations.
<code>IFilteredQuery createFilteredQueryForIHierarchyNode()</code>	Create an instance of IFilteredQuery to manage hierarchy node query-related operations.
<code>IHierarchyAccessManager createHierarchyAccessManager(IHierarchyContext hierarchyContext)</code>	Return a singleton of the IHierarchyAccessManager to manage hierarchy access.
<code>IHierarchyFolderManager createHierarchyFolderManager()</code>	Create an instance of IHierarchyFolderManager to manage hierarchy folder-related operations.
<code>IHierarchyManager createHierarchyManager(IUser user)</code>	Create an instance of IHierarchyManager to manage hierarchy-related operations.
<code>IHierarchyTypeManager createHierarchyTypeManager()</code>	Create an instance of IHierarchyTypeManager to manage hierarchy type-related operations.

Method	Description
<code>java.util.List getObjectSearchTypes(OMFTypeProperty otp)</code>	Return a list of HierSearchType objects for the specified OMFTypeProperty
<code>java.util.List getObjectSearchTypes(OMFTypeProperty omfType, IHierarchyType hierType)</code>	Return a list of hierarchy search types for specified omf type within given type of hierarchy.

IHierarchyType

This interface represents the type of hierarchy. Note that this object is persistent.

Method	Description
<code>java.lang.String getCode()</code>	A unique code to identify this hierarchy type.
<code>java.lang.String getDescription()</code>	Gets the description of the hierarchy type
<code>java.lang.Long getID()</code>	Gets the internal ID used to identify this hierarchy type.
<code>java.lang.String getName()</code>	Gets the name of the hierarchy type.

IHierarchyTypeManager

This object is used to manage hierarchy types. You can get an instance of this object through IHierarchyService.

Method	Description
<code>boolean canContain(IHierarchyType hierarchyType, OMFTypeProperty parentType, OMFTypeProperty childType)</code>	Returns true if the given type of parent and child can form a valid relationship in the specified type of hierarchy.
<code>IHierarchyType getHierarchyType(java.lang.String code)</code>	Gets an instance of IHierarchy by its unique code
<code>java.util.List getHierarchyTypes()</code>	Gets all the supported hierarchy types.
<code>java.lang.String getOMFObjectTypeName(java.lang.String omfType)</code>	Gets the name of the link target with the specified object type
<code>java.util.List getOMFObjectTypes(IHierarchyType hierarchyType)</code>	Each hierarchy type allows a list of link target objects to be linked to it.

Method	Description
java.util.List getOMFTypePropertyList(IHierarchyType hierarchyType)	This function calls the getOMFObjectTypes(IHierarchyType hierarchyType) internally, and return OMF type and the type name pairs in a list

IHierarchyUserRef

This interface represents a user being assigned to a hierarchy node.

Method	Description
java.lang.String getExternalID()	Get the user id.

ILinkTargetConfig

This interface provides a contract for accessing the link target configuration. The link target provides business meaning to a hierarchy node.

Method	Description
java.lang.String getDescription()	Gets the method name which will be used to retrieve description string.
java.lang.String getDisplayName()	Gets the method name for displaying display name.
java.lang.String getExternalKey()	Returns the method name to be used as display external key properties.
java.util.List getLinkTargetEventHandlers()	Returns a list of event handler class names.
java.lang.String getTargetType()	Returns target type string for the link target.
java.lang.String getTargetTypeName()	Gets name string assigned for given target class.
IUnassignedObjectProvider getUnassignedObjectProvider()	Gets the name of class for retrieving unassigned objects.
IHierarchyXMLExchangeHandler getXMLExchangeHandler()	Returns full class name of XML exchange handler for handle this type of element in the XML import process.
java.lang.String getXMLTag()	Returns XML element tag name used for this type of link target.

Method	Description
boolean isStoredHierXRef()	Returns a Boolean indicating whether this type of link target, when handled in the OLAP side, is flattened out outside the hierarchy node table.

