30

Oracle XML DB Repository Events

The use of repository events to trigger application actions is deprecated in Oracle Database 21c (21.3). There is no replacement.

> **Note:**
>
> The Oracle XML DB Repository is deprecated with Oracle Database 23ai.

Repository events are events that can be used to trigger application actions. Repository events include repository changes, such as creating, deleting, locking, unlocking, rendering, linking, unlinking, placing under version control, checking in, checking out, unchecking out (reverting a checked out version), opening, and updating a resource. The deprecation of the use of repository events includes deprecation of the `DBMS_XEVENT` package, and the following subprogram groups:

You can use Oracle XML DB Repository to store and access data of any kind, in the form of repository resources. You can access repository data from any application. Sometimes your application needs to perform certain actions whenever a particular repository operation occurs. You can do this using repository events, but doing so is deprecated.

- Overview of Repository Events
  Your application can perform specific actions when events associated with repository operations occur. For example, you might want to perform a move-to-wastebasket or another backup action whenever a resource is deleted.
- Possible Repository Events
  Repository operations are associated with predefined events. Except for a rendering operation, events come in pre and post pairs.
- Repository Operations and Events
  The same repository event can occur with different Oracle XML DB Repository operations, and a given repository operation can produce more than one repository event.
- Repository Event Handler Considerations
  Some considerations are listed for defining handlers for Oracle XML DB Repository events.
- Configuration of Repository Events
  In a resource configuration file, you use element `event-listeners`, child of element `ResConfig`, to configure Oracle XML DB Repository event handling.

Related Topics

- Configuration of Oracle XML DB Repository
  Overall configuration of Oracle XML DB Repository applies to all repository resources. It does not include configuring parameters for handling events or managing XLink and XInclude processing. You use resource configuration files to configure resources.

Overview of Repository Events

Your application can perform specific actions when events associated with repository operations occur. For example, you might want to perform a move-to-wastebasket or another backup action whenever a resource is deleted.

Repository resource operations include creating, deleting, locking, unlocking, rendering, linking, unlinking, placing under version control, checking in, checking out, unchecking out (reverting a checked out version), opening, and updating.

- Repository Events: Use Cases
  Examples of cases where you can use repository events include moving a resource to a wastebasket and categorizing a resource based on its MIME type.
- Repository Events and Database Triggers
  You cannot use a database trigger to let your application react to repository operations. A given repository operation can consist of multiple database operations on multiple underlying, internal tables. Because these underlying tables are internal to Oracle XML DB, you cannot easily map them to specific repository operations.
- Repository Event Listeners and Event Handlers
  Each repository operation is associated with events. Your application can configure event listeners for particular resources or the entire repository. A listener can be restricted by a node-existence precondition. A listener is a set of PL/SQL or Java handlers, each of which processes a single event.
- Repository Event Configuration
  Repository event configuration involves defining resource configuration files and defining the order in which they are processed. The files define event listeners. Such configuration applies to events on individual resources and events for the repository as a whole.

Repository Events: Use Cases

Examples of cases where you can use repository events include moving a resource to a wastebasket and categorizing a resource based on its MIME type.

- *Wastebasket* – You can use an `UnLink` pre-event handler to effectively move a resource to a wastebasket instead of deleting it. Create a link in a wastebasket folder before removing the original link. The link in the wastebasket ensures that the resource is not removed. When you subsequently undelete a resource from the waste basket, the original link can be created again and the wastebasket link removed. The wastebasket link name can be different from the name of the link being removed because a resource at a certain path could be unlinked more than once from that path. The wastebasket would then have multiple links corresponding to that path, with different link properties and possibly pointing to different resources.
- *Categorization* – An application might categorize the resources it manages based on MIME type or other properties. It might keep track of GIF, text, and XML files by maintaining links to them from repository folders `/my-app/gif`, `/my-app/txt`, and `/my-app/xml`. Three post-event handlers could be used here: `LinkIn`, `UnlinkIn`, and `Update`. The `LinkIn` post-event handler would examine the resource and create a link in the appropriate category folder, if not already present. The `UnlinkIn` post-event handler would remove the link from the category folder. The `Update` post-event handler would effectively move the resource from one category folder to another if its category changes.

Repository Events and Database Triggers

You cannot use a database trigger to let your application react to repository operations. A given repository operation can consist of multiple database operations on multiple underlying, internal tables. Because these underlying tables are internal to Oracle XML DB, you cannot easily map them to specific repository operations.

For example, internal table `XDB$H_INDEX` might be updated by either a database update operation, if an ACL is changed, or a link operation. Even in cases where you might be able to accomplish the same thing using database triggers, you would not want to do that: A repository event is a higher-level abstraction than would be a set of database triggers on the underlying tables.

When a repository event occurs, information associated with the operation, such as the resource path used, can be passed to the corresponding event handler. Such information is not readily available using database triggers.

