Oracle® Configurator

Oracle Configuration Interface Object (CIO) Developer's Guide

Release 11*i*

March 2000

Part No. A81001-03

This document describes Functional Companions, which augment the functionality of an Oracle SellingPoint application, and the Oracle Configuration Interface Object (CIO), which is used by Functional Companions to access the Oracle Configurator Active Model.

ORACLE®

Oracle Configuration Interface Object (CIO) Developer's Guide, Release 11*i*

Part No. A81001-03

Primary Author: Mark Sawtelle

Contributors: Brent Benson, Jim Carlson, Ivan Lazarov, Marty Plotkin, Brian Ross

Contents

2 The Configuration Interface Object (CIO)

List of Examples

List of Figures

List of Tables

Send Us Your Comments

Oracle Configuration Interface Object (ClO) Developer's Guide, Release 11*i*

Part No. A81001-03

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available). You can send comments through your call to Oracle Support Services or by sending them to:

Oracle Configurator
Oracle Corporation
Documentation
21 North Avenue
Burlington, MA 01803
USA

If you would like a reply, please give your name, address, and telephone number below.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

You can use Functional Companions to augment the functionality of your Oracle SellingPoint application beyond what is provided by Oracle Configurator Developer. You create Functional Companion objects, which use the Configuration Interface Object (CIO) to perform various tasks, including accessing the Model, setting and getting logic states, and adding optional components. You can also use the CIO in your own applications, to interact with the Model.

Intended Audience

This manual is intended primarily for software developers writing Functional Companions. The language recommended for developing Functional Companions is Java.

This manual assumes that you are an experienced programmer and that you understand Oracle databases, the SQL and Java programming languages, and the principles of JDBC.

> **Note:** For specialized purposes, Functional Companions can be written in Oracle's GSL (Generative Specification Language), and by using COM with the Microsoft Java Virtual Machine. This is only possible on Windows 95/98 and Windows NT 4.0.

This manual also provides background and reference information on the CIO, which is needed by developers of applications having customized user interfaces that need access to the Oracle Configurator Active Model.

Structure

This manual contains:

- Chapter 1, "Functional Companions"
- Chapter 2, "The Configuration Interface Object (CIO)"
- Chapter 3, "Reference Documentation for the CIO"
- Chapter 4, "Examples"

Related Documents

For more information, see the following manuals in Release 11*i* of the Oracle Configurator documentation set:

- *Oracle SellingPoint Configurator Administration Guide*
- *Oracle Configurator Developer User's Guide*
- *Oracle SellingPoint CompanionBuilder Help*
- *Oracle Configurator Developer Tutorial*

The following documents are also relevant:

- *Oracle8i JDBC Developer's Guide and Reference*
- Oracle White Paper: "Using COM with Oracle SellingPoint 4.2" (Available through Products Online, http://products.us.oracle.com. Look under "Supply Chain", "SellingPoint", "Collateral".)

Conventions

In examples, an implied carriage return occurs at the end of each line, unless otherwise noted. You must press the Return key at the end of a line of input.

The following conventions are also used in this manual:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.

Convention	Meaning
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement orcommand not directly related to the example have been omitted
boldface text	Boldface type in text indicates a term defined in the text, the glossary, or in both locations.
< >	Angle brackets enclose user-supplied names.
[ ]	Brackets enclose optional clauses from which you can choose one or none.
>	The left bracket alone sign represents the MS DOS prompt.

1

Functional Companions

Functional Companions extend your Oracle SellingPoint application by attaching custom code through established interfaces.

1.1 What Are Functional Companions?

A Functional Companion is a programming object that you attach to your Model in order to extend the functionality of your Oracle SellingPoint application in ways that are not provided by Oracle Configurator Developer.

You can write a Functional Companion object in several languages, depending on the functionality needed by your application. The Functional Companion communicates with your Model through an API (application programming interface) called the Configuration Interface Object (CIO). The Oracle Configuration Interface Object is written in Java. See Chapter 2, "The Configuration Interface Object (CIO)".

You connect Functional Companions to specific nodes in your Model using Oracle Configurator Developer. You also specify the type of action that you want the specified Functional Companion to perform when your end users select its associated node. Then you generate the logic and user interface, as you normally do for your Oracle SellingPoint application. This action associates the Functional Companion with your application so that when your end users select a node in the Model, the Functional Companion on that node is automatically invoked.

1.1.1 Types of Functional Companions

You can assign a Functional Companion to perform any or all of these three types of actions:

Table 1–1 Types of Functional Companions

Type	Description
Auto-configuration	Configures the state of the Model. You can use this to modify the shape of the Model tree, and the state of its nodes. For instance, your application might gather initial needs assessment information and use it to set up the appropriate set of choices for your end user to make. In your Oracle SellingPoint application, your end user will explicitly choose to run an auto-configuration Functional Companion See Section 2.9.2, "The autoConfigure() Interface Method".
Validation	Validates the logical choice that the end user has just made. The Functional Companion can perform complex operations beyond the scope of what you can develop in Oracle Configurator Developer. For instance, you can perform sophisticated numeric comparisons. A Java Functional Companion returns null if the validation is successful. If the validation fails, it returns a List of CompanionValidationFailure objects. A COM Functional Companion returns true if the validation is successful. If the validation fails, it returns a COM Array of Strings. In your Oracle SellingPoint application, all validation Functional Companions are run every time your end user chooses an Option. After each action, the end user gets the collection of strings returned by each Functional Companion that failed. Validation companions query the model to determine validity, but should *not* modify the model. Modifying the model in a validation Functional Companion can cause unexpected application failures. See Section 2.9.3, "The validate() Interface Method".
Output	Generates some form of output from the configuration. This output might be a report, a performance graph, a geometric rendering, or a graphical representation of the configuration. In your Oracle SellingPoint application, your end user will explicitly choose to run an output Functional Companion. See Section 2.9.4, "The generateOutput() Interface Method".

1.1.2 Background to Building Functional Companions

To build a Functional Companion, you implement an object class in the language that you choose as being most appropriate for the operation that you want to perform. The language choices are:

Java	This is the recommended choice for developing Functional Companions. Java Functional Companions can run on any platform supported by Java. The other language choices are recommended only for special purposes.
GSL	GSL (Generative Specification Language) is an Oracle proprietary object-oriented dynamic language recommended primarily for geometric visualization and modeling. Using it requires CompanionBuilder, which is only available on Windows 95/98 and Windows NT 4.0. The resultant Functional Companions can only be used in a "fat client" deployment, not, for example, in a web browser.
COM	Functional Companions can be written to the Microsoft COM standard (using Visual Basic, for instance), but are restricted to Windows 95/98 and Windows NT 4.0, and require the Microsoft Java Virtual Machine.

When an Oracle SellingPoint application runs, it creates an instance of the CIO, which creates runtime instances of all the Components in the Model. If you used Oracle Configurator Developer to associate a Functional Companion with a Component, then the application creates, for each instance of that Component, an instance of the class that you defined for your Functional Companion and attaches the Functional Companion instance to the Component.

You can associate more than one Functional Companion with a particular Component; the CIO will create instances of all of them.

If any Functional Companions cannot be loaded when you create a new configuration (for instance, due to internal errors or an incorrect CLASSPATH), the configuration will fail to open.

You can also associate Functional Companions with Products.

- For Functional Companions built with Java, you implement a class that extends `oracle.apps.cz.cio.FunctionalCompanion`. See Section 1.3, "Building Functional Companions in Java".

- For Functional Companions built with COM, you implement an object that supports IDispatch. See Section 1.4, "Building Functional Companions in COM". For details and more background, see the Oracle White Paper "Using COM with Oracle SellingPoint 4.2".
- For Functional Companions built with (GSL) (the Generative Specification Language), you implement a library object to be used as the basis of the companion instances. See the *Oracle SellingPoint CompanionBuilder Help* for details.
- For *all* language choices, you also implement one or more of the standard interface methods of `oracle.apps.cz.cio.IFunctionalCompanion`, which are described in Section 2.9, "Standard Interface Methods for Functional Companions".

In order to communicate with the Model of your application, the Functional Companion uses Oracle's CIO API. The CIO can also be used to develop a custom user interface for an Oracle SellingPoint application, in order to access the Model. As a point of information, both Oracle Configurator Developer and the default user interface for the Oracle SellingPoint application communicate in just this way with the Model, using the Oracle Configurator Database to store structure, rules, and user interface information (in addition to your end user's data).

1.2 Functional Companions and the CIO

Functional Companions are invoked by the CIO through the Oracle SellingPoint application, and Functional Companions call the CIO to get information from the Active Model. The CIO is like a broker for the Active Model, in that it passes information both ways. Programmers writing Functional Companions need to know how to use the CIO.

Each Functional Companion is an object class. For every Component instance in your Model that is associated with a Functional Companion, the CIO creates an instance of this class.

1.2.1 Using the CIO Interface

Your Functional Companion is a client of the CIO. When you program against the CIO, you create instances of a set of public interface objects, which are defined in `oracle.apps.cz.cio`.

Your code should refer only to these public interface objects. See Section 2.2, "The CIO's Runtime Node Interface Classes".

Reference

For reference documentation, see: Package oracle.apps.cz.cio.

1.2.2 Implementing Standard Interface Methods

You provide functionality for your Functional Companion by implementing body code for the methods:

- `initialize`
- `autoConfigure`
- `validate`
- `generateOutput`
- `terminate`

These methods are described in Section 2.9, "Standard Interface Methods for Functional Companions".

For particulars that apply to the languages currently supported by the CIO, and examples, see Section 1.3, "Building Functional Companions in Java", and the *Oracle SellingPoint CompanionBuilder Help.*

1.3 Building Functional Companions in Java

1.3.1 Procedure for Building Functional Companions in Java

Here is an overview of the tasks for Building Functional Companions in Java. See also Section 1.3.2, "Installation Requirements for Java Functional Companions".

1. Use a Java development environment or text editor to create a .java file in which to define a Java class.
2. Import the classes for the CIO (`oracle.apps.cz.cio.*`).

   ```
   import oracle.apps.cz.cio.*;
   ```

3. Define a class in which to determine the behavior of your Functional Companion.

   Here is the relevant line from Example 1–1:

   ```
   public class MyClass extends FunctionalCompanion // line 6
   ```

When you define your Functional Companion class, you can do one of the following:

- Normally: Extend the base class for Functional Companions—`oracle.apps.cz.cio.FunctionalCompanion`—and override just the particular methods that you need. In this case, you gain the functionality of the `FunctionalCompanion` base class. This functionality includes: saving references to the runtime node with which the Functional Companion is associated (with the `FunctionalCompanion.getRuntimeNode()` method), and returning the name of the Functional Companion (with the `FunctionalCompanion.getName()` method). See the reference for: FunctionalCompanion.
- More rarely: Implement the interface class for Functional Companions—`oracle.apps.cz.cio.IFunctionalCompanion`—and implement all its methods. You do not extend `oracle.apps.cz.cio.FunctionalCompanion`. In this case, you lose the functionality of the `FunctionalCompanion` base class. See the reference for: FunctionalCompanion.

4. You may want to override `oracle.apps.cz.cio.FunctionalCompanion.initialize()`. (See Section 2.9.1, "The initialize() Interface Method".)

   You should ordinarily never directly call `FunctionalCompanion.initialize()`, since the CIO does that for you. However, if your Functional Companion overrides `FunctionalCompanion` as its base class, then the `initialize()` method of *your* class should call `super.initialize()`. This passes some necessary variables to the superclass (`oracle.apps.cz.cio.FunctionalCompanion`) so that its methods will work.

   For an example in context, see Line 35 in Example 4–2, "Basic Functional Companion: FuncCompTest1" on page 4-2, which is shown below:

```
    public void initialize(IRuntimeNode comp_node, String name, String
description, int id)
    {
    this.comp_node = comp_node;
    super.initialize(comp_node, name, description, id); // line 35
    }
```

5. Override one or more of the other interface methods of `oracle.apps.cz.cio.IFunctionalCompanion` (see Section 2.9, "Standard Interface Methods for Functional Companions"):

```
autoConfigure
validate
generateOutput
terminate
```

For examples in context, see Example 4–2, "Basic Functional Companion: FuncCompTest1" on page 4-2:

```
public void autoConfigure()

public List validate()

public String generateOutput()
```

6. Optionally, call the methods of the other interface classes of the CIO (see Section 2.2, "The CIO's Runtime Node Interface Classes").

> **Note:** Basic Functional Companions, ones that only use the standard interface methods listed in step 5, do not need to use the interface classes of the CIO.

7. Compile the .java file into a .class file for example, with JDK 1.1.x:

```
javac FuncCompTest1.java
```

8. Put the resulting .class file in your classpath, or into a JAR file in your classpath. For example:

```
jar cvf FuncComps.jar FuncCompTest1.class

set CLASSPATH=%CLASSPATH%;D:\companions\FuncComps.jar
```

9. Run Oracle Configurator Developer with this classpath. Associate your Functional Companion with a Component in your Model. See Section 1.5, "Incorporating Functional Companions in your Application" on page 1-13. Generate the Active Model and User Interface.

10. To test your Functional Companion, click the Test button in Oracle Configurator Developer. When the Oracle SellingPoint application runs, click the buttons that have been generated in the UI for activating your Functional Companions. See Section 1.5.2, "Testing Functional Companions in the Oracle SellingPoint Application" on page 1-17.

1.3.2 Installation Requirements for Java Functional Companions

1.3.2.1 Requirements for Developing Functional Companions

In order to develop Java Functional Companions, you must install a Java development environment that enables you to compile Java classes, such as:

- Oracle JDeveloper
- Sun JDK 1.1.x or JDK 1.2.x (JDK 1.1.x is recommended for compatibility with Oracle Applications Release 11*i*)
- Microsoft Visual J++

You do not need JDBC drivers or database access to compile a Functional Companion, although these are required to run one.

1.3.2.2 Requirements for Running Functional Companions

At runtime, an Oracle SellingPoint application using Functional Companions requires:

- The Microsoft Java Virtual Machine (JVM)
- Microsoft JDBC/ODBC drivers
- An ODBC datasource

The Oracle SellingPoint application automatically sets up a JDBC database connection for use by the CIO. Custom user interfaces that take the place of the Oracle SellingPoint application must perform this task. See Section 2.3, "Initializing the CIO" for details.

In order to run Java Functional Companions, the software described in Table 1–2 must be installed and recognized by your operating system environment in the indicated locations.

Table 1–2 Required Software for Functional Companions

File name	Location	Required for	Source
config.jar	CLASSPATH	Any use of CIO.	Oracle Configurator (OC) installation.
confw32.jar	CLASSPATH	Functional Companions using COM or GSL.	OC installation.
collections.jar	CLASSPATH	Any use of CIO.	OC installation.

Table 1–2 (Cont.) Required Software for Functional Companions

File name	Location	Required for	Source
swingall.jar	CLASSPATH	Use of Swing UI widgets.	OC installation.
xmlparser.jar	CLASSPATH	Custom application user interfaces.	OC installation.
cz.dll czjni.dll	PATH	CIO and Functional Companion access to the Oracle Configurator logic engine.	OC installation.
JDBC OCI driver (such as oci805jdbc.dll)	PATH	Functional Companions using Oracle JDBC OCI ("thick")drivers. For use with Javasoft JDK 1.1.x.	Oracle Technology Network download area, under "Oracle 8 JDBC OCI and JDBC Thin Drivers".
classes111.zip	CLASSPATH	Functional Companions using the Oracle JDBC Thin drivers. For use with Javasoft JDK 1.1.x.	Oracle Technology Network download area, under "Oracle 8 JDBC OCI and JDBC Thin Drivers".

For background on JDBC drivers, see the *Oracle8i JDBC Developer's Guide and Reference*.

1.3.2.3 Requirements for Testing Java Functional Companions

The class(es) that implement your Functional Companions must be included in your CLASSPATH environment variable. Otherwise, you are likely to get an error message like the following when you try to create a new configuration:

```
New Configuration: Cannot create configuration:
oracle.apps.cz.cio.FuncCompCreationException:
java.lang.ClassNotFoundException: classname
```

Where *classname* is the name of the first Functional Companion to be loaded.

1.3.3 Minimal Example of a Java Functional Companion

Example 1–1 illustrates the minimal coding required for a Functional Companion that does not perform any work. (See Section 4.2, "Basic Java Functional Companion" for a fuller example.)

Example 1–1 Elementary Java Functional Companion: MyClass.java

```
import oracle.apps.cz.cio.*;
import com.sun.java.util.collections.List;                    // line 2
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

public class MyClass extends FunctionalCompanion              // line 6
{

    // constructor
    public MyClass(CIO cio, IRuntimeNode node) {
    }

    public void initialize(IRuntimeNode node, String name, String description,
int id) {
    // implement body, if necesary
        super.initialize(node, name, description, id);
    }

    public void autoConfigure() throws LogicalException {
    // implement body as desired
    }

    public List validate() {
    // implement body as desired
    return null;
    }

    // for thin client
    public void generateOutput(HttpServletResponse response) throws IOException
{
    // implement body as desired
    }

    // for thick client
    public String generateOutput() {
    // implement body as desired
    return null;
    }

    public void terminate() {
    // implement body, if necesary
    super.terminate();
    }
}
```

Line 2

```
import com.sun.java.util.collections.List; // line 2
```

If you are using JDK 1.1.x, import `com.sun.java.util.collections.List`, which is provided in `collections.jar` (see "Installation Requirements for Java Functional Companions" on page 1-8). If you are using JDK 1.2, then import `java.util.List`.

Line 4

```
public class MyClass extends FunctionalCompanion // line 6
```

This class extends the base class for Functional Companions: `oracle.apps.cz.cio.FunctionalCompanion`. See the explanation under Step 3.

1.4 Building Functional Companions in COM

With certain restrictions, you can build Functional Companions as objects conforming to the Microsoft Component Object Model (COM) standard.

It is currently only possible to run COM Functional Companions on Windows 95/98 and Windows NT 4.0, using the Microsoft Java Virtual Machine (JVM). (This precludes using COM Functional Companions in a server context using Oracle Application Server.)

The Java wrapper classes needed to access the CIO are installed with Oracle Configurator. (See Section 1.3.2, "Installation Requirements for Java Functional Companions" on page 1-8.)

A COM Functional Companion should be implemented as an object that supports `IDispatch` (every instance of a Visual Basic class fits this criteria). The object then needs to implement the `initialize, autoConfigure, generateOutput` and `terminate` methods.)

The ProgID of the COM object should be specified as the Program String in Oracle Configurator Developer. See Section 1.5.1, "Associating Functional Companions with your Model".

For details, a fuller example, and more background, see the Oracle White Paper “Using COM with Oracle SellingPoint 4.2” (as noted under "Related Documents" on page -xiv).

Example 1–2 illustrates a simple COM Functional Companion implemented in Visual Basic.

Example 1–2 COM Functional Companion

```
Option Explicit

Private m_component As Object
Private m_name As String
Private m_description As String
Private m_id As Long

' The initialize method stores the initialization information in instance
variables
Public Sub initialize(component As Object, name As String, description As
String, id As Long)
  Set m_component = component
  m_name = name
  m_description = description
  m_id = id
End Sub

' This autoConfigure method selects a particular option in a feature of the
component
Public Sub autoConfigure()
  Dim feature1 As Object
  Dim option1 As Object

  Set feature1 = m_component.getChildByName("Feature1")
  Set option1 = feature1.getChildByName("Option1")
  Call option1.Select
End Sub

' This validate method requires that a particular option not be set.   If the
option is
' set, a validation failure is raised by returning the failure message in an
array,
' otherwise, True is returned.
Public Function validate() As Variant
  Dim feature1 As Object
  Dim option2 As Object

  Set feature1 = m_component.getChildByName("Feature1")
  Set option2 = feature1.getChildByName("Option2")
  If option2.isSelected() Then
```

```
    validate = Array("Option2 cannot be selected")
  Else
    validate = True
  End If
End Function

' This generateOutput method returns the name of the component.
Public Function generateOutput() As String
  generateOutput = m_component.getName()
End Function

Public Sub terminate()
End Sub
```

1.5 Incorporating Functional Companions in your Application

1.5.1 Associating Functional Companions with your Model

To enable your Functional Companion to work with your Oracle SellingPoint application, you must associate it with a Component (or Product) in your Model. You create this association in Oracle Configurator Developer, as a type of Configuration Rule that specifies the Functional Companion method(s) that you have implemented, and the path to be used by the Oracle SellingPoint application to locate the Functional Companion object.

To create an association between a Component and a Functional Companion:

1. Click on the Rules button on the main toolbar.

   A list of the Configuration Rule types appear in the lower-left pane.

2. Choose **New Functional Companion** from the Create menu. You can also highlight the Functional Companions node, click on the right mouse button, and select New Functional Companion from the popup menu.

3. Type a name for the Functional Companion rule.

4. In the Description section, type a short explanation of the Functional Companion rule. If necessary, open the Description section by clicking on the blue arrow to the left of it.

5. If necessary, open the Definition section by clicking on the blue arrow to the left of it. In the Model view, select the Component or Product that you want to include in this rule. Drag it with the left-hand mouse button to the Base

Component field in the Definition section. Only one Base Component may be specified per rule.

6. Choose one or more roles for the Functional Companion. The choices are:

Type	Associated Functional Companion method
Auto-configuration	`autoConfigure`
Validation	`validate`
Output	`generateOuput`

See Section 1.1.1, "Types of Functional Companions" and Section 2.9, "Standard Interface Methods for Functional Companions" for background. Note that you do not associate the `initialize()` and `terminate()` methods, since they are invoked automatically by the Oracle SellingPoint application.

7. Indicate how the Functional Companion is implemented:
   - Java
   - GSL
   - COM
8. Type in the Program String that identifies the Functional Companion:
   - For Java, this is the name of the class that implements the Functional Companion, such as:

     ```
     FuncCompTest1
     ```

     The full class specification must be accessible through your CLASSPATH environment variable. For instance, if `FuncCompTest1` is contained this way in a JAR file `tests.jar`:

     ```
     com\java\tests\FuncCompTest1.class
     ```

     then you would specify the Functional Companion this way in Configurator Developer:

     ```
     com.java.tests.FuncCompTest1
     ```

     See Step 8 under Section 1.3.1, "Procedure for Building Functional Companions in Java".
   - For GSL, this is a top-level path specifying a library and building block, such as:

```
fc_test_lib.fc_test
```

See *Oracle SellingPoint CompanionBuilder Help* for details on working with GSL.

- For COM, this is a ProgID, such as:

```
FCTestProject.FCTest
```

See Section 1.4, "Building Functional Companions in COM".

9. Choose Generate Active Model from the Tools menu.
10. When the Generate Active Model command completes successfully, click on the UI button on the main toolbar, then choose Refresh from the Edit menu.

Figure 1–1 shows what the Rules module screen of Oracle Configurator Developer might look like after you associate a Component with a Functional Companion.

Figure 1–1 Associating a Component with a Functional Companion

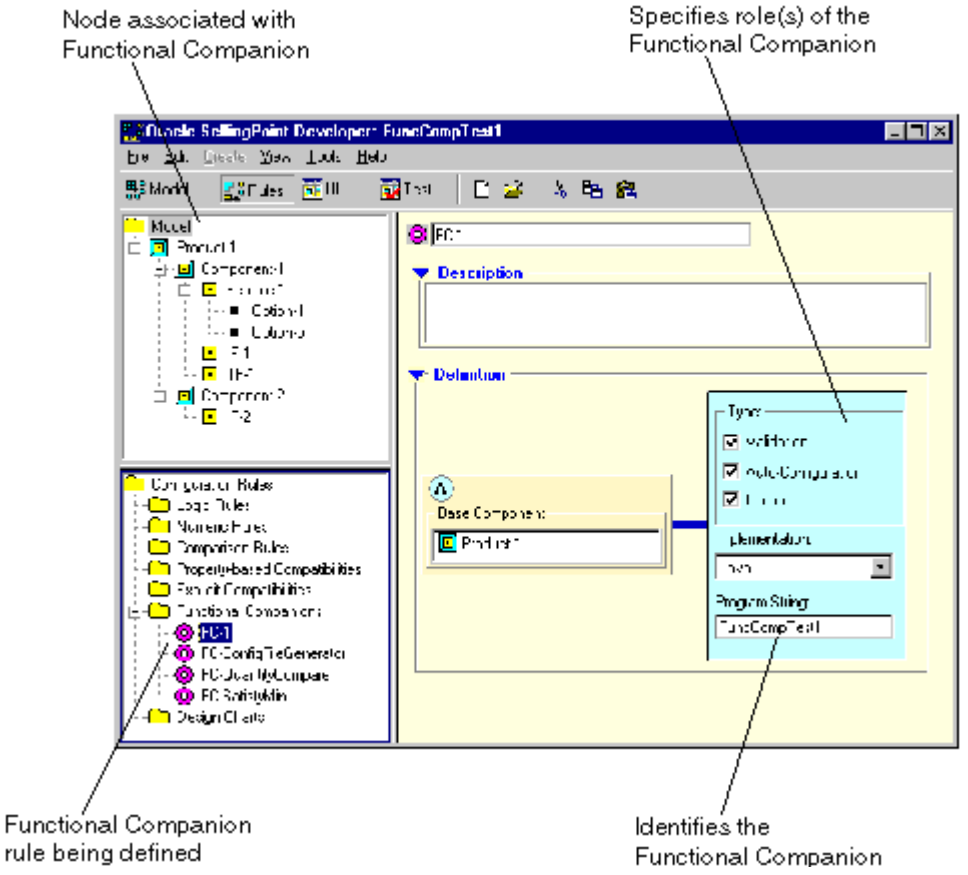
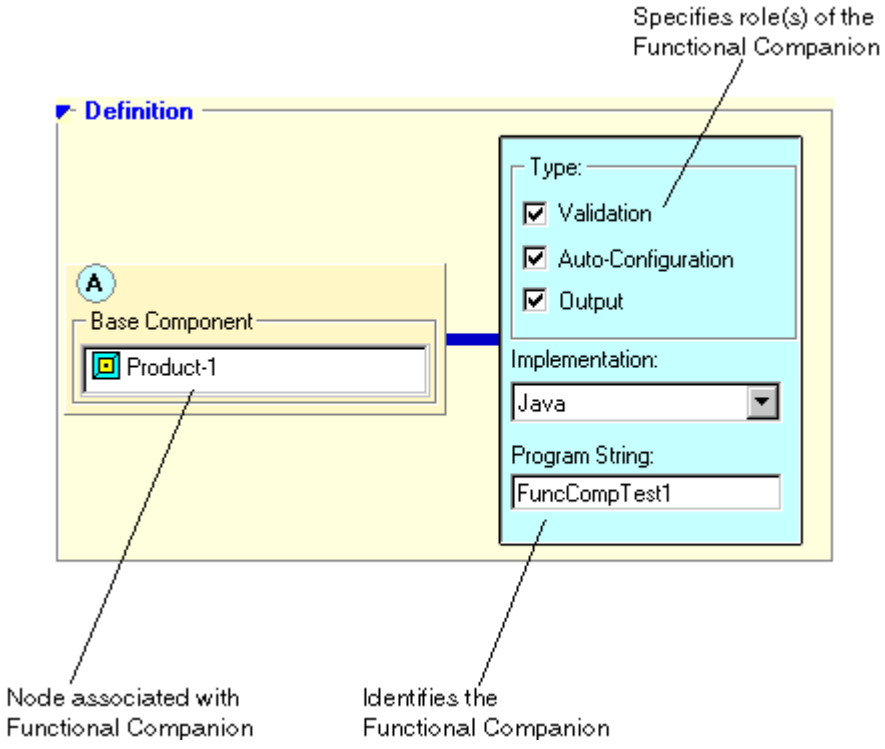


Figure 1–2 shows details of the Attributes view of the screen in Figure 1–1.

Figure 1–2 Functional Companion Rule: Detail of the Attributes view



1.5.2 Testing Functional Companions in the Oracle SellingPoint Application

Note: The Oracle SellingPoint application is provided with Release 11*i* of Oracle Configurator.

After you generate the Active Model and UI, you can test your Functional Companions by running the Oracle SellingPoint application. You can run the Oracle SellingPoint application in the ways described below.

1.5.2.1 Testing from the Windows Start Menu

To run the Oracle SellingPoint application from the Windows Start Menu:

- Follow the instructions in the *Oracle Configurator Developer CD-ROM Insert* on running the Oracle SellingPoint application.

1.5.2.2 Testing from Oracle Configurator Developer

To run the Oracle SellingPoint application from Oracle Configurator Developer:

1. Use Tools > Options to chose the Oracle SellingPoint application as your test environment.
2. Click the Test button.
3. Follow the instructions in the *Oracle Configurator Developer CD-ROM Insert* on running the Oracle SellingPoint application.

1.5.2.3 Test Functionality in the Oracle SellingPoint Application

The Active User Interface for the Oracle SellingPoint application allows you to test your Functional Companions as appropriate:

Type	User Interface feature
Auto-configuration	A button allows the user to run the `autoConfigure()` method on the associated Component instance.
Validation	The `validate()` method is called automatically when the user selects anything.
Output	A button allows the user to run the `generateOuput()` method on the associated Component instance.

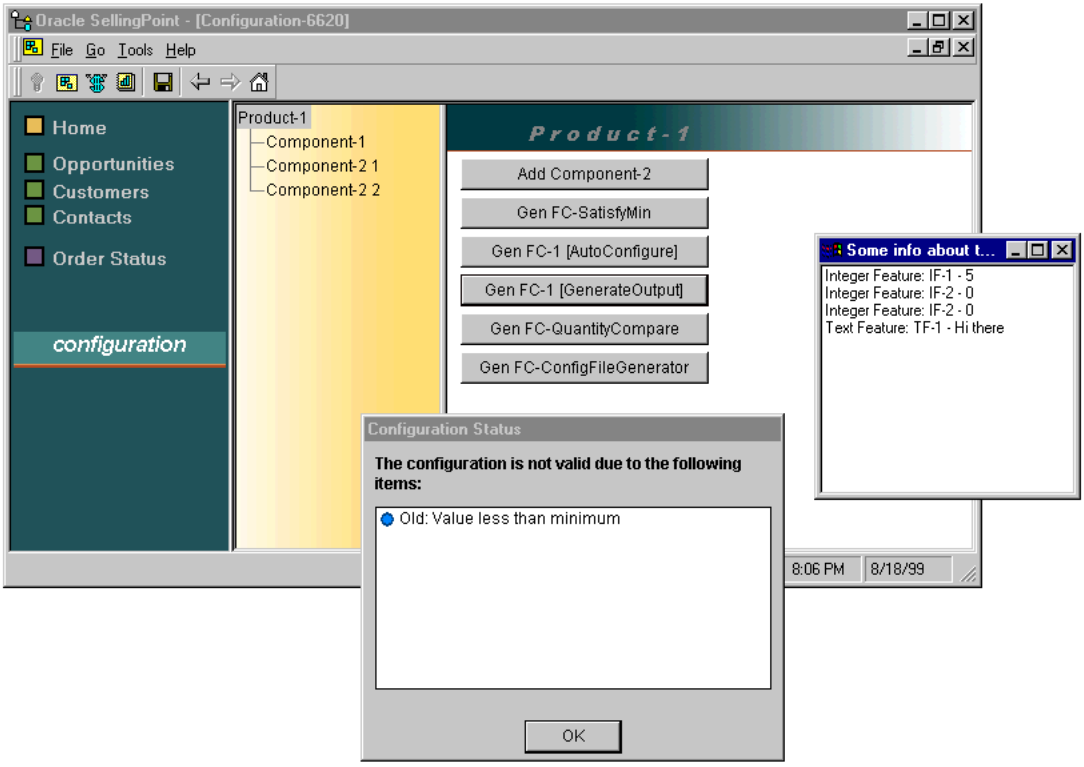
Figure 1–3 illustrates testing several Functional Companions in the Oracle SellingPoint application. The Functional Companions illustrated are the ones defined in the example in Section 4.2, "Basic Java Functional Companion" on page 4-2.

- Clicking the GenerateOutput Functional Companion button produces a window that displays the current value of several Features. (This uses the "thick client" version of `generateOutput()`. For a thin-client example, see Section 4.3, "Thin-Client generateOutput() Functional Companion".)
- There is no button for the Validate Functional Companion. The `validate()` method is run whenever there is a change in the value of an Option. If the value

violates a specified range, or a Configuration Rule, then the application displays a Configuration Status message.

- Clicking the AutoConfigure Functional Companion button changes the value of a numeric Feature (not shown here), in this case violating a specified minimum and thereby triggering the Configuration Status message.

