Document Conversion Framework v. 1.0 User Guide

EMEA Data – JSON, Spatial and Graph Specialists

May 2025

I. INTRODUCTION

Oracle Database 23ai supports multiple different document formats, including two the most popular: XML and JSON.
Solution described in this document builds an abstraction layer over documents stored in the database, allowing to work on them, regardless of their formats.
Possible use cases can cover:

- Extend functionality of existing applications by adding new data sources/supported data format
- Multi-format document-oriented applications
- Modernization of existing workloads by easy migrating to more modern data models
- Expose on-the-fly XML document collections to MongoDB applications (by using Oracle API for MongoDB)

II. INSTALLATION AND DEINSTALLATION

Requirements

- Oracle Database 23ai
- A separate schema, which will own the framework with the following privileges/roles granted
    - DB_DEVELOPER_ROLE
    - CREATE ROLE

To install the framework there is need to execute **doc_conv_install.sql** script in a database schema, which will own the framework.

To remove the framework from the database there is need to execute **doc_conv_remove.sql** script being connected to the schema, which owns the framework.

During the installation process DC_ROLE database role is created. This role groups privileges allowing to work with converter.

III. COMPONENTS

Below table provides information about components of the conversion framework

Component	Type	Access	Description
DocComponent	PL/SQL Object Type	Internal	Root of inheritance hierarchy. Used only internally
DocElement	PL/SQL Object Type	Public	Abstraction layer used to store and manipulate document's data
DocAttribute	PL/SQL Object Type	Public	Class used to store and manipulate XML Attributes
CompArray	PL/SQL Nested Table	Internal	Used internally to store nested elements
AttrArray	PL/SQL Nested Table	Internal	Used internally to store XML attributes
DOC_UTL	PL/SQL Package	Internal	Used internally to process the document's data
DOC_CONV	PL/SQL Package	Public	High-level PL/SQL interface used to work on documents
DOC_PARAMS	Table	Internal	Stores parameters with their values

IV. USAGE

Basic and the most fundamental component is DocElement object type (class). Its constructors allows for creating documents from XML and JSON sources. Two main non-constructor methods allow for getting data in any format we need.

Example 1
Converting JSON data into XML document

```
SQL> set serveroutput on
SQL> declare
  2     jd JSON_ELEMENT_T := JSON_ELEMENT_T.parse('{field1:"data1",field2:"data2"}');
  3     ed DocElement := DocElement(jd);
  4     xd XMLType := ed.getAsXML;
  5  begin
  6     dbms_output.put_line(xd.getclobval);
  7  end;
  8* /
<_dc_object_><field1>data1</field1><field2>data2</field2></_dc_object_>


PL/SQL procedure successfully completed.
```

Example 2
Converting XML data into JSON document

```
SQL> set serveroutput on
SQL> declare
  2     xd XMLType:=
  3     XMLType('<rootElement><nested1>val1</nested1><nested2>val2</nested2></rootElement>');
  4     ed DocElement := DocElement(xd);
  5     jd JSON_ELEMENT_T := ed.getAsJSON;
  6  begin
  7     dbms_output.put_line(jd.to_String);
  8  end;
  9* /
{"rootElement":{"nested1":"val1","nested2":"val2"}}


PL/SQL procedure successfully completed.
```

Converter allows also for manipulating documents, for example adding or removing elements

Example 3:
Adding and removing document elements

```
SQL> set serveroutput on
SQL> declare
  2     xd XMLType;
  3     jd JSON_ELEMENT_T;
  4     ed DocElement := DocElement(
  5                           XMLType(
  6                           '<root><nested1>val1</nested1><nested2>val2</nested2></root>'));
  7
  8  begin
  9     jd := ed.getAsJSON;
 10     xd := ed.getAsXML;
 11     dbms_output.put_line(jd.to_String);
 12     dbms_output.put_line(xd.getclobval);
 13     ed.addElement('nested3','val3');
 14     jd := ed.getAsJSON;
 15     xd := ed.getAsXML;
 16     dbms_output.put_line(jd.to_String);
 17     dbms_output.put_line(xd.getclobval);
 18  end;
 19* /
{"root":{"nested1":"val1","nested2":"val2"}}
<root><nested1>val1</nested1><nested2>val2</nested2></root>
{"root":{"nested1":"val1","nested2":"val2","nested3":"val3"}}
<root><nested1>val1</nested1><nested2>val2</nested2><nested3>val3</nested3></root>
PL/SQL procedure successfully completed.
```

Appendix A contains full list of public DocElement methods with their descriptions

Additionally, converter contains DOC_CONV PL/SQL package allowing for using the tool in SQL statements. It can be used, for example, to easy create an XML collection table

Example 4
Using Converter to easy creation of XML collection table

```
SQL> create table dept_xml_table of XMLType
  2  as
  3  select doc_conv.json2xml(JSON{*}) department
  4* from hr.departments;

Table DEPT_XML_TABLE created.

SQL> select *
  2* from dept_xml_table;

SYS_NC_ROWINFO$
------------------------------------------------------------
<_dc_object_>
  <DEPARTMENT_ID>10</DEPARTMENT_ID>
  <DEPARTMENT_NAME>Administration</DEPARTMENT_NAME>
  <MANAGER_ID>200</MANAGER_ID>
  <LOCATION_ID>1700</LOCATION_ID>
</_dc_object_>
…
```