Repository events and database triggers can both be applied to XML data. You can use triggers on `XMLType` tables, for instance. However, if an `XMLType` table is also a repository table (hierarchy-enabled), then do not duplicate in an event handler any trigger code that applies to the table. Otherwise, that code is executed twice.

Repository Event Listeners and Event Handlers

Each repository operation is associated with events. Your application can configure event listeners for particular resources or the entire repository. A listener can be restricted by a node-existence precondition. A listener is a set of PL/SQL or Java handlers, each of which processes a single event.

A repository **event listener** is a Java class or a PL/SQL package or object type. It comprises a set of PL/SQL procedures or Java methods, each of which is called an **event handler**.

You associate a repository event listener with a resource by mapping a **resource configuration file** to the resource. You use PL/SQL package `DBMS_RESCONFIG` to manipulate resource configuration files, including associating them with the resources they configure. In particular, PL/SQL function `DBMS_RESCONFIG.getListeners` lists all event listeners for a given resource.

Repository Event Configuration

Repository event configuration involves defining resource configuration files and defining the order in which they are processed. The files define event listeners. Such configuration applies to events on individual resources and events for the repository as a whole.

A given resource can be configured by multiple resource configuration files. These are stored in a **resource configuration list**, and they are processed in list order. Events for the repository as a whole can also be configured by multiple resource configuration files. Similarly, the repository also has a resource configuration list. Event handling that is configured for the repository as a whole takes effect before any resource-specific event handling. All applicable repository-wide events are processed before any resource-specific events.

A given resource configuration file can define multiple event listeners for the resources it configures, and each event listener can define multiple event handlers.

Related Topics

- Configuration of Repository Events
  In a resource configuration file, you use element `event-listeners`, child of element `ResConfig`, to configure Oracle XML DB Repository event handling.

> **See Also:**
>
> Resource Configuration Files for general information about resource configuration and resource configuration lists

Possible Repository Events

Repository operations are associated with predefined events. Except for a rendering operation, events come in pre and post pairs.

A rendering operation is associated with a single repository event. Except for rendering, all repository operations are associated with one or more *pairs* of events.

For example, a resource creation is associated with three pairs of events, with the events occurring in this order:

1. Pre-creation event
2. Post-creation event
3. Pre-link-in event
4. Pre-link-to event
5. Post-link-to event
6. Post-link-in event

Table 30-1 lists the events associated with each repository operation. Their order is indicated in the handler columns.

Table 30-1 Predefined Repository Events

Repository Event Type	Description	Pre Handler Execution	Post Handler Execution
`Render`	A `Render` event occurs only for *file* resources, never for folder resources. Occurs when resource contents are accessed using any of the following: • Protocols • `XDBURIType` methods `getCLOB()`, `getBLOB()`, and `getXML()` Does *not* occur when resource contents are accessed using any of the following: • `SELECT ... FROM RESOURCE_VIEW` • `XDBURIType` method `getResource()` Only one handler for a `Render` event can set the rendered output. The first handler to call `setRenderStream` or `setRenderPath` controls the rendering.	N/A	N/A
`Create`	Occurs when a resource is created. The pre and post handlers executed are those defined on the folder of the new resource.	After pre-parsing, after validating the parent resource ACL and locks, and before assigning default values to undefined properties.	After inserting the resource into the system resource table.
`Delete`	Occurs when the resource and its contents are removed from disk, that is, when the resource `REF` count is zero (0).	After validating the resource ACL and locks and before removing the resource from disk.	After removing the resource and its contents from disk and after touching the parent folder to update its last modifier and modification time.
`Update`	Occurs when a resource is updated on disk.	After validating the resource ACL and locks and before updating the last modifier and modification time.	After writing the resource to disk.
`Lock`	Occurs during a lock-resource operation.	After validating the resource ACL and locks and before creating the new lock on the resource.	After creating the new lock.
`Unlock`	Occurs during an unlock-resource operation.	After validating the resource ACL and delete token.	After removing the lock.
`LinkIn`	Occurs before a `LinkTo` event during a link operation. The event target is the folder in which the link is created. Always accompanied by a `LinkTo` event.	After validating the resource ACL and locks and before creating the link.	After executing `LinkTo` post handler.
`LinkTo`	Occurs after a `LinkIn` event during a link operation. The event target is the resource that is the link destination.	After executing `LinkIn` pre handler and before creating the link.	After creating the link and after updating the last modifier and modification time of the parent folder.

