9 XML Schema Evolution

This chapter describes how you can update your XML schema after you have registered it with Oracle XML DB. XML schema evolution is the process of updating your registered XML schema.

This chapter contains these topics:

Oracle XML DB supports the W3C XML Schema recommendation. XML instance documents that conform to an XML schema can be stored and retrieved using SQL and protocols such as FTP, HTTP(S), and WebDAV. In addition to specifying the structure of XML documents, XML schemas determine the mapping between XML and object-relational storage.

See Also: Chapter 6, "XML Schema Storage and Query: Basic"

Overview of XML Schema Evolution

A major challenge for developers using an XML schema with Oracle XML DB is how to deal with changes in the content or structure of XML documents. In some environments, the need for changes may be frequent or extensive, arising from new regulations, internal needs, or external opportunities. For example, new elements or attributes may need to be added to an XML schema definition, a data type may need to be modified, or certain minimum and maximum occurrence requirements may need to be relaxed or tightened.

In such cases, you need to "evolve" the XML schema so that new requirements are accommodated, while any existing instance documents (the data) remain valid (or can be made valid), and existing applications can continue to run.

If you do not care about any existing documents, you can of course simply drop the `XMLType` tables that are dependent on the XML schema, delete the old XML schema, and register the new XML schema at the same URL. In most cases, however, you need to keep the existing documents, possibly transforming them to accommodate the new XML schema.

Oracle XML DB supports two kinds of schema evolution:

- **Copy-based schema evolution**, in which all instance documents that conform to the schema are copied to a temporary location in the database, the old schema is deleted, the modified schema is registered, and the instance documents are inserted into their new locations from the temporary area

- **In-place schema evolution**, which does not require copying, deleting, and inserting existing data and thus is much faster than copy-based evolution, but which has restrictions that do not apply to copy-based evolution

  In general, in-place evolution is permitted if you are not changing the storage model and if the changes do not invalidate existing documents (that is, if existing documents are conformant with the new schema or can be made conformant with it). A more detailed explanation of restrictions and guidelines is presented in "Using In-Place XML Schema Evolution" on page 9-15.

Each approach has its own PL/SQL procedure: `DBMS_XMLSCHEMA.copyEvolve` for copy-based evolution and `DBMS_XMLSCHEMA.inPlaceEvolve` for in-place evolution. Separate sections in this chapter explain the use of each procedure, as well as guidelines for using its associated approach to schema evolution.

Using Copy-Based Schema Evolution

You perform copy-based XML schema evolution using procedure `copyEvolve` of PL/SQL package `DBMS_XMLSCHEMA`. Procedure `copyEvolve` copies existing instance documents to temporary `XMLType` tables to back them up, drops the old version of the XML schema (which also deletes the associated instance documents), registers the new version, and copies the backed-up instance documents to new `XMLType` tables. In case of a problem, the backup copies are restored—see "Rollback When Procedure DBMS_XMLSCHEMA.COPYEVOLVE Raises an Error" on page 9-9.

With procedure `copyEvolve` you can evolve your registered XML schema in such a way that existing XML instance documents continue to be valid.

Scenario for Copy-Based Evolution

Example 9–1 shows a *partial* listing of a revised version of the purchase-order XML schema of Example 3–8. See Example A–2 on page A-29 for the *complete* revised schema listing. Text that is in **bold** here is new or different from that in the original schema.

Example 9–1 Revised Purchase-Order XML Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xdb="http://xmlns.oracle.com/xdb"
           version="1.0">
  <xs:element
    name="PurchaseOrder" type="PurchaseOrderType"
    xdb:defaultTable="PURCHASEORDER"
    xdb:columnProps=
      "CONSTRAINT purchaseorder_pkey PRIMARY KEY (XMLDATA.reference),
       CONSTRAINT valid_email_address FOREIGN KEY (XMLDATA.userid)
         REFERENCES hr.employees (EMAIL)"
    xdb:tableProps=
      "VARRAY XMLDATA.ACTIONS.ACTION STORE AS TABLE ACTION_TABLE
        ((CONSTRAINT action_pkey PRIMARY KEY (NESTED_TABLE_ID, SYS_NC_ARRAY_INDEX$)))
       VARRAY XMLDATA.LINEITEMS.LINEITEM STORE AS TABLE LINEITEM_TABLE
        ((constraint LINEITEM_PKEY primary key (NESTED_TABLE_ID, SYS_NC_ARRAY_INDEX$)))
       lob (XMLDATA.NOTES) STORE AS (ENABLE STORAGE IN ROW STORAGE(INITIAL 4K NEXT 32K))"/>
  <xs:complexType name="PurchaseOrderType" xdb:SQLType="PURCHASEORDER_T">
    <xs:sequence>
      <xs:element name="Actions" type="ActionsType" xdb:SQLName="ACTIONS"/>
      <xs:element name="Reject" type="RejectionType" minOccurs="0" xdb:SQLName="REJECTION"/>
      <xs:element name="Requestor" type="RequestorType" xdb:SQLName="REQUESTOR"/>
      <xs:element name="User" type="UserType" xdb:SQLName="USERID"/>
      <xs:element name="CostCenter" type="CostCenterType" xdb:SQLName="COST_CENTER"/>
      <xs:element name="BillingAddress" type="AddressType" minOccurs="0"
```