7

Appendix A Hierarchy XML Exchange Schema

Located at C:\Siebel\CBA\estatement\config\xml\ common-hierarchy-interchange-1.0.xsd

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" version="1.0">
  <xs:annotation>
    <xs:documentation>
      This file contains the XML schema definition that edocs hierarchy module
      uses for interchanging hierarchical business structures.
      Do not modify this file for deployment specific requirements.
      Any deployment specific information should be made in the
      instance-hierarchy-interchange-1.0.xsd schema definition file.
    </xs:documentation>
  </xs:annotation>
  <xs:include schemaLocation="instance-hierarchy-interchange-1.0.xsd">
    <xs:annotation>
      <xs:documentation>
        Includes the document containing instance/deployment specific schema details.
      </xs:documentation>
    </xs:annotation>
  </xs:include>
  <xs:element name="ListOfHierarchies">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="DeltaHierarchy" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Move" maxOccurs="unbounded" minOccurs="0">
                <xs:complexType>
                  <xs:all>
                    <xs:element name="SrcHierarchy" type="HierarchyDef"/>
                    <xs:element name="SrcNode" type="DeltaNodeDef"/>
                    <xs:element name="DestHierarchy" type="HierarchyDef"/>
                    <xs:element name="DestNode" type="DeltaNodeDef"/>
                  </xs:all>
                </xs:complexType>
              </xs:element>
              <xs:element name="Add" maxOccurs="unbounded" minOccurs="0">
                <xs:complexType>
                  <xs:all>
                    <xs:element name="SrcNode" type="DeltaNodeDef"/>
                    <xs:element name="DestHierarchy" type="HierarchyDef"/>
                    <xs:element name="DestNode" type="DeltaNodeDef"/>
                  </xs:all>
                </xs:complexType>
              </xs:element>
              <xs:element name="Delete" maxOccurs="unbounded" minOccurs="0">
                <xs:complexType>
                  <xs:all>
                    <xs:element name="SrcNode" type="DeltaNodeDef"/>
                    <xs:element name="DestHierarchy" type="HierarchyDef"/>
                  </xs:all>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```

        <xs:element name="CompleteHierarchy" type="Hierarchy" minOccurs="0"
maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
</xs:element>
<xs:complexType name="DeltaNodeDef">
    <xs:sequence>
        <xs:element name="BusinessObject" type="BusinessObjectType"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="Hierarchy">
    <xs:annotation>
        <xs:documentation>
            Always the hierarchy will be created using the user given name,
            and the name in the XML file will be ignored.
        </xs:documentation>
    </xs:annotation>
    <xs:sequence>
        <xs:element name="AcceptableBusinessObjectTypes" type="AcceptableBusinessObject"
minOccurs="0"/>
        <xs:element name="RootNode" type="HierarchyNode"/>
    </xs:sequence>
    <xs:attributeGroup ref="HierarchyAttrs"/>
</xs:complexType>
<xs:complexType name="AcceptableBusinessObject">
    <xs:annotation>
        <xs:documentation>
            A list of xml tag names of acceptable business object types to be
            considered when importing a hierarchy.
            If there were any tags which were not specified here in the XML file,
            then those and their child nodes will be ignored.
            When overriding an existing hierarchy, any nodes with any other
            type than specified here will be unaltered by the import process.
            If this section is not present then the content of the whole XML file will be imported.
        </xs:documentation>
    </xs:annotation>
    <xs:sequence maxOccurs="unbounded">
        <xs:element name="tagName" type="xs:string"/>
    </xs:sequence>
</xs:complexType>
<xs:attributeGroup name="HierarchyAttrs">
    <xs:attribute name="domainID" type="xs:string" use="optional">
        <xs:annotation>
            <xs:documentation>
                When the domain ID is not available, user's domain ID will be used.
            </xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="type" type="HierarchyType" use="optional">
        <xs:annotation>
            <xs:documentation>
                When the hierarchy type is not available, user's default hierarchy type will be used
                which may be set by the importer/exporter hooks.
            </xs:documentation>
        </xs:annotation>
    </xs:attribute>
    <xs:attribute name="name" type="xs:string" use="required"/>
    <xs:attribute name="displayName" type="xs:string" use="optional">
        <xs:annotation>
            <xs:documentation>
                When the display name is not available, name will be used.
            </xs:documentation>
        </xs:annotation>
    </xs:attribute>

```