Table 30-1 (Cont.) Predefined Repository Events

Repository Event Type	Description	Pre Handler Execution	Post Handler Execution
`UnLinkIn`	Occurs before an `UnlinkFrom` event during an unlink operation. Always accompanied by an `UnlinkFrom` event.	After validating the resource ACL and locks and before removing the link.	After executing the `UnlinkFrom` post handler.
`UnlinkFrom`	Occurs after an `UnlinkIn` event during an unlink operation.	After executing the `UnlinkIn` pre handler.	After removing the link.
`CheckIn`	Occurs during check-in of a resource.	After validating the resource ACL and locks and after verifying that the resource is version-controlled and has been checked out.	After checking in the resource.
`CheckOut`	Occurs during check-out of a resource.	After validating the resource ACL and locks and after verifying that the resource is version-controlled and is not already checked out.	After checking out the resource.
`UncheckOut`	Occurs during uncheck-out of a resource.	Before removing the record that the resource is checked out.	After unchecking out the resource.
`VersionControl`	Occurs when a version history is created for a resource. Note: You can call `DBMS_XDB_VERSION.MakeVersioned()` multiple times, but the version history is created only at the first call. Subsequent calls have no effect, so no `VersionControl` event occurs.	Before creating the version history for the resource.	After creating the first version of the resource.

For simplicity, the documentation generally treats both members of a repository event pair together, referring, for example, to the `LinkIn` event type as shorthand for the pre-link-in and post-link-in event types. For the same reason, the event-type names used here are derived from the Java interface `XDBRepositoryEventListener` by dropping the prefixes `handlePre` and `handlePost`.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for the PL/SQL repository event types

Repository Operations and Events

The same repository event can occur with different Oracle XML DB Repository operations, and a given repository operation can produce more than one repository event.

Table 30-2 lists the events that are associated with each repository operation. See Table 30-1 for the event order when multiple repository events occur for the same operations.

Table 30-2 Oracle XML DB Repository Operations and Events

Operation	Repository Events Occurring
Get binary representation of resource contents by path name	`Render`
Get XML representation of resource contents by path name	`Render`
Create or update a resource	If the resource already exists: `Create`, `LinkIn`, `LinkTo` If resource doe not yet exist (HTTP and FTP only): `Update`
Create a folder	`Create`, `LinkIn`, `LinkTo`
Create a link to an existing resource	`LinkIn` on the folder containing the link target, `LinkTo` on the target resource to be linked
Unlink a file resource or an empty folder resource. (Decrement `RefCount`, and if it becomes zero then delete the resource from disk.)	`UnlinkIn`, `UnlinkFrom`, and, if `RefCount` is zero, `Delete`
Forcibly delete a folder and its contents	Recursively produce events for unlinking a resource. Folder child resources are deleted recursively, then the folder is deleted.
Forcibly remove all links to a resource	Produce unlinking events for each link removed.
Update the contents, properties, or ACL of a resource by path name	`Update`
Put a depth-zero WebDAV lock on a resource	`Lock`
Remove a depth-zero WebDAV lock from a resource	`Lock`
Rename (move) a resource	`LinkIn` and `LinkTo` on the new location, `UnlinkIn` and `UnlinkFrom` on the old location
Copy a resource	`Create`, `LinkIn`, and `LinkTo` on the new location
Check out a resource	`CheckOut`
Check in a resource	`CheckIn`
Place a resource under version control	`VersionControl`
Uncheck out a resource	`UncheckOut`

All operations listed in Table 30-2 are atomic, except for these:

- Forced deletion of a folder and its contents
- Update of resource properties by path name using HTTP (only)
- Copy of a folder using HTTP (only)

> **See Also:**
>
> Table 21-3 for information on accessing resources using APIs and protocols

Repository Event Handler Considerations

Some considerations are listed for defining handlers for Oracle XML DB Repository events.

- In any handler: Do not use `COMMIT`, `ROLLBACK`, or data definition language (DDL) statements in a handler. Do not call PL/SQL functions or procedures, such as `DBMS_XMLSCHEMA.registerSchema`, that behave similarly to DDL statements. In a `Render` handler: Do not use data manipulation language (DML) statements.

  To work around these restrictions, a handler can use such statements inside an autonomous transaction, but it must ensure that lock conflicts cannot arise.
- In a `Render` handler, do not close an output stream. (You can append to a stream.)
- Do *not* use modifier methods from class `XDBResource` in a handler, unless it is a `Pre-Create` or `Pre-Update` handler. Do *not* use method `XDBResource.save()` in any handler.
- Oracle recommends that you develop only safe repository event handlers. In particular:
  - Write only resource properties that are in namespaces owned by your application, never in the `xdb` namespace.
  - Do not delete a resource while it is being created.
- A repository event handler is passed an `XDBRepositoryEvent` object, which exists only during the current SQL statement or protocol operation. You can use PL/SQL procedures and Java methods on this object to obtain information about the resource, the event, and the associated event handlers.
- When an event handler performs operations that cause other repository events to occur, those cascading events occur immediately. They are not queued to occur after the handlers for the current event are finished. Each event thus occurs in the context of its corresponding operation.
- Repository event handlers are called synchronously. They are executed in the process, session, and transaction context of the corresponding operation. However, handlers can use Oracle Database Advanced Queuing (AQ) to queue repository events that are then handled asynchronously by some other process.
- Because a repository event handler is executed in the transaction context of its corresponding operation, any locks acquired by that operation, or by other operations run previously in the transaction, are still active. An event handler must not start a separate session or transaction that tries to acquire such a lock. Otherwise, the handler hangs.
- Repository event handlers are called in the order that they appear in a resource configuration file. If preconditions are defined for a resource configuration, then only those handlers are called for which the precondition is satisfied.
- Although handlers are called in the order they are defined in a configuration file, avoid letting your code depend upon this. If the user who is current when a handler is invoked has privilege `write-config`, then the handler invocation order could be changed inside an executing handler.
- The entire list of handlers applicable to a given repository event occurrence is determined before any of the handlers is invoked. This means, in particular, that the precondition for each handler is evaluated before any handlers are invoked.
- The following considerations apply to *error handling* for repository events:
  - A pre-operation event handler is never invoked if access checks for the operation fail.