```
                xdb:SQLName="BILLING_ADDRESS"/>
      <xs:element name="ShippingInstructions" type="ShippingInstructionsType"
                  xdb:SQLName="SHIPPING_INSTRUCTIONS"/>
      <xs:element name="SpecialInstructions" type="SpecialInstructionsType"
                  xdb:SQLName="SPECIAL_INSTRUCTIONS"/>
      <xs:element name="LineItems" type="LineItemsType" xdb:SQLName="LINEITEMS"/>
      <xs:element name="Notes" type="NotesType" minOccurs="0" xdb:SQLType="CLOB"
                  xdb:SQLName="NOTES"/>
    </xs:sequence>
    <xs:attribute name="Reference" type="ReferenceType" use="required" xdb:SQLName="REFERENCE"/>
    <xs:attribute name="DateCreated" type="xs:dateTime" use="required"
                  xdb:SQLType="TIMESTAMP WITH TIME ZONE"/>
  </xs:complexType>
  <xs:complexType name="LineItemsType" xdb:SQLType="LINEITEMS_T">
    <xs:sequence>
      <xs:element name="LineItem" type="LineItemType" maxOccurs="unbounded" xdb:SQLName="LINEITEM"
                  xdb:SQLCollType="LINEITEM_V"/>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="LineItemType" xdb:SQLType="LINEITEM_T">
    <xs:sequence>
      <xs:element name="Part" type="PartType" xdb:SQLName="PART"/>
      <xs:element name="Quantity" type="quantityType"/>
    </xs:sequence>
    <xs:attribute name="ItemNumber" type="xs:integer" xdb:SQLName="ITEMNUMBER"
                  xdb:SQLType="NUMBER"/>
  </xs:complexType>
  <xs:complexType name="PartType" xdb:SQLType="PART_T">
    <xs:simpleContent>
      <xs:extension base="UPCCodeType">
        <xs:attribute name="Description" type="DescriptionType" use="required"
                      xdb:SQLName="DESCRIPTION"/>
        <xs:attribute name="UnitCost" type="moneyType" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
  <xs:simpleType name="ReferenceType">
    <xs:restriction base="xs:string">
      <xs:minLength value="18"/>
      <xs:maxLength value="30"/>
    </xs:restriction>
  </xs:simpleType>

. . .

  <xs:complexType name="RejectionType" xdb:SQLType="REJECTION_T">
    <xs:all>
      <xs:element name="User" type="UserType" minOccurs="0" xdb:SQLName="REJECTED_BY"/>
      <xs:element name="Date" type="DateType" minOccurs="0" xdb:SQLName="DATE_REJECTED"/>
      <xs:element name="Comments" type="CommentsType" minOccurs="0" xdb:SQLName="REASON_REJECTED"/>
    </xs:all>
  </xs:complexType>
  <xs:complexType name="ShippingInstructionsType" xdb:SQLType="SHIPPING_INSTRUCTIONS_T">
    <xs:sequence>
      <xs:element name="name" type="NameType" minOccurs="0" xdb:SQLName="SHIP_TO_NAME"/>
      <xs:choice>
        <xs:element name="address" type="AddressType" minOccurs="0"/>
        <xs:element name="fullAddress" type="FullAddressType" minOccurs="0"
                    xdb:SQLName="SHIP_TO_ADDRESS"/>
      </xs:choice>
      <xs:element name="telephone" type="TelephoneType" minOccurs="0" xdb:SQLName="SHIP_TO_PHONE"/>
    </xs:sequence>
  </xs:complexType>

. . .
```

```
  <xs:simpleType name="NameType">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="20"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="FullAddressType">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="256"/>
    </xs:restriction>
  </xs:simpleType>

. . .

  <xs:simpleType name="DescriptionType">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="256"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="AddressType" xdb:SQLType="ADDRESS_T">
    <xs:sequence>
      <xs:element name="StreetLine1" type="StreetType"/>
      <xs:element name="StreetLine2" type="StreetType" minOccurs="0"/>
      <xs:element name="City" type="CityType"/>
      <xs:choice>
        <xs:sequence>
          <xs:element name="State" type="StateType"/>
          <xs:element name="ZipCode" type="ZipCodeType"/>
        </xs:sequence>
        <xs:sequence>
          <xs:element name="Province" type="ProvinceType"/>
          <xs:element name="PostCode" type="PostCodeType"/>
        </xs:sequence>
        <xs:sequence>
          <xs:element name="County" type="CountyType"/>
          <xs:element name="Postcode" type="PostCodeType"/>
        </xs:sequence>
      </xs:choice>
      <xs:element name="Country" type="CountryType"/>
    </xs:sequence>
  </xs:complexType>
  <xs:simpleType name="StreetType">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="128"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="CityType">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="64"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="StateType">
    <xs:restriction base="xs:string">
      <xs:minLength value="2"/>
      <xs:maxLength value="2"/>
      <xs:enumeration value="AK"/>
      <xs:enumeration value="AL"/>
      <xs:enumeration value="AR"/>

 . . . -- A value for each US state abbreviation

      <xs:enumeration value="WY"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="ZipCodeType">
    <xs:restriction base="xs:string">
      <xs:pattern value="\d{5}"/>
      <xs:pattern value="\d{5}-\d{4}"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="CountryType">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="64"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="CountyType">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="32"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="PostCodeType">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="12"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="ProvinceType">
    <xs:restriction base="xs:string">
      <xs:minLength value="2"/>
      <xs:maxLength value="2"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="NotesType">
    <xs:restriction base="xs:string">
      <xs:maxLength value="32767"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:simpleType name="UPCCodeType">
    <xs:restriction base="xs:string">
      <xs:minLength value="11"/>
      <xs:maxLength value="14"/>
      <xs:pattern value="\d{11}"/>
      <xs:pattern value="\d{12}"/>
      <xs:pattern value="\d{13}"/>
      <xs:pattern value="\d{14}"/>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```

copyEvolve Parameters and Errors

This is the signature of procedure `DBMS_XMLSCHEMA.copyEvolve`:

```
procedure copyEvolve(schemaURLs       IN XDB$STRING_LIST_T,
                     newSchemas       IN XMLSequenceType,
                     transforms       IN XMLSequenceType := NULL,
                     preserveOldDocs  IN BOOLEAN := FALSE,
                     mapTabName       IN VARCHAR2 := NULL,
                     generateTables   IN BOOLEAN := TRUE,
                     force            IN BOOLEAN := FALSE,
                     schemaOwners     IN XDB$STRING_LIST_T := NULL
                     parallelDegree   IN PLS_INTEGER := 0,
                     options          IN PLS_INTEGER := 0);
```

Table 9–1 describes the individual parameters. Table 9–2 describes the errors associated with the procedure.