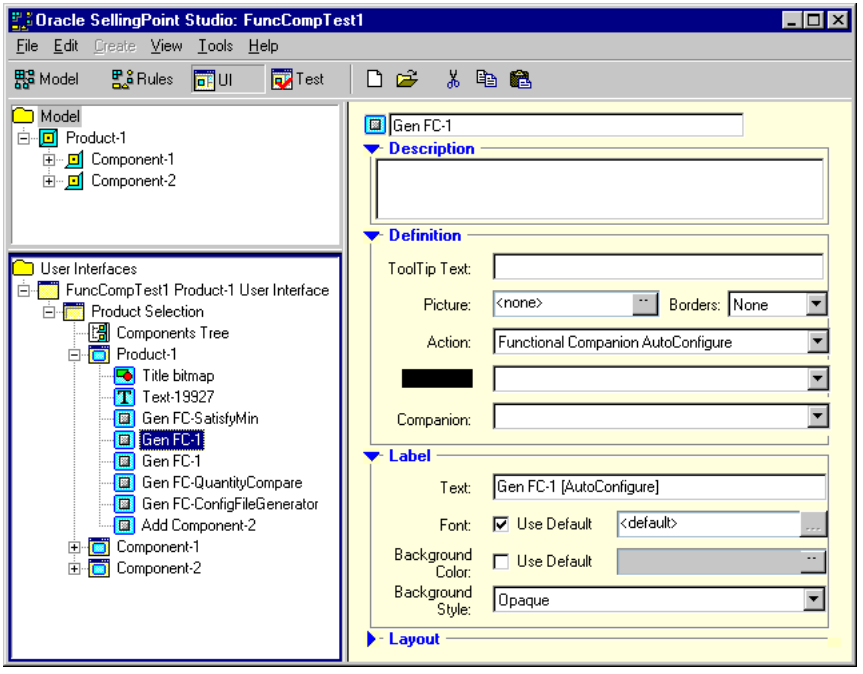
Figure 1–3 Testing Functional Companions in the Oracle SellingPoint application.



Each button that runs a Functional Companion is labelled with default text that identifies the Functional Companion that the button activates. You can use the User Interface module of Oracle Configurator Developer to modify these labels. The labels buttons generated by the Functional Companion shown in Figure 1–1 have

been so modified, by adding the self-identifying text `[AutoConfigure]` and `[GenerateOuput]`, as shown in Figure 1–4.

Figure 1–4 Modifying Functional Companion Buttons



2

The Configuration Interface Object (CIO)

2.1 Background

2.1.1 What is the CIO?

The Configuration Interface Object (CIO) is an API (application programming interface) that provides your programs access to the Model used by a Oracle SellingPoint application, which you construct with Oracle Configurator Developer.

The CIO is also used by Functional Companions. See Section 1.2, "Functional Companions and the CIO".

The CIO is a top-level configuration server. The CIO is responsible for creating, saving and destroying objects representing configurations, which themselves contain objects representing Products, Components, Features, Options, Totals and Resources. The runtime configuration model can be completely controlled and manipulated through these interfaces, using methods for getting and setting logical, numeric and string values, and creating optional subcomponents.

Internally, the CIO performs its tasks through interfaces to *logic net objects* (to get and set logic states), to *runtime model subschema objects* (to create the appropriate runtime Model based on the design-time model), and to *configuration subschema objects* (to save and restore configurations created by a user).

The Oracle Configuration Interface Object is written in Java, and implemented as a set of Java packages. The only one that you will usually need to import is:

```
oracle.apps.cz.cio
```

Note: All references in this document to classes, methods, and properties refer to the package `oracle.apps.cz.cio`, and all code examples are in Java, unless otherwise stated.

Interfaces are also provided for Oracle's Generative Specification Language (GSL). The current version of Oracle Configurator Developer supports the ability to connect your Model directly to GSL objects built with CompanionBuilder.

2.1.2 The CIO and Functional Companions

A Functional Companion is a Java client of the CIO.

Functional Companions are invoked by the CIO through the Oracle SellingPoint application, and Functional Companions call the CIO to get information from the running Model. The CIO is like a broker for the Active Model, in that it passes information both ways. Programmers writing Functional Companions need to have some knowledge of how to use the CIO.

Each Functional Companion is an object class. For every Component instance in your Model that is associated with a Functional Companion, the CIO creates an instance of this class.

2.2 The CIO's Runtime Node Interface Classes

When you program against the CIO, you only create instances of the classes `CIO` (see Section 2.3, "Initializing the CIO") and `Configuration` (see Section 2.4.1, "Creating and Deleting Configurations"). You then use the public interfaces listed in Table 2–1 to access fields in the runtime node objects created by your instances of `CIO` and `Configuration`. Apart from `CIO` and `Configuration`, your code should refer only to these public runtime node interface objects. You should not implement any of the runtime node interface classes, but only use them as references to runtime node objects.

These interfaces are all defined in the Java package `oracle.apps.cz.cio`.

Table 2–1 Runtime node interface classes for the CIO

Interface	Role of implementing classes
IBomItem	Implemented by all selectable BOM items.

Table 2–1 (Cont.) Runtime node interface classes for the CIO

Interface	Role of implementing classes
IConfigEventListener	Implemented by objects that want to find out about added components.
ICompSetEventListener	Implemented by objects that want to find out about added components.
ICount	Implemented by objects that have an associated integer count.
IDecimal	Implemented by objects that have a decimal value.
IDecimalMinMax	Implemented by objects that have a decimal minimum and maximum value.
IFunctionalCompanion	Implemented by Functional Companion objects attached to Components in order to provide programmatic functionality to a configuration model.
IInteger	Implemented by objects that have an integer value.
IIntegerMinMax	Implemented by objects that have an integer minimum and maximum.
IOption	Implemented by objects that act as options. The defining characteristic of an option is that it can be selected and deselected.
IOptionFeature	Implemented by objects that contain selectable options. This interface provides a mechanism for selecting and deselecting options, and for determining which options are currently selected.
IReadOnlyDecimal	Implemented by objects that have a decimal value.
IRuntimeNode	This interface implements behavior common to all nodes in the runtime configuration tree, including Components, Features, Options, Totals, and Resources.
IState	Implemented by objects that have logic state. This interface contains a set of input states, used to specify a new state for an object, a set of output states, returned when querying an object for its state, and a set of methods for getting and setting the object's state.
IText	Implemented by objects that have a textual value.

The functionality underlying the CIO interfaces is implemented by other classes in `oracle.apps.cz.cio,` which are subject to revision by Oracle. This

interface/implementer architecture protects your code from the effects of such revisions, since the interfaces will remain constant.

Reference

For reference documentation, see: Package oracle.apps.cz.cio.

2.3 Initializing the CIO

In order to use any of the features of the CIO, an application must initialize it, using a JDBC driver to make a connection to the Oracle Configurator Database. This connection enables the CIO to obtain and store data about Model structure, Configuration Rules, and User Interface.

If you are using the CIO in a custom user interface, you will have to initialize the CIO.

> **Note:** When you run Functional Companions through the Oracle SellingPoint application (or test them by using the Test button in Oracle Configurator Developer), this initialization and connection work is automatically handled for you by the application; you do not have to write your own code to initialize the CIO.

1. Import the necessary packages.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import oracle.apps.cz.cio.*;
import oracle.apps.cz.common.*;
```

2. Load the database driver that you have installed. For instance, load one of the following:

```
Class.forName("com.ms.jdbc.odbc.JdbcOdbcDriver");
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Class.forName("oracle.jdbc.Driver.OracleDriver");
```

3. Create a CZContext context object and pass to it the information needed to make a database connection: the database URL, the user ID and password of the current user, and the owner of the database. The context object manages the

database connection; you should *not* create a separate connection object (e.g., with `java.sql.DriverManager.getConnection`).

```
CZContext contextObject = new CZContext("jdbc:subprotocol:datasource",
"userID", "password", "schemaOwner");
```

4. Create a CIO object.

```
CIO cioObject = new CIO();
```

5. Pass the location of the Active Model to the CIO object. Ordinarily, this will be `ORACLE_HOME/OSP/Shared/ActiveModel/`. This is only necessary if your application logic is not generated in the database.

```
cioObject.setActiveModelPath("modelPath");
```

Example 2–1 shows how Steps 1 through 5 are combined together. See Section 4.1, "Initializing the CIO" for a fuller example of initializing the CIO.

Example 2–1 Initializing the CIO (Short Example)

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import oracle.apps.cz.cio.*;
import oracle.apps.cz.common.*;

public class InitCIO
{
    private void InitializeCIO() throws SQLException
    {
    CIO cio;
    CZContext context;

    try{Class.forName("com.ms.jdbc.odbc.JdbcOdbcDriver");}
        catch (ClassNotFoundException c){System.out.println(c);}
    context = new CZContext("jdbc:odbc:TutorialLite", "spx", "spx", "spx");
    cio = new CIO();
    cio.setActiveModelPath("D:/orant/OSP/Shared/ActiveModel/");
    }
}
```

2.4 Access to Configurations

The Configuration object, `oracle.apps.cz.cio.Configuration`, represents a complete configuration. You can use the CIO to work with multiple configurations within the same session.

A configuration communicates through the Configuration object. It supports accessing the containing CIO, the root Component, the project ID, a collection of current validation failures, access to any runtime node based on its runtime ID, and an indication if the complete configuration is satisfied. In addition, there are methods for starting, ending, and rolling back configuration-level logic transactions; these transactions are to maintain logic consistency and are not database transactions. See Section 2.4.4, "Logic Transactions".

Reference

For reference documentation, see: `Configuration`.

2.4.1 Creating and Deleting Configurations

Use `CIO.createConfiguration()` to create a Configuration object, which is the top-level entry point to a configuration. There are different ways to create a Configuration, depending on your requirements.

- To create a Configuration using the name of a Project as specified in Oracle Configurator Developer, use this form:

  ```
  createConfiguration(java.lang.String projectName,
                      oracle.apps.fnd.common.Context ctx)
  ```

- To create a Configuration using the ID of the root node of your Model, use this form:

  ```
  createConfiguration(int rootNodeID,
                      oracle.apps.fnd.common.Context ctx)
  ```

  To determine the root node ID, you would query the Oracle Configurator Database, which is described in the *Oracle Configurator Technical Reference Manual*. Such a query might be:

  ```
  SELECT PS_NODE_ID FROM CZ_PS_NODES WHERE NAME = 'CN9744';
  ```

Both ways of creating a Configuration object require a database context object, as discussed in Step 3 of Section 2.3 on page 2-4.

For reference documentation on the database context object, see: `CZContext`.

To delete all runtime structure and memory associated with a configuration, use `CIO.closeConfiguration()`

To get the CIO that created the configuration, use `Configuration.getCIO()`.

Example 2–2 Creating New Configuration Objects

```
// create the database context object
ctx = new CZContext("jdbc:odbc:testdb101", "myusername", "mypasswd", "admin01");

// create Configuration using Project name and Context
cfg_prj = createConfiguration(“Project 10”, ctx);

// create Configuration using ID of root node of Model and Context
cfg_id = createConfiguration(1221, ctx);
```

2.4.2 Saving and Restoring Configurations

Use `Configuration.saveNew()`to Save an entirely new Configuration object into the Oracle Configurator Database.

Use `Configuration.save()` to save subsequent changes to a Configuration object created with `saveNew()`, or to a Configuration object restored with `CIO.restoreConfiguration()`.

Use `CIO.restoreConfiguration()` to restore a Configuration object from the Oracle Configurator Database.

Use `Configuration.saveNewRev()` to save a new revision of the restored Configuration object.

Use `Configuration.saveAs(configHeaderID, revNumber)` to save the current Configuration object over a different Configuration already saved in the database. You use the `configHeaderID` and `revNumber` to open a configuration header object and replace the configuration in it.

Note: Do not save a Configuration object during a logic transaction (see Section 2.4.4). You may miss some validation messages that are not available until the transaction is committed.

2.4.3 Access to Configuration Parameters

If you are using Oracle Configurator Internet Edition for a web-based application, you can use the CIO to allow a Functional Companion to obtain a list of the configuration inputs that were passed from your application to your configuration Model.

Create a Functional Companion that calls `Configuration.getInitParameters()`, which returns a NameValuePairSet object. This object contains all the parameter values stored by the Internet Edition UI Server when it processed the initialization message sent by your application to the Internet Edition UI Servlet.

See the *Oracle Configurator Internet Edition Developer's Guide* for information on Internet Edition.

Reference

For reference documentation, see: `NameValuePairSet`.

2.4.4 Logic Transactions

In order to help you maintain consistency in interactions with the Oracle Configurator logic engine, you can use *configuration-level logic transactions*. A logic transaction comprises all the logical assertions that constitute a user interaction. At the end of a transaction, the CIO returns a list of all validation failures. See Section 2.8, "Validating Configurations".

The Configuration object, `oracle.apps.cz.cio.Configuration`, provides a set of methods for starting, ending, and rolling back configuration-level logic transactions. Note that logic transactions are not database transactions.

Inside a transaction, the normal course of action is to set the logical states and numeric values of runtime nodes (see Section 2.5.4 and Section 2.5.5).

- Use `Configuration.beginConfigTransaction()` to create a new transaction, returning a ConfigTransaction object. After performing the desired series of operations (e.g., setting states and values), you must end, commit, or roll back the transaction by passing the ConfigTransaction object to one of the mutually exclusive methods that finish the transaction:

  ```
  endConfigTransaction
  commitConfigTransaction
  rollbackConfigTransaction
  ```

- `Configuration.endConfigTransaction(transaction)` ends the transaction begun with `beginConfigTransaction()`, without committing it (thus skipping validation checking).
- `Configuration.commitConfigTransaction(transaction)` commits the given transaction or series of nested transactions, propagates the effect of user selections throughout the configuration Model, and triggers validation checking (see Section 2.8, "Validating Configurations").
- `Configuration.rollbackConfigTransaction(transaction)`rolls back the unfinished transaction, undoing the operations performed inside it.

You can nest intermediate transactions with `beginConfigTransaction()` and `endConfigTransaction`, delaying validation checking until you call `commitConfigTransaction()`. You must end or commit inner transactions before ending or committing the outer ones that contain them. When rolling back unfinished transactions, with `rollbackConfigTransaction()`, you can roll back outer transactions, which automatically rolls back the inner transactions.

When beginning a transaction, you can autocommit it, by setting the optional boolean `autoCommit` argument to `beginConfigTransaction()` to TRUE. If no argument is set, then the transaction inherits the autocommit state of its parent (outer) transaction. If an outer transaction sets `autoCommit` to TRUE, then inner transactions can override it to either TRUE or FALSE. If an outer transaction sets `autoCommit` to FALSE, then inner transactions cannot override it; they will always inherit FALSE.

Reference

For reference documentation, see: `Configuration`.

2.5 Access to Nodes of the Model at Runtime

The root Component, and every other node in the underlying runtime Model tree, implements the IRuntimeNode interface. This interface exposes the type of the node (based on a set of node type constants), its name, the database ID, the database node of which this runtime node is an instance, a runtime ID that is unique to this node across all nodes created by this particular CIO, the parent node (which is null for the root Component), a (possibly empty) collection of children, and information about whether this part of the runtime tree has been satisfied. See Section 2.6, "Introspection through IRuntimeNode".

Use `IRuntimeNode.getConfiguration()` to get the configuration to which a node belongs.

Reference

For reference documentation, see: `IRuntimeNode.getConfiguration()`.

2.5.1 Accessing Components

Use `Component.getFunctionalCompanions()` to return a list of all the Functional Companions associated with this Component.

2.5.2 Adding and Deleting Optional Components

The Component set represents a set of similar Components that can be added and deleted dynamically. Each Component set implements the IRuntimeNode interface.

Use `ComponentSet.add()` to an optional Component. The `add()` method can throw a LogicalException exception if adding the Component causes a logical contradiction.

Use `ComponentSet.delete(component)` to delete an optional Component.

2.5.3 Accessing Features

There are several specialized types of Features. Each Feature type supports the `IRuntimeNode` interface, enabling you to use its general methods for working with runtime nodes (see "Introspection through IRuntimeNode" on page 2-15). Each type also supports its own interface with appropriately specialized methods.

BooleanFeatures have a boolean (true/false) value.

CountFeatures have both a boolean value, and an associated integer-valued numeric count. The minimum value of the count must be greater than or equal to zero. The boolean value a CountFeatures object is returned by its methods `hasMax()` and `hasMin()`.

IntegerFeatures have an integer numeric value. The value can be positive, negative, or zero.

DecimalFeatures have a floating point value.

TextFeatures have a string value.

OptionFeatures have a logic value, and a set of options as children. You can use the methods `getMinSelected()` and `getMaxSelected()`, of `IOptionFeature`, to determine the minimum and maximum number of a Feature's child Options that can be selected. If you do, first use `hasMinSelected()` or `hasMaxSelected()` to determine whether there is a minimum or maximum number of Options.

Note: If, in Oracle Configurator Developer, you set the minimum count of a Feature greater than or equal to zero, then the CIO treats this as a CountFeature object. If you set the minimum count less than zero, then the CIO treats this as a IntegerFeature object.

2.5.4 Getting and Setting Logic States

To interact with objects that have logic state, you implement the IState interface. This interface contains:

- a set of input states, used to specify a new state for an object

FALSE	The input state used to set an object to false.
TRUE	The input state used to set an object to true.
TOGGLE	The input state used to turn an object state to true if it is false or unknown, and to make it unknown or false if it is true.

- a set of output states, returned when querying an object for its state

LFALSE	The logically false output state, indicating that the state is false as a consequence of a rule.
LTRUE	The logically true output state, indicating that the state is true as a consequence of a rule.
UFALSE	The user false output state, indicating that a user has set this object to false.
UNKNOWN	The unknown output state.
UTRUE	The user true output state, indicating that a user has set this object to true.

- a set of methods for getting and setting the object's state

`getState()`	Gets the current logic state of this object.
`setState(int state)`	Change the current logic state of this object.

Example 2–3 Getting the state of a node

The following code fragment, which uses `getState()` with `UTRUE`, is taken from Section 4.2, "Basic Java Functional Companion", after the comment `"//get the necessary components from the configuration // line 61"`.

```
//get the necessary components from the configuration    // line 61
baseComponent = (oracle.apps.cz.cio.Component)comp_
node.getChildByName("Component-1");
            of = (OptionFeature)baseComponent.getChildByName("Feature-1");
            op = (Option)of.getChildByName("Option-1");
            intFeat = (IntegerFeature)baseComponent.getChildByName("IF-1");
            //check if the option is set to UTRUE.  If so, set the Integer value
to 5
            if( op.getState() == IState.UTRUE )
                intFeat.setIntValue(5);
        }
```

Example 2–4 Setting the state of a node

The following code fragment, which uses `setState()` with `TOGGLE`, toggles the state of the selected item in the Model tree.

```
private void toggleSelectedItem() {
    IState node = (IState)tree.getLastSelectedPathComponent();
    try {
      node.setState(IState.TOGGLE);
    }
    catch (LogicalException le) {}
    catch (TransactionException te) {}

    tree.repaint();
```

2.5.5 Getting and Setting Numeric Values

You can use the following methods to get and set the values of objects that have numeric values.

For decimal values, use:

`IDecimal.setDecimalValue()`
`IReadOnlyDecimal.getDecimalValue()`
For integer values, use:

`IInteger.setIntValue()`

`IInteger.getIntValue()`

The code fragment in Example 2–5 uses `setIntValue()` to change the value of an Integer Feature. Note that you can use the generalized IRuntimeNode interface for flexibility in selecting a child node, and then cast the node object to a particular interface to perform the desired operation on it.

Example 2–5 Setting a numeric value

```
// select a node by name
IRuntimeNode limit = baseComp.getChildByName("Current Limit");

// use an interface cast to set the node's value by the desired type
((IInteger)limit).setIntValue(5);
```

To determine whether a numeric value has violated its Minimum or Maximum range, you may need to iterate through the collection of validation failures returned by `Configuration.getValidationFailures()` after setting a value, for instance with `IInteger.setIntValue()`. See Section 2.8, "Validating Configurations" for more background.

There is a subtlety that you should take note of. `IDecimal.setDecimalValue()`does not throw a LogicalException when setting the value of a decimal feature that exceeds the feature's Min/Max limits. The collection of validation failures returned by `Configuration.getValidationFailures()` does not include any failures that result from setting a numeric value until the logic transaction has been closed, so there is no way to roll back a transaction in which a Min/Max violation has occurred. Here is a suggested method for dealing with this situation:

1. Open a transaction.
2. Set the new value.
3. Close the transaction.
4. Get the collection of validation failures for the configuration.
5. If the last transaction caused a Min/Max violation, then call `Configuration.undo()`, which retracts the last transaction.

This situation illustrates why it is a good practice to perform the setting of a single value inside a logic transaction. You can always undo it if the result is unsatisfactory.

2.5.6 Accessing Properties

You can determine which Properties belong to a runtime node, then use methods of the class `Property` to obtain information about the Properties.

Use `IRuntimeNode.getProperties()` to get a collection of the properties associated with a node.

Use `IRuntimeNode.getPropertyByName()` to get a particular property of a node, based on its name.

When you have the Property, use methods of the class `Property`, such as `getStringValue()`, to obtain specific information.

2.5.7 Access to Options

Option features have special methods for selecting options and querying for selected options. The `selectOption()` method implements mutual exclusion behavior for option features with a min/max of 1/1 by deselecting a currently selected option before selecting the new option. The `getSelectedOption()` method throws the TooManySelectedException if more than one option is selected in the feature.

An option is a child of an option feature which supports a true/false logic state and a count. Options implement the IRuntimeNode interface.

You can use the interface class IOption to select, deselect, and determine the selection state of Options.

Table 2–2 Methods of the Interface Class IOption

Method	Action
`deselect()`	Deselect this Option.
`isSelected()`	Returns true if this Option is selected, and false otherwise.
`select()`	Select this Option.

2.5.7.1 Example for IOption

The following code fragment displays a "check" icon if an Option of a runtime node is selected, and displays an "unsatisfied" icon if the node is logically unsatisfied:

```
IRuntimeNode rtNode = (IRuntimeNode)value;
    if (value instanceof IOption) {
      IOption optionNode = (IOption)value;
      if (optionNode.isSelected()) {
```

```
    setIcon(checkIcon);
  }
} else if (rtNode.isUnsatisfied()) {
  setIcon(unsatIcon);
}
return this;
```

2.6 Introspection through IRuntimeNode

You can get information about a node in a Model at runtime by using methods of the interface class IRuntimeNode. This helps you to write "generic" Functional Companions, which can interact with a Model tree dynamically, without having prior knowledge of its structure.

Table 2–3 Methods of the interface class IRuntimeNode

Method	Action
`getChildByID(id)`	Gets a particular child identified by its ID.
`getChildByName(name)`	Gets a particular child identified by its name.
`getChildren()`	Gets the children of this runtime configuration node.
`getChildrenByType(type)`	Gets all of the children of a particular type.
`getConfiguration()`	Gets the configuration to which this node belongs.
`getDatabaseID()`	Gets the database ID of the node. This is the field CZ_PS_NODES.PS_NODE_ID in the Oracle Configurator Database, described in the *Oracle Configurator Technical Reference Manual*.
`getDescription()`	Returns the design-time description of the runtime node.
`getName()`	Gets the name of the node.
`getParent()`	Gets the parent of the node.
`getProperties()`	Returns a collection of the properties associated with this node. The collection contains items of the type Property.
`getPropertyByName(name)`	Returns a particular property of the node, based on its name. Returns null if a property of the given name does not exist.
`getRuntimeID()`	Gets the runtime ID of the node.
`getSelectionLineID()`	Returns selection line ID (configuration output database ID) for node.

Table 2–3 (Cont.) Methods of the interface class IRuntimeNode

Method	Action
`getType()`	Gets the type of this node.
`hasCount()`	Returns true if the node has an object count.
`hasDecimalValue()`	Returns true if the node has a decimal value.
`hasSelectionLineID()`	Returns true if node has a selection line ID (configuration output ID), false if no
`hasState()`	Returns true if the node has a logical state.
`hasTextValue()`	Returns true if the node has a text value
`isUnsatisfied()`	Returns true if this particular node, or any one of its children, has not been completely configured.
`isUnsatisfiedNode()`	Returns true if this particular node has not been completely configured.

Reference

For reference documentation, see the Methods summary for: `IRuntimeNode`.

Example 2–6 Testing whether a node is selected, or satisfied

The following code fragment displays a "check" icon if an Option of a runtime node is selected, and displays an "unsatisfied" icon if the node is logically unsatisfied:

```
IRuntimeNode rtNode = (IRuntimeNode)value;
    if (value instanceof IOption) {
      IOption optionNode = (IOption)value;
      if (optionNode.isSelected()) {
        setIcon(checkIcon);
      }
    } else if (rtNode.isUnsatisfied()) {
      setIcon(unsatIcon);
    }
    return this;
```

Example 2–7 Getting a child node by name

The following code fragment creates a Configuration object `config`, sets `homeTheater` to the root Component of the configuration, and sets `userType` to the child node with the user-visible name "User Type".

```
Configuration config = m_cio.createConfiguration(m_product);
```

```
IRuntimeNode homeTheater = config.getRootComponent();

IRuntimeNode userType = homeTheater.getChildByName("User Type");
```

Example 2–8 Collecting all child nodes by type

The following code fragment, which uses `getChildrenByType()`, is taken from Section 4.2, "Basic Java Functional Companion", after the comment `"//get all the text features // line 167"`.

```
//get all the text features                                  // line 167
            textFeatList = comp.getChildrenByType(comp.TEXT_FEATURE);
            traverseTree(comp.getChildren(),
                         comp.TEXT_FEATURE,
                         textFeatList);
            iter = textFeatList.iterator();
```

2.7 Handling Logical Contradictions

When you make a request to modify the state of a logic network, for instance by using `IState.setState`, the result may be a failure of the request because of a logical contradiction. Such a failure will create and throw a *logical exception*, accessed through either the `LogicalException` or `LogicalOverridableException` objects. A `LogicalException` cannot be overriden.

See "Overriding Contradictions" for details on using `LogicalOverridableException` to override the contradiction.

`Use LogicalException.isOverridable()` to determine whether the exception is an instance of `LogicalOverridableException,` which can be overriden with its `override()` method.

Use `LogicalException.getCause()` to get the runtime node that caused the failure.

Use `LogicalException.getReasons()`to get a list of reason strings for the failure.

Use `LogicalException.getMessage()` to provide a message containing either the cause or the reasons.

Reference

For reference documentation, see: `LogicalException`.

2.7.1 Generating Error Messages from Contradictions

You can use the Reason object to wrap the information returned by a contradiction, in order to include information about internal error messages.

```
Reason(int type,
       IRuntimeNode node,
       java.lang.String msg)
```

Constructs a Reason given all of its information:

`type` What type of reason this is.

`node` The node that caused the problem.

`msg` The message returned.

Use `Reason.getMsg()` to get the message associated with this reason.

Use `Reason.getNode()` to get the node associated with this reason.

Use `Reason.getType()` to get the type of reason held in this object.

Use `Reason.toString()` to convert this object to a string.

Reference

For reference documentation, see: `Reason`.

2.7.2 Overriding Contradictions

Your Oracle SellingPoint application or Functional Companion can provide a message to your user, and ask whether the contradiction should be overridden.

If a logical contraction can be overriden, then a `LogicalOverridableException` is signalled, instead of a `LogicalException`. `LogicalOverridableException` is a subclass of `LogicalException` that adds an `override()` method. Use `LogicalOverridableException.override()` to override the contradiction.

Both types of exceptions (`LogicalException` and `LogicalOverridableException`) may be thrown back from any of the "set" methods (like `setState`) or from `Configuration.commitConfigTransaction()`. If you want to override the

overridable exception you have to call its `override()` method, which can also throw a `LogicalException`. This means that even when you try to override the exception you still trigger a contradiction and cannot continue. If the override succeeds then you still need to call `commitConfigTransaction()` to close the transaction. If you don't want to override or if you get a `LogicalException` you need to call `rollbackConfigTransaction()` to purge it. Example 2–9 is a code fragment that illustrates this point. Note that the operations represented with <ASK "text"> and <SHOW "text"> are not part of the CIO but suggest where your own Functional Companion should try to handle the situation.

Example 2–9 Handling and overriding Logical Exceptions

```
try {
      // begin a transaction
    ConfigTransaction tr = config.beginConfigTransaction();

      // call the "set" method
    opt1.setState();

      // commit the transaction
    config.commitConfigTransaction(tr);
  }
  catch(LogicalOverridableException loe) {
    proceed = <ASK "Do you want to override?">;
    if (! proceed) {
      rollbackConfigTransaction();
    }
    else {
      try {
          // override the contradiction and ...
        loe.override();
          // ... finish the transaction
        commitConfigTransaction();
      }
      catch (LogicalException le) {
          // we cannot do anything
        <SHOW "Cannot be overriden">
        config.rollbackConfigTransaction(tr);
      }
    }
  }
  catch (LogicalException le) {
      // we cannot do anything
    <SHOW "Cannot be overriden">
```

```
    config.rollbackConfigTransaction(tr);
  }
```

Reference

For reference documentation, see: `LogicalOverridableException`.

2.8 Validating Configurations

You want to be able to check whether a Configuration is valid (that is, does not violate the rules associated with it).

The CIO validates a Configuration after all logical assertions that constitute a user interaction are performed. This corresponds exactly to the length of a logical transaction. See Section 2.4.4, "Logic Transactions".

Validation checking and reporting occur when a logical transaction is ended by using `Configuration.commitConfigTransaction(transaction)` or `Configuration.rollbackConfigTransaction(transaction)`.

After a committal or rollback, the CIO traverses the nodes of the Model, checking for validation failures, selected items and unsatisfied items. These are kept in a set of collections maintained on the Configuration.

At this point, you can call the following methods of `oracle.apps.cz.cio.Configuration`:

`getValidationFailures()`	Actually returns a collection of "ValidationFailure" objects. Call this after committing or rolling back a transaction, in order to inspect the list of validation failures.
`getSelectedItems()`	Returns a collection of selected items as a `StatusInfo` structure indicating the set of selected (true) items in the Configuration.
`getUnsatisfiedItems()`	Returns a collection of unsatisfied items as a `StatusInfo` structure indicating the set of unsatisfied items in the Configuration.

As nodes become selected they have a status of STATUS_NEW. If they continue to be selected since the last transaction their status is STATUS_EXISTING. If they become unselected, their status becomes STATUS_DELETED until the next transaction at which time they will be removed from the collection.

If you are writing a Functional Companion, the `validate()` method should return a list of `CompanionValidationFailure` objects in the event of a validation failure. This allows you to return more than one failure. Your `validate()` method can include several tests; you can track which ones failed, and why. See Section 2.9.3, "The validate() Interface Method".

Reference

For reference documentation, see: `ValidationFailure` and `Configuration`.

2.9 Standard Interface Methods for Functional Companions

You provide functionality for your Functional Companion by implementing body code for the methods described in this section. For particulars that apply to the languages currently supported by the CIO, and examples, see Section 1.3, "Building Functional Companions in Java".

These methods are invoked by your Oracle SellingPoint application, through the CIO, in response to program events or the actions of end users. The type of method invoked for each Component is determined when you associate the Component with a Functional Companion in Oracle Configurator Developer. See Section 1.5, "Incorporating Functional Companions in your Application" for details.

These methods are invoked by the CIO for each Functional Companion object that it creates for the Components in your Model. Note that your code does not invoke these methods directly; that is done by the CIO. Rather, you implement the body of each method, using the API provided by the CIO to communicate with your Model.

The body of any or all of these methods can be empty. Your Functional Companion object has to implement only those methods indicated in Oracle Configurator Developer.

The interface class that defines these methods is:

`oracle.apps.cz.cio.IFunctionalCompanion`

Reference

For reference documentation, see: `IFunctionalCompanion`.

Table 2–4 Standard methods of the IFunctionalCompanion interface

Method	Purpose	Details in
`initialize`	Saves information about the Model and performs any actions needed to initialize the Functional Companion.	Section 2.9.1
`autoConfigure`	Performs a programmatic configuration step.	Section 2.9.2
`validate`	Programmatically checks that a configuration is valid and throws a LogicalException object if the Model is not valid.	Section 2.9.3
`generateOutput`	Generates output for this Component, for either a thick or thin client.	Section 2.9.4
`terminate`	Performs any cleanup on this Functional Companion that needs to occur before the Companion is destroyed.	Section 2.9.5

2.9.1 The initialize() Interface Method

The `IFunctionalCompanion.initialize()` method is called when the companion is created. It connects a Functional Companion object to its configuration modeling environment (for example, a running instance of the Oracle SellingPoint application). Be aware that Functional Companions are created and initialized after all subcomponent instances are created for the current Component instance.

Your implementation of `initialize()` can include tasks that you wish to perform when the Functional Companion is first created. For example, you might wish to start writing audit messages to a log file, tracking the actions performed by your end users.

When an Oracle SellingPoint application runs, it creates runtime instances of all the Components in the Model and their associated Functional Companions. When a Functional Companion object is created, the CIO calls `initialize()` and passes the following input parameters:

Name	Type	Description
`node`	IRuntimeNode	The node instance associated with the Functional Companion being created. Specified in Configurator Developer. Currently, only Components can be specified in Configurator Developer.
`name`	String	The name of the Functional Companion. Specified in Configurator Developer.

Name	Type	Description
`description`	String	A description of the Functional Companion. Specified in Configurator Developer.
`id`	int	The database ID of the Functional Companion. Created internally.