  - All handlers for a given event are checked before any of them are called. If any of them is not usable (for example, no longer exists), then *none* of them are called.
  - If an error is raised during event handling, then other, subsequent event handlers are not invoked for the same SQL statement or protocol operation. The current statement or operation is canceled and all of its changes are rolled back.
- The following considerations apply to *resource security* for repository events:
  - An event handler can have invoker's rights or definer rights. You specify the execution rights of a PL/SQL package when you create the package. You specify the execution rights of Java classes when you load them into the database using the `loadjava` utility. If you specify invoker's rights, but a given handler is not configured for invoker's rights, then an insufficient-privilege error is raised.
  - Within an event handler, the current user privileges, whether obtained by invoker or definer rights, are determined in detail for a given resource by its ACL. These privileges determine what the handler can do with the resource. For example, if the current user has privileges `read-properties` and `read-contents` for a particular resource, then an event handler can read that resource.
- The following considerations apply to repository events for *linking* and *unlinking*:
  - After creating a link to a resource, if you want any resource configuration files of the parent folder to also apply to the linked resource, then use procedure `DBMS_RESCONFIG.appendResConfig` to add the configuration files to the linked resource. You can invoke this procedure from a `Post-LinkTo` event handler for the linked resource.
  - After unlinking a resource, if you want to remove any such resource configuration files added when linking, then use procedure `DBMS_RESCONFIG.deleteResConfig` to remove them from the unlinked resource. You can invoke this procedure from a `Post-UnlinkFrom` event handler for the unlinked resource.
- Do not define handlers for events on folder `/sys/schemas` or on resources under this folder. Events do not occur for any such resources, so such event handlers are ignored. This implies that XML schema operations that affect the repository (registration, deletion, and so on) do not produce events.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information about PL/SQL functions and procedures for manipulating repository events
> - *Oracle Database XML Java API Reference*, classes `XDBRepositoryEvent` and `XDBEvent` for information about Java methods for manipulating repository events
> - Configuration of Repository Events for information about defining repository event handlers with invoker's rights

Configuration of Repository Events

In a resource configuration file, you use element `event-listeners`, child of element `ResConfig`, to configure Oracle XML DB Repository event handling.

You configure event treatment for Oracle XML DB Repository resources as you would configure any other treatment of repository resources — see "Configuring a Resource".

By default, repository events are enabled, but you can disable them by setting parameter `XML_DB_EVENTS` to `DISABLE`. To disable repository events at the session level, use the following SQL*Plus command. You must have role `XDBADMIN` to do this.

```
ALTER SESSION SET XML_DB_EVENTS = DISABLE;
```

To disable repository events at the system level, use the following SQL*Plus command, and then restart your database. Repository events are disabled for subsequent sessions. You must have privilege `ALTER SYSTEM` to do this.

```
ALTER SYSTEM SET XML_DB_EVENTS = DISABLE;
```

To enable repository events again, set the value of `XML_DB_EVENTS` to `ENABLE`.

A resource configuration file is an XML file that conforms to the XML schema `XDBResConfig.xsd`, which is accessible in Oracle XML DB Repository at path `/sys/schemas/PUBLIC/xmlns.oracle.com/xdb/XDBResConfig.xsd`. You use element `event-listeners`, child of element `ResConfig`, to configure repository event handling.

- Configuration Element event-listeners
  Each resource configuration file can have one `event-listeners` element, as a child of element `ResConfig`. This configures all event handling for the target resource. If the resource configuration file applies to the entire repository, not to a particular resource, then it defines event handling for all resources in the repository.
- Configuration Element listener
  Element `listener` is a child of element `event-listeners`, and it configures an individual repository event listener.
- Repository Events Configuration Examples
  Examples of configuring repository events are presented. Resource configuration files define Java and PL/SQL event listeners, with and without preconditions, respectively. An example categorizes resources according to MIME type. It includes PL/SQL code to create the resource configuration file. Examples implement listeners in Java and PL/SQL.