Table 9–1 Parameters of Procedure DBMS_XMLSCHEMA.COPYEVOLVE

Parameter	Description
`schemaURLs`	Varray of URLs of XML schemas to be evolved (varray of `VARCHAR2(4000)`. This should include the dependent schemas as well. Unless the force parameter is `TRUE`, the URLs should be in the dependency order, that is, if URL A comes before URL B in the varray, then schema A should not be dependent on schema B but schema B may be dependent on schema A.
`newSchemas`	Varray of new XML schema documents (`XMLType` instances). Specify this in exactly the same order as the corresponding URLs. If no change is necessary in an XML schema, provide the unchanged schema.
`transforms`	Varray of XSL documents (`XMLType` instances) that will be applied to XML schema based documents to make them conform to the new schemas. Specify these in exactly the same order as the corresponding URLs. If no transformations are required, this parameter need not be specified.
`preserveOldDocs`	If this is `TRUE`, then the temporary tables holding old data are not dropped at the end of schema evolution. See also "Guidelines for Using copyEvolve".
`mapTabName`	Specifies the name of table that maps old `XMLType` table or column names to names of corresponding temporary tables.
`generateTables`	By default this parameter is `TRUE`; if this is `FALSE`, `XMLType` tables or columns will not be generated after registering new schemas. If this is `FALSE`, `preserveOldDocs` must be `TRUE` and `mapTabName` must not be `NULL`.
`force`	If this is `TRUE`, then errors during the registration of new schemas are ignored. If there are circular dependencies among the schemas, set this flag to `TRUE` to ensure that each schema is stored even though there may be errors in registration.
`schemaOwners`	Varray of names of schema owners. Specify these in exactly the same order as the corresponding URLs.
`parallelDegree`	Specifies the degree of parallelism to be used in a `PARALLEL` hint during the data-copy stage. If this is `0` (default value), a `PARALLEL` hint is absent from the data-copy statements.
`options`	Miscellaneous options. The only option is `COPYEVOLVE_BINARY_XML`, which means to register the new XML schemas for binary XML data and create the new tables or columns with binary XML as the storage model.

Table 9–2 Errors Associated with Procedure DBMS_XMLSCHEMA.COPYEVOLVE

Error Number and Message	Cause	Action
30942 XML Schema Evolution error for schema '<schema_url>' table "<owner_name>.<table_name>" column '<column_name>'	The given `XMLType` table or column that conforms to the given schema had errors during evolution. In the case of a table the column name will be empty. See also the more specific error that follows this.	Based on the schema, table, and column information in this error and the more specific error that follows, take corrective action.

Table 9–2 (Cont.) Errors Associated with Procedure DBMS_XMLSCHEMA.COPYEVOLVE

Error Number and Message	Cause	Action
30943 XML Schema '<schema_url>' is dependent on XML schema '<schema_url>'	Not all dependent XML schemas were specified or the schemas were not specified in dependency order, that is, if schema S1 is dependent on schema S, S must appear before S1.	Include the previously unspecified schema in the list of schemas or correct the order in which the schemas are specified. Then retry the operation.
30944 Error during rollback for XML schema '<schema_url>' table "<owner_name>.<table_name>" column '<column_name>'	The given `XMLType` table or column that conforms to the given schema had errors during a rollback of XML schema evolution. For a table the column name will be empty. See also the more specific error that follows this.	Based on the schema, table, and column information in this error and the more specific error that follows, take corrective action.
30945 Could not create mapping table '<table_name>'	A mapping table could not be created during XML schema evolution. See also the more specific error that follows this.	Ensure that a table with the given name does not exist and retry the operation.
30946 XML Schema Evolution warning: temporary tables not cleaned up	An error occurred after the schema was evolved while cleaning up temporary tables. The schema evolution was successful.	If you need to remove the temporary tables, use the mapping table to get the temporary table names and drop them.

Limitations When Using copyEvolve

Keep in mind the following limitations when you use procedure `DBMS_XMLSCHEMA.copyEvolve`:

- Indexes, triggers, constraints, row-level security (RLS) policies, and other metadata related to the `XMLType` tables that are dependent on the schemas are not preserved. These must be re-created after evolution.
- If top-level element names are changed, additional steps are required after `copyEvolve` finishes executing. See "Top-Level Element Name Changes" on page 9-8.
- Copy-based evolution cannot be used if there is a table with an object-type column that has an `XMLType` attribute that is dependent on any of the schemas to be evolved. For example, consider this table:

```
CREATE TYPE t1 AS OBJECT (n NUMBER, x XMLType);
CREATE TABLE tab1 (e NUMBER, o t1) XMLType
 COLUMN o.x XMLSchema "s1.xsd" ELEMENT "Employee";
```

  This assumes that an XML schema with a top-level element `Employee` has been registered under URL `s1.xsd`. It is not possible to evolve this XML schema, because table `tab1` with column `o` with `XMLType` attribute `x` is dependent on the XML schema. Note that although `copyEvolve` does not handle `XMLType` object attributes, it does raise an error in such cases.

Guidelines for Using copyEvolve

The following general guideline applies to using procedure `DBMS_XMLSCHEMA.copyEvolve`. The rest of this section describes specific guidelines that can also be appropriate in particular contexts.

1. Turn off the recycle bin, to prevent dropped tables from being copied to it:

   ```
   ALTER SESSION SET RECYCLEBIN=off;
   ```

2. Identify the XML schemas that are dependent on the XML schema that is to be evolved. You can acquire the URLs of the dependent XML schemas using the following query, where *`schema_to_be_evolved`* is the schema to be evolved, and *`owner_of_schema_to_be_evolved`* is its owner (database user).

   ```
   SELECT dxs.SCHEMA_URL, dxs.OWNER
       FROM DBA_DEPENDENCIES dd, DBA_XML_SCHEMAS dxs
       WHERE dd.REFERENCED_NAME = (SELECT INT_OBJNAME
                                     FROM DBA_XML_SCHEMAS
                                     WHERE SCHEMA_URL = schema_to_be_evolved
                                       AND OWNER = owner_of_schema_to_be_evolved)
         AND dxs.INT_OBJNAME = dd.NAME;
   ```

   In many cases, no changes are needed in the dependent XML schemas. But if the dependent XML schemas need to be changed, you must also prepare new versions of those XML schemas.

3. If the existing instance documents do not conform to the new XML schema, then you must provide an XSL style sheet that, when applied to an instance document, transforms it to conform to the new schema. You must do this for each XML schema identified in Step 2. The transformation must handle documents that conform to all top-level elements in the new XML schema.

4. Call procedure `DBMS_XMLSCHEMA.copyEvolve`, specifying the XML schema URLs, new schemas, and transformation style sheet.