> **Note:** It is worth emphasizing that the node passed as the first input parameter to `initialize()` is specified in Oracle Configurator Developer, when you create the Functional Companion rule that associates a Model node with your Functional Companion.

Your Functional Companion should ordinarily never directly call `FunctionalCompanion.initialize()`, since the CIO does that for you automatically. However, if your Functional Companion extends `FunctionalCompanion` as its base class, and you wish to perform some specialized initialization tasks, then the overriding `initialize()` method in *your* class should call `super.initialize()`. This passes some necessary variables to the superclass (`oracle.apps.cz.cio.FunctionalCompanion`) so that its methods will work.

It is not normally necessary to implement your own `initialize()` method in your Functional Companion. If you need to obtain the values of the input parameters of `FunctionalCompanion.initialize()` for use elsewhere in your Functional Companion, you can use the set of accessor methods of `FunctionalCompanion` already provided in the `oracle.apps.cz.cio.FunctionalCompanion` base class. Each of these methods returns the value of the corresponding input parameter:

`getRuntimeNode()`	Returns the runtime node to which this functional is associated.
`getName()`	Returns the name of the functional companion.
`getDescription()`	Returns the description of the functional companion.
`getID()`	Returns the database ID of the functional companion.

Note: Currently, in Configurator Developer, you can only associate a Functional Companion with a Component (which corresponds to the `node` parameter of `initialize()`. However, to accommodate possible future enhancement of Configurator Developer, the `IFunctionalCompanion` interface allows any runtime node to be associated with your Functional Companion.

Reference

For reference documentation, see: `initialize(IRuntimeNode, String, String, int)`.

2.9.2 The autoConfigure() Interface Method

The `IFunctionalCompanion.autoConfigure()` method is called at the request of the controlling User Interface, and can set states in the Model, add optional Components, etc.

This method performs an automatic configuration on the Model. This action can include changing the logical state of Options, or adding nodes underneath the selected Component instance in the Model tree.

Your implementation of `autoConfigure()` can include configuration actions that you wish to be performed before your end users arrive at a certain point in a configuration session, or as the result of certain choices that they make.

Reference

For reference documentation, see: `autoConfigure()`.

2.9.3 The validate() Interface Method

The `IFunctionalCompanion.validate()` method is called automatically when a logical transaction takes place, and should return a `List` of `CompanionValidationFailure` objects if the Model is not valid.

This method performs a functional validation for the selected Component instance each time the end user selects a node in the Model (for example, in the Configurations section of the Oracle SellingPoint application).

You should not modify the Model in a validation function. Doing so can cause unexpected application failures.

Your implementation of `validate()` can include tasks that you wish to perform whenever your end users make any selection. For example, you might wish to perform a calculation based on the object count of the selected Component, and present the end user with a notification if the result is outside a range that you define.

If the validation fails, then information about the failure is gathered by the CIO in a `List` of `CompanionValidationFailure` objects.

The general structure of your implementation of `validate()` should be:

1. Collect inputs from the Model.
2. Call a generic validation function that you define outside the body of `validate()`.
3. Propagate the result back as the value of the function, either null or a `List` of `CompanionValidationFailure` objects.

Reference

For reference documentation, see: `validate()`.

2.9.4 The generateOutput() Interface Method

The `generateOutput()` method is invoked at the request of the controlling User Interface.

Your implementation of `generateOutput()` might include tasks such as writing to a database, creating a report, or producing a visualization of the end user's configuration choices.

There are two versions of `generateOutput()`:

- "thick client" version

  ```
  public String generateOutput();
  ```

  A thick client architecture is one in which the configuration Model, and the user interface for manipulating it, both reside on the same client machine. The thick client architecture allows your Functional Companion's Output method to produce output windows directly on the client machine.

  This version is invoked when your Functional Companion operates with the Oracle SellingPoint application. (Note: The returned string is ignored.)

- "thin client" version

```
public void generateOutput(HttpServletResponse response) throws IOException
```

A thin client, browser-based architecture is one in which the configuration Model resides on a server, and the user interface resides on a client machine's web browser. The thin-client architecture allows your Functional Companion's Output method to produce out in web-browser windows.

This version is invoked when your Functional Companion operates in a web-based context.

See Section 4.3, "Thin-Client generateOutput() Functional Companion" for an example.

Currently, there is no mechanism for output generated through `generateOutput()` to provide feedback to the User Interface or the runtime Model.

Reference

For reference documentation, see: `generateOutput()` and `generateOutput(HttpServletResponse)`.

2.9.5 The terminate() Interface Method

The `IFunctionalCompanion.terminate()` method is called automatically by the CIO when the Component that the Functional Companion is attached to is deleted from the running Model.

Your implementation of this method can include tasks that you wish to perform when the Functional Companion is deleted. For example, if `initialize()` opens a file and reads some data, `terminate()` would close the file.

Your Functional Companion should ordinarily never directly call `FunctionalCompanion.terminate()`, since the CIO does that for you automatically. However, if your Functional Companion extends `FunctionalCompanion` as its base class, and you wish to perform some specialized termination tasks, then the overriding `terminate()` method in *your* class should call `super.terminate()`.

Reference

For reference documentation, see: `terminate()`.

3

Reference Documentation for the CIO

Reference documentation for the Oracle Configuration Interface Object is provided in the form of HTML pages generated by the Javadoc tool from the source code for the CIO.

For the main entry point to these pages, follow this link:

- CIO Package and Related Classes

Tips

Here are some tips on using the generated reference documentation:

- Use the Bookmarks pane to navigate through the reference.
- Use the Contents and Index to look up items alphabetically.
- Reminder: Constants are referred to in Java as “static variables,” and are listed under the heading “Fields” in the class in which they are defined.

4

Examples

This chapter contains code examples illustrating the use of Functional Companions and the CIO. These examples are fuller and longer than the examples provided in the rest of this document, which are often fragments. The examples here can be compiled and used. See the cited background sections for details.

The examples given here are all in Java, and were compiled with JDK 1.1.8.

4.1 Initializing the CIO

For background, see Section 2.3, "Initializing the CIO". This example is intended for custom user interfaces that use the CIO.

Example 4–1 Initializing the CIO (Long Example)

```
import java.io.*;
import java.sql.*;
import oracle.apps.cz.cio.*;
import oracle.apps.cz.common.*;

class cioExample

{
    private CIO InitializeCIO()
    {
        CIO cio = null;
        CZContext context = null;

        String jdbcDriver      = "com.ms.jdbc.odbc.JdbcOdbcDriver";    // Class
name of the JDBC driver
        String dbURL           = "jdbc:odbc:cioExample";
        String dbOwner         = "cioExample";
```

```
        String dbUsername      = "spx";
        String dbPassword      = "spx";
        String activeModelPath = "D:/orant/OSP/Shared/ActiveModel/";    //
Location of the LCE file

        try {
            // Load the JDBC driver
            Class.forName(jdbcDriver);

            // Establish a connection to the database
            context = new CZContext(dbURL, dbUsername, dbPassword, dbOwner);
        }
        catch (ClassNotFoundException cnfe) {
            System.out.println("Error loading class " + jdbcDriver);
            System.exit(0);
        }
        catch (SQLException sqle) {
            System.out.println("Error in creating Context");
            System.exit(0);
        }

        try {
            // Initialize the CIO
            cio = new CIO();
            cio.setActiveModelPath(activeModelPath);
        }
        catch (Exception e) {
            System.out.println("Exception in InitializeCIO");
            cio = null;
        }

        return cio;
    }
}
```

4.2 Basic Java Functional Companion

For background, see Section 1.3, "Building Functional Companions in Java".

Example 4–2 implements all of the types of Functional Companions, which are described in Section 1.1.1. The example implements the methods described in Section 2.9, and assumes the structure of the Model shown in Figure 1–1.

Example 4–2 Basic Functional Companion: FuncCompTest1

```
import oracle.apps.cz.cio.*;
import com.sun.java.util.collections.List;        // line 2
import com.sun.java.util.collections.ArrayList;
import com.sun.java.util.collections.Iterator;
import java.awt.*;
import java.awt.event.*;

public class FuncCompTest1 extends FunctionalCompanion
{
    oracle.apps.cz.cio.IRuntimeNode comp_node;   // currently, only Components

    Frame f;
    java.awt.List uiList;

    /**
     * Constructor:
     * Can be used for any necessary setup.
     */
    public FuncCompTest1()
    {
    }

    /**
     * Initialize: calls 'super' to get access to its functions.
     * @param comp_node - base node of functional companion  (currently, only
Components)
     * @param name - the name of the companion
     * @param description  - a description of the companion
     * @param id - the db id of the companion
     * All of these parameters, except 'id', are specified in the companion
     * definition in Developer.  id is created internally.
     */
    public void initialize(IRuntimeNode comp_node, String name, String
description, int id)
    {
        this.comp_node = comp_node;
        super.initialize(comp_node, name, description, id);  // line 35
    }

    /**
     * Functionality implementation:
     * There are three types of functionality for companions
     * Any number of them can be implemented in each companion.
```

```
     */

    /**
     * Type 1: Auto-configuration
     * If this method is defined, a button will appear in the UI and the code
will
     * be run when the button is clicked.  It is used to make changes to the
     * configuration.
     */
    public void autoConfigure()
        //This example simply checks if a certain value is true and, if so, sets
        //an integer feature value to 5;
    {
        OptionFeature of;
        Option op;
        IntegerFeature intFeat;
        oracle.apps.cz.cio.Component baseComponent;

        try
        {
            //get the necessary components from the configuration    // line 61
            baseComponent = (oracle.apps.cz.cio.Component)comp_
node.getChildByName("Component-1");
            of = (OptionFeature)baseComponent.getChildByName("Feature-1");
            op = (Option)of.getChildByName("Option-1");
            intFeat = (IntegerFeature)baseComponent.getChildByName("IF-1");
            //check if the option is set to UTRUE.  If so, set the Integer value
to 5
            if( op.getState() == IState.UTRUE )
                intFeat.setIntValue(5);
        }
        catch(Exception e){System.out.println(e);}
    }

    /**
     * Type 2: Validation
     * If this method is defined, the code will automatically be run any time
     * a change is made to the base node or one of its children in the
     * configuration.  It is used for ensuring that changes made result in a
     * valid configuration.  If not, the method returns a list of validation
failures.
     */
    public List validate()
        //This example defines 'min' (presumably the minimum amount this
customer
```

```
        //may order) and checks to see if the value equals at least this amount.
        //In the real world, you would want to get this value from your customer
        //database. For example, customers in foreign countries may have higher
minimums
        //since shipping is expensive.
    {
        int min = 8;
        int val = 0;
        IntegerFeature intFeat;
        ArrayList failures = new ArrayList();

        try
        {
            //get the value of the integer feature in the configuration
            oracle.apps.cz.cio.Component c = (oracle.apps.cz.cio.Component)comp_
node.getChildByName("Component-1");
            intFeat = (IntegerFeature)c.getChildByName("IF-1");
            val = intFeat.getIntValue();
        }
        catch(NoSuchChildException e){e.printStackTrace();}


       //check to see if the value in the config is not at least the min value
// line 102
        if( !(val >= min) )
            failures.add( new CompanionValidationFailure("Value less than
minimum", comp_node, this) );

        if(failures.isEmpty())
            return null;
        else
            return failures;
    }

    /**
     * Type 3: Output
     * If this method is defined, a button will appear in the UI which, when
     * pressed, will run the code below.  It is used to generate output to the
     * user. Note: this uses the "thick client" version of generateOutput().
     */
    public String generateOutput()
        //This example opens up a window with a list of some of the current
        //components of the configuration and their values.  This example is
very basic but
        //the idea here is that data from the configuration can be used to
```

```
generate reports,
        //graphs, models, etc.
    {
        IntegerFeature intFeat;
        TextFeature textFeat;
        com.sun.java.util.collections.List intFeatList, textFeatList;
        Iterator iter;

        //setup the UI
        if(f == null)
            {
                f = new Frame("Some info about this config");
                uiList = new java.awt.List();
                f.add(uiList);
                f.addWindowListener(new WindowAdapter(){
                    public void windowClosing(WindowEvent e)
                    {
                        f.dispose();
                    }
                });
            }
            uiList.removeAll();

        try
        {
            //get all the integer features
            intFeatList = comp_node.getChildrenByType(comp_node.INTEGER_
FEATURE);
            traverseTree(comp_node.getChildren(),
                         comp_node.INTEGER_FEATURE,
                         intFeatList);
            iter = intFeatList.iterator();

            //add the integer features to the UI
            while(iter.hasNext())
            {
                intFeat = (IntegerFeature)iter.next();
                String name = intFeat.getName();
                int val = intFeat.getIntValue();
                uiList.add("Integer Feature: " + name + " - " + val);
            }

            //get all the text features                              // line 167
            textFeatList = comp_node.getChildrenByType(comp_node.TEXT_FEATURE);
            traverseTree(comp_node.getChildren(),
```

```
                        comp_node.TEXT_FEATURE,
                        textFeatList);
            iter = textFeatList.iterator();

            //add the text features to the UI
            while(iter.hasNext())
            {
                textFeat = (TextFeature)iter.next();
                String name = textFeat.getName();
                String str = textFeat.getTextValue();
                uiList.add("Text Feature: " + name + " - " + str);
            }

            f.setSize(200,200);
            f.show();
        }
        catch(Exception e){e.printStackTrace();}

        return null;
    }

    /**
     * This function is used by generateOutput() to run through the config tree
and
     * pull out all of the items of a specified type.  It is not part of the
FunctionalCompanion API,
     * but was written for this specific companion.
     * @param children this is a list of all the children of the current node
     * @param type this is the type we are currently searching for
     * @param resultList all items of the specified type which are found are
added to this list
     */
    private void traverseTree(com.sun.java.util.collections.List children,
                              int type,
                              com.sun.java.util.collections.List resultList)
    {
        if(!children.isEmpty())
        {
            Iterator iter = children.iterator();

            while(iter.hasNext())
            {
                RuntimeNode rtn = (RuntimeNode)iter.next();
                resultList.addAll(rtn.getChildrenByType(type));
                traverseTree(rtn.getChildren(),
```

```
                                type,
                                resultList);
                }
            }
        }
}
```

Notes on the example

Line 2

```
import com.sun.java.util.collections.List; // line 2
```

If you are using JDK 1.1.x, import `com.sun.java.util.collections.List`, which is provided in `collections.jar` (see Section 1.3.2, "Installation Requirements for Java Functional Companions" on page 1-8). If you are using JDK 1.2, then import `java.util.List`.

Line 35

```
super.initialize(comp_node, name, description, id); // line 35
```

In the `initialize()` method, call `super.initialize()`. This passes some of the necessary variables to the superclass so that its methods will work.

Lines 61-68

```
//get the necessary components from the configuration // line 61
```

This block illustrates how to get the logical state of an Option (with `getState`), test the logical state (with the expression `== IState.UTRUE`), and set the value of a Feature (with `setIntValue`).

Lines 102-109

```
//check to see if the value in the config is not at least the min value // line
102
```

This block produces the Configuration Status message shown in Figure 1–3, "Testing Functional Companions in the Oracle SellingPoint application."

4.3 Thin-Client generateOutput() Functional Companion

This Functional Companion uses the "thin-client" version of `generateOutput()` (see Section 2.9.4 on page 2-25). When you invoke the Functional Companion from a web browser, it produces an HTML representation of the runtime Model tree, beginning at the node to which the companion is attached.

In order to use this type of Functional Companion, you must use Oracle Configurator Internet Edition (OCIE). See the *Oracle Configurator Internet Edition Developer's Guide* for details not covered in this document. Here is a summary of the tasks:

- Compile the Java source code into a class file.
- In Configurator Developer, define a Functional Companion rule:
  - Type is Output
  - Base Component is the Component to which you want to attach the Functional Companion
  - Implementation language is Java
  - Program String is the name of the class file
- In Configurator Developer's User Interface module, define a button for the Component that invokes the Functional Companion.
- In Oracle Application Server, create an OCIE servlet.
- Add the new class file for the Functional Companion to the CLASSPATH environment variable for the servlet.
- You can test the Functional Companion from Configurator Developer, by specifying the URL of the servlet (in Tools>Options>Test>Servlet URL) and clicking the Test button. This opens a web browser, passing it a URL that includes an XMLmsg parameter containing the necessary OCIE initialization message. This message contains database connection and login strings, and specifies the Model to display, by means of the `ui_def_id` parameter that identifies the User Interface definition you created in Configurator Developer.
- You can test the Functional Companion outside Configurator Developer, by creating an HTML test page that substitutes for your host application. (Examples are provided in the *Oracle Configurator Internet Edition Developer's Guide*.) This page sends an OCIE initialization message that specifies database connection and login information, and the Model containing the Component. You can copy these parameters from the URL produced by the Test button in

Configurator Developer. Test the Functional Companion by opening the HTML test page.

The example first calls the `response.setContentType()` method of the HttpServletResponse class, passing "text/html" as the output type. Then it calls `response.getWriter()` to get an output stream to which the Functional Companion can write HTML.

You can also write non-HTML output by setting another content type (a MIME type) and writing appropriate data to the output stream.

Example 4–3 Thin-client Output Functional Companion

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.http.HttpServletResponse;
import com.sun.java.util.collections.Iterator;
import oracle.apps.cz.cio.FunctionalCompanion;
import oracle.apps.cz.cio.IRuntimeNode;

public class ShowStructure extends FunctionalCompanion {

  public void generateOutput(HttpServletResponse response) throws IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Runtime Model Structure</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<h3>Runtime Model Structure</h3>");
    IRuntimeNode rootNode = getRuntimeNode();
    generateNode(out, rootNode, 0);
    out.println("</body>");
    out.println("</html>");
  }

  private static void generateNode(PrintWriter out, IRuntimeNode node, int
level) throws IOException {
    for (int i = 0; i < level; ++i) {
      out.print("--");
    }
    out.println(node.getName() + " <br> ");
    for (Iterator i = node.getChildren().iterator(); i.hasNext(); ) {
      IRuntimeNode childNode = (IRuntimeNode)i.next();
```

```
      generateNode(out, childNode, (level + 1));
    }
  }
}
```

Glossary

This glossary for Oracle Configurator is followed by a Glossary of Acronyms

Acceptance Test

Test for validating the system (the correctness of results). Acceptance tests are based on acceptance criteria specified in the project's Test Plan.

Active Model

The part of Oracle Configurator runtime architecture that processes model structure and rules to create configurations. Interfaces dynamically with the end user Active UI and data.

Active User Interface

The part of Oracle Configurator runtime architecture that provides the views necessary to create configurations interactively. Interfaces with the Active Model and data to give users access to customer requirements gathering, product selection, and customer-centric extensions.

Alpha

An internal release of the application before implementation is complete, delivered as a build, and subject to integration, verification, and system testing.

Application

The Oracle Configurator or Oracle SellingPoint application. The end-user runtime environment that provides configuration functionality and output. Also called sales configuration application or enterprise selling system. See also Oracle Configurator.

Application Architecture

The software structure of an application at runtime. Architecture affects how an application is used, maintained, extended, and changed.

Application Architecture and Design Document

Document presenting the overall architecture for the application and how the application will be implemented.

Application Design

The task in the Oracle Configurator Deployment Methodology Implementation stage of a project for determining, documenting, reviewing, and delivering the scheme that will turn user requirements into an operational application. Occurs in parallel with the end of the Test Case Definition task and the beginning of the Construction task. Application Design results in an Application Architecture and Design Document.

Application Development

See Construction.

Application Implementer

The person who uses Oracle Configurator Developer to construct an Oracle Configurator application or the model structure, rules, and UI customizations for a Oracle Configurator. The test application generated by Oracle Configurator Developer is the Oracle SellingPoint application.

Application Testing

See Full Application Testing.

Architecture

The software structure of a system. Architecture affects how a system is used, maintained, extended, and changed. See also Application Architecture.

Beta

An external release, delivered as an installable application, and subject to system, validation, and acceptance testing. Specially selected and prepared end users may participate in beta testing.

Bill of Material

A list of component items associated with a parent item (assembly) and information about how each item relates to the parent item.

BOM

See Bill of Material.

BOM Item

The nodes imported into the Oracle Configurator Developer Model that correspond to an Oracle BOM.

BOM Model

The imported Model node in the Oracle Configurator Developer that corresponds to Standard Model in an Oracle BOM.

BOM OptionClass

The imported Model node in the Oracle Configurator Developer that corresponds to Option Class in an Oracle BOM.

BOM StandardItem

The imported Model node in the Oracle Configurator Developer that corresponds to Standard Item in an Oracle BOM.

Boolean Expression

An element of a component in the Model that has two options: true or false.

Bug

See Defect.

Build

A specific instance of an application during its construction. A build must have an install early in the project so that application implementers can unit test their latest work in the context of the entire available application.

Change Control Board

A group of people responsible for evaluating Change Request Forms, approving or rejecting them, and notifying affected parties of how each one was resolved.

Change Control Procedures

A plan that describes how change control will be conducted during a project.

Change Request Form

A form used to propose changes as part of a standard change control process. A Change Request Form typically includes a description of the proposed change and an evaluation of impacts on cost and schedule.

CIO

See Oracle Configuration Interface Object.

CIO protocols support creating and navigating the Model, querying and modifying selection states, and saving and restoring configurations.

Client

A runtime program using a server to access functionality shared with other clients.

Comparison Rule

A relationship that determines the selection state of a logical item (option, boolean feature, or list-of-options feature) based on a comparison of two numeric values (numeric features, totals, resources, option counts, or numeric constants). The numeric values being compared can be computed or they can be discrete intervals in a continuous numeric input.

Compatibility

A relationship among features in the Model that specifies the allowable combinations of options.

Compatibility Rule

A kind of compatibility relationship where the allowable combinations of options are specified implicitly by relationships between property values of the options.

Compatibility Table

A type of compatibility relationship where the allowable combination of options are explicitly enumerated.

Component

Represents a configurable element in a product. An element of the Model typically containing features.

Component Set

An element of the Model that contains a number of components of the same type, where each component of the set is independently configured.

Configuration Management

A process for managing the versions of the application and its documentation during construction.

Configuration Model

The model structure and rules-based content of an Oracle Configurator or Oracle SellingPoint application. The configuration model is constructed and maintained using Oracle Configurator Developer, and is interpreted at runtime by the Active Model.

Configuration Rules

The logic rules and numeric rules available for defining configurations.

Configurator

The part of the Oracle Configurator or Oracle SellingPoint application that provides custom configuration capabilities.

Constraint Rule

A logical relationship amongst features and options. See also Rules.

Construction

The task in the Oracle Configurator Deployment Methodology Implementation stage of a project for building the Oracle Configurator using Oracle Configurator Developer. Construction is based on the user's requirements and an approved application design. Occurs in parallel with completion of the Design task. Construction includes reviews and testing.

Contributes

A numeric rule for accumulating a total value.

Consumes

A numeric rule for specifying the quantity of a resource used.

Core Functionality

Also called Phase 1. The first release of the application delivered in 14 weeks. After validation with a subset of users, the core functionality application can be fully deployed to all intended users, maintained, or extended to offer additional product families or more functionality. See Full Deployment, Additional Product Families, and Extended Functionality.

CRM

Customer Relationship Management. The aspect of the enterprise that involves contact with customers, from lead generation to support services.

Customer

The person or persons for whom products are configured by end users of the Oracle Configurator or other Order Management and CRM applications.

Customer-centric Views

Optional extensions to core functionality that supplement product selection with rules for pre-selection, validation, and intelligent views. View capabilities include generative geometry, drawings, sketches and schematics, charts, performance analyses, and ROI calculations.

Customer-centric Extensions

See Customer-centric Views.

Customer Requirements

The needs of the customer that serve as the basis for determining the configuration of products, systems, and/or services. Not to be confused with Requirements, a task in Oracle Configurator Deployment Methodology.

Data Import

Populating the Oracle Configurator Database with enterprise data from ERP or legacy systems via import tables.

Data Integration Object

Data Integration Object. A server in the runtime application that creates and manages the interface between the client (usually a user interface like the Active User Interface) and the Oracle Configurator Database.

Data Maintenance Environment

The environment in which the Oracle Configurator or Oracle SellingPoint application data is maintained.

Data Replication

The activity of downloading and uploading configuration, quote, and order data between the Oracle Configurator Database on the enterprise server and Oracle Configurator Mobile Database on end-user mobile laptop PCs. See also Data Synchronization.

Datasource

A programmatic reference to a database.

Data Synchronization

A process for matching the data in the Oracle Configurator Database and the data available to client processes such as the Oracle Configurator. See also Data Replication.

Default

The automatic selection of an option based on the pre-selection rules or the selection of another option.

Defaults

A logic rule to determine the logic state of features or options in a default relation to other features and options. For instance, if you set A to True by selecting it, B becomes true (selected) if it is available (not false) and can be set to True without contradicting a non-default rule or a previous default setting for B.

Defect

A failure in a product to satisfy the users' requirements. Defects are prioritized as critical, major, or minor, and fixes range from corrections or workarounds to enhancements. Also known as a “bug”.

Defect Tracking

A system of identifying defects for managing additional tests, testing, and approval for release to users.

Definition

Defining and scoping the first phase of a project, and selecting a vendor such as Oracle to implement a sales configuration application.

Deliverable

A work product that is specified for review and delivery.

Delivery

The task in the Oracle Configurator Deployment Methodology Implementation stage of a project for organizing a deployment of the application. Includes beta testing.

Demonstration

A presentation of the tested application, showing a particular usage scenario.

Deployment

The stage in a project between Implementation and Maintenance when the fully operational application is distributed to users. See also Pilot and Full Deployment.

Design

See Application Design.

Design Chart

An Oracle Configurator Developer rule type for defining advanced Explicit Compatibilities interactively in a chart view.

Design Review

A technical review that focuses on application or system design.

Development

See Construction.

DIO

See Data Integration Object.

Domain Expert

A member of the customer's staff who has specific product or process knowledge needed in the Oracle Configurator or Oracle SellingPoint application.

End User

The ultimate user of the Oracle Configurator or Oracle SellingPoint application. The types of end users vary by project but may include salespeople or distributors, administrative office staff, marketing personnel, order entry personnel, product engineers, or customers directly accessing the application via web or kiosk.

Enterprise

The systems and resources of a business.

Environment

The arena in which software tools are used, such as operating system, applications, and server processes.

ERP

Enterprise Resource Planning. A software system and process that provides automation for the customer's back-room operations, including order processing.

Excludes

A logic rule to determine the logic state of features or options in an excluding relation to other features and options. For instance, if you set A to True, B becomes false, since it is not allowed when A is true. If you set A to False, there is no effect on B, meaning it could be true, false, or unknown.

Extended Functionality

A release after delivery of core functionality that extends that core functionality with customer-centric views, more complex proposal generation, discounting, quoting, and expanded integration with ERP, OMS, and COM-compliant third-party software.

Feature

An element of the Model. A configurable parameter of a component. Features can either have a value (numeric or boolean) or enumerated options.

Full Application Testing

The task in the Oracle Configurator Deployment Methodology Implementation stage of a project for testing the constructed application prior to delivery. Full Application Testing results in a completely validated application approved for delivery to users.

Full Deployment

A release of the application to all intended users after implementation and validation of the core functionality.

Full Roll Out

An external release delivered to all intended end users of the application. See also Full Deployment.

Functional Companion

An object associated with a component that supplies methods that can be used to initialize, validate and generate customer-centric views and outputs for the configuration.

Functional Specification

Document describing the functionality of the application based on user requirements.

Incremental Construction

The process of organizing the construction of the application into builds, where each build is designed to meet a specified portion of the overall requirements and is unit tested.

Increments

A logical relation that increments a count or value associated with an item by an integer quantity.

Implementation

The stage in a project between defining the problem by selecting a configuration technology vendor, such as Oracle, and deploying the completed sales configuration application. The Implementation stage includes gathering requirements, defining test cases, designing the application, constructing and testing the application, and delivering it to users.

Implies

A logic rule that determines the logic state of features or options in an implied relation to other features and options. For instance, if you set A to True by selecting it, B becomes true, since selecting A implies that B is also selected. If you set A to False by deselecting it, there is no effect on B, meaning it could be true false or unknown based on other relations B participates in. And if you set B to True by selecting it, there is no effect on A, meaning it could be true false or unknown based on other relations A participates in. But if you set B to False by deselecting it, the relation of A implies B is preserved only by having A be false (deselected) as well.

Import Tables

Tables mirroring the Oracle Configurator Database Item Master structure, but without integrity constraints. Import Tables allow batch population of the Oracle Configurator Database Item Master. Import Tables are used in conjunction with extractions from Oracle Applications or legacy data to create, update, or delete records in the Oracle Configurator Database Item Master.

Install

A program that sets up the local machine and installs the application for testing and use.

Integration

The process of combining multiple software components and making them work together.

Integration Testing

Testing the interaction among software programs that have been integrated into an application or system.

Intelligent Views

Configuration output, such as reports, graphs, schematics, and diagrams, that help to illustrate the value proposition of what is being sold.

Item Master

A table in the Oracle Configurator Database containing data used to structure the product. Data in the item master is either entered manually or imported from Oracle Applications or legacy data.

Item Type

A table in the Oracle Configurator Database containing data used to classify the product data in the item master table.

Logic Rules

Logic rules directly or indirectly set the logical state (true, false, or unknown) of features and options in the Model.

There are four (4) primary logic rules: Implies, Requires, Excludes, and Negates. Each of these rules takes a list of features or options as operands. See also Logic, Implies, Requires, Excludes, and Negates.

Maintenance

The effort of keeping a system running once it has been deployed, through bug fixes, procedure changes, infrastructure adjustments, data replication schedules, etc.

Maintainability

The characteristic of a product or process to allow straightforward maintenance, alteration, and extension. Maintainability must be built into the product or process from inception.

Maintenance Guide

A guide for maintaining a specific application or system. The maintenance guide covers all aspects of maintenance described in the generic Maintenance Plan.

Maintenance Plan

A document that outlines what is required for successful maintenance, and who is responsible for all the actions and deliverables of carrying out maintenance on a system. Oracle's Application and System Maintenance Plan presents a generic model of activities and deliverables necessary for successful maintenance.

OC

See Oracle Configurator.

Methodology

A standard, step-by-step process designed to achieve consistent, reliable results. Oracle Configurator Deployment Methodology is a repeatable implementation process based on software development standards and Oracle Configurator implementation best practices.

Mobile Database

See Oracle Configurator Mobile Database.

Model

The entire hierarchical “tree” view of all the data required for configurations, including model structure, variables such as resources and totals, and elements in support of intermediary rules. May consist of BOM Items.

Oracle Configurator

The views of the model structure and rules generated by the Active UI to present end users with interactive product selection based on configuration models.

Model Structure

Hierarchical, “tree” view of data in terms of product elements (models, components, features, and options). May include reusable components.

MRP

Manufacturing Resource Planning. A software system and process for monitoring and maintaining the customer's manufacturing systems.

Negates

A logic rule to determine the logic state of features or options in a negating relation to other features and options. For instance, if you set one item in the relationship to True, the other item must be false. And if you set one item to False, the other item must be true.

Next Phase

The phase following Phase 1 Core Functionality, consisting of Full Deployment of Phase 1, expanding the application for coverage of additional Product Families or extending the functionality to include more complexity and customer-centric views.

Node

The place in a Model occupied by a component, feature, option or variable, BOM Model, BOM OptionClass, or BOM StandardItem.

Numeric Rules

Rules that are used to set the global parameters specified in product structuring. These include Contributes, Supplies, and Consumes. See also Numeric Rules, Contributes, Supplies, and Consumes.

OC

See Oracle Configurator.

Opportunity

The workspace in the Oracle SellingPoint application and Oracle Field Sales in which products, systems, and/or services are configured, quotes and proposals are generated, and orders are submitted.

Option

An element of the Model. A choice for the value of an enumerated feature.

A logical selection made by the end user when configuring a component.

Oracle Configurator

The product family consisting of development tools and runtime applications such as Oracle Configurator Developer and Oracle Configurator Sales Edition, variously packaged for use in networked, mobile, or web deployments.

Oracle Configurator Database

The implementation version of the standard Oracle Configurator or Oracle SellingPoint application data-warehousing schema that manages data for the configuration model. The implementation schema includes all the data required for the runtime system as well as specific tables used during the construction of the application.

Oracle Configurator Deployment Methodology

The methodology of stages, tasks, steps, and activities to deliver a core functionality, Phase 1 Oracle SellingPoint application or Oracle Configurator to users in 14 weeks. Oracle Configurator Deployment Methodology includes templates and checklists for organizing and managing an Oracle Configurator project.

Oracle Configurator Deployment Methodology Program

The rapid application development program that combines the tools of Oracle Configurator Developer with the application construction methodology and project management methods of Oracle Configurator Deployment Methodology to ensure delivery of a core functionality Phase 1 Oracle SellingPoint application or Oracle Configurator in 14-weeks. Oracle Education provides books and courses in support of using Oracle Configurator Developer and Oracle Configurator Deployment Methodology most effectively. See also Oracle Configurator Developer and Oracle Configurator Deployment Methodology.

Oracle Configuration Interface Object

A server in the runtime application that creates and manages the interface between the client (usually a user interface like the Active User Interface) and the underlying representation of model structure and rules in the Active Model.

Oracle Configurator Mobile Database

The runtime version of the standard Oracle Configurator Database that manages data for the configuration model. The runtime schema includes customer, product, and pricing data as well as data created during operation of an Oracle Configurator.

Oracle Configurator Developer

The suite of tools in the Oracle Configurator product for constructing and maintaining sales configuration applications.

Oracle Configurator

The end-user application created with the Oracle Configurator Developer product. See also Application, Oracle Configurator and Oracle SellingPoint application.

Oracle Configurator Architecture

The application runtime architecture consists of the Active User Interface, the Active Model, and the Oracle Configurator Database or Oracle Configurator Mobile Database. The application development architecture consists of Oracle Configurator Developer and the Oracle Configurator Database.

Oracle SellingPoint application

The test application generated by Oracle Configurator Developer.

Output

The output generated by the sales configuration application, such as quotes, proposals, bills of material (BOM), and customer-centric views.

PDM

Product Data Management. A software system that manages the version control of product data.

Phase 1 (one)

All the sales configuration functionality that can be implemented within the Active User Interface and Oracle Configurator Developer within the 14-week Oracle Configurator Deployment Methodology program, and nothing that requires programming outside of these environments. See also Core Functionality.

Pilot

An external release to a subset of 20-25 end users for validation of the system.

Populator

An entity in the Oracle Configurator Developer that defines how to create a Model from information in the item master.

Preliminary Project Plan

An initial high-level project plan and schedule describing the events in the project in terms of time relations rather than specific delivery dates.

Pre-selection

The default state in a sales configuration application that defines an initial selection of components, features, and options for sales configuration.

A process that is implemented to select the initial element(s) of the configuration.

Principal Design Consultant

Member of the project team responsible for architecting the design of the application.

Product

Whatever is subjected to configuration and is the output of the application.

The root element of the Model.

Product Family

A collection of products or product lines, which are organized as a group by a provider or manufacturer.

Product Maintenance

A release of the application after delivery of core functionality that adds wider product coverage or keeps product data correct.

Product Structure

See Model Structure

Project

A project is the process of implementing and delivering an Oracle Configurator or Oracle SellingPoint application.

A Project in Oracle Configurator Developer is the workspace in which sales configuration applications are constructed.

Project Manager

A member of the project team who is responsible for directing the project during implementation.

Project Plan

A document that outlines the logistics of successfully implementing the project, including the schedule.

Property

A named value associated with an object in the Model or the item master. A set of properties may be associated with an item type.

Prototype

A construction technique in which a preliminary version of the application, or part of the application, is built to facilitate user feedback, to prove feasibility or examine other implementation issues.

Reference

The use of a reusable component within the Model. Not implemented in Release 11*i* or before.

Regression Test

An automated test that ensures the newest build still meets previously tested requirements and functionality.

Requirements

The task in the Oracle Configurator Deployment Methodology Implementation stage of a project when the project team explores and understands what the application will do. Occurs in parallel with starting the Implementation task Test Case Definition. Requirements gathering results in a Functional Specification.

Requires

A logic rule to determine the logic state of features or options in a requirement relation to other features and options. For instance, if you set one item in the relationship to True, the other item is required to be true as well. And if you set one item to False, the other item must be false as well.

Resource

Staff or materials available or needed within an enterprise.

A variable in the Model used to maintain the balance of features not consuming more of a specific resource than has been provided by other features.

Reusable Component

A component that is referenced from multiple locations in the Model. Not implemented in Release 11*i* or before.

Reusability

The extent to and ease with which parts of a system can be put to use in other systems.

Roll Out

See Full Roll Out.

Rules

Also called business rules or configuration rules. Constraints applied among elements of the product to ensure that defined relationships are preserved during configuration. Elements of the product are components, features, and options. Rules express logic, numeric parameters, implicit compatibility, or explicit compatibility. Rules are used to provide pre-selection and validation capability in an application.

See also Logic Rules and Numeric Rules.

Runtime

The environment and context in which applications are run or used, rather than developed.

Sales Configuration

A part of the sales process to which configuration technology has been applied in order to increase sales effectiveness and decrease order errors. Commonly identifies needs assessment and product configuration.

SellingPoint

The configuration engine used in the Oracle Configurator.

Server

Centrally located software processes or hardware, shared by clients.

Solution

The deployed system as a response to a problem or problems.

Statement of Work

Document describing the work required to deliver an application based on pre-sales scoping activities.

Studio

See Oracle Configurator Developer.

Supplies

A numeric rule for specifying how much of a resource is available.

System

The hardware and software components and infrastructure integrated to satisfy sales configuration requirements.

System Project

The project of implementing the solution, including the Oracle Configurator or Oracle SellingPoint application. See also Project.

System User

Any user in contact with the system.

Test Case

A description of inputs, execution instructions, and expected results, which are created for the purpose of determining whether a specific software feature works correctly or a specific requirement has been met.

Test Case Definition

The task in the Oracle Configurator Deployment Methodology Implementation stage of a project for defining the Test Cases and describing the tests that will be performed to validate the application. Occurs in parallel with the end of the Requirements task and much of the Design task. Test Case Definition results in a detailed Test Plan for the project.

Testing

See Full Application Testing

Test Log

A record of what tests were run at what time on which version of the application to what effect.

Test Plan

The plan for defining and executing tests.

Timeline

The schedule for completing the tasks and activities required to implement a phase of the sales configuration application project.

Total

A variable in the Model used to accumulate a numeric total, such as total price or total weight.

Training

Training that prepares Oracle Configurator Deployment Methodology program users for creating, changing, extending, and supporting the application.

Training that prepares the application end user for operating the system.

Training Guide

A guide created by members of the project team to instruct end users how to use the system. See also User's Guide.

Undetermined

The logic state that is neither true nor false, but unknown at the time a logic rule is executed.

Unit Test

Execution of individual routines and modules by the application implementer or by an independent test consultant for the purposes of finding defects.

User

The person using the Oracle Configurator tools and methods to build an Oracle Configurator or Oracle SellingPoint application. See also End user.

User Interface

The visible part of the application, including menus, dialog boxes, and other on-screen elements. The part of a system where the user interacts with the software.

User Requirements

A description of what the Oracle Configurator or Oracle SellingPoint application is expected to do from the end user's perspective.

User's Guide

Documentation on using the application to solve the intended problem. See also Training Guide.

Validation

Tests that ensure that the configured components will meet specific performance or acceptance criteria.

A type of functional companion that is implemented to ensure that the configured components will meet specific performance or acceptance criteria.

Variable

Parts of the Model that represent either totals or resources.

Verification

Tests that check whether the result agrees with the specification.

Glossary of Acronyms

API

Application Programming Interface

ATP

Available to Promise

BOM

Bill of Material

CIO

Configuration Interface Object

CM

Configuration Management

COM

Component Object Module

CRM

Customer Relationship Management

DBMS

Database Management System

DCOM

Distributed Component Object Modeling

DHTML

Dynamic Hypertext Markup Language

DIO

Data Integration Object

DLL

Dynamically Linked Library

DXF

Drawing Exchange Format (AutoCAD drawings)

ECO

Engineering Change Order

ERM

Enterprise Relationship Management

ERP

Enterprise Resource Planning

ESD

Electronic Software Distribution

ESP

External Service Provider

ESS

Enterprise Selling System

GSE

Generative Specification Environment

GSL

Generative Specification Language

HT

High Tech

HTML

Hypertext Markup Language

IP

Industrial Products

IS

Information Services

ISS

Interactive Selling System

ISV

Independent Software Vendor

LAN

Local Area Network

LCE

Logical Configuration Engine

MAPI

Messaging Application Programming Interface

MC/S

Mobile Client/Server System

MDUI

Model-Driven User Interface

MES

Marketing Encyclopedia System (Catalog)

MIS

Management Information Systems

MRP

Manufacturing Resource Planning

MS
Microsoft

OC
Oracle Configurator

OCX
Object Control File, OLE custom controls

ODBC
Open Database Connectivity

OLE
Object linking and embedding

OMS
Opportunity Management System

OOD
Object-Oriented Design

ORB
Object Request Broker

PDM
Product Data Management

PIA
Project Impact Assessment

PM
Project Manager

POS
Point of Sale

QA
Quality Assurance

RAD

Rapid Application Development

RDBMS

Relational Database Management System

RFQ

Request for Quote

ROI

Return on Investment

SAS

Sales Analysis System

SCM

Supply Chain Management

SCS

Sales Configuration System

SE

Sales Engineer

SFA

Sales Force Automation

SI

System Integrator

SOT

Strategic Options Theory

SOW

Statement of Work

CDBI

Configurator Database Interface

SQA
Software Quality Assurance

SQL
Structured Query Language

TERM
Technology-Enabled Relationship Management

TES
Technology-Enabled Selling

UI
User Interface

VAR
Value-Added Reseller

VB
Microsoft Visual Basic

WAN
Wide Area Network

WIP
Work In Progress

Y2K
Year 2000 Compliant

A

CIO Package and Related Classes

Package Summary	
Packages	
Package oracle.apps.cz.cio	Provides classes used to create, save and restore configurations.
Package oracle.apps.cz.common	
Package oracle.apps.cz.utilities	

B

Package oracle.apps.cz.cio

Description

Provides classes used to create, save and restore configurations. The top-level entry point to this package is the configuration integration object, which provides methods for creating, saving, restoring and deleting configurations. The configuration can be manipulated by calling methods on the configuration object and its tree of runtime objects.

Class Summary

Interfaces

IAtp	Implemented by objects that can have ATP calculated.
IBomItem	Implemented by all selectable BOM items.
ICompSetEventListener	Implemented by objects that want to find out about added components.
IConfigEventListener	Implemented by objects that want to find out about added components.
ICount	Implemented by objects that have an associated integer count.
IDecimal	Implemented by objects that can both get and set a decimal value.
IDecimalMinMax	Implemented by objects that have a decimal minimum and maximum value.
IFunctionalCompanion	Implemented by functional companion objects attached to components in order to provide programatic functionality to a configuration model.
IInteger	Implemented by objects that have an integer value.
IIntegerMinMax	Implemented by objects that have an integer minimum and maximum.
IOption	Implemented by objects that act as options.
IOptionFeature	Implemented by objects that contain selectable options.

Class Summary	
IPrice	Implemented by objects that can be priced.
IReadOnlyDecimal	Implemented by objects that have a decimal value.
IRuntimeNode	Implemented by all objects in the runtime configuration tree.
IState	Implemented by objects that have logic state.
IText	Implemented by objects that have a textual value.
Classes	
BomModel	Represents configurable BOM Models.
BomNode	Represents configurable BOM option classes.
BomOptionClass	Represents configurable BOM option classes.
BomStdItem	Represents configurable BOM models.
BooleanFeature	Represents a feature with a boolean value.
CIO	Implements a configuration integration object that can be used to create, save, restore and delete configurations.
CompanionNode	Provides methods for extracting values from a model based on property annotations.
CompanionRoot	Provides functional companion implementors with property-based feature extraction.
CompanionValidationFailure	Failure produced by a functional companion object.
Component	Implements the runtime configuration behavior of products and components.
ComponentNode	Represents a set of configurable components.
ComponentSet	Represents a set of configurable components.
ConfigTransaction	Represents a configuration transaction.
Configuration	The top-level entry point to a configuration.
CountFeature	Represents a countable integer feature.
DecimalFeature	Represents a feature with a decimal value.
DecimalNode	An abstract class implementing behavior common to objects with a decimal value.
Factory	Provides a class factory for the CIO to clients that do not support arguments to constructors.
FunctionalCompanion	Base object on which user functional companions can be based.

Class Summary	
FunctionalCompanionException	This exception is used to indicate that an error occured somewhere inside the functional companion.
IntegerFeature	Represents a feature with an integer value.
IntegerNode	Represents a feature with an integer value.
Option	Represents an option of an option feature.
OptionFeature	Represents a feature with selectable options.
OptionFeatureNode	An abstract class implementing behavior commont to all features with options.
OptionNode	An abstract class implementing behavior common to all option-like objects.
Property	Represents name/value properties associated with runtime nodes.
ReadOnlyDecimalNode	An abstract class implementing behavior common to objects with a decimal value.
Reason	This class wraps the information returned by a contradiction in order to include information about internal error messages.
Resource	Represents a consumable resource.
RestoreValidationFailure	Failure produced when restoring a configuration over a changed model.
RuntimeNode	Abstract class implementing common behavior across all runtime nodes.
StateCountNode	Abstract class implementing common behavior for nodes with a logic state and count.
StateNode	Abstract class implementing common behavior across nodes with logic state.
StatusInfo	Contains information about a status change for a particular runtime node.
TextFeature	Represents a feature that has a textual value.
TextNode	Represents a feature that has a textual value.
Total	Represents a total that has a decimal numeric value.
ValidationFailure	Implements behavior common to all validation failures.
Exceptions	
AtpUnavailableException	Signals that the CIO ATP calculation functionality is not available.
BomExplosionException	Exception which is thrown when a client tries to create a configuration directly from an Apps bill of material and there is a problem with the explosion of the bill.
FuncCompCreationException	Signalled if a functional companion cannot be created.

Class Summary	
IncompatibleInputException	Signalled if a particular input is of different type than the node it is trying to restore over.
LogicalException	Signalled if a logical failure occurs.
LogicalOverridableException	Signalled if a logical contradiction occurs that can be overriden.
LogicalRuntimeException	Signalled if a fatal logic exception occured.
MissingFileException	Signalled if a particular logic file is missing.
MissingLogicException	Signalled if a particular logic record is missing.
NoAtpCalculatedException	Exception which is thrown when an ATP method is called on an item for which ATP is not calculated.
NoConfigHeaderException	Signalled if the configuration hasn't been saved yet.
NonPricedNodeException	Exception which is thrown when a pricing method is called on an item which should not be priced.
NoSuchChildException	Signalled if a requested child does not exist.
NotOneProductException	Exception which is thrown when a client tries to create a configuration by specifying the name of the project and the project contains more than one or no products.
NotOneProjectException	Exception which is thrown when a client tries to create a configuration by specifying the name of the project and the project name identifies more than one or no projects.
PricingUnavailableException	Signals that the CIO pricing functionality is not available.
SelectionNotMutexedException	Signalled when an mutexed selection operation is performed on an option feature that does not support mutexed selection.
TransactionException	Signalled if a particular logic file is missing.

oracle.apps.cz.cio
AtpUnavailableException

Syntax