> **See Also:**
>
> Configuration of Oracle XML DB Repository for general information about configuring repository resources

Configuration Element event-listeners

Each resource configuration file can have one `event-listeners` element, as a child of element `ResConfig`. This configures all event handling for the target resource. If the resource configuration file applies to the entire repository, not to a particular resource, then it defines event handling for all resources in the repository.

Element `event-listeners` has the following optional attributes:

- `set-invoker` – Set this to `true` to if the resource configuration defines one or more repository event handlers to have invoker's rights. The default value is `false`, meaning that definer rights are used.

To define an invoker-rights repository event handler, you must have database role `XDB_SET_INVOKER`. This role is granted to `DBA`, but not to `XDBADMIN`. Role `XDB_SET_INVOKER` is checked only when a resource configuration file is created or updated. Only attribute `set-invoker`, not role `XDB_SET_INVOKER`, is checked at run time to ensure sufficient privilege.

> **See Also:**
>
> Repository Event Handler Considerations for information about insufficient-privilege errors

- `default-schema` – The default schema value, used for listeners for which no `schema` element is defined.
- `default-language` –The default language value, used for listeners for which no `language` element is defined.

Element `event-listeners` has a sequence of `listener` elements as children. These configure individual repository event listeners. The listeners are processed at run time in the order of the `listener` elements.

Configuration Element listener

Element `listener` is a child of element `event-listeners`, and it configures an individual repository event listener.

Each `listener` element has the following child elements. All of these are optional except `source`, and they can appear in any order (their order is irrelevant).

- `description` – Description of the listener.
- `schema` – Database schema for the Java or PL/SQL implementation of the repository event handlers. If neither this nor `default-schema` is defined, then an error is raised.
- `source` (required) – Name of the Java class, PL/SQL package, or object type that provides the handler methods. Java class names must be qualified with a package name. Use an empty `source` element to indicate that the repository event handlers are standalone PL/SQL stored procedures.
- `language` – Implementation language of the listener class (Java) or package (PL/SQL). If neither this nor `default-language` is defined, then an error is raised.
- `pre-condition` – Precondition to be met for any repository event handlers in this listener to be executed. This is identical to the `pre-condition` child of general resource configuration element `configuration` – see Configuration Elements defaultChildConfig and configuration.
- `events` – Sequence of unique repository event type names: `Render`, `Pre-Create`, and so on. Only handlers for repository events of these types are enabled for the listener. See Possible Repository Events for the list of possible repository event types. If element `events` is not present, then handlers of repository events of all types are enabled for the listener, which can be wasteful. Provide element `events` to eliminate handler invocations for insignificant repository events.

Repository Events Configuration Examples

Examples of configuring repository events are presented. Resource configuration files define Java and PL/SQL event listeners, with and without preconditions, respectively. An example categorizes resources according to MIME type. It includes PL/SQL code to create the resource configuration file. Examples implement listeners in Java and PL/SQL.

Example 30-1 shows the content of a resource configuration file that defines two event listeners. Each listener defines handlers for repository events of types `Post-LinkIn`, `Post-UnlinkIn`, and `Post-Update`. It defines preconditions, the default language (Java), and the default database schema.

The implementation of the handlers of the first listener is in Java class `oracle.cm.quota` defined in database schema `CM`. These handlers are invoked only for events on resources of `ContentType image/gif`.

The implementation of the handlers of the second listener is in Java class `oracle.ifs.quota` defined in database schema `IFS` (the default schema for this resource configuration file). These handlers are invoked only for events on resources of type `ifs-file` in namespace `http://foo.xsd`.

> **See Also:**
>
> Configuration Elements defaultChildConfig and configuration for a description of elements `defaultChildConfig` and `applicationData`

As a simple end-to-end illustration, suppose that an application needs to categorize the resources in folder `/public/res-app` according to their MIME types. It creates links to resources in folders `/public/app/XML-TXT`, `/public/app/IMG`, and `/public/app/FOLDER`, depending on whether the resource MIME type is `text/xml`, `image/gif`, or `application/octet-stream`, respectively. This is illustrated in Example 30-2, Example 30-3, and Example 30-5.

Example 30-2 shows the PL/SQL code to create the configuration file for this categorization illustration. It defines a single listener that handles events of types `Pre-UnlinkIn` and `Post-LinkIn`. It explicitly defines the language (PL/SQL) and database schema. No preconditions are defined.