```

</xs:attribute>
<xs:attribute name="period" type="xs:date" use="optional">
  <xs:annotation>
    <xs:documentation>
      Period is optional and will be ignored for the first release.
    </xs:documentation>
  </xs:annotation>
</xs:attribute>
</xs:attributeGroup>
<xs:complexType name="NodeList">
  <xs:sequence maxOccurs="unbounded">
    <xs:element name="Node" type="HierarchyNode"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="AccessDef">
  <xs:sequence>
    <xs:element name="userId" type="xs:string" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="HierarchyNode">
  <xs:sequence>
    <xs:element name="BusinessObject" type="BusinessObjectType"/>
    <xs:element name="CanBeAccessedBy" type="AccessDef" minOccurs="0"/>
    <xs:element name="ChildNodeList" type="NodeList" minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="FolderDef">
  <xs:annotation>
    <xs:documentation>
      Folder is the default business object type.
    </xs:documentation>
  </xs:annotation>
<xs:all>
  <xs:element name="Description" type="xs:string" minOccurs="0"/>
  <xs:element name="AttributeList" minOccurs="0">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Attribute" minOccurs="0" maxOccurs="unbounded">
          <xs:complexType>
            <xs:attribute name="name" type="xs:string"/>
            <xs:attribute name="value" type="xs:string"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:all>
  <xs:attribute name="name" type="xs:string"/>
  <xs:attribute name="externalID" type="xs:string"/>
</xs:complexType>
<xs:complexType name="ServiceAgreementDef">
  <xs:sequence>
    <xs:element name="ExtAttr1" type="xs:string" minOccurs="0"/>
    <xs:element name="ExtAttr2" type="xs:string" minOccurs="0"/>
    <xs:element name="ExtAttr3" type="xs:string" minOccurs="0"/>
    <xs:element name="ExtAttr4" type="xs:string" minOccurs="0"/>
    <xs:element name="ExtAttr5" type="xs:string" minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="serviceNo" use="required">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:minLength value="1"/>
        <xs:maxLength value="255"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>

```

```

        </xs:restriction>
    </xs:simpleType>
</xs:attribute>
<xs:attribute name="accountNo" use="optional">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="255"/>
        </xs:restriction>
    </xs:simpleType>
</xs:attribute>
<xs:attribute name="billerId" use="optional">
    <xs:simpleType>
        <xs:restriction base="xs:string">
            <xs:minLength value="1"/>
            <xs:maxLength value="255"/>
        </xs:restriction>
    </xs:simpleType>
</xs:attribute>
</xs:complexType>
<xs:complexType name="ServiceChargeDef">
    <xs:attribute name="serviceNo" use="required">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:minLength value="1"/>
                <xs:maxLength value="255"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="accountNo" type="xs:string"/>
    <xs:attribute name="billerId" type="xs:string"/>
    <xs:attribute name="chargeType" use="required">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:minLength value="1"/>
                <xs:maxLength value="32"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
</xs:complexType>
<xs:complexType name="CompanyDef">
    <xs:attribute name="fiscalCode" type="xs:string"/>
    <xs:attribute name="companyTitle" type="xs:string"/>
</xs:complexType>
<xs:complexType name="HierarchyDef">
    <xs:attributeGroup ref="HierarchyAttrs"/>
</xs:complexType>
<xs:complexType name="AccountDef">
    <xs:attribute name="accountNo" use="required">
        <xs:simpleType>
            <xs:restriction base="xs:string"><xs:minLength value="1"/>
                <xs:maxLength value="255"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="billerId" use="optional">
        <xs:simpleType>
            <xs:restriction base="xs:string">
                <xs:minLength value="1"/>
                <xs:maxLength value="255"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>

```