```
public class AtpUnavailableException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.apps.cz.cio.AtpUnavailableException
```

All Implemented Interfaces:

`java.io.Serializable`

Description

Signals that the CIO ATP calculation functionality is not available.

Member Summary

Constructors

AtpUnavailableException(String)

AtpUnavailableException(String, Object, Log)

Inherited Member Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructors

AtpUnavailableException(String)

```
public  AtpUnavailableException(java.lang.String reason)
```

AtpUnavailableException(String, Object, Log)

```
public  AtpUnavailableException(java.lang.String reason, java.lang.Object
source, oracle.apps.fnd.common.Log log)
```

oracle.apps.cz.cio
BomExplosionException

Syntax

```
public class BomExplosionException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.apps.cz.cio.BomExplosionException
```

All Implemented Interfaces:

java.io.Serializable

Description

Exception which is thrown when a client tries to create a configuration directly from an Apps bill of material and there is a problem with the explosion of the bill.

Member Summary

Methods

getExplosionDate()

getInventoryItemId()

getOrganizationId()

Inherited Member Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

Inherited Member Summary

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods

getExplosionDate()

```
public java.util.Date getExplosionDate()
```

getInventoryItemId()

```
public int getInventoryItemId()
```

getOrganizationId()

```
public int getOrganizationId()
```

oracle.apps.cz.cio
BomModel

Syntax

```
public class BomModel extends BomNode

java.lang.Object
  |
  +--RuntimeNode
        |
        +--StateNode
              |
              +--StateCountNode
                    |
                    +--OptionFeatureNode
                          |
                          +--BomNode
                                |
                                +--oracle.apps.cz.cio.BomModel
```

All Implemented Interfaces:

IAtp, IBomItem, ICount, IOption, IOptionFeature, IPrice, IRuntimeNode, IState

Description

Represents configurable BOM Models.

Member Summary

Methods

getType()

Inherited Member Summary

Fields inherited from interface IState

FALSE, LFALSE, LTRUE, TOGGLE, TRUE, UFALSE, UNKNOWN, UTRUE

Inherited Member Summary

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class BomNode

addAtpNotification(String), calculateAtpDate(), clearAtpDate(), clearAtpNotifications(), deselect(), getAtpDate(), getAtpNotifications(), getComponentCode(), getDefaultQuantity(), getDiscountedPrice(), getInventoryItemId(), getItemKey(), getListPrice(), getMaxQuantity(), getMaxSelected(), getMinQuantity(), getMinSelected(), getOrganizationId(), getPrimaryUomCode(), getState(), getUomCode(), hasDefaultQuantity(), hasMaxQuantity(), hasMaxSelected(), hasMinQuantity(), hasMinSelected(), isOptionMutexed(), isRequired(), isSelected(), isSelectionMutexed(), select(), select(IOption), setAtpDate(Date), setState(int)

Methods inherited from class OptionFeatureNode

deselect(IOption), getSelectedOption(), getSelectedOptions()

Methods inherited from class StateCountNode

addPricingNotification(String), clearDiscountedPrice(), clearPricingNotifications(), getCount(), getExtendedPrice(), getPricingNotifications(), setCount(int), setDiscountedPrice(double), setListPrice(double), toString()

Methods inherited from class StateNode

isDefaultState(int), isFalseState(int), isLogicState(int), isTrueState(int), isUnknownState(int), isUserState(int), statesMatch(int, int), unset()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IBomItem

getComponentCode(), getInventoryItemId(), getMaxQuantity(), getMinQuantity(), getOrganizationId(), getPrimaryUomCode(), hasMaxQuantity(), hasMinQuantity(), isRequired()

Methods inherited from interface IOptionFeature

deselect(IOption), getMaxSelected(), getMinSelected(), getSelectedOption(), getSelectedOptions(), hasMaxSelected(), hasMinSelected(), isSelectionMutexed(), select(IOption)

Methods inherited from interface IState

getState(), setState(int), unset()

Methods inherited from interface ICount

Inherited Member Summary

getCount(), setCount(int), unset()

Methods inherited from interface IOption

deselect(), isOptionMutexed(), isSelected(), select()

Methods inherited from interface IPrice

getDatabaseID(), getDiscountedPrice(), getExtendedPrice(), getItemKey(), getListPrice(), getPricingNotifications(), getUomCode()

Methods inherited from interface IAtp

getAtpDate(), getAtpNotifications(), getDatabaseID(), getItemKey(), getUomCode()

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getType()

```
public int getType()
```

Overrides:

getType() in class RuntimeNode

oracle.apps.cz.cio
BomNode

Syntax

```
public abstract class BomNode extends OptionFeatureNode implements IBomItem

java.lang.Object
  |
  +--RuntimeNode
        |
        +--StateNode
              |
              +--StateCountNode
                    |
                    +--OptionFeatureNode
                          |
                          +--oracle.apps.cz.cio.BomNode
```

Direct Known Subclasses:

BomModel, BomOptionClass, BomStdItem

All Implemented Interfaces:

IAtp, IBomItem, ICount, IOption, IOptionFeature, IPrice, IRuntimeNode, IState

Description

Represents configurable BOM option classes.

Member Summary

Methods

addAtpNotification(String)

calculateAtpDate()

clearAtpDate()

clearAtpNotifications()

deselect()

Member Summary

getAtpDate()

getAtpNotifications()

getComponentCode()

getDefaultQuantity()

getDiscountedPrice()

getInventoryItemId()

getItemKey()

getListPrice()

getMaxQuantity()

getMaxSelected()

getMinQuantity()

getMinSelected()

getOrganizationId()

getPrimaryUomCode()

getState()

getUomCode()

hasDefaultQuantity()

hasMaxQuantity()

hasMaxSelected()

hasMinQuantity()

hasMinSelected()

isOptionMutexed()

isRequired()

isSelected()

isSelectionMutexed()

select()

select(IOption)

setAtpDate(Date)

Member Summary

setState(int)

Inherited Member Summary

Fields inherited from interface IState

FALSE, LFALSE, LTRUE, TOGGLE, TRUE, UFALSE, UNKNOWN, UTRUE

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class OptionFeatureNode

deselect(IOption), getSelectedOption(), getSelectedOptions()

Methods inherited from class StateCountNode

addPricingNotification(String), clearDiscountedPrice(), clearPricingNotifications(), getCount(), getExtendedPrice(), getPricingNotifications(), setCount(int), setDiscountedPrice(double), setListPrice(double), toString()

Methods inherited from class StateNode

isDefaultState(int), isFalseState(int), isLogicState(int), isTrueState(int), isUnknownState(int), isUserState(int), statesMatch(int, int), unset()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IOptionFeature

deselect(IOption), getSelectedOption(), getSelectedOptions()

Methods inherited from interface IState

unset()

Methods inherited from interface ICount

getCount(), setCount(int), unset()

Methods inherited from interface IPrice

Inherited Member Summary

getDatabaseID(), getExtendedPrice(), getPricingNotifications()

Methods inherited from interface IAtp

getDatabaseID()

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

addAtpNotification(String)

```
public void addAtpNotification(java.lang.String message)
```

calculateAtpDate()

```
public void calculateAtpDate()
```

clearAtpDate()

```
public void clearAtpDate()
```

clearAtpNotifications()

```
public void clearAtpNotifications()
```

deselect()

```
public void deselect()
```

Specified By:

deselect() in interface IOption

getAtpDate()

```
public java.util.Date getAtpDate()
```

Specified By:

getAtpDate() in interface IAtp

getAtpNotifications()

```
public java.lang.String getAtpNotifications()
```

Specified By:
getAtpNotifications() in interface IAtp

getComponentCode()

```
public java.lang.String getComponentCode()
```

Specified By:
getComponentCode() in interface IBomItem

getDefaultQuantity()

```
public int getDefaultQuantity()
```

getDiscountedPrice()

```
public double getDiscountedPrice()
```

Specified By:
getDiscountedPrice() in interface IPrice

Specified By:
getDiscountedPrice() in interface IPrice

Overrides:
getDiscountedPrice() in class StateCountNode

getInventoryItemId()

```
public int getInventoryItemId()
```

Specified By:
getInventoryItemId() in interface IBomItem

getItemKey()

```
public java.lang.String getItemKey()
```

Specified By:
getItemKey() in interface IPrice

Specified By:
getItemKey() in interface IAtp

Specified By:
getItemKey() in interface IPrice

Overrides:
getItemKey() in class StateCountNode

getListPrice()

```
public double getListPrice()
```

Specified By:
getListPrice() in interface IPrice

Specified By:
getListPrice() in interface IPrice

Overrides:
getListPrice() in class StateCountNode

getMaxQuantity()

```
public int getMaxQuantity()
```

Specified By:
getMaxQuantity() in interface IBomItem

getMaxSelected()

```
public int getMaxSelected()
```

Specified By:
getMaxSelected() in interface IOptionFeature

Specified By:
getMaxSelected() in interface IOptionFeature

getMinQuantity()

```
public int getMinQuantity()
```

Specified By:
getMinQuantity() in interface IBomItem

getMinSelected()

```
public int getMinSelected()
```

Specified By:
getMinSelected() in interface IOptionFeature

Specified By:
getMinSelected() in interface IOptionFeature

getOrganizationId()

```
public int getOrganizationId()
```

Specified By:
getOrganizationId() in interface IBomItem

getPrimaryUomCode()

```
public java.lang.String getPrimaryUomCode()
```

Specified By:
getPrimaryUomCode() in interface IBomItem

getState()

```
public int getState()
```

Specified By:
getState() in interface IState

Specified By:
getState() in interface IState

Specified By:
getState() in interface IState

Specified By:
getState() in interface IState

Overrides:
getState() in class StateNode

getUomCode()

```
public java.lang.String getUomCode()
```

Specified By:
getUomCode() in interface IPrice

Specified By:
getUomCode() in interface IAtp

Specified By:
getUomCode() in interface IPrice

Overrides:
getUomCode() in class StateCountNode

hasDefaultQuantity()

```
public boolean hasDefaultQuantity()
```

hasMaxQuantity()

```
public boolean hasMaxQuantity()
```

Specified By:
hasMaxQuantity() in interface IBomItem

hasMaxSelected()

```
public boolean hasMaxSelected()
```

Specified By:
hasMaxSelected() in interface IOptionFeature

Specified By:
hasMaxSelected() in interface IOptionFeature

hasMinQuantity()

```
public boolean hasMinQuantity()
```

Specified By:
hasMinQuantity() in interface IBomItem

hasMinSelected()

```
public boolean hasMinSelected()
```

Specified By:
hasMinSelected() in interface IOptionFeature

Specified By:
hasMinSelected() in interface IOptionFeature

isOptionMutexed()

```
public boolean isOptionMutexed()
```

Specified By:
isOptionMutexed() in interface IOption

isRequired()

```
public boolean isRequired()
```

Specified By:
isRequired() in interface IBomItem

isSelected()

```
public boolean isSelected()
```

Specified By:
isSelected() in interface IOption

isSelectionMutexed()

```
public boolean isSelectionMutexed()
```

Specified By:
isSelectionMutexed() in interface IOptionFeature

Specified By:
isSelectionMutexed() in interface IOptionFeature

Overrides:
isSelectionMutexed() in class OptionFeatureNode

select()

```
public void select()
```

Specified By:
select() in interface IOption

select(IOption)

```
public void select(IOption option)
```

Specified By:
select(IOption) in interface IOptionFeature

Specified By:
select(IOption) in interface IOptionFeature

Overrides:
select(IOption) in class OptionFeatureNode

setAtpDate(Date)

```
public void setAtpDate(java.util.Date atpDate)
```

setState(int)

```
public void setState(int newState)
```

Specified By:
setState(int) in interface IState

Specified By:
setState(int) in interface IState

Specified By:
setState(int) in interface IState

Specified By:
setState(int) in interface IState

Overrides:
setState(int) in class StateNode

oracle.apps.cz.cio
BomOptionClass

Syntax

```
public class BomOptionClass extends BomNode

java.lang.Object
  |
  +--RuntimeNode
        |
        +--StateNode
              |
              +--StateCountNode
                    |
                    +--OptionFeatureNode
                          |
                          +--BomNode
                                |
                                +--oracle.apps.cz.cio.BomOptionClass
```

All Implemented Interfaces:

IAtp, IBomItem, ICount, IOption, IOptionFeature, IPrice, IRuntimeNode, IState

Description

Represents configurable BOM option classes.

Member Summary

Methods

getType()

Inherited Member Summary

Fields inherited from interface IState

FALSE, LFALSE, LTRUE, TOGGLE, TRUE, UFALSE, UNKNOWN, UTRUE

Inherited Member Summary

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class BomNode

addAtpNotification(String), calculateAtpDate(), clearAtpDate(), clearAtpNotifications(), deselect(), getAtpDate(), getAtpNotifications(), getComponentCode(), getDefaultQuantity(), getDiscountedPrice(), getInventoryItemId(), getItemKey(), getListPrice(), getMaxQuantity(), getMaxSelected(), getMinQuantity(), getMinSelected(), getOrganizationId(), getPrimaryUomCode(), getState(), getUomCode(), hasDefaultQuantity(), hasMaxQuantity(), hasMaxSelected(), hasMinQuantity(), hasMinSelected(), isOptionMutexed(), isRequired(), isSelected(), isSelectionMutexed(), select(), select(IOption), setAtpDate(Date), setState(int)

Methods inherited from class OptionFeatureNode

deselect(IOption), getSelectedOption(), getSelectedOptions()

Methods inherited from class StateCountNode

addPricingNotification(String), clearDiscountedPrice(), clearPricingNotifications(), getCount(), getExtendedPrice(), getPricingNotifications(), setCount(int), setDiscountedPrice(double), setListPrice(double), toString()

Methods inherited from class StateNode

isDefaultState(int), isFalseState(int), isLogicState(int), isTrueState(int), isUnknownState(int), isUserState(int), statesMatch(int, int), unset()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IBomItem

getComponentCode(), getInventoryItemId(), getMaxQuantity(), getMinQuantity(), getOrganizationId(), getPrimaryUomCode(), hasMaxQuantity(), hasMinQuantity(), isRequired()

Methods inherited from interface IOptionFeature

deselect(IOption), getMaxSelected(), getMinSelected(), getSelectedOption(), getSelectedOptions(), hasMaxSelected(), hasMinSelected(), isSelectionMutexed(), select(IOption)

Methods inherited from interface IState

getState(), setState(int), unset()

Methods inherited from interface ICount

Inherited Member Summary

getCount(), setCount(int), unset()

Methods inherited from interface IOption

deselect(), isOptionMutexed(), isSelected(), select()

Methods inherited from interface IPrice

getDatabaseID(), getDiscountedPrice(), getExtendedPrice(), getItemKey(), getListPrice(), getPricingNotifications(), getUomCode()

Methods inherited from interface IAtp

getAtpDate(), getAtpNotifications(), getDatabaseID(), getItemKey(), getUomCode()

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getType()

```
public int getType()
```

Overrides:

getType() in class RuntimeNode

oracle.apps.cz.cio
BomStdItem

Syntax

```
public class BomStdItem extends BomNode

java.lang.Object
  |
  +--RuntimeNode
        |
        +--StateNode
              |
              +--StateCountNode
                    |
                    +--OptionFeatureNode
                          |
                          +--BomNode
                                |
                                +--oracle.apps.cz.cio.BomStdItem
```

All Implemented Interfaces:

IAtp, IBomItem, ICount, IOption, IOptionFeature, IPrice, IRuntimeNode, IState

Description

Represents configurable BOM models.

Member Summary

Methods

getType()

Inherited Member Summary

Fields inherited from interface IState

FALSE, LFALSE, LTRUE, TOGGLE, TRUE, UFALSE, UNKNOWN, UTRUE

Inherited Member Summary

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class BomNode

addAtpNotification(String), calculateAtpDate(), clearAtpDate(), clearAtpNotifications(), deselect(), getAtpDate(), getAtpNotifications(), getComponentCode(), getDefaultQuantity(), getDiscountedPrice(), getInventoryItemId(), getItemKey(), getListPrice(), getMaxQuantity(), getMaxSelected(), getMinQuantity(), getMinSelected(), getOrganizationId(), getPrimaryUomCode(), getState(), getUomCode(), hasDefaultQuantity(), hasMaxQuantity(), hasMaxSelected(), hasMinQuantity(), hasMinSelected(), isOptionMutexed(), isRequired(), isSelected(), isSelectionMutexed(), select(), select(IOption), setAtpDate(Date), setState(int)

Methods inherited from class OptionFeatureNode

deselect(IOption), getSelectedOption(), getSelectedOptions()

Methods inherited from class StateCountNode

addPricingNotification(String), clearDiscountedPrice(), clearPricingNotifications(), getCount(), getExtendedPrice(), getPricingNotifications(), setCount(int), setDiscountedPrice(double), setListPrice(double), toString()

Methods inherited from class StateNode

isDefaultState(int), isFalseState(int), isLogicState(int), isTrueState(int), isUnknownState(int), isUserState(int), statesMatch(int, int), unset()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IBomItem

getComponentCode(), getInventoryItemId(), getMaxQuantity(), getMinQuantity(), getOrganizationId(), getPrimaryUomCode(), hasMaxQuantity(), hasMinQuantity(), isRequired()

Methods inherited from interface IOptionFeature

deselect(IOption), getMaxSelected(), getMinSelected(), getSelectedOption(), getSelectedOptions(), hasMaxSelected(), hasMinSelected(), isSelectionMutexed(), select(IOption)

Methods inherited from interface IState

getState(), setState(int), unset()

Methods inherited from interface ICount

Inherited Member Summary

getCount(), setCount(int), unset()

Methods inherited from interface IOption

deselect(), isOptionMutexed(), isSelected(), select()

Methods inherited from interface IPrice

getDatabaseID(), getDiscountedPrice(), getExtendedPrice(), getItemKey(), getListPrice(), getPricingNotifications(), getUomCode()

Methods inherited from interface IAtp

getAtpDate(), getAtpNotifications(), getDatabaseID(), getItemKey(), getUomCode()

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getType()

```
public int getType()
```

Overrides:

getType() in class RuntimeNode

oracle.apps.cz.cio
BooleanFeature

Syntax

```
public class BooleanFeature extends StateNode

java.lang.Object
  |
  +--RuntimeNode
        |
        +--StateNode
              |
              +--oracle.apps.cz.cio.BooleanFeature
```

All Implemented Interfaces:

IRuntimeNode, IState

Description

Represents a feature with a boolean value.

Member Summary

Methods

getType()

Inherited Member Summary

Fields inherited from interface IState

FALSE, LFALSE, LTRUE, TOGGLE, TRUE, UFALSE, UNKNOWN, UTRUE

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class StateNode

Inherited Member Summary

getState(), isDefaultState(int), isFalseState(int), isLogicState(int), isTrueState(int), isUnknownState(int), isUserState(int), setState(int), statesMatch(int, int), unset()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IState

getState(), setState(int), unset()

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getType()