Example 30-3 shows the PL/SQL code that implements the event handlers that are configured in Example 30-2. The `Post-LinkIn` event handler creates a link to the `eventObject` resource in one of the folders `/public/app/XML-TXT`, `/public/app/IMG`, and `/public/app/FOLDER`, depending on the resource MIME type. The `Pre-UnlinkIn` event handler deletes the links that are created by the `Post-LinkIn` event handler.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information about PL/SQL package `DBMS_XDBRESOURCE`
> - *Oracle Database PL/SQL Packages and Types Reference* for information about PL/SQL package `DBMS_XEVENT`
> - *Oracle Database PL/SQL Packages and Types Reference* for information about PL/SQL package `DBMS_XDB_REPOS`

A Java example would be configured the same as in Example 30-2, with the exception of these two lines, which would replace the elements with the same names in Example 30-2:

```
<source>category</source>
<language>Java</language>
```

Example 30-4 shows the Java code that implements the event handlers. The logic is identical to that in Example 30-3.

Example 30-5 demonstrates the invocation of the event handlers that are implemented in Example 30-3 or Example 30-4.

Example 30-1 Resource Configuration File for Java Event Listeners with Preconditions

```
<ResConfig xmlns="http://xmlns.oracle.com/xdb/XDBResConfig.xsd"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://xmlns.oracle.com/xdb/XDBResConfig.xsd
                               http://xmlns.oracle.com/xdb/XDBResConfig.xsd">
  <event-listeners default-language="Java" default-schema="IFS">
    <listener>
      <description>Category application</description>
      <schema>CM</schema>
      <source>oracle.cm.category</source>
      <events>
        <Post-LinkIn/>
        <Post-UnlinkIn/>
        <Post-Update/>
      </events>
      <pre-condition>
        <existsNode>
          <XPath>/Resource[ContentType="image/gif"]</XPath>
        </existsNode>
      </pre-condition>
    </listener>
    <listener>
      <description>Check quota</description>
      <source>oracle.ifs.quota</source>
      <events>
        <Post-LinkIn/>
        <Post-UnlinkIn/>
        <Post-Update/>
      </events>
      <pre-condition>
        <existsNode>
          <XPath>r:/Resource/[ns:type="ifs-file"]</XPath>
          <namespace>xmlns:r="http://xmlns.oracle.com/xdb/XDBResource.xsd"
                     xmlns:ns="http://foo.xsd"
          </namespace>
```

```
                    </existsNode>
                  </pre-condition>
                </listener>
              </event-listeners>
              <defaultChildConfig>
                <configuration>
                  <path>/sys/xdb/resconfig/user_rc.xml</path>
                </configuration>
              </defaultChildConfig>
              <applicationData>
                <foo:data xmlns:foo="http://foo.xsd">
                  <foo:item1>1234</foo:item1>
                </foo:data>
              </applicationData>
            </ResConfig>
```

Example 30-2 Resource Configuration File for PL/SQL Event Listeners with No Preconditions

```
DECLARE
  b BOOLEAN := FALSE;
BEGIN
  b := DBMS_XDB_REPOS.createFolder('/public/resconfig');
  b := DBMS_XDB_REPOS.createResource(
         '/public/resconfig/appcatg-rc1.xml',
         '<ResConfig xmlns="http://xmlns.oracle.com/xdb/XDBResConfig.xsd"
                     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                     xsi:schemaLocation="http://xmlns.oracle.com/xdb/XDBResConfig.xsd
                                         http://xmlns.oracle.com/xdb/XDBResConfig.xsd">
            <event-listeners>
              <listener>
                <description>Category application</description>
                <schema>APPCATGUSER1</schema>
                <source>APPCATG_EVT_PKG1</source>
                <language>PL/SQL</language>
                <events>
                  <Pre-UnlinkIn/>
                  <Post-LinkIn/>
                </events>
              </listener>
            </event-listeners>
            <defaultChildConfig>
              <configuration>
                <path>/public/resconfig/appcatg-rc1.xml</path>
              </configuration>
            </defaultChildConfig>
          </ResConfig>',
         'http://xmlns.oracle.com/xdb/XDBResConfig.xsd',
         'ResConfig');
END;
/
BEGIN
  DBMS_RESCONFIG.appendResConfig('/public/res-app',
                                 '/public/resconfig/appcatg-rc1.xml',
                                 DBMS_RESCONFIG.APPEND_RECURSIVE);
END;
/
```