Top-Level Element Name Changes

Procedure `DBMS_XMLSCHEMA.copyEvolve` assumes that top-level elements have not been dropped and that their names have not been changed in the new XML schemas. If there are such changes in your new XML schemas, then you can call procedure `copyEvolve` with parameter `generateTables` set to `FALSE` and parameter `preserveOldDocs` set to `TRUE`. In this way, new tables are not generated, and the temporary tables holding the old documents (backup copies) are not dropped at the end of the procedure. You can then store the old documents in whatever form is appropriate and drop the temporary tables. See "copyEvolve Parameters and Errors" on page 9-5 for more details on using these parameters.

User-Created Virtual Columns of Tables Other Than Default Tables

For tables that are not default tables, any virtual columns that you create are not re-created during copy-based evolution. If the columns are needed, then set parameter `preserveOldDocs` to `TRUE`, create the tables, and copy the old documents after procedure `copyEvolve` has finished.

Ensure that the XML Schema and Dependents Are Not Used by Concurrent Sessions

Ensure that the XML schema and its dependents are not used by any concurrent session during the XML schema evolution process. If other, concurrent sessions have shared locks on this schema at the beginning of the evolution process, then procedure DBMS_XMLSCHEMA.copyEvolve waits for these sessions to release the locks so that it can acquire an exclusive lock. However, this lock is released immediately to allow the rest of the process to continue.

Rollback When Procedure DBMS_XMLSCHEMA.COPYEVOLVE Raises an Error

Procedure DBMS_XMLSCHEMA.copyEvolve either completely succeeds or raises an error, in which case it attempts to roll back as much of the operation as possible. Evolving an XML schema involves many database DDL statements. When an error occurs, compensating DDL statements are executed to undo the effect of all steps executed to that point. If the old tables or schemas have been dropped, they are re-created, but any table, column, and storage properties and any auxiliary structures (such as indexes, triggers, constraints, and RLS policies) associated with the tables and columns are lost.

Failed Rollback From Insufficient Privileges

In certain cases you cannot roll back the copy-based evolution operation. For example, if table creation fails due to reasons not related to the new XML schema, then there is no way to roll back. An example is failure due to insufficient privileges. The temporary tables are not deleted even if preserveOldDocs is FALSE, so the data can be recovered. If the mapTabName parameter is null, the mapping table name is XDB$MAPTAB followed by a sequence number. The exact table name can be found using a query such as the following:

```
SELECT TABLE_NAME FROM USER_TABLES WHERE TABLE_NAME LIKE 'XDB$MAPTAB%';
```

Privileges Needed for XML Schema Evolution

Copy-based XML schema evolution may involve dropping or creating data types. Hence, you need type-related privileges such as DROP TYPE, CREATE TYPE, and ALTER TYPE.

You need privileges to delete and register the XML schemas involved in the evolution. You need all privileges on XMLType tables that conform to the schemas being evolved. For XMLType columns, the ALTER TABLE privilege is needed on corresponding tables. If there are schema-based XMLType tables or columns in other database schemas, you need privileges such as the following:

- CREATE ANY TABLE
- CREATE ANY INDEX
- SELECT ANY TABLE
- UPDATE ANY TABLE
- INSERT ANY TABLE
- DELETE ANY TABLE
- DROP ANY TABLE
- ALTER ANY TABLE
- DROP ANY INDEX

To avoid needing to grant all these privileges to the database- schema owner, Oracle recommends that a DBA perform the evolution if there are XML schema-based XMLType table or columns belonging to other database schemas.

Using a Style Sheet to Update Existing Instance Documents

After you modify a registered XML schema, you must update any existing XML instance documents that use the schema. You do this by applying an XSLT style sheet to each of the instance documents. The style sheet represents the difference between the old and new schemas.

Example 9–2 is a style sheet, in file `evolvePurchaseOrder.xsl`, that transforms existing purchase-order documents that use the old schema, so they will use the new schema instead.

Example 9–2 evolvePurchaseOrder.xsl: Style Sheet to Update Instance Documents

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xsl:output method="xml" encoding="UTF-8"/>
  <xsl:template match="/PurchaseOrder">
    <PurchaseOrder>
      <xsl:attribute name="xsi:noNamespaceSchemaLocation">
        http://localhost:8080/source/schemas/poSource/xsd/purchaseOrder.xsd
      </xsl:attribute>
      <xsl:for-each select="Reference">
        <xsl:attribute name="Reference">
          <xsl:value-of select="."/>
        </xsl:attribute>
      </xsl:for-each>
      <xsl:variable name="V264_394" select="'2004-01-01T12:00:00.000000-08:00'"/>
      <xsl:attribute name="DateCreated">
        <xsl:value-of select="$V264_394"/>
      </xsl:attribute>
      <xsl:for-each select="Actions">
        <Actions>
          <xsl:for-each select="Action">
            <Action>
              <xsl:for-each select="User">
                <User>
                  <xsl:value-of select="."/>
                </User>
              </xsl:for-each>
              <xsl:for-each select="Date">
                <Date>
                  <xsl:value-of select="."/>
                </Date>
              </xsl:for-each>
            </Action>
          </xsl:for-each>
        </Actions>
      </xsl:for-each>
      <xsl:for-each select="Reject">
        <Reject>
          <xsl:for-each select="User">
            <User>
              <xsl:value-of select="."/>
            </User>
          </xsl:for-each>
          <xsl:for-each select="Date">
            <Date>
              <xsl:value-of select="."/>
            </Date>
          </xsl:for-each>
          <xsl:for-each select="Comments">
            <Comments>
        <xsl:value-of select="."/>
      </Comments>
    </xsl:for-each>
  </Reject>
</xsl:for-each>
<xsl:for-each select="Requestor">
  <Requestor>
    <xsl:value-of select="."/>
  </Requestor>
</xsl:for-each>
<xsl:for-each select="User">
  <User>
    <xsl:value-of select="."/>
  </User>
</xsl:for-each>
<xsl:for-each select="CostCenter">
  <CostCenter>
    <xsl:value-of select="."/>
  </CostCenter>
</xsl:for-each>
<ShippingInstructions>
  <xsl:for-each select="ShippingInstructions">
    <xsl:for-each select="name">
      <name>
        <xsl:value-of select="."/>
      </name>
    </xsl:for-each>
  </xsl:for-each>
  <xsl:for-each select="ShippingInstructions">
    <xsl:for-each select="address">
      <fullAddress>
        <xsl:value-of select="."/>
      </fullAddress>
    </xsl:for-each>
  </xsl:for-each>
  <xsl:for-each select="ShippingInstructions">
    <xsl:for-each select="telephone">
      <telephone>
        <xsl:value-of select="."/>
      </telephone>
    </xsl:for-each>
  </xsl:for-each>
</ShippingInstructions>
<xsl:for-each select="SpecialInstructions">
  <SpecialInstructions>
    <xsl:value-of select="."/>
  </SpecialInstructions>
</xsl:for-each>
<xsl:for-each select="LineItems">
  <LineItems>
    <xsl:for-each select="LineItem">
      <xsl:variable name="V22" select="."/>
      <LineItem>
        <xsl:for-each select="@ItemNumber">
          <xsl:attribute name="ItemNumber">
            <xsl:value-of select="."/>
          </xsl:attribute>
        </xsl:for-each>
        <xsl:for-each select="$V22/Part">
          <xsl:variable name="V24" select="."/>
          <xsl:for-each select="@Id">
            <Part>
              <xsl:for-each select="$V22/Description">
                <xsl:attribute name="Description">
                  <xsl:value-of select="."/>
                </xsl:attribute>
```