```
public int getType()
```

Overrides:

getType() in class RuntimeNode

oracle.apps.cz.cio
CIO

Syntax

```
public class CIO extends java.lang.Object

java.lang.Object
  |
  +--oracle.apps.cz.cio.CIO
```

Description

Implements a configuration integration object that can be used to create, save, restore and delete configurations.

Member Summary

Constructors	
CIO()	Constructs a newly allocated configuration integration object.
Methods	
clearLogicFile(String)	Clears only the key specified file from the LCE file cache
clearLogicFileCache()	Clears all LogicFile objects from the logic file cache.
close()	Closes the CIO object and all associated runtime objects.
closeConfiguration(Configuration)	Deletes all runtime structure and memory associated with a configuration.
createConfiguration(int, Context)	Creates a new configuration based on a root model node ID representing a configurable product or component.
createConfiguration(int, int, Context)	
createConfiguration(int, int, Date, Context)	Creates a new BOM explosion configuration based on inventoryItemId, organizationId, and explosionDate representing a configurable product or component.
createConfiguration(String, Context)	Creates a new configuration based on a project name representing a configurable product or component.
getActiveModelPath()	Gets the current active model path.
restoreConfiguration(DbConfigHeader, Context)	Restores a configuration from the database.

Member Summary	
restoreConfiguration(int, int, Context)	Restores a configuration from the database.
setActiveModelPath(String)	Sets the path to the directory where the CIO will look for logic files, and where it will store logic files when generating them out of the database.

Inherited Member Summary
Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

CIO()

```
public  CIO()
```
Constructs a newly allocated configuration integration object.

Methods

clearLogicFile(String)

```
public void clearLogicFile(java.lang.String key)
```
Clears only the key specified file from the LCE file cache

clearLogicFileCache()

```
public void clearLogicFileCache()
```
Clears all LogicFile objects from the logic file cache.

close()

```
public void close()
```
Closes the CIO object and all associated runtime objects.

closeConfiguration(Configuration)

```
public void closeConfiguration(Configuration config)
```
Deletes all runtime structure and memory associated with a configuration.

Parameters:

`config` - the configuration to be deleted.

See Also:

Configuration

createConfiguration(int, Context)

```
public Configuration createConfiguration(int rootNodeID,
oracle.apps.fnd.common.Context ctx)
```

Creates a new configuration based on a root model node ID representing a configurable product or component.

Parameters:

`rootNodeID` - the ID of the DIO model node representing the product or configuration to be configured.

`ctx` - the Context object representing the application context

Returns:

a new configuration.

Throws:

LogicalException - if a logic failure is encountered when initializing the configuration.

MissingFileException - if a logic file cannot be found in the active model path

See Also:

Configuration

createConfiguration(int, int, Context)

```
public Configuration createConfiguration(int projectID, int rootNodeID,
oracle.apps.fnd.common.Context ctx)
```

Deprecated.

Creates a new configuration based on a project ID and root model node ID both representing a configurable product or component.

Parameters:

`projectID` - the ID of the DIO project representing the product or configuration to be configured.

`rootNodeID` - the ID of the DIO model node representing the product or configuration to be configured.

`ctx` - the Context object representing the application context

Returns:

a new configuration.

Throws:

LogicalException - if a logic failure is encountered when initializing the configuration.

MissingFileException - if a logic file cannot be found in the active model path

See Also:

Configuration

createConfiguration(int, int, Date, Context)

```
public Configuration createConfiguration(int inventoryItemId, int
organizationId, java.util.Date explosionDate, oracle.apps.fnd.common.Context
ctx)
```

Creates a new BOM explosion configuration based on inventoryItemId, organizationId, and explosionDate representing a configurable product or component.

Parameters:

`inventoryItemId` - the inventory item id of the BOM explosion model

`organizationId` - the organization id of the BOM explosion model

`explosionDate` - the effective date of the BOM explosion model

`ctx` - the Context object representing the application context

Returns:

a new configuration.

Throws:

NotOneProductException - if the specified project contains more than one or no products

LogicalException - if a logic failure is encountered when initializing the configuration.

See Also:

Configuration

createConfiguration(String, Context)

`public Configuration createConfiguration(java.lang.String projectName, oracle.apps.fnd.common.Context ctx)`
Creates a new configuration based on a project name representing a configurable product or component.

Parameters:

`projectName` - the name of the DIO project representing the product or configuration to be configured.

`ctx` - the Context object representing the application context

Returns:

a new configuration.

Throws:

NotOneProductException - if the specified project contains more than one or no products

LogicalException - if a logic failure is encountered when initializing the configuration.

MissingFileException - if a logic file cannot be found in the active model path

See Also:

Configuration

getActiveModelPath()

`public java.lang.String getActiveModelPath()`
Gets the current active model path.

Returns:

the active model path.

restoreConfiguration(DbConfigHeader, Context)

```
public Configuration
restoreConfiguration(oracle.apps.cz.dio.config.DbConfigHeader header,
oracle.apps.fnd.common.Context ctx)
```

Restores a configuration from the database.

Parameters:

`the` - header containing information identifying the configuration to be restored.

`ctx` - the Context object representing the application context

Returns:

the restored configuration.

Throws:

LogicalException - if a logic failure is encountered when initializing the configuration.

MissingFileException - if a logic file cannot be found in the active model path

See Also:

Configuration

restoreConfiguration(int, int, Context)

```
public Configuration restoreConfiguration(int configHeaderID, int revNumber,
oracle.apps.fnd.common.Context ctx)
```

Restores a configuration from the database.

Parameters:

`the` - ID of the header containing information identifying the configuration to be restored.

`the` - revision number of the header containing information identifying the configuration to be restored.

`ctx` - the Context object representing the application context

Returns:

the restored configuration.

Throws:

LogicalException - if a logic failure is encountered when initializing the configuration.

MissingFileException - if a logic file cannot be found in the active model path

See Also:

Configuration

setActiveModelPath(String)

```
public void setActiveModelPath(java.lang.String path)
```

Sets the path to the directory where the CIO will look for logic files, and where it will store logic files when generating them out of the database.

Parameters:

`path` - the path to the active model directory which should include the trailing path separator.

oracle.apps.cz.cio
CompanionNode

Syntax

```
public class CompanionNode extends java.lang.Object

java.lang.Object
  |
  +--oracle.apps.cz.cio.CompanionNode
```

Description

Provides methods for extracting values from a model based on property annotations.

Created by a parent CompanionRoot from a functional companion and used to get property-mapped feature information.

Member Summary

Methods	
getBoolean(String)	Returns the value of a boolean feature.
getBoolean(String, boolean)	Returns the value of a boolean feature, or the default if features is not present.
getChildren()	Returns all of the CompanionNode children of this CompanionNode.
getDouble(String)	Returns the value of a double feature.
getDouble(String, double)	Returns the value of a double feature, or the default if features is not present.
getFeature(String)	Get the runtime node representing a particular feature based on its property annotation.
getInteger(String)	Returns the value of an integer feature.
getInteger(String, int)	Returns the value of an integer feature, or the default if features is not present.
getString(String)	Returns the value of a string feature.
getString(String, String)	Returns the value of a string feature, or the default if the feature is not present.
hasFeature(String)	Returns true if this CompanionNode contains the named feature.

Inherited Member Summary

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods

getBoolean(String)

```
public boolean getBoolean(java.lang.String name)
```
Returns the value of a boolean feature.

getBoolean(String, boolean)

```
public boolean getBoolean(java.lang.String name, boolean dflt)
```
Returns the value of a boolean feature, or the default if features is not present.

getChildren()

```
public com.sun.java.util.collections.List getChildren()
```
Returns all of the CompanionNode children of this CompanionNode.

getDouble(String)

```
public double getDouble(java.lang.String name)
```
Returns the value of a double feature.

getDouble(String, double)

```
public double getDouble(java.lang.String name, double dflt)
```
Returns the value of a double feature, or the default if features is not present.

getFeature(String)

```
public IRuntimeNode getFeature(java.lang.String name)
```
Get the runtime node representing a particular feature based on its property annotation.

getInteger(String)

```
public int getInteger(java.lang.String name)
```
Returns the value of an integer feature.

getInteger(String, int)

```
public int getInteger(java.lang.String name, int dflt)
```
Returns the value of an integer feature, or the default if features is not present.

getString(String)

```
public java.lang.String getString(java.lang.String name)
```
Returns the value of a string feature.

getString(String, String)

```
public java.lang.String getString(java.lang.String name, java.lang.String dflt)
```
Returns the value of a string feature, or the default if the feature is not present.

hasFeature(String)

```
public boolean hasFeature(java.lang.String name)
```
Returns true if this CompanionNode contains the named feature.

oracle.apps.cz.cio
CompanionRoot

Syntax

```
public abstract class CompanionRoot extends java.lang.Object

java.lang.Object
  |
  +--oracle.apps.cz.cio.CompanionRoot
```

Description

Provides functional companion implementors with property-based feature extraction.

An abstract class to be extended by functional companion implementors which attaches CompanionNode objects to runtime nodes and makes model features available to the function companion through a flexible system of property-based annotations.

In order to use this class, the implementor should provide implementations of getNodeIdentifier() , which returns the name of the property used to identify runtime nodes to which `CompanionNode`s will be attached, getFeatureIdentifier() which returns the name of the property used to identify features of the `CompanionNode`, and getNodeClass(String) which maps the value of the `getNodeIdentifier` property to the subclass of `CompanionNode` that should be instantiated to represent a particular node.

Member Summary

Constructors	
CompanionRoot(IRuntimeNode)	Creates a tree of companion node objects based on property annotations.
Methods	
getFeatureIdentifier()	Returns the name of the property used to identify companion features.
getNodeClass(String)	Maps a node type to the class used to represent the node.

Member Summary	
getNodeIdentifier()	Returns the name of the property used to identify companion nodes.
getRootNodes()	Returns the root CompanionNode objects.

Inherited Member Summary
Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

CompanionRoot(IRuntimeNode)

```
public  CompanionRoot(IRuntimeNode root)
```
Creates a tree of companion node objects based on property annotations.

Methods

getFeatureIdentifier()

```
public abstract java.lang.String getFeatureIdentifier()
```
Returns the name of the property used to identify companion features.

getNodeClass(String)

```
public abstract java.lang.String getNodeClass(java.lang.String nodeType)
```
Maps a node type to the class used to represent the node.

getNodeIdentifier()

```
public abstract java.lang.String getNodeIdentifier()
```
Returns the name of the property used to identify companion nodes. Runtime nodes that have a property of this name will be mapped to `CompanionNode` objects. The value of the property will be mapped through the getNodeClass(String) method to determine which subclass of `CompanionNode` to instantiate.

getRootNodes()

```
public com.sun.java.util.collections.List getRootNodes()
```

Returns the root CompanionNode objects.

oracle.apps.cz.cio CompanionValidationFailure

Syntax

```
public class CompanionValidationFailure extends ValidationFailure

java.lang.Object
  |
  +--StatusInfo
        |
        +--ValidationFailure
              |
              +--oracle.apps.cz.cio.CompanionValidationFailure
```

Description

Failure produced by a functional companion object.

Member Summary

Constructors

CompanionValidationFailure(String, IRuntimeNode, IFunctionalCompanion)	

Methods

equals(Object)	
getCompanion()	Returns the companion in which this validation failure occurred.
hashCode()	

Inherited Member Summary

Fields inherited from class ValidationFailure

COMPANION_FAILURE, MAX_FAILURE, MIN_FAILURE, MIN0_FAILURE, MINMAX_FAILURE, RESOURCE_FAILURE, RESTORE_FAILURE

Fields inherited from class StatusInfo

Inherited Member Summary

STATUS_DELETED, STATUS_EXISTING, STATUS_NEW

Methods inherited from class ValidationFailure

getMessage(), getMessage(String), getType(), toString()

Methods inherited from class StatusInfo

getNode(), getStatus(), statusToString(int), toString(boolean)

Methods inherited from class java.lang.Object

getClass, notify, notifyAll, wait, wait, wait

Constructors

CompanionValidationFailure(String, IRuntimeNode, IFunctionalCompanion)

```
public  CompanionValidationFailure(java.lang.String message, IRuntimeNode node,
IFunctionalCompanion companion)
```

Methods

equals(Object)

```
public boolean equals(java.lang.Object obj)
```

Overrides:

equals(Object) in class ValidationFailure

getCompanion()

```
public IFunctionalCompanion getCompanion()
```

Returns the companion in which this validation failure occurred.

hashCode()

```
public int hashCode()
```

Overrides:

hashCode() in class StatusInfo

oracle.apps.cz.cio
Component

Syntax

```
public class Component extends ComponentNode

java.lang.Object
  |
  +--RuntimeNode
        |
        +--ComponentNode
              |
              +--oracle.apps.cz.cio.Component
```

All Implemented Interfaces:

IIntegerMinMax, IRuntimeNode

Description

Implements the runtime configuration behavior of products and components.

Member Summary

Methods

getChildren()	Returns a list of this node's children.
getCount()	Returns the count of the associated component set.
getFuncCompByID(int)	Returns a particular functional companion based on its ID, null if no match.
getFuncCompByName(String)	Returns a particular functional companion based on its name, null if no match.
getFunctionalCompanions()	Returns a list of all functional companions associated with this component.
getInstanceNumber()	Returns the instance number of this component (1 if not in a component set).
getMax()	Returns the maximum of the design-time component.
getMin()	Returns the minimum of the design-time component.
getName()	Returns the name of this runtime node.
getType()	Returns the type of this runtime node.

Member Summary	
hasMax()	Returns true if the design-time component has a maximum.
hasMin()	Returns true if the design-time component has a minimum.
instanceTypeToString(int)	
isRoot()	Returns true if this is the root component in the runtime tree.
isVirtual()	Returns true if this component is a virtual component.
setName(String)	Sets the name of this component.

Inherited Member Summary

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class ComponentNode

getChildrenByType(int), isActive()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getConfiguration(), getDatabaseID(), getDescription(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getChildren()

```
public com.sun.java.util.collections.List getChildren()
```

Returns a list of this node's children.

Overrides:

getChildren() in class RuntimeNode

getCount()

```
public int getCount()
```

Returns the count of the associated component set.

getFuncCompByID(int)

```
public IFunctionalCompanion getFuncCompByID(int id)
```

Returns a particular functional companion based on its ID, null if no match.

getFuncCompByName(String)

```
public IFunctionalCompanion getFuncCompByName(java.lang.String name)
```

Returns a particular functional companion based on its name, null if no match.

getFunctionalCompanions()

```
public com.sun.java.util.collections.List getFunctionalCompanions()
```

Returns a list of all functional companions associated with this component.

getInstanceNumber()

```
public int getInstanceNumber()
```

Returns the instance number of this component (1 if not in a component set).

getMax()

```
public int getMax()
```

Returns the maximum of the design-time component.

getMin()

```
public int getMin()
```

Returns the minimum of the design-time component.

getName()

```
public java.lang.String getName()
```

Returns the name of this runtime node.

Overrides:

getName() in class RuntimeNode

getType()

```
public int getType()
```
Returns the type of this runtime node.

Overrides:

getType() in class RuntimeNode

hasMax()

```
public boolean hasMax()
```
Returns true if the design-time component has a maximum.

hasMin()

```
public boolean hasMin()
```
Returns true if the design-time component has a minimum.

instanceTypeToString(int)

```
public static java.lang.String instanceTypeToString(int instanceType)
```

isRoot()

```
public boolean isRoot()
```
Returns true if this is the root component in the runtime tree.

isVirtual()

```
public boolean isVirtual()
```
Returns true if this component is a virtual component.

setName(String)

```
public void setName(java.lang.String newName)
```
Sets the name of this component. NOTE: The method setName() shouldn't be used and may be removed in a future release.

oracle.apps.cz.cio
ComponentNode

Syntax

```
public abstract class ComponentNode extends RuntimeNode implements
IIntegerMinMax

java.lang.Object
  |
  +--RuntimeNode
        |
        +--oracle.apps.cz.cio.ComponentNode
```

Direct Known Subclasses:

Component, ComponentSet

All Implemented Interfaces:

IIntegerMinMax, IRuntimeNode

Description

Represents a set of configurable components.

Member Summary

Methods	
getChildrenByType(int)	Returns a list of all children of a given type.
isActive()	Returns true if this node has been activated.

Inherited Member Summary

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class RuntimeNode

Inherited Member Summary

getChildByID(int), getChildByName(String), getChildren(), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IIntegerMinMax

getMax(), getMin(), hasMax(), hasMin()

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getChildrenByType(int)

```
public com.sun.java.util.collections.List getChildrenByType(int type)
```
Returns a list of all children of a given type.

Specified By:

getChildrenByType(int) in interface IRuntimeNode

Overrides:

getChildrenByType(int) in class RuntimeNode

isActive()

```
public boolean isActive()
```
Returns true if this node has been activated.

oracle.apps.cz.cio
ComponentSet

Syntax

```
public class ComponentSet extends ComponentNode

java.lang.Object
  |
  +--RuntimeNode
        |
        +--ComponentNode
              |
              +--oracle.apps.cz.cio.ComponentSet
```

All Implemented Interfaces:

IIntegerMinMax, IRuntimeNode

Description

Represents a set of configurable components.

Member Summary

Methods

add()	
addConfigEventListener(ICompSetEventListener)	Add a listener that is notified when a component is added or deleted.
delete(Component)	
getChildByInstanceNumber(int)	
getCount()	
getMax()	
getMin()	
getType()	
hasMax()	
hasMin()	

Member Summary

removeConfigEventListener(ICompSetEventListener)	Remove a listener that is notified when a component is added or deleted.

Inherited Member Summary

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class ComponentNode

getChildrenByType(int), isActive()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

add()

```
public Component add()
```

addConfigEventListener(ICompSetEventListener)

```
public void addConfigEventListener(ICompSetEventListener listener)
```

Add a listener that is notified when a component is added or deleted.

delete(Component)

```
public void delete(Component component)
```

getChildByInstanceNumber(int)

```
public Component getChildByInstanceNumber(int instNum)
```

getCount()

```
public int getCount()
```

getMax()

```
public int getMax()
```

getMin()

```
public int getMin()
```

getType()

```
public int getType()
```

Overrides:

getType() in class RuntimeNode

hasMax()

```
public boolean hasMax()
```

hasMin()

```
public boolean hasMin()
```

removeConfigEventListener(ICompSetEventListener)

```
public void removeConfigEventListener(ICompSetEventListener listener)
```

Remove a listener that is notified when a component is added or deleted.

oracle.apps.cz.cio
ConfigTransaction

Syntax

```
public class ConfigTransaction extends oracle.apps.cz.cio.BasicConfigAction

java.lang.Object
  |
  +--oracle.apps.cz.cio.BasicConfigAction
        |
        +--oracle.apps.cz.cio.ConfigTransaction
```

Description

Represents a configuration transaction.

Inherited Member Summary

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

oracle.apps.cz.cio
Configuration

Syntax

```
public class Configuration
```

`oracle.apps.cz.cio.Configuration`

Description

The top-level entry point to a configuration.

Member Summary

Methods

addConfigEventListener(IConfigEventListener)	Add a listener that is notified when a component is added or deleted.
addConfigMessage(String, String)	Adds a configuration message to be saved to the cz_config_messages table.
beginConfigTransaction()	Creates a new transaction.
beginConfigTransaction(boolean)	Creates a new transaction and specifies the auto commit mode.
calculateAtpDates()	Calculates ATP dates for all IAtp nodes in the tree.
calculateListPrices()	Calculates list prices of all IPrice nodes within configuration.
canPerform()	Returns true if there is at least one undone or not commited transaction that can be performed.
canUndo()	Returns true if there are performed transactions that can be undone.
clearConfigMessages()	Removes all configuration messages added by addConfigMessage.
close()	Close the Configuration object and all associated runtime objects.
commitConfigTransaction(ConfigTransaction)	Commits the given transaction if it matches with current one in the configuration.
endConfigTransaction(ConfigTransaction)	Ends the given transaction if it matches with current one in the configuration.
finalizeWorkaround()	
getAltPricingAtpContext()	Returns context that was added to the configuration through setAltPricingAtpContext, or null if no alternate pricing/ATP context exists.

Member Summary

getCIO()	Gets the CIO that created this configuration.
getConfigHeaderCheckoutUser()	Gets the user who has the config header checked out.
getConfigHeaderDateCreated()	Gets the date when the config header was creaed.
getConfigHeaderDescription()	Gets the description of the config header.
getConfigHeaderEffectiveFrom()	Gets the date from which the config header is effective.
getConfigHeaderEffectiveTo()	Gets the date towhich the config header is effective.
getConfigHeaderId()	Gets the id of the config header.
getConfigHeaderLastUpdateDate()	Gets the date when the config header was last updated.
getConfigHeaderName()	Gets the name of the config header.
getConfigHeaderNote()	Gets the note of the config header.
getConfigHeaderNumberQuotesUsedIn()	Gets the config header number quotes used in.
getConfigHeaderOpportunityHeaderId()	Gets the opportunity header id of the config header .
getConfigHeaderRevision()	Gets the revision of the config header.
getConfigHeaderStatus()	Gets the status of the config header.
getConfigHeaderUiDefinitionId()	Gets the UI definition id in the config header.
getConfigHeaderUserIdCreated()	Gets the id of the user who created the config header.
getContext()	Returns the Context object associated with this configuration
getInitParameters()	Gets initialization parameters, which are stored in a NameValuePairSet object.
getLastContradiction()	Returns the most recent contradiction.
getProjectID()	Gets the database ID of the project from which this configuration was created.
getRootBomModel()	Returns the root BOM Model node, if there is associated with this configuration.
getRootBomModel(int, int)	Returns the root BOM Model node with the given inventory item ID and organization ID
getRootComponent()	Gets the root product or component of the configuration.
getRootComponentDbId()	Gets the database id of the root component.
getRuntimeNode(int)	Fetches a runtime node based on a runtime ID.

Member Summary

getSelectedItems()	Gets a collection of status info objects describing all selected items in the configuration.
getTotalDiscountedPrice()	Returns rolled up discounted price of the configuration.
getTransactionDepth()	Returns the number of nested transactions (depth).
getUnsatisfiedItems()	Gets a collection of status info objects describing all unsatisfied items in the configuration.
getValidationFailures()	Gets a collection of validation failures describing current problems with the configuration.
isUnsatisfied()	Returns `true` if the configuration is incomplete.
perform()	Perform the next pending transaction.
removeConfigEventListener(IConfigEventListener)	Remove a listener that is notified when a component is added or deleted.
restartConfiguration(boolean)	Restart the current configuration instance without destroying the objects NOTE: Currently, it works only for values and states.
rollbackConfigTransaction(ConfigTransaction)	Rolls back the given transaction if it matches with current one in the configuration.
save()	Saves the restored configuration.
saveAs(int, int)	Saves over the current configuration.
saveNew()	Saves a whole new configuration.
saveNewRev()	Saves a new revision of the restored configuration.
setAltPricingAtpContext(Context)	If pricing and ATP information should be retrieved from an alternate Apps database, the setAltPricingAtpContext method should be called immediately after the Configuration is created.
setConfigHeaderCheckoutUser(String)	Sets the user who has checked out the config header.
setConfigHeaderDateCreated(Timestamp)	Sets the config header creation date.
setConfigHeaderDescription(String)	Sets the config header description.
setConfigHeaderEffectiveFrom(Timestamp)	Sets the date from which the config header is effective.
setConfigHeaderEffectiveTo(Timestamp)	Sets the date to which the config header is effective.
setConfigHeaderName(String)	Sets the config header name.

Member Summary	
setConfigHeaderNote(String)	Sets the config header note.
setConfigHeaderOpportunityHeaderId(int)	Sets the config header opportunity header id.
setConfigHeaderUiDefinitionId(int)	Sets the config header UI definition id.
setInitParameters(NameValuePairSet)	Sets configuration initialization parameters, e.g.
undo()	Undo the previous transaction.

Methods

addConfigEventListener(IConfigEventListener)

`public void addConfigEventListener(IConfigEventListener listener)`
Add a listener that is notified when a component is added or deleted.

addConfigMessage(String, String)

`public void addConfigMessage(java.lang.String keyword, java.lang.String message)`
Adds a configuration message to be saved to the cz_config_messages table. Messages are cleared from the Configuration object when the configuration and messages are saved.

Parameters:

`keyword` - keyword describing the type of message, e.g. "CONTRADICTION"

`message` - message string

beginConfigTransaction()

`public ConfigTransaction beginConfigTransaction()`
Creates a new transaction.

Returns:

a reference to the newly created transaction.

beginConfigTransaction(boolean)

`public ConfigTransaction beginConfigTransaction(boolean autoCommit)`
Creates a new transaction and specifies the auto commit mode.

Returns:

a reference to the newly created transaction.

calculateAtpDates()

```
public java.util.Date calculateAtpDates()
```

Calculates ATP dates for all IAtp nodes in the tree. ATP values can then be retrieved using IAtp.getAtpDate().

Returns:

configuration level ATP date if calculated, null if not

Throws:

AtpUnavailableException - thrown if configuration initialization parameters required to run ATP check have not all been provided

calculateListPrices()

```
public void calculateListPrices()
```

Calculates list prices of all IPrice nodes within configuration. Prices are retrieved through IPrice.getListPrice.

canPerform()

```
public boolean canPerform()
```

Returns true if there is at least one undone or not commited transaction that can be performed.

canUndo()

```
public boolean canUndo()
```

Returns true if there are performed transactions that can be undone.

clearConfigMessages()

```
public void clearConfigMessages()
```

Removes all configuration messages added by addConfigMessage.

close()

```
public void close()
```

Close the Configuration object and all associated runtime objects.

commitConfigTransaction(ConfigTransaction)

`public void commitConfigTransaction(ConfigTransaction transaction)`
Commits the given transaction if it matches with current one in the configuration.

Parameters:

a - transaction reference.

endConfigTransaction(ConfigTransaction)

`public void endConfigTransaction(ConfigTransaction transaction)`
Ends the given transaction if it matches with current one in the configuration.

Parameters:

a - transaction reference.

finalizeWorkaround()

`public void finalizeWorkaround()`

getAltPricingAtpContext()

`public oracle.apps.fnd.common.Context getAltPricingAtpContext()`
Returns context that was added to the configuration through setAltPricingAtpContext, or null if no alternate pricing/ATP context exists.

getCIO()

`public CIO getCIO()`
Gets the CIO that created this configuration.

Returns:

the CIO that created this configuration.

See Also:

CIO

getConfigHeaderCheckoutUser()

`public java.lang.String getConfigHeaderCheckoutUser()`
Gets the user who has the config header checked out.

Returns:

the config header checkout user.

getConfigHeaderDateCreated()

```
public java.sql.Timestamp getConfigHeaderDateCreated()
```

Gets the date when the config header was creaed.

Returns:

the config header creation date.

getConfigHeaderDescription()

```
public java.lang.String getConfigHeaderDescription()
```

Gets the description of the config header.

Returns:

the config header description.

getConfigHeaderEffectiveFrom()

```
public java.sql.Timestamp getConfigHeaderEffectiveFrom()
```

Gets the date from which the config header is effective.

Returns:

the config header 'Effective From' date.

getConfigHeaderEffectiveTo()

```
public java.sql.Timestamp getConfigHeaderEffectiveTo()
```

Gets the date towhich the config header is effective.

Returns:

the config header 'Effective To' date.

getConfigHeaderId()

```
public int getConfigHeaderId()
```

Gets the id of the config header.

Returns:

the config header id.

getConfigHeaderLastUpdateDate()

`public java.sql.Timestamp getConfigHeaderLastUpdateDate()`
Gets the date when the config header was last updated.

Returns:

the config header last update date.

getConfigHeaderName()

`public java.lang.String getConfigHeaderName()`
Gets the name of the config header.

Returns:

the config header name.

getConfigHeaderNote()

`public java.lang.String getConfigHeaderNote()`
Gets the note of the config header.

Returns:

the config header note.

getConfigHeaderNumberQuotesUsedIn()

`public int getConfigHeaderNumberQuotesUsedIn()`
Gets the config header number quotes used in.

Returns:

the config header number quotes used in.

getConfigHeaderOpportunityHeaderId()

`public int getConfigHeaderOpportunityHeaderId()`
Gets the opportunity header id of the config header .

Returns:

the config header opportunity header id.

getConfigHeaderRevision()

`public int getConfigHeaderRevision()`
Gets the revision of the config header.

Returns:

the config header revision.

getConfigHeaderStatus()

`public java.lang.String getConfigHeaderStatus()`
Gets the status of the config header.

Returns:

the config header status.

getConfigHeaderUiDefinitionId()

`public int getConfigHeaderUiDefinitionId()`
Gets the UI definition id in the config header.

Returns:

the config header UI definition id.

getConfigHeaderUserIdCreated()

`public int getConfigHeaderUserIdCreated()`
Gets the id of the user who created the config header.

Returns:

the config header user id created.

getContext()

`public oracle.apps.fnd.common.Context getContext()`
Returns the Context object associated with this configuration

getInitParameters()

`public NameValuePairSet getInitParameters()`
Gets initialization parameters, which are stored in a NameValuePairSet object.

Returns:

initParameters object

getLastContradiction()

`public LogicalException getLastContradiction()`
Returns the most recent contradiction.

getProjectID()

`public int getProjectID()`
Gets the database ID of the project from which this configuration was created.

Returns:

the project ID.

getRootBomModel()

`public BomModel getRootBomModel()`
Returns the root BOM Model node, if there is associated with this configuration.

getRootBomModel(int, int)

`public BomModel getRootBomModel(int inventoryItemId, int organizationId)`
Returns the root BOM Model node with the given inventory item ID and organization ID

getRootComponent()

`public Component getRootComponent()`
Gets the root product or component of the configuration.

Returns:

the root product or component.

See Also:

Component

getRootComponentDbId()

```
public int getRootComponentDbId()
```
Gets the database id of the root component.

Returns:

the root component db id.

getRuntimeNode(int)

```
public IRuntimeNode getRuntimeNode(int runtimeID)
```
Fetches a runtime node based on a runtime ID.

Parameters:

`runtimeID` - the runtime ID of the desired node.

Returns:

the corresponding runtime node.

See Also:

IRuntimeNode

getSelectedItems()

```
public com.sun.java.util.collections.Collection getSelectedItems()
```
Gets a collection of status info objects describing all selected items in the configuration.

Returns:

the collection of status info objects.

getTotalDiscountedPrice()

```
public double getTotalDiscountedPrice()
```
Returns rolled up discounted price of the configuration. Discounted prices on selected items are available after this call through IPrice.getDiscountedPrice.

getTransactionDepth()

```
public int getTransactionDepth()
```
Returns the number of nested transactions (depth).

Returns:

the transaction depth.

getUnsatisfiedItems()

`public com.sun.java.util.collections.Collection getUnsatisfiedItems()`
Gets a collection of status info objects describing all unsatisfied items in the configuration.

Returns:

the collection of status info objects.

getValidationFailures()

`public com.sun.java.util.collections.Collection getValidationFailures()`
Gets a collection of validation failures describing current problems with the configuration.

Returns:

the collection of validation failures.

isUnsatisfied()

`public boolean isUnsatisfied()`
Returns `true` if the configuration is incomplete.

Returns:

a boolean indicating whether the configuration is unsatisfied.

perform()

`public void perform()`
Perform the next pending transaction.

removeConfigEventListener(IConfigEventListener)

`public void removeConfigEventListener(IConfigEventListener listener)`
Remove a listener that is notified when a component is added or deleted.

restartConfiguration(boolean)

`public void restartConfiguration(boolean checkValidations)`

Restart the current configuration instance without destroying the objects NOTE: Currently, it works only for values and states. Additions and deletions are not being restarted. Therefore, after restart() you get the latest (before the restart) component instances and cannot undo an instance addition or deletion.

rollbackConfigTransaction(ConfigTransaction)

```
public void rollbackConfigTransaction(ConfigTransaction transaction)
```
Rolls back the given transaction if it matches with current one in the configuration.

Parameters:

a - transaction reference.

save()

```
public void save()
```
Saves the restored configuration.

Throws:

NoConfigHeaderException - when this configuration hasn't been previously saved. Consider calling SaveNew().

`ConfigOverwriteNotAllowedException` - when this configuration is "read only". Consider calling SaveNewRev() or SaveNew().

saveAs(int, int)

```
public void saveAs(int configHeaderID, int revNumber)
```
Saves over the current configuration. It uses the passed ID and revision number to open a ConfigHeader object and to save the configuration int it

Throws:

`ConfigOverwriteNotAllowedException` - when this configuration is "read only". Consider calling SaveNewRev() or SaveNew().

saveNew()

```
public void saveNew()
```
Saves a whole new configuration.

saveNewRev()