Example 30-3 PL/SQL Code Implementing Event Listeners

```
CREATE OR REPLACE PACKAGE appcatg_evt_pkg1 AS

  PROCEDURE handlePreUnlinkIn (eventObject DBMS_XEVENT.XDBRepositoryEvent);
```

```
  PROCEDURE handlePostLinkIn (eventObject DBMS_XEVENT.XDBRepositoryEvent);

END;
/
CREATE OR REPLACE PACKAGE BODY appcatg_evt_pkg1 AS

  PROCEDURE handlePreUnlinkIn (eventObject DBMS_XEVENT.XDBRepositoryEvent) AS
      XDBResourceObj DBMS_XDBRESOURCE.XDBResource;
      ResDisplayName VARCHAR2(100);
      ResPath        VARCHAR2(1000);
      ResOwner       VARCHAR2(1000);
      ResDeletedBy   VARCHAR2(1000);
      XDBPathobj     DBMS_XEVENT.XDBPath;
      XDBEventobj    DBMS_XEVENT.XDBEvent;
      SeqChar        VARCHAR2(1000);
      LinkName       VARCHAR2(10000);
      ResType        VARCHAR2(100);
      LinkFolder     VARCHAR2(100);
    BEGIN
      XDBResourceObj := DBMS_XEVENT.getResource(eventObject);
      ResDisplayName := DBMS_XDBRESOURCE.getDisplayName(XDBResourceObj);
      ResOwner       := DBMS_XDBRESOURCE.getOwner(XDBResourceObj);
      XDBPathobj     := DBMS_XEVENT.getPath(eventObject);
      ResPath        := DBMS_XEVENT.getName(XDBPathObj);
      XDBEventobj    := DBMS_XEVENT.getXDBEvent(eventObject);
      ResDeletedBy   := DBMS_XEVENT.getCurrentUser(XDBEventobj);
      BEGIN
        SELECT XMLCast(
                 XMLQuery(
                   'declare namespace ns = "http://xmlns.oracle.com/xdb/XDBResource.xsd";
                    /ns:Resource/ns:ContentType'
                   PASSING r.RES RETURNING CONTENT) AS VARCHAR2(100))
          INTO ResType
          FROM PATH_VIEW r WHERE r.PATH=ResPath;
          EXCEPTION WHEN OTHERS THEN NULL;
      END;
      IF ResType = 'text/xml' THEN LinkFolder := '/public/app/XML-TXT/';
      END IF;
      IF ResType = 'image/gif' THEN LinkFolder := '/public/app/IMG/';
      END IF;
      IF ResType = 'application/octet-stream' THEN LinkFolder := '/public/app/FOLDER/';
      END IF;
      DBMS_XDB_REPOS.deleteResource(LinkFolder || ResDisplayName);
  END;

  PROCEDURE handlePostLinkIn (eventObject DBMS_XEVENT.XDBRepositoryEvent) AS
      XDBResourceObj DBMS_XDBRESOURCE.XDBResource;
      ResDisplayName VARCHAR2(100);
      ResPath        VARCHAR2(1000);
      ResOwner       VARCHAR2(1000);
      ResDeletedBy   VARCHAR2(1000);
      XDBPathobj     DBMS_XEVENT.XDBPath;
      XDBEventobj    DBMS_XEVENT.XDBEvent;
      SeqChar        VARCHAR2(1000);
      LinkName       VARCHAR2(10000);
      ResType        VARCHAR2(100);
      LinkFolder     VARCHAR2(100);
    BEGIN
      XDBResourceObj := DBMS_XEVENT.getResource(eventObject);
      ResDisplayName := DBMS_XDBRESOURCE.getDisplayName(XDBResourceObj);
      ResOwner       := DBMS_XDBRESOURCE.getOwner(XDBResourceObj);
      XDBPathobj     := DBMS_XEVENT.getPath(eventObject);
```

```
    ResPath        := DBMS_XEVENT.getName(XDBPathObj);
    XDBEventobj    := DBMS_XEVENT.getXDBEvent(eventObject);
    ResDeletedBy   := DBMS_XEVENT.getCurrentUser(XDBEventobj);
    SELECT XMLCast(
             XMLQuery(
               'declare namespace ns = "http://xmlns.oracle.com/xdb/XDBResource.xsd";
                /ns:Resource/ns:ContentType'
               PASSING r.RES RETURNING CONTENT) AS VARCHAR2(100))
      INTO ResType
      FROM PATH_VIEW r WHERE r.PATH=ResPath;
    IF ResType = 'text/xml' THEN LinkFolder := '/public/app/XML-TXT';
    END IF;
    IF ResType = 'image/gif' THEN LinkFolder := '/public/app/IMG';
    END IF;
    IF ResType = 'application/octet-stream' THEN LinkFolder := '/public/app/FOLDER';
    END IF;
    DBMS_XDB_REPOS.link(ResPath, LinkFolder, ResDisplayName);
  END;
END;
/
```