```
                    </xsl:for-each>
                    <xsl:for-each select="$V24/@UnitPrice">
                      <xsl:attribute name="UnitCost">
                        <xsl:value-of select="."/>
                      </xsl:attribute>
                    </xsl:for-each>
                    <xsl:value-of select="."/>
                  </Part>
                </xsl:for-each>
              </xsl:for-each>
              <xsl:for-each select="$V22/Part">
                <xsl:for-each select="@Quantity">
                  <Quantity>
                    <xsl:value-of select="."/>
                  </Quantity>
                </xsl:for-each>
              </xsl:for-each>
            </LineItem>
          </xsl:for-each>
        </LineItems>
      </xsl:for-each>
    </PurchaseOrder>
  </xsl:template>
</xsl:stylesheet>
```

Examples of Using Procedure copyEvolve

Example 9–3 loads a revised XML schema and evolution XSL style sheet into Oracle XML DB Repository.

Example 9–3 Loading Revised XML Schema and XSL Style Sheet

```
DECLARE
  res BOOLEAN;
BEGIN
  res := DBMS_XDB.createResource(                -- Load revised XML schema
           '/source/schemas/poSource/revisedPurchaseOrder.xsd',
           bfilename('XMLDIR', 'revisedPurchaseOrder.xsd'),
           nls_charset_id('AL32UTF8'));
  res := DBMS_XDB.createResource(                -- Load revised XSL style sheet
           '/source/schemas/poSource/evolvePurchaseOrder.xsl',
           bfilename('XMLDIR', 'evolvePurchaseOrder.xsl'),
           nls_charset_id('AL32UTF8'));
END;
/
```

Example 9–4 shows how to use procedure `DBMS_XMLSCHEMA.copyEvolve` to evolve the XML schema `purchaseOrder.xsd` to `revisedPurchaseOrder.xsd` using the XSL style sheet `evolvePurchaseOrder.xsl`.

Example 9–4 Using DBMS_XMLSCHEMA.COPYEVOLVE to Update an XML Schema

```
BEGIN
  DBMS_XMLSCHEMA.copyEvolve(
    xdb$string_list_t('http://localhost:8080/source/schemas/poSource/xsd/purchaseOrder.xsd'),
    XMLSequenceType(XDBURIType('/source/schemas/poSource/revisedPurchaseOrder.xsd').getXML()),
    XMLSequenceType(XDBURIType('/source/schemas/poSource/evolvePurchaseOrder.xsl').getXML()));
END;

SELECT extract(object_value, '/PurchaseOrder/LineItems/LineItem[1]') LINE_ITEM
  FROM purchaseorder
  WHERE existsNode(object_value, '/PurchaseOrder[@Reference="SBELL-2003030912333601PDT"]') = 1
```

```
/

LINE_ITEM
----------------------------------------------------------------------------
<LineItem ItemNumber="1">
  <Part Description="A Night to Remember" UnitCost="39.95">715515009058</Part>
  <Quantity>2</Quantity>
</LineItem>
```

The same query would have produced the following result before the schema evolution:

```
LINE_ITEM
---------------------------------------------------------
<LineItem ItemNumber="1">
  <Description>A Night to Remember</Description>
  <Part Id="715515009058" UnitPrice="39.95" Quantity="2"/>
</LineItem>
```

Procedure `DBMS_XMLSCHEMA.copyEvolve` evolves registered XML schemas in such a way that existing instance documents continue to remain valid.

Caution: Before executing procedure `DBMS_XMLSCHEMA.copyEvolve`, always *back up* all registered XML schemas and all XML documents that conform to them. Procedure `copyEvolve` *deletes* all documents that conform to registered XML schemas.

First, procedure `copyEvolve` copies the data in XML schema-based `XMLType` tables and columns to temporary tables. It then drops the original tables and columns, and deletes the old XML schemas. After registering the new XML schemas, it creates `XMLType` tables and columns and populates them with data (unless parameter `GENTABLES` is `FALSE`) but it does not create any auxiliary structures such as indexes, constraints, triggers, and row-level security (RLS) policies. Procedure `copyEvolve` creates the tables and columns as follows:

- It creates default tables while registering the new schemas.
- It creates tables that are not default tables using a statement of the following form:

  ```
  CREATE TABLE <TABLE_NAME> OF XMLType OID '<OID>'
     XMLSCHEMA <SCHEMA_URL> ELEMENT <ELEMENT_NAME>
  ```

  where `<OID>` is the original OID of the table, before it was dropped.
- It adds `XMLType` columns using a statement of the following form:

  ```
  ALTER TABLE <Table_Name> ADD (<Column_Name> XMLType) XMLType column
    <Column_Name> xmlschema <Schema_Url> ELEMENT <Element_Name>
  ```

When a new XML schema is registered, types are generated if the registration of the corresponding old schema had generated types. If an XML schema was global before the evolution, then it will also be global after the evolution. Similarly, if an XML schema was local before the evolution, then it will also be local (owned by the same user) after the evolution.

You have the option to preserve the temporary tables that contain the old documents, by setting parameter `preserveOldDocs` to `TRUE`. All temporary tables are created in the database schema of the current user. For XMLType tables, the temporary table has the columns shown in Table 9–3.

Table 9–3 XML Schema Evolution: XMLType Table Temporary Table Columns

Name	Type	Comment
Data	CLOB	XML document from the old table, in CLOB format.
OID	RAW(16)	OID of the corresponding row in the old table.
ACLOID	RAW(16)	This column is present only if the old table is hierarchy enabled. ACLOID of corresponding row in old table.
OWNERID	RAW(16)	This column is present only if old table is hierarchy enabled. OWNERID of corresponding row in old table.

For XMLType columns, the temporary table has the columns shown in Table 9–4.

Table 9–4 XML Schema Evolution: XMLType Column Temporary Table Columns

Name	Type	Comment
Data	CLOB	XML document from the old column, in CLOB format.
RID	ROWID	ROWID of the corresponding row in the table containing this column.

Procedure copyEvolve stores information about the mapping from the old table or column name to the corresponding temporary table name in a separate table specified by parameter mapTabName. If preserveOldDocs is TRUE, then the mapTabName parameter must not be NULL, and it must not be the name of any existing table in the current database schema. Each row in the mapping table has information about one of the old tables/columns. Table 9–5 shows the mapping table columns.

Table 9–5 Procedure copyEvolve Mapping Table

Column Name	Column Type	Comment
SCHEMA_URL	VARCHAR2(700)	URL of the schema to which this table or column conforms.
SCHEMA_OWNER	VARCHAR2(30)	Owner of the schema.
ELEMENT_NAME	VARCHAR2(256)	Element to which this table or column conforms.
TABLE_NAME	VARCHAR2(65)	Qualified name of the table (<owner_name>.<table_name>).
TABLE_OID	RAW(16)	OID of table.
COLUMN_NAME	VARCHAR2(4000)	Name of the column (NULL for XMLType tables).
TEMP_TABNAME	VARCHAR2(30)	Name of temporary table that holds the data for this table or column.

You can avoid generating any tables or columns after registering the new XML schema by setting parameter GENTABLES to FALSE. If GENTABLES is FALSE, parameter PRESERVEOLDDOCS must be TRUE and parameter MAPTABNAME must not be NULL.

This ensures that the data in the old tables is not lost. This is useful if you do not want the tables to be created by the procedure, as described in section "copyEvolve Parameters and Errors" on page 9-5.

By default, it is assumed that all XML schemas are owned by the current user. If this is not true, then you must specify the owner of each XML schema in the `schemaOwners` parameter.

> **See Also:** *Oracle Database SQL Language Reference* for the complete description of `ALTER TABLE`

Using In-Place XML Schema Evolution

In-place XML schema evolution makes changes to an XML schema without requiring that existing data be copied, deleted, and reinserted. In-place evolution is thus much faster than copy-based evolution. However, in-place evolution also has several restrictions that do not apply to copy-based evolution.

You use procedure `DBMS_XMLSCHEMA.inPlaceEvolve` to perform in-place evolution. Using this procedure, you identify the changes to be made to an existing XML schema by specifying an XML schema-differences document, and you optionally specify flags to be applied to the evolution process.

In-place evolution constructs a new version of an XML schema by applying changes specified in a `diffXML` document, validates that new XML schema (against the XML schema for XML schemas), constructs DDL statements to evolve the disk structures used to store the XML instance documents associated with the XML schema, executes these DDL statements, and replaces the old version of the XML schema with the new, in that order. If the new version of the XML schema is not a valid schema, then in-place evolution fails.

Restrictions for In-Place XML Schema Evolution

Because in-place XML schema evolution avoids copying data, it does not permit arbitrary changes to an XML schema. This section describes why certain changes are not permitted. It does not list the supported XML schema changes; for that, see "Supported Operations for In-Place XML Schema Evolution" on page 9-17.

The primary restriction on using in-place evolution can be stated generally as a requirement that a given XML schema can be evolved in place in only a **backward-compatible** way. This means that any possible instance document that would validate against a given XML schema must also validate against a later (evolved) version of that XML schema. Note that this applies not only to existing instance documents; it applies to *all possible* conforming instance documents. For XML data that is stored as binary XML, backward compatibility also means that any XML schema annotations that affect binary XML treatment must not change during evolution. Backward compatibility is described in section "Backward-Compatibility Restrictions" on page 9-15.

In addition to this general backward-compatibility restriction, there are some other restrictions for in-place evolution. These are described in section "Other Restrictions on In-Place Evolution" on page 9-17.

Backward-Compatibility Restrictions

The restrictions described in this section ensure backward compatibility of an evolved XML schema, so that any possible instance documents that satisfy the old XML schema also satisfy the new schema.

Changes in On-Disk Data Layout Certain changes to an XML schema alter the layout of the associated instance documents on disk, and are therefore not permitted. This situation is more common when the storage layer is tightly integrated with information derived from the XML schema, as is the case for object-relational storage.

One such example is an XML schema, registered for object-relational storage mapping, that is evolved by splitting a complex type into two complex types. In Example 9–5, complex type `ShippingInstructionsType` is split into two complex types, `Person-Name` and `Contact-Info`, and the `ShippingInstructionsType` complex type is deleted.

Example 9–5 Splitting a Complex Type into Two Complex Types

These code excerpts show the definitions of the original `ShippingInstructionsType` type and the new `Person-Name` and `Contact-Info` types.