```
</xs:complexType>
<xs:complexType name="BusinessObjectType">
  <xs:choice>
    <xs:group ref="DeploymentSpecificBusinessObjectType"/>
    <xs:element name="Folder" type="FolderDef"/>
    <xs:element name="ServiceAgreement" type="ServiceAgreementDef"/>
    <xs:element name="ServiceCharge" type="ServiceChargeDef"/>
    <xs:element name="Company" type="CompanyDef"/>
    <xs:element name="Account" type="AccountDef"/>
  </xs:choice>
</xs:complexType>
</xs:schema>
```


8

Appendix B: Sample Hierarchy XML File

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ListOfHierarchies xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="common-hierarchy-interchange-1.0.xsd">
  <CompleteHierarchy name="Jul_AmericanHighTech" domainID="American HighTech1" type="BILLING"
displayName="Jul_AmericanHighTech">
    <RootNode>
      <BusinessObject>
        <Company fiscalCode="American HighTech1" companyTitle="American HighTech1"/>
      </BusinessObject>
      <ChildNodeList>
        <Node>
          <BusinessObject>
            <Account accountNo="1|29006120" billerId="1"/>
          </BusinessObject>
          <ChildNodeList>
            <Node>
              <BusinessObject>
                <ServiceAgreement serviceNo="5458039028"
accountNo="29006120" billerId="1"/>
              </BusinessObject>
            </Node>
          </ChildNodeList>
        </Node>
        <Node>
          <BusinessObject>
            <Account accountNo="1|31569801" billerId="1"/>
          </BusinessObject>
          <ChildNodeList>
            <Node>
              <BusinessObject>
                <ServiceAgreement serviceNo="4943929463"
accountNo="31569801" billerId="1"/>
              </BusinessObject>
            </Node>
            <Node>
              <BusinessObject>
                <ServiceAgreement serviceNo="4943942893"
accountNo="31569801" billerId="1"/>
              </BusinessObject>
            </Node>
          </ChildNodeList>
        </Node>
        <Node>
          <BusinessObject>
            <Account accountNo="1|41251761" billerId="1"/>
          </BusinessObject>
          <ChildNodeList>
            <Node>
              <BusinessObject>
                <ServiceAgreement serviceNo="7379289372"
accountNo="41251761" billerId="1"/>
              </BusinessObject>
            </Node>
            <Node>
              <BusinessObject>
            </BusinessObject>
            </Node>
          </ChildNodeList>
        </Node>
      </ChildNodeList>
    </RootNode>
  </CompleteHierarchy>
</ListOfHierarchies>
```

```

        <ServiceAgreement serviceNo="7379830382"
            accountNo="41251761" billerId="1"/>
    </BusinessObject>
</Node>
</ChildNodeList>
</Node>
<Node>
    <BusinessObject>
        <Account accountNo="1|5128140" billerId="1"/>
    </BusinessObject>
    <ChildNodeList>
        <Node>
            <BusinessObject>
                <ServiceAgreement serviceNo="4513783743" accountNo="5128140" billerId="1"/>
            </BusinessObject>
        </Node>
        <Node>
            <BusinessObject>
                <ServiceAgreement serviceNo="4514724956" accountNo="5128140" billerId="1"/>
            </BusinessObject>
        </Node>
        <Node>
            <BusinessObject>
                <ServiceAgreement serviceNo="4519382734" accountNo="5128140" billerId="1"/>
            </BusinessObject>
        </Node>
    </ChildNodeList>
</Node>
<Node>
    <BusinessObject>
        <Account accountNo="1|61362310" billerId="1"/>
    </BusinessObject>
    <ChildNodeList>
        <Node>
            <BusinessObject>
                <ServiceAgreement serviceNo="3184732174"
                    accountNo="61362310" billerId="1"/>
            </BusinessObject>
        </Node>
    </ChildNodeList>
</Node>
<Node>
    <BusinessObject>
        <Account accountNo="1|71385461" billerId="1"/>
    </BusinessObject>
    <ChildNodeList>
        <Node>
            <BusinessObject>
                <ServiceAgreement serviceNo="7634076300"
                    accountNo="71385461" billerId="1"/>
            </BusinessObject>
        </Node>
    </ChildNodeList>
</Node>
<Node>
    <BusinessObject>
        <Account accountNo="1|80011008" billerId="1"/>
    </BusinessObject>
    <ChildNodeList>
        <Node>
            <BusinessObject>
                <ServiceAgreement serviceNo="4070620806"
                    accountNo="80011008" billerId="1"/>
            </BusinessObject>
        </Node>
    </ChildNodeList>
</Node>

```

```
        </BusinessObject>
    </Node>
  <Node>
    <BusinessObject>
      <ServiceAgreement serviceNo="4071135451"
        accountNo="80011008" billerId="1"/>
    </BusinessObject>
  </Node>
</ChildNodeList>
</Node>
</ChildNodeList>
</RootNode>
</CompleteHierarchy>
</ListOfHierarchies>
```