Example 30-4 Java Code Implementing Event Listeners

```
import oracle.xdb.event.*;
import oracle.xdb.spi.*;
import java.sql.*;
import java.io.*;
import java.net.*;
import oracle.jdbc.*;
import oracle.sql.*;
import oracle.xdb.XMLType;
import oracle.xdb.dom.*;

public class category
extends oracle.xdb.event.XDBBasicEventListener
{
  public Connection connectToDB() throws java.sql.SQLException
  {
    try
    {
      String strUrl="jdbc:oracle:kprb:";
      String strUname="appcatguser1";
      String strPwd="appcatguser1 ";
      Connection conn=null;
      OraclePreparedStatement stmt=null;
      DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
      conn = DriverManager.getConnection(strUrl, strUname, strPwd);
      return conn;
    }
    catch(Exception e1)
    {
      System.out.println("Exception in connectToDB java function");
      System.out.println("e1:" + e1.toString());
      return null;
    }
  }
  public void handlePostLinkIn (XDBRepositoryEvent eventObject)
  {
    XDBPath objXDBPath = null;
    String strPathName="";
    objXDBPath = eventObject.getPath();
    strPathName = objXDBPath.getName();
```

```
  XDBResource objXDBResource1;
  objXDBResource1 = eventObject.getResource();
  String textResDisplayName = objXDBResource1.getDisplayName();
  String resType = objXDBResource1.getContentType();
  String linkFolder="";
  System.out.println("resType" + resType+"sumit");
  System.out.println("strPathName:" + strPathName);
  System.out.println("textResDisplayName:" + textResDisplayName);
  if (resType.equals("text/xml")) linkFolder = "/public/app/XML-TXT/";
  else if (resType.equals("image/gif")) linkFolder = "/public/app/IMG/";
  else if (resType.equals("application/octet-stream"))
          linkFolder = "/public/app/FOLDER/";
  System.out.println("linkFolder:" + linkFolder);
  try
  {
    Connection con1 = connectToDB();
    OraclePreparedStatement stmt=null;
    stmt = (OraclePreparedStatement)con1.prepareStatement(
             "CALL DBMS_XDB_REPOS.link(?,?,?)");
    stmt.setString(1,strPathName);
    stmt.setString(2,linkFolder);
    stmt.setString(3,textResDisplayName);
    stmt.execute();
    stmt.close();
    con1.close();
  }
  catch(java.sql.SQLException ej1)
  {
    System.out.println("ej1:" + ej1.toString());
  }

/* Make sure the link is not in the category folders.
   Then check the target resource's mime type and create a link
   in the appropriate category folder. */
}
public void handlePreUnlinkIn (XDBRepositoryEvent eventObject)
{
  XDBPath objXDBPath = null;
  String strPathName="";
  objXDBPath = eventObject.getPath();
  strPathName = objXDBPath.getName();
  XDBResource objXDBResource1;
  objXDBResource1 = eventObject.getResource();
  String textResDisplayName = objXDBResource1.getDisplayName();
  String resType = objXDBResource1.getContentType();
  String linkFolder="";
  if (resType.equals("text/xml")) linkFolder = "/public/app/XML-TXT/";
  else if (resType.equals("image/gif")) linkFolder = "/public/app/IMG/";
  else if (resType.equals("application/octet-stream"))
          linkFolder = "/public/app/FOLDER/";
  try
  {
    Connection con1 = connectToDB();
    OraclePreparedStatement stmt=null;
    stmt = (OraclePreparedStatement)con1.prepareStatement(
             "CALL DBMS_XDB_REPOS.deleteResource(?)");
    stmt.setString(1,linkFolder+textResDisplayName);
    stmt.execute();
    stmt.close();
    con1.close();
  }
  catch(java.sql.SQLException ej1)
```

```
    {
      System.out.println("ej1:" + ej1.toString());
    }
  }
}
```

Example 30-5 Invoking Event Handlers

```
DECLARE
  ret BOOLEAN;
BEGIN
  ret := DBMS_XDB_REPOS.createResource('/public/res-app/res1.xml',
                                       '<name>TestForEventType-1</name>');
END;
/
DECLARE
  b BOOLEAN := FALSE;
  dummy_data CLOB := 'AAA';
BEGIN
  b := DBMS_XDB_REPOS.createResource('/public/res-app/res2.gif', dummy_data);
END;
/
DECLARE
  b BOOLEAN := FALSE;
  dummy_data CLOB := 'AAA';
BEGIN
  b := DBMS_XDB_REPOS.createFolder('/public/res-app/res-appfolder1');
END;

SELECT PATH FROM PATH_VIEW WHERE PATH LIKE '/public/app/%' ORDER BY PATH;

PATH
--------------------------------
/public/app/FOLDER
/public/app/FOLDER/res-appfolder1
/public/app/IMG
/public/app/IMG/res2.gif
/public/app/XML-TXT
/public/app/XML-TXT/res1.xml

6 rows selected.

-- Delete the /res-app resources. The /app resources are deleted also.
EXEC DBMS_XDB_REPOS.deleteResource('/public/res-app/res2.gif');
EXEC DBMS_XDB_REPOS.deleteResource('/public/res-app/res1.xml');
EXEC DBMS_XDB_REPOS.deleteResource('/public/res-app/res-appfolder1');

SELECT PATH FROM PATH_VIEW WHERE PATH LIKE '/public/app/%' ORDER BY PATH;

PATH
-------------------
/public/app/FOLDER
/public/app/IMG
/public/app/XML-TXT

3 rows selected.
```