```
<complexType name="ShippingInstructionsType">
    <sequence>
        <element name="name"   type="NameType" minOccurs="0"/>
        <element name="address" type="AddressType" minOccurs="0"/>
        <element name="telephone" type="TelephoneType" minOccurs="0"/>
    </sequence>
</complexType>

<complexType name="Person-Name">
    <sequence>
        <element name="name" type="NameType" minOccurs="0"/>
    </sequence>
</complexType>

<complexType name="Contact-Info">
    <sequence>
        <element name="address" type="AddressType" minOccurs="0"/>
        <element name="telephone" type="TelephoneType" minOccurs="0"/>
    </sequence>
</complexType>
```

Even if this XML schema has no associated instance documents, and therefore no data copy is required, a change in the layout of existing tables is required to accommodate future instance documents.

Reordering of XML Schema Constructs You cannot use in-place evolution to reorder schema elements in a way that affects the DOM fidelity of instance documents. For example, you cannot change the order of elements within a `<sequence>` element in a complex type definition. As an example, if a complex type named `ShippingInstructionsType` requires that its child elements `name`, `address`, and `telephone` be in that order, you cannot use in-place evolution to change the order to `name`, `telephone`, and `address`.

Changes from a Collection to a Non-Collection You cannot use in-place evolution to change a collection to a non-collection. An example would be changing from a `maxOccurs` value greater than one to a `maxOccurs` value of one. You cannot use in-place evolution to delete an element from a complex type if the deletion requires that a collection be evolved to a non-collection.

Other Restrictions on In-Place Evolution

The restrictions on in-place XML schema evolution that are described in this section are necessary for reasons other than backward compatibility of the evolved XML schema.

Changes to Attributes in Namespace xdb Except for attribute `xdb:defaultTable`, you cannot use in-place evolution to modify any attributes in namespace `http://xmlns.oracle.com/xdb` (which has the predefined prefix `xdb`).

Changes from a Non-Collection to a Collection When XML data is stored object-relationally, you cannot use in-place evolution to change a non-collection object type to a collection object type. An example would be adding an element to a complex type with the element name matching the name of an element already present in the type (or in another type that is related to the first type through inheritance).

Supported Operations for In-Place XML Schema Evolution

This section describes operations that are supported for in-place schema evolution. This list of supported operations is not necessarily exhaustive. Some of the operations listed here are not permitted in specific contexts; these contexts are specified. In particular, some of the operations described here are not permitted for XML schemas that are used with binary XML.

- **Add an optional element to a complex type or group:** Always permitted. An example is the addition of the optional element `shipmethod` in the following complex type definition:

```
<xs:complexType name="ShippingInstructionsType">
    <xs:sequence>
        <xs:element name="name" type="NameType" minOccurs="0"/>
        <xs:element name="address" type="AddressType" minOccurs="0"/>
        <xs:element name="telephone" type="TelephoneType" minOccurs="0"/>
        <xs:element name = "shipmethod" type = "xs:string" minOccurs = "0"/>
    </xs:sequence>
</xs:complexType>
```

- **Add an optional attribute to a complex type or attribute group:** Always permitted. An example is the addition of the optional attribute `shipbydate` in the following complex type definition:

```
<xs:complexType name="ShippingInstructionsType">
    <xs:sequence>
        <xs:element name="name" type="NameType" minOccurs="0"/>
        <xs:element name="address" type="AddressType" minOccurs="0"/>
        <xs:element name="telephone" type="TelephoneType" minOccurs="0"/>
    </xs:sequence>
    <xs:attribute name="shipbydate" type="DateType" use="optional"/>
</xs:complexType>
```

- **Convert an element from simple type to complex type with simple content:** Supported only if the storage model is binary XML.
- **Modify the value attribute of an existing maxLength element:** Always permitted. The value can only be increased, not decreased.
- **Add an enumeration value:** You can add a new enumeration value only to the end of an enumeration list.

- **Add a global element:** Always permitted. An example is the addition of the global element `PurchaseOrderComment` in the following schema definition:

```
<xs:schema ...>
...
   <xs:element name="PurchaseOrderComment" type="string" xdb:defaultTable=""/>
..
</xs:schema>
```

- **Add a global attribute:** Always permitted.
- **Add or delete a global complex type:** Always permitted. An example is the addition of the global complex type `ComplexAddressType` in the following schema definition:

```
<xs:schema ...>
....
    <xs:complexType name="ComplexAddressType">
        <xs:sequence>
            <xs:element name="street" type="string"/>
            <xs:element name="city" type="string"/>
            <xs:element ref="zip" type="positiveInteger"/>
            <xs:element name="country"  type="string"/>
        </xs:sequence>
     </xs:complexType>
...
</xs:schema>
```

- **Add or delete a global simple type:** Always permitted.
- **Change the minOccurs attribute value:** The value of `minOccurs` can only be decreased.
- **Change the maxOccurs attribute value:** The value of `maxOccurs` can only be increased, and this is only possible for data stored as binary XML. That is, you cannot make any change to the `maxOccurs` attribute for data stored object-relationally.
- **Add or delete a global group or attributeGroup:** Always permitted. An example is the addition of an `Instructions` group in the following type definition:

```
<xsd:schema ...>
...
  <xsd:group name="Instructions">
    <xsd:sequence>
     <xsd:element name="ShippingInstructions" type="ShippingInstructionsType"/>
      <xsd:element name="SpecialInstructions" type=" SpecialInstructionsType"/>
    </xsd:sequence>
  </xsd:group>
...
</xsd:schema>
```

- **Change the xdb:defaultTable attribute value:** Always permitted. Changes are *not* permitted to any other attributes in the `xdb` namespace.
- **Add, modify, or delete a comment or processing instruction:** Always permitted.

Guidelines for Using In-Place XML Schema Evolution

The following guidelines apply to in-place XML-schema evolution:

- *Before* you perform an in-place XML-schema evolution:

- *Back up all existing data* (instance documents) for the XML schema that will be evolved.

  Caution: Make sure that you back up your data before performing in-place XML schema evolution, in case the result is not what you intended. There is *no rollback* possible after an in-place evolution. If any errors occur during evolution, or if you make a major mistake and need to redo the entire operation, you must be able to go back to the backup copy of your original data.

- *Perform a dry run* using trace only, that is, without actually evolving the XML schema or updating any instance documents, produce a trace of the update operations that would be performed during evolution. To do this, set the `flag` parameter value to only `INPLACE_TRACE`. Do not also use `INPLACE_EVOLVE`.

  After performing the dry run, examine the trace file, verifying that the listed DDL operations are in fact those that you intend.

- *After* you perform an in-place XML-schema evolution:

  If you are accessing the database using a client that caches data, or if you are not sure whether this is the case, then *restart your client*. Otherwise, the pre-evolution version of the XML schema might continue to be used locally, with unpredictable results.

  See Also: *Oracle Database Administrator's Guide* for information about using trace files

inPlaceEvolve Parameters

This is the signature of procedure `DBMS_XMLSCHEMA.inPlaceEvolve`:

```
procedure inPlaceEvolve(schemaURL IN VARCHAR2,
                        diffXML   IN XMLType,
                        flags     IN NUMBER);
```

Table 9–6 describes the individual parameters.

Table 9–6 Parameters of Procedure DBMS_XMLSCHEMA.INPLACEEVOLVE

Parameter	Description
`schemaURL`	URL of the XML schema to be evolved (`VARCHAR2`).
`diffXML`	XML document (`XMLType` instance) that conforms to the `xdiff` XML schema, and that specifies the changes to apply and the locations in the XML schema where the changes are to be applied. For information about how to create the document for this parameter, see "Creating the Document for the diffXML Parameter" on page 9-20.

Table 9–6 (Cont.) Parameters of Procedure DBMS_XMLSCHEMA.INPLACEEVOLVE

Parameter	Description
`flags`	A bit mask that controls the behavior of the procedure. You can set the following bit values in this mask independently, summing them to define the overall effect. The default `flags` value is 1 (bit 1 on, bit 2 off), meaning that in-place evolution is performed and no trace is written. ■ `INPLACE_EVOLVE` (value 1, meaning that bit 1 is on) – Perform in-place XML schema evolution: construct a new XML schema and validate it (against the XML schema for XML schemas); construct the DDL statements needed to evolve the instance-document disk structures, execute the DDL statements, and replace the old XML schema with the new. ■ `INPLACE_TRACE` (value 2, meaning that bit 2 is on) – Perform all steps necessary for in-place evolution, *except* executing the DDL statements and overwriting the old XML schema with the new, then write both the DDL statements and the new XML schema to a trace file. That is, each of the bits constructs the new XML schema, validates it, and determines the steps needed to evolve the disk structures underlying the instance documents. In addition: ■ Bit `INPLACE_EVOLVE` carries out those evolution steps and replaces the old XML schema with the new. ■ Bit `INPLACE_TRACE` saves the evolution steps and the new XML schema in a trace file (it does not carry out the evolution steps).

Procedure `DBMS_XMLSCHEMA.inPlaceEvolve` raises an error in the following cases:

- An XPath expression is invalid, or is syntactically correct but does not target a node in the XML schema.
- The `diffXML` document does not conform to the `xdiff` XML schema.
- The change makes the XML schema invalid or not well formed.
- A generated DDL statement (`CREATE TYPE`, `ALTER TYPE`, and so on) causes a problem when it is executed.

Creating the Document for the diffXML Parameter

The value of the `diffXML` parameter to procedure `DBMS_XMLSCHEMA.inPlaceEvolve` is an XML document (as an XMLType instance) that specifies the changes to be applied to an XML schema for in-place evolution. This `diffXML` document contains a sequence of operations that describe the changes between the old XML schema and the new (the intended evolution result). The changes specified by the `diffXML` document are applied in order.

You must create the XML document to be used for the `diffXML` parameter. To do this, you can use any of the following methods:

- The `XMLDiff` JavaBean (`oracle.xml.differ.XMLDiff`)
- The `xmldiff` command-line utility
- SQL function `XMLDiff`

The `diffXML` parameter document must conform to the `xdiff` XML schema.

The rest of this section presents examples of some operations in a document that conforms to the `xdiff` XML schema.

See Also:

- "xdiff.xsd: XML Schema for Comparing Schemas for In-Place Evolution" on page A-24
- *Oracle XML Developer's Kit Programmer's Guide* for information on using the `XMLDiff` JavaBean
- *Oracle XML Developer's Kit Programmer's Guide* for information on command-line utility `xmldiff`
- *Oracle Database SQL Language Reference* for information on SQL function `XMLDiff`

diffXML Operations and Examples

This section describes some operations that can be specified in the document for the `diffXML` document supplied to procedure `DBMS_XMLSCHEMA.inPlaceEvolve`. It presents an example XML document that conforms to the `xdiff` XML schema.

The `<append-node>` element is used for most of the supported changes, such as adding a new attribute to a complex type or appending a new element to a group.

The `<insert-node-before>` element specifies that a node of the given type should be inserted before the specified node. The `xpath` attribute specifies the location of the specified node and the `node-type` attribute specifies the type of node to be inserted. The node to be inserted is specified by the `<content>` child element. The `<insert-node-before>` element is mainly used for inserting comments and processing instructions, and for changing and adding add annotation elements.

The `<delete-node>` element specifies that the node with the given XPath (specified by the `xpath` attribute) should be deleted along with all its children. For example, you can use this element to delete comments and annotation elements. You can also use this element, in conjunction with `<append-node>` or `<insert-node-before>`, to make changes to an existing node.

Example 9–6 shows an XML document for the `diffXML` parameter that specifies the following changes:

- Delete complex type `PartType`.
- Add complex type `PartType` with a maximum length of 28.
- Add a comment before element `ShippingInstructions`.
- Add a required element `shipmethod` to element `ShippingInstructions`.

Example 9–6 diffXML Parameter Document

```
<xd:xdiff  xmlns="http://www.w3c.org/2001/XMLSchema"
           xmlns:xd="http://xmlns.oracle.com/xdb/xdiff.xsd"
           xmlns:xsi="http://www.w3c.org/2001/XMLSchema-Instance"
           xsi:schemaLocation="http://xmlns.oracle.com/xdb/xdiff.xsd
           http://xmlns.oracle.com/xdb/xdiff.xsd">
 <xd:delete-node xpath="/schema/complexType[@name=&quote;PartType&quote;]//maxLength/>
 <xd:append-node
  parent-xpath = "/schema/complexType[@name=&quote;PartType&quote;]//restriction"
  node-type = "element">
  <xd:content>
    <xs:maxLength value = "28"/>
  </xd:content>
 </xd:append-node>
 <xd:insert-node-before
```

```
  xpath="/schema/complexType[@name =&quote;ShippingInstructionsType&quote;]/sequence"
  node-type="comment">
  <xd:content>
    <!-- A type representing instructions for shipping -->
  </xd:content>
 </xd:insert-node-before>
 <xd:append-node
  parent-xpath="/schema/complexType[@name=&quote;ShippingInstructionsType&quote;]/sequence"
  node-type="element">
  <xd:content>
   <xs:element name = "shipmethod" type = "xs:string" minOccurs = "1"/>
  </xd:content>
 </xd:append-node>
</xd:xdiff>
```