```
public void saveNewRev()
```

Saves a new revision of the restored configuration.

Throws:

NoConfigHeaderException - when this configuration hasn't been previously saved. Consider calling SaveNew().

setAltPricingAtpContext(Context)

`public void setAltPricingAtpContext(oracle.apps.fnd.common.Context ctx)`
If pricing and ATP information should be retrieved from an alternate Apps database, the setAltPricingAtpContext method should be called immediately after the Configuration is created.

Parameters:

`ctx` - Context which represents session on database from which pricing and ATP information is retrieved

setConfigHeaderCheckoutUser(String)

`public void setConfigHeaderCheckoutUser(java.lang.String user)`
Sets the user who has checked out the config header.

setConfigHeaderDateCreated(Timestamp)

`public void setConfigHeaderDateCreated(java.sql.Timestamp dateCreated)`
Sets the config header creation date.

setConfigHeaderDescription(String)

`public void setConfigHeaderDescription(java.lang.String description)`
Sets the config header description.

setConfigHeaderEffectiveFrom(Timestamp)

`public void setConfigHeaderEffectiveFrom(java.sql.Timestamp effFrom)`
Sets the date from which the config header is effective.

setConfigHeaderEffectiveTo(Timestamp)

`public void setConfigHeaderEffectiveTo(java.sql.Timestamp effTo)`
Sets the date to which the config header is effective.

setConfigHeaderName(String)

```
public void setConfigHeaderName(java.lang.String name)
```
Sets the config header name.

setConfigHeaderNote(String)

```
public void setConfigHeaderNote(java.lang.String note)
```
Sets the config header note.

setConfigHeaderOpportunityHeaderId(int)

```
public void setConfigHeaderOpportunityHeaderId(int id)
```
Sets the config header opportunity header id.

setConfigHeaderUiDefinitionId(int)

```
public void setConfigHeaderUiDefinitionId(int id)
```
Sets the config header UI definition id.

setInitParameters(NameValuePairSet)

```
public void setInitParameters(NameValuePairSet initParameters)
```
Sets configuration initialization parameters, e.g. order header information. All parameter values should be provided as String objects.

To use the callback pricing mechanism, the following parameters must be provided: "pricing_package_name" (required), "price_mult_items_proc" or "price_single_item_proc" (one is required), "configurator_session_key" (required)

To use Apps 10.7/11.0 pricing for BomNodes ("AMNT" pricing method only), the following parameters must be provided: "price_list_id" (required), "pricing_attribute1" (optional), "pricing_attribute2" (optional), "pricing_attribute3" (optional), "pricing_attribute4" (optional), "pricing_attribute5" (optional), "pricing_attribute6" (optional), "pricing_attribute7" (optional), "pricing_attribute8" (optional), "pricing_attribute9" (optional), "pricing_attribute10" (optional), "pricing_attribute11" (optional), "pricing_attribute12" (optional), "pricing_attribute13" (optional), "pricing_attribute14" (optional), "pricing_attribute15" (optional), "ship_to_site_use_id" (optional), "customer_id" (optional), "invoice_to_site_use_id" (optional), "po_number" (optional), "agreement_id" (optional), "agreement_type_code" (optional), "order_type_id" (optional), "gsa" (optional).

NOTE: If the callback parameters and price_list_id are both provided, then the pricing callback will be run to determine prices.

To use the callback ATP mechanism, the following parameters must be provided: "atp_package_name" (required), "get_atp_dates_proc" (required), "configurator_session_key" (required), "warehouse_id" (required), "requested_date" (optional), and either "ship_to_org_id" (required) or "customer_id" and "customer_site_id" (required),

To use Apps 10.7/11.0 ATP calculation methods, the following parameters must be provided: "user_id" (required), "application_id" (required), "responsibility_id" (required), "atp_timeout" (required)

undo()

```
public void undo()
```

Undo the previous transaction.

oracle.apps.cz.cio
CountFeature

Syntax

```
public class CountFeature extends StateCountNode implements IInteger,
IIntegerMinMax

java.lang.Object
  |
  +--RuntimeNode
        |
        +--StateNode
              |
              +--StateCountNode
                    |
                    +--oracle.apps.cz.cio.CountFeature
```

All Implemented Interfaces:

ICount, IInteger, IIntegerMinMax, IPrice, IRuntimeNode, IState

Description

Represents a countable integer feature. A count feature is similar to an integer feature except that its minimum value must be greater than or equal to zero.

Member Summary

Methods

getIntValue()

getMax()

getMin()

getType()

hasMax()

hasMin()

setIntValue(int)

Inherited Member Summary

Fields inherited from interface IState

FALSE, LFALSE, LTRUE, TOGGLE, TRUE, UFALSE, UNKNOWN, UTRUE

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class StateCountNode

addPricingNotification(String), clearDiscountedPrice(), clearPricingNotifications(), getCount(), getDiscountedPrice(), getExtendedPrice(), getItemKey(), getListPrice(), getPricingNotifications(), getUomCode(), setCount(int), setDiscountedPrice(double), setListPrice(double), toString()

Methods inherited from class StateNode

getState(), isDefaultState(int), isFalseState(int), isLogicState(int), isTrueState(int), isUnknownState(int), isUserState(int), setState(int), statesMatch(int, int), unset()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IInteger

unset()

Methods inherited from interface ICount

getCount(), setCount(int)

Methods inherited from interface IPrice

getDatabaseID(), getDiscountedPrice(), getExtendedPrice(), getItemKey(), getListPrice(), getPricingNotifications(), getUomCode()

Methods inherited from interface IState

getState(), setState(int)

Methods inherited from interface IRuntimeNode

Inherited Member Summary

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getIntValue()

```
public int getIntValue()
```

Specified By:
getIntValue() in interface IInteger

getMax()

```
public int getMax()
```

Specified By:
getMax() in interface IIntegerMinMax

getMin()

```
public int getMin()
```

Specified By:
getMin() in interface IIntegerMinMax

getType()

```
public int getType()
```

Specified By:
getType() in interface IRuntimeNode

Overrides:
getType() in class RuntimeNode

hasMax()

```
public boolean hasMax()
```

Specified By:
hasMax() in interface IIntegerMinMax

hasMin()

```
public boolean hasMin()
```

Specified By:
hasMin() in interface IIntegerMinMax

setIntValue(int)

```
public void setIntValue(int newValue)
```

Specified By:
setIntValue(int) in interface IInteger

oracle.apps.cz.cio
DecimalFeature

Syntax

```
public class DecimalFeature extends DecimalNode implements IDecimalMinMax

java.lang.Object
  |
  +--RuntimeNode
        |
        +--ReadOnlyDecimalNode
              |
              +--DecimalNode
                    |
                    +--oracle.apps.cz.cio.DecimalFeature
```

All Implemented Interfaces:

IDecimal, IDecimalMinMax, IReadOnlyDecimal, IRuntimeNode

Description

Represents a feature with a decimal value.

Member Summary

Methods

getMax()

getMin()

getType()

hasMax()

hasMin()

Inherited Member Summary

Fields inherited from interface IRuntimeNode

Inherited Member Summary

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class DecimalNode

setDecimalValue(double), toString(), unset()

Methods inherited from class ReadOnlyDecimalNode

getDecimalValue()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IDecimal

setDecimalValue(double), unset()

Methods inherited from interface IReadOnlyDecimal

getDecimalValue()

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getMax()

```
public double getMax()
```

Specified By:

getMax() in interface IDecimalMinMax

getMin()

```
public double getMin()
```

Specified By:
getMin() in interface IDecimalMinMax

getType()

```
public int getType()
```

Specified By:
getType() in interface IRuntimeNode

Overrides:
getType() in class RuntimeNode

hasMax()

```
public boolean hasMax()
```

Specified By:
hasMax() in interface IDecimalMinMax

hasMin()

```
public boolean hasMin()
```

Specified By:
hasMin() in interface IDecimalMinMax

oracle.apps.cz.cio
DecimalNode

Syntax

```
public abstract class DecimalNode extends ReadOnlyDecimalNode implements
IDecimal

java.lang.Object
  |
  +--RuntimeNode
        |
        +--ReadOnlyDecimalNode
              |
              +--oracle.apps.cz.cio.DecimalNode
```

Direct Known Subclasses:

DecimalFeature, Resource, Total

All Implemented Interfaces:

IDecimal, IReadOnlyDecimal, IRuntimeNode

Description

An abstract class implementing behavior common to objects with a decimal value.

Member Summary

Methods

setDecimalValue(double)

toString()

unset()

Inherited Member Summary

Fields inherited from interface IRuntimeNode

Inherited Member Summary

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class ReadOnlyDecimalNode

getDecimalValue()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IReadOnlyDecimal

getDecimalValue()

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

setDecimalValue(double)

```
public void setDecimalValue(double newValue)
```

Specified By:

setDecimalValue(double) in interface IDecimal

toString()

```
public java.lang.String toString()
```

Overrides:

toString() in class ReadOnlyDecimalNode

unset()

```
public void unset()
```

Specified By:

unset() in interface IDecimal

oracle.apps.cz.cio
Factory

Syntax

```
public class Factory extends java.lang.Object

java.lang.Object
  |
  +--oracle.apps.cz.cio.Factory
```

Description

Provides a class factory for the CIO to clients that do not support arguments to constructors.

Member Summary

Constructors

Factory()	

Methods

createCIO()	Creates an instance of the CIO.
createContext(String, String, String, String)	Creates the database context object which is required by most CIO methods.
createContext(String, String, String, String, String, String, String)	Creates the database context object which is required by most CIO methods.
loadDriver(String)	Loads the JDBC driver named by the argument.

Inherited Member Summary

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

Factory()

```
public  Factory()
```

Methods

createCIO()

```
public CIO createCIO()
```
Creates an instance of the CIO.

createContext(String, String, String, String)

```
public oracle.apps.fnd.common.Context createContext(java.lang.String url,
java.lang.String uname, java.lang.String pwd, java.lang.String owner)
```
Creates the database context object which is required by most CIO methods. This method requires a database user and password.

Parameters:

`url` - database connection URL that specifies JDBC driver and datasource

`uname` - database username

`pwd` - database password

`owner` - SellingPoint schema owner

createContext(String, String, String, String, String, String, String)

```
public oracle.apps.fnd.common.Context createContext(java.lang.String url,
java.lang.String appsUsername, java.lang.String appsPassword, java.lang.String
gatewayUsername, java.lang.String gatewayPassword, java.lang.String fndNam,
java.lang.String applServerId)
```
Creates the database context object which is required by most CIO methods. This method uses Apps FND authentication to validate the user.

Parameters:

`url` - database connection URL, identifies JDBC driver and data source

`appsUsername` - Oracle Applications user name

`appsPassword` - Oracle Applications password for appsUsername

`gatewayUsername` - gateway user name

`gatewayPassword` - password for gatewayUsername

`fndNam` - Apps schema owner

`applServerId` - application server ID, only used if its security feature is ON (OFF by default)

loadDriver(String)

```
public void loadDriver(java.lang.String driver)
```
Loads the JDBC driver named by the argument.

Throws:

`ClassNotFoundException` - if the driver cannot be loaded.

oracle.apps.cz.cio
FuncCompCreationException

Syntax

```
public class FuncCompCreationException extends java.lang.RuntimeException

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--oracle.apps.cz.cio.FuncCompCreationException
```

All Implemented Interfaces:

`java.io.Serializable`

Description

Signalled if a functional companion cannot be created.

Member Summary

Methods

getDescrption()	Returns the description of the failed companion.
getID()	Returns the database ID of the failed companion.
getName()	Returns the name of the failed companion.
getProgString()	Returns the program string used when trying to create the companion.

Inherited Member Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Inherited Member Summary

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods

getDescrption()

```
public java.lang.String getDescrption()
```

Returns the description of the failed companion.

getID()

```
public int getID()
```

Returns the database ID of the failed companion.

getName()

```
public java.lang.String getName()
```

Returns the name of the failed companion.

getProgString()

```
public java.lang.String getProgString()
```

Returns the program string used when trying to create the companion.

oracle.apps.cz.cio
FunctionalCompanion

Syntax

```
public class FunctionalCompanion extends java.lang.Object implements
IFunctionalCompanion

java.lang.Object
  |
  +--oracle.apps.cz.cio.FunctionalCompanion
```

All Implemented Interfaces:

IFunctionalCompanion

Description

Base object on which user functional companions can be based.

Member Summary

Constructors

FunctionalCompanion()	

Methods

autoConfigure()	Does nothing.
generateOutput()	Does nothing.
generateOutput(HttpServletResponse)	Does nothing.
getDescription()	Returns the description of the functional companion.
getID()	Returns the database ID of the functional companion.
getName()	Returns the name of the functional companion.
getRuntimeNode()	Returns the runtime node to which this functional is associated.
initialize(IRuntimeNode, String, String, int)	Saves the parameters in member variables.
terminate()	Does nothing.

Member Summary	
toString()	
validate()	Does nothing.

Inherited Member Summary
Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructors

FunctionalCompanion()

```
public  FunctionalCompanion()
```

Methods

autoConfigure()

```
public void autoConfigure()
```
Does nothing.

Specified By:

autoConfigure() in interface IFunctionalCompanion

generateOutput()

```
public java.lang.String generateOutput()
```
Does nothing. Returns null.

Specified By:

generateOutput() in interface IFunctionalCompanion

generateOutput(HttpServletResponse)

```
public void generateOutput(javax.servlet.http.HttpServletResponse response)
```
Does nothing.

Specified By:

generateOutput(HttpServletResponse) in interface IFunctionalCompanion

getDescription()

```
public java.lang.String getDescription()
```

Returns the description of the functional companion.

Specified By:

getDescription() in interface IFunctionalCompanion

getID()

```
public int getID()
```

Returns the database ID of the functional companion.

Specified By:

getID() in interface IFunctionalCompanion

getName()

```
public java.lang.String getName()
```

Returns the name of the functional companion.

Specified By:

getName() in interface IFunctionalCompanion

getRuntimeNode()

```
public IRuntimeNode getRuntimeNode()
```

Returns the runtime node to which this functional is associated.

Specified By:

getRuntimeNode() in interface IFunctionalCompanion

initialize(IRuntimeNode, String, String, int)

```
public void initialize(IRuntimeNode node, java.lang.String name,
java.lang.String description, int id)
```

Saves the parameters in member variables.

Specified By:

initialize(IRuntimeNode, String, String, int) in interface IFunctionalCompanion

terminate()

```
public void terminate()
```

Does nothing.

Specified By:

terminate() in interface IFunctionalCompanion

toString()

```
public java.lang.String toString()
```

Overrides:

java.lang.Object.toString() in class java.lang.Object

validate()

```
public com.sun.java.util.collections.List validate()
```

Does nothing.

Specified By:

validate() in interface IFunctionalCompanion

oracle.apps.cz.cio
FunctionalCompanionException

Syntax

```
public class FunctionalCompanionException
```

oracle.apps.cz.cio.FunctionalCompanionException

Description

This exception is used to indicate that an error occured somewhere inside the functional companion.

Member Summary

Constructors

FunctionalCompanionException(Throwable)	The message of the original exception will be the message of this exception.
FunctionalCompanionException(Throwable, String)	

Constructors

FunctionalCompanionException(Throwable)

```
public  FunctionalCompanionException(java.lang.Throwable ex)
```

The message of the original exception will be the message of this exception.

FunctionalCompanionException(Throwable, String)

```
public  FunctionalCompanionException(java.lang.Throwable ex, java.lang.String
message)
```

Parameters:

`message` - the message of the exception

oracle.apps.cz.cio
IAtp

Syntax

```
public interface IAtp extends ICount
```

All Known Subinterfaces:

IBomItem

All Superinterfaces:

ICount

Description

Implemented by objects that can have ATP calculated. This interface contains methods for getting available-to-promise (ATP) information, and methods to retrieve ATP errors/warnings/messages.

Member Summary

Methods	
getAtpDate()	Retrieves last ATP date calculated by Configuration.calculateAtpDates for this item.
getAtpNotifications()	Returns string containing any ATP messages, warnings or errors generated for this node by the latest Configuration.calculateAtpDates call.
getDatabaseID()	Returns the database ID of the runtime node.
getItemKey()	Returns item key for items imported from Oracle Inventory / BOM.
getUomCode()	Returns unit of measure code for items imported from Oracle Inventory/ BOM.

Inherited Member Summary

Methods inherited from interface ICount

getCount(), setCount(int), unset()

Methods

getAtpDate()

`public java.util.Date getAtpDate()`
Retrieves last ATP date calculated by Configuration.calculateAtpDates for this item.

Returns:

ATP date

Throws:

AtpUnavailableException - thrown if ATP initialization parameters were not provided

NoAtpCalculatedException - thrown if ATP was never demanded or if the ATP procedure did not calculate an ATP date for this node

getAtpNotifications()

`public java.lang.String getAtpNotifications()`
Returns string containing any ATP messages, warnings or errors generated for this node by the latest Configuration.calculateAtpDates call.

getDatabaseID()

`public int getDatabaseID()`
Returns the database ID of the runtime node.

getItemKey()

`public java.lang.String getItemKey()`
Returns item key for items imported from Oracle Inventory / BOM. Item key is constructed from BOM_EXPLOSIONS field values: "[COMPONENT_CODE]:[EXPLOSION_TYPE]:[ORGANIZATION_ID]:[TOP_ITEM_ID]" Item key may be used by PL/SQL ATP procedures to calculate ATP for nodes. Returns null if node was not imported from Oracle Inventory / BOM.

getUomCode()

`public java.lang.String getUomCode()`
Returns unit of measure code for items imported from Oracle Inventory/ BOM. The unit of measure may be used by PL/SQL pricing procedures to calculate ATP for nodes. Returns null if node was not imported from Oracle Inventory / BOM.

oracle.apps.cz.cio
IBomItem

Syntax

```
public interface IBomItem extends IOptionFeature, IOption, IPrice, IAtp
```

All Superinterfaces:

IAtp, ICount, IOption, IOptionFeature, IPrice, IState

All Known Implementing Classes:

BomNode

Description

Implemented by all selectable BOM items.

Member Summary

Methods	
getComponentCode()	Returns component code of item.
getInventoryItemId()	Returns Oracle Applications inventory_item_id of item.
getMaxQuantity()	Gets the maximum quantity.
getMinQuantity()	Gets the minimum quantity.
getOrganizationId()	Returns Oracle Applications organization_id of item.
getPrimaryUomCode()	Gets primary unit of measure code for item.
hasMaxQuantity()	Reuturns true if the BOM item has maximum quantity
hasMinQuantity()	Returns true if the BOM item has minimum quantity
isRequired()	Returns `true` if this is a required BOM item.

Inherited Member Summary

Fields inherited from interface IState

Inherited Member Summary

FALSE, LFALSE, LTRUE, TOGGLE, TRUE, UFALSE, UNKNOWN, UTRUE

Methods inherited from interface IOptionFeature

deselect(IOption), getMaxSelected(), getMinSelected(), getSelectedOption(), getSelectedOptions(), hasMaxSelected(), hasMinSelected(), isSelectionMutexed(), select(IOption)

Methods inherited from interface IState

getState(), setState(int), unset()

Methods inherited from interface ICount

getCount(), setCount(int), unset()

Methods inherited from interface IOption

deselect(), isOptionMutexed(), isSelected(), select()

Methods inherited from interface IPrice

getDatabaseID(), getDiscountedPrice(), getExtendedPrice(), getItemKey(), getListPrice(), getPricingNotifications(), getUomCode()

Methods inherited from interface IAtp

getAtpDate(), getAtpNotifications(), getDatabaseID(), getItemKey(), getUomCode()

Methods

getComponentCode()

```
public java.lang.String getComponentCode()
```

Returns component code of item. Component code is used to identify the item within an exploded bill of materials.

getInventoryItemId()

```
public int getInventoryItemId()
```

Returns Oracle Applications inventory_item_id of item.

getMaxQuantity()

```
public int getMaxQuantity()
```

Gets the maximum quantity.

getMinQuantity()

```
public int getMinQuantity()
```

Gets the minimum quantity.

getOrganizationId()

```
public int getOrganizationId()
```
Returns Oracle Applications organization_id of item.

getPrimaryUomCode()

```
public java.lang.String getPrimaryUomCode()
```
Gets primary unit of measure code for item.

hasMaxQuantity()

```
public boolean hasMaxQuantity()
```
Reuturns true if the BOM item has maximum quantity

hasMinQuantity()

```
public boolean hasMinQuantity()
```
Returns true if the BOM item has minimum quantity

isRequired()

```
public boolean isRequired()
```
Returns `true` if this is a required BOM item.

oracle.apps.cz.cio
ICompSetEventListener

Syntax

```
public interface ICompSetEventListener extends java.util.EventListener
```

All Superinterfaces:

```
java.util.EventListener
```

Description

Implemented by objects that want to find out about added components.

Member Summary

Methods

notifyComponentAdded(Component)	Called when a component is added to the component set.
notifyComponentDeleted(Component)	Called when a component is deleted from the component set.

Methods

notifyComponentAdded(Component)

```
public void notifyComponentAdded(Component component)
```

Called when a component is added to the component set.

notifyComponentDeleted(Component)

```
public void notifyComponentDeleted(Component component)
```

Called when a component is deleted from the component set.

oracle.apps.cz.cio
IConfigEventListener

Syntax

```
public interface IConfigEventListener extends java.util.EventListener
```

All Superinterfaces:

```
java.util.EventListener
```

Description

Implemented by objects that want to find out about added components. This listener's methods are called as the result of user interaction, after a functional companion is initialized.

Member Summary

Methods

notifyComponentAdded(Component)	Called when a component is added to the configuration as the result of user interaction, after a functional companion is initialized.
notifyComponentDeleted(Component)	Called when a component is deleted from the configuration as the result of user interaction, after a functional companion is initialized.

Methods

notifyComponentAdded(Component)

```
public void notifyComponentAdded(Component component)
```

Called when a component is added to the configuration as the result of user interaction, after a functional companion is initialized.

notifyComponentDeleted(Component)

```
public void notifyComponentDeleted(Component component)
```

Called when a component is deleted from the configuration as the result of user interaction, after a functional companion is initialized.

oracle.apps.cz.cio
ICount

Syntax
```
public interface ICount
```

All Known Subinterfaces:
IAtp, IBomItem, IOption, IOptionFeature, IPrice

All Known Implementing Classes:
StateCountNode

Description
Implemented by objects that have an associated integer count.

Member Summary	
Methods	
getCount()	Gets the current count of this object.
setCount(int)	Sets the count of this object.
unset()	Retracts any user selection made toward this node

Methods

getCount()

```
public int getCount()
```
Gets the current count of this object.

Returns:
the current count of this object.

setCount(int)

```
public void setCount(int newCount)
```

Sets the count of this object.

unset()

```
public void unset()
```

Retracts any user selection made toward this node

oracle.apps.cz.cio
IDecimal

Syntax

```
public interface IDecimal extends IReadOnlyDecimal
```

All Superinterfaces:

IReadOnlyDecimal

All Known Implementing Classes:

DecimalNode

Description

Implemented by objects that can both get and set a decimal value.

Member Summary	
Methods	
setDecimalValue(double)	Sets the current value of this object.
unset()	Retracts any user selection made toward this node

Inherited Member Summary
Methods inherited from interface IReadOnlyDecimal
getDecimalValue()

Methods

setDecimalValue(double)

```
public void setDecimalValue(double newValue)
```

Sets the current value of this object.

unset()

```
public void unset()
```

Retracts any user selection made toward this node

oracle.apps.cz.cio
IDecimalMinMax

Syntax

```
public interface IDecimalMinMax
```

All Known Implementing Classes:

DecimalFeature

Description

Implemented by objects that have a decimal minimum and maximum value.

Member Summary	
Methods	
getMax()	Get the maximum allowable value.
getMin()	Get the minimum allowable value.
hasMax()	Returns true if there is a maximum limit.
hasMin()	Returns true if there is a minimum limit.

Methods

getMax()

```
public double getMax()
```
Get the maximum allowable value.

getMin()

```
public double getMin()
```
Get the minimum allowable value.

hasMax()

```
public boolean hasMax()
```
Returns true if there is a maximum limit.

hasMin()

```
public boolean hasMin()
```

Returns true if there is a minimum limit.

oracle.apps.cz.cio
IFunctionalCompanion

Syntax

```
public interface IFunctionalCompanion
```

All Known Implementing Classes:

FunctionalCompanion

Description

Implemented by functional companion objects attached to components in order to provide programatic functionality to a configuration model.

Member Summary

Methods

autoConfigure()	Performs a programmatic configuration step.
generateOutput()	Generates output for this component.
generateOutput(HttpServletResponse)	Generates output for this component.
getDescription()	Returns the description of the functional companion.
getID()	Returns the database ID of the functional companion.
getName()	Returns the name of the functional companion.
getRuntimeNode()	Returns the runtime node to which this component is attached.
initialize(IRuntimeNode, String, String, int)	Saves information about the model and performs any actions needed to initialize the companion.
terminate()	Performs any cleanup on this companion that needs to occur before the companion is destroyed.
validate()	Programatically checks that a configuration is valid and returns a list of ValidationFailure objects if there are failures, and null otherwise.

Methods

autoConfigure()

```
public void autoConfigure()
```
Performs a programmatic configuration step. Any modifications to the model should be performed here.

generateOutput()

```
public java.lang.String generateOutput()
```
Generates output for this component. This version is called in a thick client context where the user's machine can be addressed directly. Can modify the model, but this is not recommended practice.

generateOutput(HttpServletResponse)

```
public void generateOutput(javax.servlet.http.HttpServletResponse response)
```
Generates output for this component. This version is called in a thin client context where the user's browser is addressed indirectly by writing to the HttpServletResponse object. Can modify the model, but this is not recommended practice.

getDescription()

```
public java.lang.String getDescription()
```
Returns the description of the functional companion.

getID()

```
public int getID()
```
Returns the database ID of the functional companion.

getName()

```
public java.lang.String getName()
```
Returns the name of the functional companion.

getRuntimeNode()

```
public IRuntimeNode getRuntimeNode()
```
Returns the runtime node to which this component is attached.

initialize(IRuntimeNode, String, String, int)

```
public void initialize(IRuntimeNode node, java.lang.String name,
java.lang.String description, int id)
```

Saves information about the model and performs any actions needed to initialize the companion. Should never attempt to modify the model.

terminate()

```
public void terminate()
```

Performs any cleanup on this companion that needs to occur before the companion is destroyed.

validate()

```
public com.sun.java.util.collections.List validate()
```

Programatically checks that a configuration is valid and returns a list of ValidationFailure objects if there are failures, and null otherwise. Should never attempt to modify the model.

oracle.apps.cz.cio
IInteger

Syntax
```
public interface IInteger
```

All Known Implementing Classes:
CountFeature, IntegerNode

Description
Implemented by objects that have an integer value.

Member Summary	
Methods	
getIntValue()	Get the current integer value of this object.
setIntValue(int)	Set the current integer value of this object.
unset()	Retracts any user selection made toward this node

Methods

getIntValue()

```
public int getIntValue()
```
Get the current integer value of this object.

setIntValue(int)

```
public void setIntValue(int newValue)
```
Set the current integer value of this object.

unset()

```
public void unset()
```
Retracts any user selection made toward this node

oracle.apps.cz.cio
IIntegerMinMax

Syntax
```
public interface IIntegerMinMax
```

All Known Implementing Classes:
CountFeature, IntegerFeature, ComponentNode

Description
Implemented by objects that have an integer minimum and maximum.

Member Summary	
Methods	
getMax()	Get the maximal allowable value for this object.
getMin()	Get the minimal allowable value for this object.
hasMax()	Returns true if there is a maximum limit.
hasMin()	Returns true if there is a minimum limit.

Methods

getMax()
```
public int getMax()
```
Get the maximal allowable value for this object.

getMin()
```
public int getMin()
```
Get the minimal allowable value for this object.

hasMax()
```
public boolean hasMax()
```
Returns true if there is a maximum limit.

hasMin()

```
public boolean hasMin()
```

Returns true if there is a minimum limit.

oracle.apps.cz.cio
IncompatibleInputException

Syntax

```
public class IncompatibleInputException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.apps.cz.cio.IncompatibleInputException
```

All Implemented Interfaces:

java.io.Serializable

Description

Signalled if a particular input is of different type than the node it is trying to restore over.

Member Summary	
Methods	
getInput()	Returns the input object where the mismatch occured
getModelNode()	Returns the corresponding model node where the mismatch occured

Inherited Member Summary
Methods inherited from class java.lang.Throwable
fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString
Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods

getInput()

`public oracle.apps.cz.dio.config.DbConfigInput getInput()`
Returns the input object where the mismatch occured

Returns:
the failed DbConfigInput object

getModelNode()

`public oracle.apps.cz.dio.model.DbModelNode getModelNode()`
Returns the corresponding model node where the mismatch occured

Returns:
the failed DbConfigInput object

oracle.apps.cz.cio
IntegerFeature

Syntax

```
public class IntegerFeature extends IntegerNode implements IIntegerMinMax

java.lang.Object
  |
  +--RuntimeNode
        |
        +--IntegerNode
              |
              +--oracle.apps.cz.cio.IntegerFeature
```

All Implemented Interfaces:

IInteger, IIntegerMinMax, IRuntimeNode

Description

Represents a feature with an integer value.

Member Summary

Methods

getMax()

getMin()

getType()

hasMax()

hasMin()

Inherited Member Summary

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Inherited Member Summary

Methods inherited from class IntegerNode

getIntValue(), setIntValue(int), unset()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IInteger

getIntValue(), setIntValue(int), unset()

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getMax()

```
public int getMax()
```

Specified By:
getMax() in interface IIntegerMinMax

getMin()

```
public int getMin()
```

Specified By:
getMin() in interface IIntegerMinMax

getType()

```
public int getType()
```

Specified By:
getType() in interface IRuntimeNode

Overrides:
getType() in class RuntimeNode

hasMax()

```
public boolean hasMax()
```

Specified By:
hasMax() in interface IIntegerMinMax

hasMin()

```
public boolean hasMin()
```

Specified By:
hasMin() in interface IIntegerMinMax

oracle.apps.cz.cio
IntegerNode

Syntax

```
public abstract class IntegerNode extends RuntimeNode implements IInteger

java.lang.Object
  |
  +--RuntimeNode
        |
        +--oracle.apps.cz.cio.IntegerNode
```

Direct Known Subclasses:

IntegerFeature

All Implemented Interfaces:

IInteger, IRuntimeNode

Description

Represents a feature with an integer value.

Member Summary

Methods

getIntValue()

setIntValue(int)

unset()

Inherited Member Summary

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class RuntimeNode

Inherited Member Summary

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getIntValue()

```
public int getIntValue()
```

Specified By:

getIntValue() in interface IInteger

setIntValue(int)

```
public void setIntValue(int newIntValue)
```

Specified By:

setIntValue(int) in interface IInteger

unset()

```
public void unset()
```

Specified By:

unset() in interface IInteger

oracle.apps.cz.cio
IOption

Syntax

```
public interface IOption extends IState, ICount
```

All Known Subinterfaces:

IBomItem

All Superinterfaces:

ICount, IState

All Known Implementing Classes:

OptionNode

Description

Implemented by objects that act as options. The defining characteristic of an option is that it can be selected and deselected.

Member Summary

Methods	
deselect()	Deslect this option.
isOptionMutexed()	Returns true if this option is a child of a mutexed parent
isSelected()	Returns `true` if this option is selected, and `false` otherwise.
select()	Select this option.

Inherited Member Summary

Fields inherited from interface IState

FALSE, LFALSE, LTRUE, TOGGLE, TRUE, UFALSE, UNKNOWN, UTRUE

Methods inherited from interface IState

Inherited Member Summary

getState(), setState(int), unset()

Methods inherited from interface ICount

getCount(), setCount(int), unset()

Methods

deselect()

```
public void deselect()
```
Deslect this option.

isOptionMutexed()

```
public boolean isOptionMutexed()
```
Returns true if this option is a child of a mutexed parent

isSelected()

```
public boolean isSelected()
```
Returns `true` if this option is selected, and `false` otherwise.

select()

```
public void select()
```
Select this option.

oracle.apps.cz.cio
IOptionFeature

Syntax

```
public interface IOptionFeature extends IState, ICount
```

All Known Subinterfaces:

IBomItem

All Superinterfaces:

ICount, IState

All Known Implementing Classes:

OptionFeatureNode

Description

Implemented by objects that contain selectable options. This interface provides a mechansim for selecting and deselecting options, and for determining which options are currently selected.

Member Summary

Methods	
deselect(IOption)	Deselect a particular option.
getMaxSelected()	Returns the maximum number of selected options.
getMinSelected()	Returns the minimum number of selected options.
getSelectedOption()	Returns the currently selected option, or `null` if no option is selected.
getSelectedOptions()	Returns a, possibly empty, collection of options that are currently selected.
hasMaxSelected()	Returns true if the Feature specifies a maximum number of selected options.
hasMinSelected()	Returns true if the Feature specifies a minimum number of selected options.
isSelectionMutexed()	Returns `true` if this feature supports mutexed selections.
select(IOption)	Select a particular option.

Inherited Member Summary

Fields inherited from interface IState

FALSE, LFALSE, LTRUE, TOGGLE, TRUE, UFALSE, UNKNOWN, UTRUE

Methods inherited from interface IState

getState(), setState(int), unset()

Methods inherited from interface ICount

getCount(), setCount(int), unset()

Methods

deselect(IOption)

```
public void deselect(IOption option)
```
Deselect a particular option.

Parameters:

`option` - the option to be de selected.

getMaxSelected()

```
public int getMaxSelected()
```
Returns the maximum number of selected options.

getMinSelected()

```
public int getMinSelected()
```
Returns the minimum number of selected options.

getSelectedOption()

```
public IOption getSelectedOption()
```
Returns the currently selected option, or `null` if no option is selected.

Returns:

the currently selected option.

Throws:

SelectionNotMutexedException - if this feature does not support mutexed selections.

getSelectedOptions()

```
public com.sun.java.util.collections.List getSelectedOptions()
```

Returns a, possibly empty, collection of options that are currently selected.

hasMaxSelected()

```
public boolean hasMaxSelected()
```

Returns true if the Feature specifies a maximum number of selected options.

hasMinSelected()

```
public boolean hasMinSelected()
```

Returns true if the Feature specifies a minimum number of selected options.

isSelectionMutexed()

```
public boolean isSelectionMutexed()
```

Returns `true` if this feature supports mutexed selections. When a selection is mutexed, it means that only one of a particular option is selectable at any one time, and selecting one option automatically deselects any other option that is mutexed and currently selected.

select(IOption)

```
public void select(IOption option)
```

Select a particular option.

Parameters:

`option` - the option to be selected.

oracle.apps.cz.cio
IPrice

Syntax

```
public interface IPrice extends ICount
```

All Known Subinterfaces:

IBomItem

All Superinterfaces:

ICount

All Known Implementing Classes:

StateCountNode

Description

Implemented by objects that can be priced. This interface contains methods for getting list, discount, and extended prices, and methods to retrieve pricing errors/warnings/messages.

Member Summary

Methods	
getDatabaseID()	Returns the database ID of the runtime node.
getDiscountedPrice()	Gets discounted price of item based on adjustments associated with price list specified in initParameters.
getExtendedPrice()	Calculates extended price of item (quantity * discounted price).
getItemKey()	Returns item key for items imported from Oracle Inventory / BOM.
getListPrice()	Gets list price of item on price list specified in initParameters.
getPricingNotifications()	Returns string containing any pricing messages, warnings, or errors.
getUomCode()	Returns unit of measure code for items imported from Oracle Inventory/ BOM.

Inherited Member Summary

Methods inherited from interface ICount

getCount(), setCount(int), unset()

Methods

getDatabaseID()

```
public int getDatabaseID()
```
Returns the database ID of the runtime node.

getDiscountedPrice()

```
public double getDiscountedPrice()
```
Gets discounted price of item based on adjustments associated with price list specified in initParameters.

getExtendedPrice()

```
public double getExtendedPrice()
```
Calculates extended price of item (quantity * discounted price).

getItemKey()

```
public java.lang.String getItemKey()
```
Returns item key for items imported from Oracle Inventory / BOM. Item key is constructed from BOM_EXPLOSIONS field values: "[COMPONENT_CODE]:[EXPLOSION_TYPE]:[ORGANIZATION_ID]:[TOP_ITEM_ID]" Item key may be used by PL/SQL pricing procedures to price nodes. Returns null if node was not imported from Oracle Inventory / BOM.

getListPrice()

```
public double getListPrice()
```
Gets list price of item on price list specified in initParameters.

getPricingNotifications()

```
public java.lang.String getPricingNotifications()
```
Returns string containing any pricing messages, warnings, or errors.

getUomCode()

```
public java.lang.String getUomCode()
```

Returns unit of measure code for items imported from Oracle Inventory/ BOM. The unit of measure may be used by PL/SQL pricing procedures to price nodes. Returns null if node was not imported from Oracle Inventory / BOM.

oracle.apps.cz.cio
IReadOnlyDecimal

Syntax
```
public interface IReadOnlyDecimal
```

All Known Subinterfaces:
IDecimal

All Known Implementing Classes:
ReadOnlyDecimalNode

Description
Implemented by objects that have a decimal value.

Member Summary

Methods	
getDecimalValue()	Gets the current value of this object.

Methods

getDecimalValue()

```
public double getDecimalValue()
```
Gets the current value of this object.

Returns:
the current value.

oracle.apps.cz.cio
IRuntimeNode

Syntax

```
public interface IRuntimeNode
```

All Known Implementing Classes:

RuntimeNode

Description

Implemented by all objects in the runtime configuration tree. This interface implements behavior common to all nodes in the runtime configuration tree, including components, features, options, totals, etc.

Member Summary

Fields

ALL_FEATURES	A pseudo-type that represents all feature types for use in getChildrenByType.
BOM_MODEL	BOM model type.
BOM_OPTION_CLASS	BOM option class type.
BOM_STD_ITEM	BOM standard item type.
BOOLEAN_FEATURE	Boolean feature type.
COMPONENT	Component type.
COMPONENT_SET	Component set type.
COUNT_FEATURE	Count feature type.
DECIMAL_FEATURE	Decimal feature type.
INTEGER_FEATURE	Integer feature type.
OPTION	Option type.
OPTION_FEATURE	Option feature type.
RESOURCE	Resource type.
TEXT_FEATURE	Text feature type.

Member Summary	
TOTAL	Total type.
Methods	
getChildByID(int)	Gets a particular child identified by its ID.
getChildByName(String)	Gets a particular child identified by its name.
getChildren()	Gets the children of this runtime configuration node.
getChildrenByType(int)	Gets all of the children of a particular type.
getConfiguration()	Gets the configuration to which this node belongs.
getDatabaseID()	Gets the database ID of the node.
getDescription()	Returns the design-time description of the runtime node.
getName()	Gets the name of the node.
getParent()	Get the parent of this node.
getProperties()	Returns a collection of the properties associated with this node.
getPropertyByName(String)	Returns a particular property of this node, based on its name.
getRuntimeID()	Gets the runtime ID of the node.
getSelectionLineID()	Returns selection line ID (configuration output database ID) for node.
getType()	Gets the type of this node.
hasCount()	Returns true if the node has an object count.
hasDecimalValue()	Returns true if the node has a decimal value.
hasDescription()	Returns true if there is a design-time description of the runtime node.
hasSelectionLineID()	Returns true if node has a selection line ID (configuration output ID), false if not.
hasState()	Returns true if the node has a logical state.
hasTextValue()	Returns true if the node has a text value.
isNative()	Returns true if this is a native BOM node
isUnsatisfied()	Returns `true` if this particular node, or any one of its children, has not been completely configured.
isUnsatisfiedNode()	Returns `true` if this particular node has not been completely configured.
toString(boolean)	Returns a String representation of this node, based on whether the client demands a description (if there is one) or just a name

Fields

ALL_FEATURES

```
public static final int ALL_FEATURES
```

A pseudo-type that represents all feature types for use in getChildrenByType.

BOM_MODEL

```
public static final int BOM_MODEL
```

BOM model type.

BOM_OPTION_CLASS

```
public static final int BOM_OPTION_CLASS
```

BOM option class type.

BOM_STD_ITEM

```
public static final int BOM_STD_ITEM
```

BOM standard item type.

BOOLEAN_FEATURE

```
public static final int BOOLEAN_FEATURE
```

Boolean feature type.

COMPONENT

```
public static final int COMPONENT
```

Component type.

COMPONENT_SET

```
public static final int COMPONENT_SET
```

Component set type.

COUNT_FEATURE

```
public static final int COUNT_FEATURE
```

Count feature type.

DECIMAL_FEATURE

```
public static final int DECIMAL_FEATURE
```

Decimal feature type.

INTEGER_FEATURE

```
public static final int INTEGER_FEATURE
```
Integer feature type.

OPTION

```
public static final int OPTION
```
Option type.

OPTION_FEATURE

```
public static final int OPTION_FEATURE
```
Option feature type.

RESOURCE

```
public static final int RESOURCE
```
Resource type.

TEXT_FEATURE

```
public static final int TEXT_FEATURE
```
Text feature type.

TOTAL

```
public static final int TOTAL
```
Total type.

Methods

getChildByID(int)

```
public IRuntimeNode getChildByID(int id)
```
Gets a particular child identified by its ID.

Returns:

a child of this node.

getChildByName(String)

`public IRuntimeNode getChildByName(java.lang.String name)`
Gets a particular child identified by its name.

Returns:

a child of this node.

getChildren()

`public com.sun.java.util.collections.List getChildren()`
Gets the children of this runtime configuration node.

Returns:

a collection of children.

getChildrenByType(int)

`public com.sun.java.util.collections.List getChildrenByType(int type)`
Gets all of the children of a particular type.

Returns:

a collection of children.

getConfiguration()

`public Configuration getConfiguration()`
Gets the configuration to which this node belongs.

Returns:

the configuration to which this node belongs.

See Also:

Configuration

getDatabaseID()

`public int getDatabaseID()`
Gets the database ID of the node.

Returns:

the database ID of the node.

getDescription()

```
public java.lang.String getDescription()
```

Returns the design-time description of the runtime node.

getName()

```
public java.lang.String getName()
```

Gets the name of the node.

Returns:

the name of the node.

getParent()

```
public IRuntimeNode getParent()
```

Get the parent of this node.

Returns:

the node's parent.

getProperties()

```
public com.sun.java.util.collections.Collection getProperties()
```

Returns a collection of the properties associated with this node. The collection contains items of the type Property.

getPropertyByName(String)

```
public Property getPropertyByName(java.lang.String name)
```

Returns a particular property of this node, based on its name. Returns null if a property of the given name does not exist.

getRuntimeID()

```
public int getRuntimeID()
```

Gets the runtime ID of the node. This ID is unique across all other nodes created by a particular CIO.

Returns:

runtime ID of the node.

getSelectionLineID()

```
public int getSelectionLineID()
```

Returns selection line ID (configuration output database ID) for node. The hasSelectionLineID() method should always be called before this method. A RuntimeException will be thrown if ID doesn't exist.

Returns:

line ID

getType()

```
public int getType()
```

Gets the type of this node.

Returns:

the type of this node.

hasCount()

```
public boolean hasCount()
```

Returns true if the node has an object count.

hasDecimalValue()

```
public boolean hasDecimalValue()
```

Returns true if the node has a decimal value.

hasDescription()

```
public boolean hasDescription()
```

Returns true if there is a design-time description of the runtime node.

hasSelectionLineID()

```
public boolean hasSelectionLineID()
```

Returns true if node has a selection line ID (configuration output ID), false if not.

hasState()

```
public boolean hasState()
```
Returns true if the node has a logical state.

hasTextValue()

```
public boolean hasTextValue()
```
Returns true if the node has a text value.

isNative()

```
public boolean isNative()
```
Returns true if this is a native BOM node

isUnsatisfied()

```
public boolean isUnsatisfied()
```
Returns `true` if this particular node, or any one of its children, has not been completely configured. The value is cached and is only updated on transaction commit or rollback.

Returns:

a boolean indicating whether the node is unsatisfied.

isUnsatisfiedNode()

```
public boolean isUnsatisfiedNode()
```
Returns `true` if this particular node has not been completely configured. The value is cached and is only updated on transaction commit or rollback.

Returns:

a boolean indicating whether the node is unsatisfied.

toString(boolean)

```
public java.lang.String toString(boolean description)
```
Returns a String representation of this node, based on whether the client demands a description (if there is one) or just a name

oracle.apps.cz.cio
IState

Syntax
```
public interface IState
```

All Known Subinterfaces:
IBomItem, IOption, IOptionFeature

All Known Implementing Classes:
StateNode

Description
Implemented by objects that have logic state. This interface contains a set of input states, used to specify a new state for an object, a set of output states, returned when querying an object for its state, and a set of methods for getting and setting the object's state.

Member Summary

Fields

FALSE	The input state used to set an object to false.
LFALSE	The logically false output state, indicating that the state is false as a consequence of a rule.
LTRUE	The logically true output state, indicating that the state is true as a consequence of a rule.
TOGGLE	The input state used to turn an object state to true if it is false or unknown, and to make it unknown or false if it is true.
TRUE	The input state used to set an object to true.
UFALSE	The user false output state, indicating that a user has set this object to false.
UNKNOWN	The unknown output state.
UTRUE	The user true output state, indicating that a user has set this object to true.

Methods

Member Summary	
getState()	Gets the current logic state of this object.
setState(int)	Change the current logic state of this object.
unset()	Retracts any user selection made toward this node

Fields

FALSE

```
public static final int FALSE
```
The input state used to set an object to false.

LFALSE

```
public static final int LFALSE
```
The logically false output state, indicating that the state is false as a consequence of a rule.

LTRUE

```
public static final int LTRUE
```
The logically true output state, indicating that the state is true as a consequence of a rule.

TOGGLE

```
public static final int TOGGLE
```
The input state used to turn an object state to true if it is false or unknown, and to make it unknown or false if it is true.

TRUE

```
public static final int TRUE
```
The input state used to set an object to true.

UFALSE

```
public static final int UFALSE
```
The user false output state, indicating that a user has set this object to false.

UNKNOWN

```
public static final int UNKNOWN
```
The unknown output state.

UTRUE

```
public static final int UTRUE
```
The user true output state, indicating that a user has set this object to true.

Methods

getState()

```
public int getState()
```
Gets the current logic state of this object.

Returns:

the current state.

setState(int)

```
public void setState(int state)
```
Change the current logic state of this object.

unset()

```
public void unset()
```
Retracts any user selection made toward this node

oracle.apps.cz.cio
IText

Syntax
```
public interface IText
```

All Known Implementing Classes:
TextNode

Description
Implemented by objects that have a textual value.

Member Summary	
Methods	
getTextValue()	Gets the current textual value of this object.
setTextValue(String)	Sets the current textual value of this object.
unset()	Retracts any user selection made toward this node

Methods

getTextValue()

```
public java.lang.String getTextValue()
```
Gets the current textual value of this object.

Returns:
the current value.

setTextValue(String)

```
public void setTextValue(java.lang.String value)
```
Sets the current textual value of this object.

unset()

```
public void unset()
```

Retracts any user selection made toward this node

oracle.apps.cz.cio
LogicalException

Syntax

```
public class LogicalException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.apps.cz.cio.LogicalException
```

Direct Known Subclasses:

LogicalOverridableException

All Implemented Interfaces:

java.io.Serializable

Description

Signalled if a logical failure occurs. This failure could either be a contradiction, or a more serious problem.

Member Summary

Constructors

LogicalException()

LogicalException(LogicException, Configuration)

LogicalException(Reason, Configuration)

LogicalException(String, Configuration)

Methods

getCause()

Member Summary

getMessage()

getMessageHeader()

getReasons()

isOverridable()

Inherited Member Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructors

LogicalException()

```
public  LogicalException()
```

LogicalException(LogicException, Configuration)

```
public  LogicalException(oracle.apps.cz.logic.LogicException le, Configuration
config)
```

LogicalException(Reason, Configuration)

```
public  LogicalException(Reason r, Configuration config)
```

LogicalException(String, Configuration)

```
public  LogicalException(java.lang.String msg, Configuration config)
```

Methods

getCause()

```
public IRuntimeNode getCause()
```

getMessage()

```
public java.lang.String getMessage()
```

Overrides:

java.lang.Throwable.getMessage() in class java.lang.Throwable

getMessageHeader()

```
public java.lang.String getMessageHeader()
```

getReasons()

```
public com.sun.java.util.collections.List getReasons()
```

isOverridable()

```
public boolean isOverridable()
```

oracle.apps.cz.cio
LogicalOverridableException

Syntax

```
public class LogicalOverridableException extends LogicalException

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--LogicalException
                    |
                    +--oracle.apps.cz.cio.LogicalOverridableException
```

All Implemented Interfaces:

java.io.Serializable

Description

Signalled if a logical contradiction occurs that can be overriden.

Member Summary

Constructors

LogicalOverridableException(LogicContradictionException, Configuration)

Methods

equals(Object)

isOverridable()

override()

Inherited Member Summary

Methods inherited from interface LogicalException

getCause(), getMessage(), getMessageHeader(), getReasons()

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructors

LogicalOverridableException(LogicContradictionException, Configuration)

```
public
LogicalOverridableException(oracle.apps.cz.logic.LogicContradictionException
lce, Configuration config)
```

Methods

equals(Object)

```
public boolean equals(java.lang.Object lce)
```

Overrides:

java.lang.Object.equals(java.lang.Object) in class java.lang.Object

isOverridable()

```
public boolean isOverridable()
```

Overrides:

isOverridable() in class LogicalException

override()

```
public void override()
```

oracle.apps.cz.cio
LogicalRuntimeException

Syntax

```
public class LogicalRuntimeException extends java.lang.RuntimeException

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--oracle.apps.cz.cio.LogicalRuntimeException
```

All Implemented Interfaces:

`java.io.Serializable`

Description

Signalled if a fatal logic exception occured.

Member Summary

Constructors

LogicalRuntimeException(LogicException)

LogicalRuntimeException(String)

Methods

getLogicException()

Inherited Member Summary

Methods inherited from class java.lang.Throwable

Inherited Member Summary

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructors

LogicalRuntimeException(LogicException)

```
public  LogicalRuntimeException(oracle.apps.cz.logic.LogicException le)
```

LogicalRuntimeException(String)

```
public  LogicalRuntimeException(java.lang.String msg)
```

Methods

getLogicException()

```
public oracle.apps.cz.logic.LogicException getLogicException()
```

oracle.apps.cz.cio
MissingFileException

Syntax

```
public class MissingFileException extends java.lang.RuntimeException

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--oracle.apps.cz.cio.MissingFileException
```

All Implemented Interfaces:

java.io.Serializable

Description

Signalled if a particular logic file is missing.

Inherited Member Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

oracle.apps.cz.cio
MissingLogicException

Syntax

```
public class MissingLogicException extends java.lang.RuntimeException

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--java.lang.RuntimeException
                    |
                    +--oracle.apps.cz.cio.MissingLogicException
```

All Implemented Interfaces:

java.io.Serializable

Description

Signalled if a particular logic record is missing.

Inherited Member Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

oracle.apps.cz.cio
NoAtpCalculatedException

Syntax

```
public class NoAtpCalculatedException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.apps.cz.cio.NoAtpCalculatedException
```

All Implemented Interfaces:

java.io.Serializable

Description

Exception which is thrown when an ATP method is called on an item for which ATP is not calculated.

Inherited Member Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

oracle.apps.cz.cio
NoConfigHeaderException

Syntax

```
public class NoConfigHeaderException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.apps.cz.cio.NoConfigHeaderException
```

All Implemented Interfaces:

java.io.Serializable

Description

Signalled if the configuration hasn't been saved yet.

Member Summary

Constructors

NoConfigHeaderException()

Inherited Member Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructors

NoConfigHeaderException()

```
public  NoConfigHeaderException()
```

oracle.apps.cz.cio NonPricedNodeException

Syntax

```
public class NonPricedNodeException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.apps.cz.cio.NonPricedNodeException
```

All Implemented Interfaces:

java.io.Serializable

Description

Exception which is thrown when a pricing method is called on an item which should not be priced.

Inherited Member Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

oracle.apps.cz.cio
NoSuchChildException

Syntax

```
public class NoSuchChildException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.apps.cz.cio.NoSuchChildException
```

All Implemented Interfaces:

java.io.Serializable

Description

Signalled if a requested child does not exist.

Member Summary

Constructors

NoSuchChildException(IRuntimeNode, int)

NoSuchChildException(IRuntimeNode, String)

Methods

getID()

getName()

getParent()

Inherited Member Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructors

NoSuchChildException(IRuntimeNode, int)

```
public  NoSuchChildException(IRuntimeNode parent, int id)
```

NoSuchChildException(IRuntimeNode, String)

```
public  NoSuchChildException(IRuntimeNode parent, java.lang.String name)
```

Methods

getID()

```
public int getID()
```

getName()

```
public java.lang.String getName()
```

getParent()

```
public IRuntimeNode getParent()
```

oracle.apps.cz.cio
NotOneProductException

Syntax

```
public class NotOneProductException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.apps.cz.cio.NotOneProductException
```

All Implemented Interfaces:

java.io.Serializable

Description

Exception which is thrown when a client tries to create a configuration by specifying the name of the project and the project contains more than one or no products.

Inherited Member Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

oracle.apps.cz.cio
NotOneProjectException

Syntax

```
public class NotOneProjectException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.apps.cz.cio.NotOneProjectException
```

All Implemented Interfaces:

java.io.Serializable

Description

Exception which is thrown when a client tries to create a configuration by specifying the name of the project and the project name identifies more than one or no projects.

Inherited Member Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

oracle.apps.cz.cio
Option

Syntax

```
public class Option extends OptionNode

java.lang.Object
  |
  +--RuntimeNode
        |
        +--StateNode
              |
              +--StateCountNode
                    |
                    +--OptionNode
                          |
                          +--oracle.apps.cz.cio.Option
```

All Implemented Interfaces:

ICount, IOption, IPrice, IRuntimeNode, IState

Description

Represents an option of an option feature.

Member Summary

Methods

getType()

Inherited Member Summary

Fields inherited from interface IState

FALSE, LFALSE, LTRUE, TOGGLE, TRUE, UFALSE, UNKNOWN, UTRUE

Fields inherited from interface IRuntimeNode

Inherited Member Summary

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class OptionNode

deselect(), isOptionMutexed(), isSelected(), select(), setState(int)

Methods inherited from class StateCountNode

addPricingNotification(String), clearDiscountedPrice(), clearPricingNotifications(), getCount(), getDiscountedPrice(), getExtendedPrice(), getItemKey(), getListPrice(), getPricingNotifications(), getUomCode(), setCount(int), setDiscountedPrice(double), setListPrice(double), toString()

Methods inherited from class StateNode

getState(), isDefaultState(int), isFalseState(int), isLogicState(int), isTrueState(int), isUnknownState(int), isUserState(int), statesMatch(int, int), unset()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IOption

deselect(), isOptionMutexed(), isSelected(), select()

Methods inherited from interface IState

getState(), setState(int), unset()

Methods inherited from interface ICount

getCount(), setCount(int), unset()

Methods inherited from interface IPrice

getDatabaseID(), getDiscountedPrice(), getExtendedPrice(), getItemKey(), getListPrice(), getPricingNotifications(), getUomCode()

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getType()

```
public int getType()
```

Overrides:
getType() in class RuntimeNode

oracle.apps.cz.cio
OptionFeature

Syntax

```
public class OptionFeature extends OptionFeatureNode

java.lang.Object
  |
  +--RuntimeNode
        |
        +--StateNode
              |
              +--StateCountNode
                    |
                    +--OptionFeatureNode
                          |
                          +--oracle.apps.cz.cio.OptionFeature
```

All Implemented Interfaces:

ICount, IOptionFeature, IPrice, IRuntimeNode, IState

Description

Represents a feature with selectable options.

Member Summary

Methods

getMaxSelected()

getMinSelected()

getType()

hasMaxSelected()

hasMinSelected()

Inherited Member Summary

Fields inherited from interface IState

FALSE, LFALSE, LTRUE, TOGGLE, TRUE, UFALSE, UNKNOWN, UTRUE

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class OptionFeatureNode

deselect(IOption), getSelectedOption(), getSelectedOptions(), isSelectionMutexed(), select(IOption)

Methods inherited from class StateCountNode

addPricingNotification(String), clearDiscountedPrice(), clearPricingNotifications(), getCount(), getDiscountedPrice(), getExtendedPrice(), getItemKey(), getListPrice(), getPricingNotifications(), getUomCode(), setCount(int), setDiscountedPrice(double), setListPrice(double), toString()

Methods inherited from class StateNode

getState(), isDefaultState(int), isFalseState(int), isLogicState(int), isTrueState(int), isUnknownState(int), isUserState(int), setState(int), statesMatch(int, int), unset()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IOptionFeature

deselect(IOption), getSelectedOption(), getSelectedOptions(), isSelectionMutexed(), select(IOption)

Methods inherited from interface IState

getState(), setState(int), unset()

Methods inherited from interface ICount

getCount(), setCount(int), unset()

Methods inherited from interface IPrice

getDatabaseID(), getDiscountedPrice(), getExtendedPrice(), getItemKey(), getListPrice(), getPricingNotifications(), getUomCode()

Methods inherited from interface IRuntimeNode

Inherited Member Summary

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getMaxSelected()

```
public int getMaxSelected()
```

getMinSelected()

```
public int getMinSelected()
```

getType()

```
public int getType()
```

Overrides:

getType() in class RuntimeNode

hasMaxSelected()

```
public boolean hasMaxSelected()
```

hasMinSelected()

```
public boolean hasMinSelected()
```

oracle.apps.cz.cio
OptionFeatureNode

Syntax

```
public abstract class OptionFeatureNode extends StateCountNode implements
IOptionFeature

java.lang.Object
  |
  +--RuntimeNode
        |
        +--StateNode
              |
              +--StateCountNode
                    |
                    +--oracle.apps.cz.cio.OptionFeatureNode
```

Direct Known Subclasses:

BomNode, OptionFeature

All Implemented Interfaces:

ICount, IOptionFeature, IPrice, IRuntimeNode, IState

Description

An abstract class implementing behavior commont to all features with options.

Member Summary

Methods

deselect(IOption)

getSelectedOption()

getSelectedOptions()

isSelectionMutexed()

select(IOption)

Inherited Member Summary

Fields inherited from interface IState

FALSE, LFALSE, LTRUE, TOGGLE, TRUE, UFALSE, UNKNOWN, UTRUE

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class StateCountNode

addPricingNotification(String), clearDiscountedPrice(), clearPricingNotifications(), getCount(), getDiscountedPrice(), getExtendedPrice(), getItemKey(), getListPrice(), getPricingNotifications(), getUomCode(), setCount(int), setDiscountedPrice(double), setListPrice(double), toString()

Methods inherited from class StateNode

getState(), isDefaultState(int), isFalseState(int), isLogicState(int), isTrueState(int), isUnknownState(int), isUserState(int), setState(int), statesMatch(int, int), unset()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IOptionFeature

getMaxSelected(), getMinSelected(), hasMaxSelected(), hasMinSelected()

Methods inherited from interface IState

getState(), setState(int), unset()

Methods inherited from interface ICount

getCount(), setCount(int), unset()

Methods inherited from interface IPrice

getDatabaseID(), getDiscountedPrice(), getExtendedPrice(), getItemKey(), getListPrice(), getPricingNotifications(), getUomCode()

Methods inherited from interface IRuntimeNode

Inherited Member Summary

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

deselect(IOption)

```
public void deselect(IOption option)
```

Specified By:

deselect(IOption) in interface IOptionFeature

getSelectedOption()

```
public IOption getSelectedOption()
```

Specified By:

getSelectedOption() in interface IOptionFeature

getSelectedOptions()

```
public com.sun.java.util.collections.List getSelectedOptions()
```

Specified By:

getSelectedOptions() in interface IOptionFeature

isSelectionMutexed()

```
public boolean isSelectionMutexed()
```

Specified By:

isSelectionMutexed() in interface IOptionFeature

select(IOption)

```
public void select(IOption option)
```

Specified By:
select(IOption) in interface IOptionFeature

oracle.apps.cz.cio
OptionNode

Syntax

```
public abstract class OptionNode extends StateCountNode implements IOption

java.lang.Object
  |
  +--RuntimeNode
        |
        +--StateNode
              |
              +--StateCountNode
                    |
                    +--oracle.apps.cz.cio.OptionNode
```

Direct Known Subclasses:

Option

All Implemented Interfaces:

ICount, IOption, IPrice, IRuntimeNode, IState

Description

An abstract class implementing behavior common to all option-like objects.

Member Summary

Methods

deselect()

isOptionMutexed()

isSelected()

select()

setState(int)

Inherited Member Summary

Fields inherited from interface IState

FALSE, LFALSE, LTRUE, TOGGLE, TRUE, UFALSE, UNKNOWN, UTRUE

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class StateCountNode

addPricingNotification(String), clearDiscountedPrice(), clearPricingNotifications(), getCount(), getDiscountedPrice(), getExtendedPrice(), getItemKey(), getListPrice(), getPricingNotifications(), getUomCode(), setCount(int), setDiscountedPrice(double), setListPrice(double), toString()

Methods inherited from class StateNode

getState(), isDefaultState(int), isFalseState(int), isLogicState(int), isTrueState(int), isUnknownState(int), isUserState(int), statesMatch(int, int), unset()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IState

getState(), unset()

Methods inherited from interface ICount

getCount(), setCount(int), unset()

Methods inherited from interface IPrice

getDatabaseID(), getDiscountedPrice(), getExtendedPrice(), getItemKey(), getListPrice(), getPricingNotifications(), getUomCode()

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

deselect()

```
public void deselect()
```

Specified By:
deselect() in interface IOption

isOptionMutexed()

```
public boolean isOptionMutexed()
```

Specified By:
isOptionMutexed() in interface IOption

isSelected()

```
public boolean isSelected()
```

Specified By:
isSelected() in interface IOption

select()

```
public void select()
```

Specified By:
select() in interface IOption

setState(int)

```
public void setState(int newState)
```

Specified By:
setState(int) in interface IState

Specified By:
setState(int) in interface IState

Overrides:

setState(int) in class StateNode

oracle.apps.cz.cio
PricingUnavailableException

Syntax

```
public class PricingUnavailableException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.apps.cz.cio.PricingUnavailableException
```

All Implemented Interfaces:

java.io.Serializable

Description

Signals that the CIO pricing functionality is not available.

Member Summary

Constructors

PricingUnavailableException(String)

PricingUnavailableException(String, Object, Log)

Inherited Member Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructors

PricingUnavailableException(String)

```
public  PricingUnavailableException(java.lang.String reason)
```

PricingUnavailableException(String, Object, Log)

```
public  PricingUnavailableException(java.lang.String reason, java.lang.Object
source, oracle.apps.fnd.common.Log log)
```

oracle.apps.cz.cio
Property

Syntax

```
public class Property extends java.lang.Object

java.lang.Object
  |
  +--oracle.apps.cz.cio.Property
```

Description

Represents name/value properties associated with runtime nodes.

Member Summary

Methods	
getBooleanValue()	Returns the property's value as a boolean.
getDecimalValue()	Returns the property's value as a double.
getDescription()	Returns the property's description.
getIntValue()	Returns the property's value as an integer.
getName()	Returns the property's name.
getStringValue()	Returns the property's value as a string.
getUnit()	Returns the property's unit of measure.
hasBooleanValue()	Returns true if property is a boolean property.
hasDecimalValue()	Returns true if property is a decimal property.
hasDefaultValue()	Checks to see if property has overridden its default value.
hasIntegerValue()	Returns true if property is an integer property.
hasStringValue()	Returns true if property is a string property.

Inherited Member Summary

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Methods

getBooleanValue()

```
public boolean getBooleanValue()
```
Returns the property's value as a boolean.

getDecimalValue()

```
public double getDecimalValue()
```
Returns the property's value as a double.

getDescription()

```
public java.lang.String getDescription()
```
Returns the property's description.

getIntValue()

```
public int getIntValue()
```
Returns the property's value as an integer.

getName()

```
public java.lang.String getName()
```
Returns the property's name.

getStringValue()

```
public java.lang.String getStringValue()
```
Returns the property's value as a string.

getUnit()

```
public java.lang.String getUnit()
```
Returns the property's unit of measure.

hasBooleanValue()

```
public boolean hasBooleanValue()
```
Returns true if property is a boolean property.

hasDecimalValue()

```
public boolean hasDecimalValue()
```
Returns true if property is a decimal property.

hasDefaultValue()

```
public boolean hasDefaultValue()
```
Checks to see if property has overridden its default value.

hasIntegerValue()

```
public boolean hasIntegerValue()
```
Returns true if property is an integer property.

hasStringValue()

```
public boolean hasStringValue()
```
Returns true if property is a string property.

oracle.apps.cz.cio ReadOnlyDecimalNode

Syntax

```
public abstract class ReadOnlyDecimalNode extends RuntimeNode implements
IReadOnlyDecimal

java.lang.Object
  |
  +--RuntimeNode
        |
        +--oracle.apps.cz.cio.ReadOnlyDecimalNode
```

Direct Known Subclasses:

DecimalNode

All Implemented Interfaces:

IReadOnlyDecimal, IRuntimeNode

Description

An abstract class implementing behavior common to objects with a decimal value.

Member Summary

Methods

getDecimalValue()

toString()

Inherited Member Summary

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class RuntimeNode

Inherited Member Summary

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getDecimalValue()

```
public double getDecimalValue()
```

Specified By:

getDecimalValue() in interface IReadOnlyDecimal

toString()

```
public java.lang.String toString()
```

Overrides:

toString() in class RuntimeNode

oracle.apps.cz.cio
Reason

Syntax

```
public class Reason extends java.lang.Object

java.lang.Object
  |
  +--oracle.apps.cz.cio.Reason
```

Description

This class wraps the information returned by a contradiction in order to include information about internal error messages.

Member Summary

Fields	
DEFAULT	This reason initiated from inability to set a state, because of a default relation.
INTL_TEXT	The message is an internationalized text string.
MINMAX	This reason initiated from an internal MINMAX relationship.
ORTHEN	This reason initiated from an internal ORTHEN relationship.
TEXT	The message is an unknown format text string.
TRUEATBIRTH	This reason initiated from an internal relationship for a group.
Constructors	
Reason(int, IRuntimeNode, String)	Construct a reason given all of it's information.
Reason(Message, String, IRuntimeNode)	Constructs a reason from an FND message.
Reason(String)	Constructs a simple TEXT reason.
Methods	
getMsg()	Get the message associated with this reason.
getNode()	Get the node associated with this reason.
getType()	Get the type of reason is held in this object.

Member Summary	
toString()	Convert this object to a string.
translate()	This method returns the translated string for the reason.
translate(String)	This method returns the translated reason string using the given name for substitution variable.

Inherited Member Summary
Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Fields

DEFAULT

```
public static int DEFAULT
```
This reason initiated from inability to set a state, because of a default relation.

INTL_TEXT

```
public static int INTL_TEXT
```
The message is an internationalized text string.

MINMAX

```
public static int MINMAX
```
This reason initiated from an internal MINMAX relationship.

ORTHEN

```
public static int ORTHEN
```
This reason initiated from an internal ORTHEN relationship.

TEXT

```
public static int TEXT
```
The message is an unknown format text string.

TRUEATBIRTH

```
public static int TRUEATBIRTH
```
This reason initiated from an internal relationship for a group.

Constructors

Reason(int, IRuntimeNode, String)

```
public  Reason(int type, IRuntimeNode node, java.lang.String msg)
```
Construct a reason given all of it's information.

Parameters:

`type` - What type of reason this is.

`node` - The node that caused the problem.

`msg` - The message returned.

Reason(Message, String, IRuntimeNode)

```
public  Reason(oracle.apps.fnd.common.Message fndMsg, java.lang.String token,
IRuntimeNode node)
```
Constructs a reason from an FND message.

Parameters:

`fndMsg` - The FND message object with all but one token substituted.

`token` - The token name left to substitute.

`node` - The node requiring substitution.

Reason(String)

```
public  Reason(java.lang.String msg)
```
Constructs a simple TEXT reason.

Parameters:

`msg` - The message string for the reason.

Methods

getMsg()

```
public java.lang.String getMsg()
```
Get the message associated with this reason.

getNode()

```
public IRuntimeNode getNode()
```
Get the node associated with this reason.

getType()

```
public int getType()
```
Get the type of reason is held in this object.

toString()

```
public java.lang.String toString()
```
Convert this object to a string.

Overrides:

java.lang.Object.toString() in class java.lang.Object

translate()

```
public java.lang.String translate()
```
This method returns the translated string for the reason. If the string has a node name substitution then the internal name is used.

translate(String)

```
public java.lang.String translate(java.lang.String nodeName)
```
This method returns the translated reason string using the given name for substitution variable.

Parameters:

`nodeName` - The node name to substitute into the string.

oracle.apps.cz.cio
Resource

Syntax

```
public class Resource extends DecimalNode

java.lang.Object
  |
  +--RuntimeNode
        |
        +--ReadOnlyDecimalNode
              |
              +--DecimalNode
                    |
                    +--oracle.apps.cz.cio.Resource
```

All Implemented Interfaces:

IDecimal, IReadOnlyDecimal, IRuntimeNode

Description

Represents a consumable resource. A resource will signal a validation failure when it is overconsumed (in other words, when its value goes below zero). NOTE: This class inherits from DecimalNode, but the functionality of a DecimalNode (specifically the method SetDecimalValue()) is 'deprecated', meaning that it shouldn't be used on new projects and may be unsupported in a future release. Use only methods inherited from ReadOnlyDecimalNode.

Member Summary

Methods

getType()

Inherited Member Summary

Fields inherited from interface IRuntimeNode

Inherited Member Summary

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class DecimalNode

setDecimalValue(double), toString(), unset()

Methods inherited from class ReadOnlyDecimalNode

getDecimalValue()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IDecimal

setDecimalValue(double), unset()

Methods inherited from interface IReadOnlyDecimal

getDecimalValue()

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getType()

```
public int getType()
```

Overrides:

getType() in class RuntimeNode

oracle.apps.cz.cio RestoreValidationFailure

Syntax

```
public class RestoreValidationFailure extends ValidationFailure

java.lang.Object
  |
  +--StatusInfo
        |
        +--ValidationFailure
              |
              +--oracle.apps.cz.cio.RestoreValidationFailure
```

Description

Failure produced when restoring a configuration over a changed model.

Member Summary

Methods	
equals(Object)	
getInput()	Returns the input object where the validation failure occured
hashCode()	

Inherited Member Summary

Fields inherited from class ValidationFailure

COMPANION_FAILURE, MAX_FAILURE, MIN_FAILURE, MIN0_FAILURE, MINMAX_FAILURE, RESOURCE_FAILURE, RESTORE_FAILURE

Fields inherited from class StatusInfo

STATUS_DELETED, STATUS_EXISTING, STATUS_NEW

Methods inherited from class ValidationFailure

getMessage(), getMessage(String), getType(), toString()

Inherited Member Summary

Methods inherited from class StatusInfo

getNode(), getStatus(), statusToString(int), toString(boolean)

Methods inherited from class java.lang.Object

getClass, notify, notifyAll, wait, wait, wait

Methods

equals(Object)

```
public boolean equals(java.lang.Object obj)
```

Overrides:

equals(Object) in class ValidationFailure

getInput()

```
public oracle.apps.cz.dio.config.DbConfigInput getInput()
```

Returns the input object where the validation failure occured

Returns:

the failed DbConfigInput object

hashCode()

```
public int hashCode()
```

Overrides:

hashCode() in class StatusInfo

oracle.apps.cz.cio
RuntimeNode

Syntax

```
public abstract class RuntimeNode extends java.lang.Object implements
IRuntimeNode

java.lang.Object
  |
  +--oracle.apps.cz.cio.RuntimeNode
```

Direct Known Subclasses:

ComponentNode, IntegerNode, ReadOnlyDecimalNode, StateNode, TextNode

All Implemented Interfaces:

IRuntimeNode

Description

Abstract class implementing common behavior across all runtime nodes.

Member Summary

Methods

getChildByID(int)	Returns the child of this node with a given database ID.
getChildByName(String)	Returns the child of this node with a given name.
getChildren()	Returns a list of all children of this runtime node.
getChildrenByType(int)	Returns a list of all children of a particular type.
getConfiguration()	Returns the configuration to which this node belongs.
getDatabaseID()	Returns the database ID of the runtime node.
getDescription()	Returns the design-time description of the runtime node.
getName()	Returns the name of the runtime node.
getParent()	Returns the parent of this runtime node, or `null` if this is the root node.
getProperties()	Returns a collection of the properties associated with this node.

Member Summary	
getPropertyByName(String)	Returns a particular property of this node, based on its name.
getRuntimeID()	Returns the runtime ID for the node.
getSelectionLineID()	
getType()	Returns the type of the runtime node.
hasCount()	Returns true if the node has an object count.
hasDecimalValue()	Returns true if the node has a decimal value.
hasDescription()	Returns true if there is a design-time description of the runtime node.
hasIntegerValue()	Returns true if the node has a integer value.
hasSelectionLineID()	
hasState()	Returns true if the node has a logical state.
hasTextValue()	Returns true if the node has a text value.
isNative()	Returns true if this is a native BOM node
isUnsatisfied()	Returns true if this runtime node, or any of its children, is not fully configured.
isUnsatisfiedNode()	Returns true if this particular node is not fully configured.
toString()	
toString(boolean)	
typeToString(int)	Returns a string representation of a given runtime node type constant.

Inherited Member Summary

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods

getChildByID(int)

```
public IRuntimeNode getChildByID(int id)
```
Returns the child of this node with a given database ID.

Specified By:

getChildByID(int) in interface IRuntimeNode

Throws:

NoSuchChildException - if there is no child with such ID.

getChildByName(String)

```
public IRuntimeNode getChildByName(java.lang.String name)
```
Returns the child of this node with a given name.

Specified By:

getChildByName(String) in interface IRuntimeNode

Throws:

NoSuchChildException - if there is no child with such name.

getChildren()

```
public com.sun.java.util.collections.List getChildren()
```
Returns a list of all children of this runtime node.

Specified By:

getChildren() in interface IRuntimeNode

getChildrenByType(int)

```
public com.sun.java.util.collections.List getChildrenByType(int type)
```
Returns a list of all children of a particular type.

Specified By:

getChildrenByType(int) in interface IRuntimeNode

getConfiguration()

`public Configuration getConfiguration()`
Returns the configuration to which this node belongs.

Specified By:

getConfiguration() in interface IRuntimeNode

getDatabaseID()

`public int getDatabaseID()`
Returns the database ID of the runtime node.

Specified By:

getDatabaseID() in interface IRuntimeNode

getDescription()

`public java.lang.String getDescription()`
Returns the design-time description of the runtime node.

Specified By:

getDescription() in interface IRuntimeNode

getName()

`public java.lang.String getName()`
Returns the name of the runtime node.

Specified By:

getName() in interface IRuntimeNode

getParent()

`public IRuntimeNode getParent()`
Returns the parent of this runtime node, or `null` if this is the root node.

Specified By:

getParent() in interface IRuntimeNode

getProperties()

```
public com.sun.java.util.collections.Collection getProperties()
```
Returns a collection of the properties associated with this node. The collection contains items of the type IProperty.

Specified By:

getProperties() in interface IRuntimeNode

getPropertyByName(String)

```
public Property getPropertyByName(java.lang.String name)
```
Returns a particular property of this node, based on its name. Returns null if a property of the given name does not exist.

Specified By:

getPropertyByName(String) in interface IRuntimeNode

getRuntimeID()

```
public int getRuntimeID()
```
Returns the runtime ID for the node. This ID is unique across all nodes in a particular configuration.

Specified By:

getRuntimeID() in interface IRuntimeNode

getSelectionLineID()

```
public int getSelectionLineID()
```

Specified By:

getSelectionLineID() in interface IRuntimeNode

getType()

```
public abstract int getType()
```
Returns the type of the runtime node. Must be implemented.

Specified By:

getType() in interface IRuntimeNode

hasCount()

```
public boolean hasCount()
```
Returns true if the node has an object count.

Specified By:

hasCount() in interface IRuntimeNode

hasDecimalValue()

```
public boolean hasDecimalValue()
```
Returns true if the node has a decimal value.

Specified By:

hasDecimalValue() in interface IRuntimeNode

hasDescription()

```
public boolean hasDescription()
```
Returns true if there is a design-time description of the runtime node.

Specified By:

hasDescription() in interface IRuntimeNode

hasIntegerValue()

```
public boolean hasIntegerValue()
```
Returns true if the node has a integer value.

hasSelectionLineID()

```
public boolean hasSelectionLineID()
```

Specified By:

hasSelectionLineID() in interface IRuntimeNode

hasState()

```
public boolean hasState()
```
Returns true if the node has a logical state.

Specified By:

hasState() in interface IRuntimeNode

hasTextValue()

```
public boolean hasTextValue()
```

Returns true if the node has a text value.

Specified By:

hasTextValue() in interface IRuntimeNode

isNative()

```
public boolean isNative()
```

Returns true if this is a native BOM node

Specified By:

isNative() in interface IRuntimeNode

isUnsatisfied()

```
public boolean isUnsatisfied()
```

Returns true if this runtime node, or any of its children, is not fully configured.

Specified By:

isUnsatisfied() in interface IRuntimeNode

isUnsatisfiedNode()

```
public boolean isUnsatisfiedNode()
```

Returns true if this particular node is not fully configured.

Specified By:

isUnsatisfiedNode() in interface IRuntimeNode

toString()

```
public java.lang.String toString()
```

Overrides:

java.lang.Object.toString() in class java.lang.Object

toString(boolean)

```
public java.lang.String toString(boolean description)
```

Specified By:

toString(boolean) in interface IRuntimeNode

typeToString(int)

```
public static java.lang.String typeToString(int type)
```

Returns a string representation of a given runtime node type constant.

oracle.apps.cz.cio SelectionNotMutexedException

Syntax

```
public class SelectionNotMutexedException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.apps.cz.cio.SelectionNotMutexedException
```

All Implemented Interfaces:

java.io.Serializable

Description

Signalled when an mutexed selection operation is performed on an option feature that does not support mutexed selection.

Member Summary

Methods

getFeature()

Inherited Member Summary

Methods inherited from class java.lang.Throwable

fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods

getFeature()

```
public IOptionFeature getFeature()
```

oracle.apps.cz.cio
StateCountNode

Syntax

```
public abstract class StateCountNode extends StateNode implements ICount, IPrice

java.lang.Object
  |
  +--RuntimeNode
        |
        +--StateNode
              |
              +--oracle.apps.cz.cio.StateCountNode
```

Direct Known Subclasses:

CountFeature, OptionFeatureNode, OptionNode

All Implemented Interfaces:

ICount, IPrice, IRuntimeNode, IState

Description

Abstract class implementing common behavior for nodes with a logic state and count.

Member Summary

Methods

addPricingNotification(String)

clearDiscountedPrice()

clearPricingNotifications()

getCount()

getDiscountedPrice()

getExtendedPrice()

getItemKey()

Member Summary

getListPrice()

getPricingNotifications()

getUomCode()

setCount(int)

setDiscountedPrice(double)

setListPrice(double)

toString()

Inherited Member Summary

Fields inherited from interface IState

FALSE, LFALSE, LTRUE, TOGGLE, TRUE, UFALSE, UNKNOWN, UTRUE

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class StateNode

getState(), isDefaultState(int), isFalseState(int), isLogicState(int), isTrueState(int), isUnknownState(int), isUserState(int), setState(int), statesMatch(int, int), unset()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface ICount

unset()

Methods inherited from interface IPrice

getDatabaseID()

Methods inherited from interface IState

Inherited Member Summary

getState(), setState(int)

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

addPricingNotification(String)

```
public void addPricingNotification(java.lang.String message)
```

clearDiscountedPrice()

```
public void clearDiscountedPrice()
```

clearPricingNotifications()

```
public void clearPricingNotifications()
```

getCount()

```
public int getCount()
```

Specified By:
getCount() in interface ICount

Specified By:
getCount() in interface ICount

getDiscountedPrice()

```
public double getDiscountedPrice()
```

Specified By:
getDiscountedPrice() in interface IPrice

getExtendedPrice()

```
public double getExtendedPrice()
```

Specified By:
getExtendedPrice() in interface IPrice

getItemKey()

```
public java.lang.String getItemKey()
```

Specified By:
getItemKey() in interface IPrice

getListPrice()

```
public double getListPrice()
```

Specified By:
getListPrice() in interface IPrice

getPricingNotifications()

```
public java.lang.String getPricingNotifications()
```

Specified By:
getPricingNotifications() in interface IPrice

getUomCode()

```
public java.lang.String getUomCode()
```

Specified By:
getUomCode() in interface IPrice

setCount(int)

```
public void setCount(int newCount)
```

Specified By:
setCount(int) in interface ICount

Specified By:

setCount(int) in interface ICount

setDiscountedPrice(double)

```
public void setDiscountedPrice(double discountedPrice)
```

setListPrice(double)

```
public void setListPrice(double listPrice)
```

toString()

```
public java.lang.String toString()
```

Overrides:

toString() in class RuntimeNode

oracle.apps.cz.cio
StateNode

Syntax

```
public abstract class StateNode extends RuntimeNode implements IState

java.lang.Object
  |
  +--RuntimeNode
        |
        +--oracle.apps.cz.cio.StateNode
```

Direct Known Subclasses:

BooleanFeature, StateCountNode

All Implemented Interfaces:

IRuntimeNode, IState

Description

Abstract class implementing common behavior across nodes with logic state.

Member Summary

Methods

getState()	
isDefaultState(int)	Returns true if the given state is default (not unknown, or user, or logic).
isFalseState(int)	Returns true if the given state is false (not unknown or true).
isLogicState(int)	Returns true if the given state is logic (not unknown, or user, or default).
isTrueState(int)	Returns true if the given state is true (not unknown or false).
isUnknownState(int)	Returns true if the given state is unknown (not true or false).
isUserState(int)	Returns true if the given state is user (not unknown, or logic, or default).
setState(int)	
statesMatch(int, int)	Returns true if the two given states match.

Member Summary

unset()

Inherited Member Summary

Fields inherited from interface IState

FALSE, LFALSE, LTRUE, TOGGLE, TRUE, UFALSE, UNKNOWN, UTRUE

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getState()

```
public int getState()
```

Specified By:

getState() in interface IState

isDefaultState(int)

```
public static boolean isDefaultState(int state)
```

Returns true if the given state is default (not unknown, or user, or logic).

isFalseState(int)

```
public static boolean isFalseState(int state)
```
Returns true if the given state is false (not unknown or true).

isLogicState(int)

```
public static boolean isLogicState(int state)
```
Returns true if the given state is logic (not unknown, or user, or default).

isTrueState(int)

```
public static boolean isTrueState(int state)
```
Returns true if the given state is true (not unknown or false).

isUnknownState(int)

```
public static boolean isUnknownState(int state)
```
Returns true if the given state is unknown (not true or false).

isUserState(int)

```
public static boolean isUserState(int state)
```
Returns true if the given state is user (not unknown, or logic, or default).

setState(int)

```
public void setState(int newState)
```

Specified By:

setState(int) in interface IState

statesMatch(int, int)

```
public static boolean statesMatch(int inputState, int outputState)
```
Returns true if the two given states match.

unset()

```
public void unset()
```

Specified By:

unset() in interface IState

oracle.apps.cz.cio
StatusInfo

Syntax

```
public class StatusInfo extends java.lang.Object

java.lang.Object
  |
  +--oracle.apps.cz.cio.StatusInfo
```

Direct Known Subclasses:

ValidationFailure

Description

Contains information about a status change for a particular runtime node. The status can be STATUS_NEW, STATUS_EXISTING, or STATUS_DELETED. The condition for which this status holds depends on which list the status exists. Possibilities include validation failure, selected nodes, and unsatisfied nodes.

Member Summary

Fields	
STATUS_DELETED	The node has newly lost this status since the last check.
STATUS_EXISTING	The already had this status during the last check, and it still does.
STATUS_NEW	The node has newly attained this status since the last check.
Methods	
equals(Object)	
getNode()	Returns the runtime node with which this status is associated.
getStatus()	Returns the current status of the node.
hashCode()	
statusToString(int)	Return a printable representation of a status constant.
toString()	
toString(boolean)	

Inherited Member Summary

Methods inherited from class java.lang.Object

getClass, notify, notifyAll, wait, wait, wait

Fields

STATUS_DELETED

```
public static final int STATUS_DELETED
```
The node has newly lost this status since the last check.

STATUS_EXISTING

```
public static final int STATUS_EXISTING
```
The already had this status during the last check, and it still does.

STATUS_NEW

```
public static final int STATUS_NEW
```
The node has newly attained this status since the last check.

Methods

equals(Object)

```
public boolean equals(java.lang.Object obj)
```

Overrides:

java.lang.Object.equals(java.lang.Object) in class java.lang.Object

getNode()

```
public IRuntimeNode getNode()
```
Returns the runtime node with which this status is associated.

getStatus()

```
public int getStatus()
```
Returns the current status of the node.

hashCode()

```
public int hashCode()
```

Overrides:

java.lang.Object.hashCode() in class java.lang.Object

statusToString(int)

```
public static java.lang.String statusToString(int status)
```

Return a printable representation of a status constant.

toString()

```
public java.lang.String toString()
```

Overrides:

java.lang.Object.toString() in class java.lang.Object

toString(boolean)

```
public java.lang.String toString(boolean description)
```

oracle.apps.cz.cio
TextFeature

Syntax

```
public class TextFeature extends TextNode

java.lang.Object
  |
  +--RuntimeNode
        |
        +--TextNode
              |
              +--oracle.apps.cz.cio.TextFeature
```

All Implemented Interfaces:

IRuntimeNode, IText

Description

Represents a feature that has a textual value.

Member Summary

Methods

getType()

Inherited Member Summary

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class TextNode

getTextValue(), setTextValue(String), unset()

Methods inherited from class RuntimeNode

Inherited Member Summary

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IText

getTextValue(), setTextValue(String), unset()

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getType()

```
public int getType()
```

Overrides:

getType() in class RuntimeNode

oracle.apps.cz.cio
TextNode

Syntax

```
public abstract class TextNode extends RuntimeNode implements IText

java.lang.Object
  |
  +--RuntimeNode
        |
        +--oracle.apps.cz.cio.TextNode
```

Direct Known Subclasses:

TextFeature

All Implemented Interfaces:

IRuntimeNode, IText

Description

Represents a feature that has a textual value.

Member Summary

Methods

getTextValue()

setTextValue(String)

unset()

Inherited Member Summary

Fields inherited from interface IRuntimeNode

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Inherited Member Summary

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), getType(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getTextValue()

```
public java.lang.String getTextValue()
```

Specified By:

getTextValue() in interface IText

setTextValue(String)

```
public void setTextValue(java.lang.String newTextValue)
```

Specified By:

setTextValue(String) in interface IText

unset()

```
public void unset()
```

Specified By:

unset() in interface IText

oracle.apps.cz.cio
Total

Syntax

```
public class Total extends DecimalNode

java.lang.Object
  |
  +--RuntimeNode
        |
        +--ReadOnlyDecimalNode
              |
              +--DecimalNode
                    |
                    +--oracle.apps.cz.cio.Total
```

All Implemented Interfaces:

IDecimal, IReadOnlyDecimal, IRuntimeNode

Description

Represents a total that has a decimal numeric value. NOTE: This class inherits from DecimalNode, but the functionality of a DecimalNode (specifically the method SetDecimalValue()) is 'deprecated', meaning that it shouldn't be used on new projects and may be unsupported in a future release. Use only methods inherited from ReadOnlyDecimalNode.

Member Summary

Methods

getType()

Inherited Member Summary

Fields inherited from interface IRuntimeNode

Inherited Member Summary

ALL_FEATURES, BOM_MODEL, BOM_OPTION_CLASS, BOM_STD_ITEM, BOOLEAN_FEATURE, COMPONENT, COMPONENT_SET, COUNT_FEATURE, DECIMAL_FEATURE, INTEGER_FEATURE, OPTION, OPTION_FEATURE, RESOURCE, TEXT_FEATURE, TOTAL

Methods inherited from class DecimalNode

setDecimalValue(double), toString(), unset()

Methods inherited from class ReadOnlyDecimalNode

getDecimalValue()

Methods inherited from class RuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasIntegerValue(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean), typeToString(int)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface IDecimal

setDecimalValue(double), unset()

Methods inherited from interface IReadOnlyDecimal

getDecimalValue()

Methods inherited from interface IRuntimeNode

getChildByID(int), getChildByName(String), getChildren(), getChildrenByType(int), getConfiguration(), getDatabaseID(), getDescription(), getName(), getParent(), getProperties(), getPropertyByName(String), getRuntimeID(), getSelectionLineID(), hasCount(), hasDecimalValue(), hasDescription(), hasSelectionLineID(), hasState(), hasTextValue(), isNative(), isUnsatisfied(), isUnsatisfiedNode(), toString(boolean)

Methods

getType()

```
public int getType()
```

Overrides:

getType() in class RuntimeNode

oracle.apps.cz.cio
TransactionException

Syntax

```
public class TransactionException extends java.lang.Exception

java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--oracle.apps.cz.cio.TransactionException
```

All Implemented Interfaces:

java.io.Serializable

Description

Signalled if a particular logic file is missing.

Member Summary	
Methods	
getAction()	Returns a String representation of the action that caused the exception

Inherited Member Summary
Methods inherited from class java.lang.Throwable
fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, printStackTrace, printStackTrace, toString
Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods

getAction()

```
public java.lang.String getAction()
```

Returns a String representation of the action that caused the exception

oracle.apps.cz.cio
ValidationFailure

Syntax

```
public class ValidationFailure extends StatusInfo

java.lang.Object
  |
  +--StatusInfo
        |
        +--oracle.apps.cz.cio.ValidationFailure
```

Direct Known Subclasses:

CompanionValidationFailure, RestoreValidationFailure

Description

Implements behavior common to all validation failures.

Member Summary

Fields

COMPANION_FAILURE

MAX_FAILURE

MIN_FAILURE

MIN0_FAILURE

MINMAX_FAILURE

RESOURCE_FAILURE

RESTORE_FAILURE

Methods

equals(Object)

getMessage()

getMessage(String)

getType()

Member Summary

toString()

Inherited Member Summary

Fields inherited from class StatusInfo

STATUS_DELETED, STATUS_EXISTING, STATUS_NEW

Methods inherited from class StatusInfo

getNode(), getStatus(), hashCode(), statusToString(int), toString(boolean)

Methods inherited from class java.lang.Object

getClass, notify, notifyAll, wait, wait, wait

Fields

COMPANION_FAILURE

```
public static final int COMPANION_FAILURE
```

MAX_FAILURE

```
public static final int MAX_FAILURE
```

MIN_FAILURE

```
public static final int MIN_FAILURE
```

MIN0_FAILURE

```
public static final int MIN0_FAILURE
```

MINMAX_FAILURE

```
public static final int MINMAX_FAILURE
```

RESOURCE_FAILURE

```
public static final int RESOURCE_FAILURE
```

RESTORE_FAILURE

```
public static final int RESTORE_FAILURE
```

Methods

equals(Object)

```
public boolean equals(java.lang.Object obj)
```

Overrides:

equals(Object) in class StatusInfo

getMessage()

```
public java.lang.String getMessage()
```

getMessage(String)

```
public java.lang.String getMessage(java.lang.String nodeName)
```

getType()

```
public int getType()
```

toString()

```
public java.lang.String toString()
```

Overrides:

toString() in class StatusInfo

C

Package oracle.apps.cz.common

Description

Class Summary	
Classes	
CZContext	Represents the runtime context of a configuration session.

oracle.apps.cz.common
CZContext

Syntax

```
public final class CZContext
```

oracle.apps.cz.common.CZContext

Description

Represents the runtime context of a configuration session. The context owns the database connection, resources, and log object. It also maintains apps, user, and language information. CZContext is a shadow of the oracle.apps.fnd.common.AppsContext implementation and is designed to operate outside of the middle-tier Oracle Apps environment. The intent is for the CIO to run with either the AppsContext or CZContext depending on the runtime environment.

Member Summary

Constructors	
CZContext(String, String)	Creates a new CZContext containing a connection established with the supplied database url.
CZContext(String, String, String, String)	Creates a new CZContext containing a connection established with the supplied database url, user name, and password.
Methods	
getAppId(String)	Returns the application ID for product configuration (CZ).
getCurrLangCode()	Returns the current language code.
getCurrLangInfo()	This method is restricted.
getDbOwner()	Returns name of SellingPoint schema owner.
getJDBCConnection(Object)	Returns the JDBC connection.
getLangCode(String)	This method is restricted.
getLangInfo(String, String)	This method is restricted.
getNLSLang(String)	This method is restricted.
getSessionManager()	This method is restricted.

Member Summary	
getUrl()	Returns the JDBC URL.
getUser()	Returns the user name.
setCurrLang(String)	Sets the current language code.

Constructors

CZContext(String, String)

`public CZContext(java.lang.String url, java.lang.String dbOwner)`
Creates a new CZContext containing a connection established with the supplied database url.

Parameters:

`url` - The full url of the database.

`dbOwner` - the schema owner of the SellingPoint tables

CZContext(String, String, String, String)

`public CZContext(java.lang.String url, java.lang.String user, java.lang.String password, java.lang.String dbOwner)`
Creates a new CZContext containing a connection established with the supplied database url, user name, and password.

Parameters:

`url` - The full url of the database.

`user` - The database user name.

`password` - The user password.

`dbOwner` - the schema owner of the SellingPoint tables

Methods

getAppId(String)

`public int getAppId(java.lang.String applShtName)`
Returns the application ID for product configuration (CZ). The parameter signature is for compatibility with the AppsContext implementation.

getCurrLangCode()

```
public java.lang.String getCurrLangCode()
```
Returns the current language code.

getCurrLangInfo()

```
public oracle.apps.fnd.common.LangInfo getCurrLangInfo()
```
This method is restricted.

getDbOwner()

```
public java.lang.String getDbOwner()
```
Returns name of SellingPoint schema owner.

getJDBCConnection(Object)

```
public java.sql.Connection getJDBCConnection(java.lang.Object pThis)
```
Returns the JDBC connection. The parameter signature includes a reference to the caller to be compatible with the AppsContext implementation.

getLangCode(String)

```
public java.lang.String getLangCode(java.lang.String pNLSLang)
```
This method is restricted.

getLangInfo(String, String)

```
public oracle.apps.fnd.common.LangInfo getLangInfo(java.lang.String pLangCode,
java.lang.String pNLSLang)
```
This method is restricted.

getNLSLang(String)

```
public java.lang.String getNLSLang(java.lang.String pLangCode)
```
This method is restricted.

getSessionManager()

```
public oracle.apps.fnd.security.SessionManager getSessionManager()
```
This method is restricted.

getUrl()

```
public java.lang.String getUrl()
```
Returns the JDBC URL.

getUser()

```
public java.lang.String getUser()
```

Returns the user name.

setCurrLang(String)

```
public boolean setCurrLang(java.lang.String pLangCode)
```

Sets the current language code.

D

Package oracle.apps.cz.utilities

Description

Class Summary	
Classes	
NameValuePair	Provides a name-value pair object combination.
NameValuePairSet	Implements an object to hold a unique set of name value pairs

oracle.apps.cz.utilities NameValuePair

Syntax

```
public class NameValuePair extends java.lang.Object

java.lang.Object
  |
  +--oracle.apps.cz.utilities.NameValuePair
```

Description

Provides a name-value pair object combination. The name cannot be changed once created.

Member Summary

Constructors	
NameValuePair(String)	Constructs a name-value pair object without a value.
NameValuePair(String, Object)	Constructs a name-value pair object.
Methods	
getName()	Retrieve the name (key).
getValue()	Retrieve the value which can be null.
setValue(Object)	Replaces the value for this pair.

Inherited Member Summary

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

NameValuePair(String)

`public NameValuePair(java.lang.String key)`
Constructs a name-value pair object without a value. Key is always stored as a lowercase string, regardless of the case of the key parameter.

Parameters:

`key` - - The String name(key) for this pair.

NameValuePair(String, Object)

`public NameValuePair(java.lang.String key, java.lang.Object value)`
Constructs a name-value pair object. Key is always stored as a lowercase string, regardless of the case of the key parameter.

Parameters:

`key` - - The String name(key) for this pair. String cannot be null or have only whitespace.

`value` - - The Object for this pair.

Methods

getName()

`public java.lang.String getName()`
Retrieve the name (key). Will always be lowercase.

getValue()

`public java.lang.Object getValue()`
Retrieve the value which can be null.

setValue(Object)

`public void setValue(java.lang.Object value)`
Replaces the value for this pair.

Parameters:

`The` - Object for this pair which can be null.

oracle.apps.cz.utilities NameValuePairSet

Syntax

```
public class NameValuePairSet extends java.lang.Object

java.lang.Object
  |
  +--oracle.apps.cz.utilities.NameValuePairSet
```

Description

Implements an object to hold a unique set of name value pairs

Member Summary

Constructors	
NameValuePairSet()	
Methods	
Add(NameValuePair)	Add a name value pair object to the set
Add(String, Object)	Create a NameValuePair and add it to the set using the name and object.
getValueByName(String)	Gets value, which may be null, of the name/value pair identified by the "name" input.
iterator()	Returns the set of keys for the name value pairs
lookupPairByName(String)	Look up a name (key) in the set

Inherited Member Summary

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructors

NameValuePairSet()

```
public  NameValuePairSet()
```

Methods

Add(NameValuePair)

```
public void Add(NameValuePair nvp)
```
Add a name value pair object to the set

Parameters:

`nameValuePair` - - The NameValuePair object to add. The key will provide the hash.

Add(String, Object)

```
public void Add(java.lang.String name, java.lang.Object value)
```
Create a NameValuePair and add it to the set using the name and object.

Parameters:

`name` - - The String key of the pair. THe name will provide the hash identifier.

`value` - - The value object which can be null.

getValueByName(String)

```
public java.lang.Object getValueByName(java.lang.String name)
```
Gets value, which may be null, of the name/value pair identified by the "name" input. Returns null if pair does not exist.

Parameters:

`name` - the name string by which to look up the value

Returns:

the value associated with name

iterator()

```
public com.sun.java.util.collections.Iterator iterator()
```

Returns the set of keys for the name value pairs

lookupPairByName(String)

```
public NameValuePair lookupPairByName(java.lang.String name)
```

Look up a name (key) in the set

Parameters:

`name` - - The String to lookup

Returns:

The NameValuePair which can be null if the string is not in the set

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

X