XML Tables can be also easily exposed as JSON collection views, available for MongoDB applications

Example 5
Using Converter to expose XML data to MongoDB applications

```
SQL> create or replace json collection view dept_json_view
  2  as
  3  select doc_conv.xml2json(value(d)) data
  4* from dept_xml_table d;

JSON collection view DEPT_JSON_VIEW created.

SQL> select *
  2* from dept_json_view;

DATA
_____________________________________________________________________________________________
{"DEPARTMENT_ID":10,"DEPARTMENT_NAME":"Administration","MANAGER_ID":200,"LOCATION_ID":1700}
{"DEPARTMENT_ID":20,"DEPARTMENT_NAME":"Marketing","MANAGER_ID":201,"LOCATION_ID":1800}
…

$ mongosh 'mongodb://….'

Current Mongosh Log ID: 689ca13016332be25fde7cdd
Connecting to:          mongodb://…
Using MongoDB:          4.2.14
Using Mongosh:          2.5.6

For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

oradev> show collections
…
DEPT_JSON_VIEW  [view]
…

oradev> db.DEPT_JSON_VIEW.find()
[
  {
    DEPARTMENT_ID: 10,
    DEPARTMENT_NAME: 'Administration',
    MANAGER_ID: 200,
    LOCATION_ID: 1700
  },
…

oradev> db.dept_json_view.find({"DEPARTMENT_ID":10})
[
  {
    PHONE_NUMBER: '1.515.555.0165',
    JOB_ID: 'AD_ASST',
    SALARY: 4400,
    COMMISSION_PCT: null,
    FIRST_NAME: 'Jennifer',
    EMPLOYEE_ID: 200,
    EMAIL: 'JWHALEN',
    LAST_NAME: 'Whalen',
    MANAGER_ID: 101,
    DEPARTMENT_ID: 10,
    HIRE_DATE: '2013-09-17T00:00:00'
  }
]
```

It is also possible to create arrays of document being results of SQL queries

Example 6
Using converter to create array of documents being results of SQL query

```
SQL> set serveroutput on
SQL> declare
  2     de DocElement := DocElement.getArray('select * from hr.employees '||
  3                                          'where department_id = 90','EMP_90','EMPLOYEE');
  4     dx XMLType;
  5     dj JSON_ELEMENT_T;
  6  begin
  7     dj := de.getAsJSON;
  8     dx := de.getAsXML;
  9     dbms_output.put_line('------- JSON DATA -------');
 10     dbms_output.put_line(dj.to_String);
 11     dbms_output.put_line('---- END OF JSON DATA ---');
 12     dbms_output.put_line('------- XML DATA --------');
 13     dbms_output.put_line(dx.getClobVal);
 14     dbms_output.put_line('---- END OF XML DATA ----');
 15  end;
 16* /
------- JSON DATA -------
{"EMP_90":[{"PHONE_NUMBER":"1.515.555.0100","JOB_ID":"AD_PRES","SALARY":24000,"COMMISSION_PCT"
:null,"FIRST_NAME":"Steven","EMPLOYEE_ID":100,"EMAIL":"SKING","LAST_NAME":"King","MANAGER_ID":
null,"DEPARTMENT_ID":90,"HIRE_DATE":"2013-06-
17T00:00:00"},{"PHONE_NUMBER":"1.515.555.0101","JOB_ID":"AD_VP","SALARY":17000,"COMMISSION_PCT
":null,"FIRST_NAME":"Neena","EMPLOYEE_ID":101,"EMAIL":"NYANG","LAST_NAME":"Yang","MANAGER_ID":
100,"DEPARTMENT_ID":90,"HIRE_DATE":"2015-09-
21T00:00:00"},{"PHONE_NUMBER":"1.515.555.0102","JOB_ID":"AD_VP","SALARY":17000,"COMMISSION_PCT
":null,"FIRST_NAME":"Lex","EMPLOYEE_ID":102,"EMAIL":"LGARCIA","LAST_NAME":"Garcia","MANAGER_ID
":100,"DEPARTMENT_ID":90,"HIRE_DATE":"2011-01-13T00:00:00"}]}
---- END OF JSON DATA ---
------- XML DATA --------
<EMP_90><EMPLOYEE><PHONE_NUMBER>1.515.555.0100</PHONE_NUMBER><JOB_ID>AD_PRES</JOB_ID><SALARY>2
4000</SALARY><COMMISSION_PCT></COMMISSION_PCT><FIRST_NAME>Steven</FIRST_NAME><EMPLOYEE_ID>100<
/EMPLOYEE_ID><EMAIL>SKING</EMAIL><LAST_NAME>King</LAST_NAME><MANAGER_ID></MANAGER_ID><DEPARTME
NT_ID>90</DEPARTMENT_ID><HIRE_DATE>2013-06-
17T00:00:00</HIRE_DATE></EMPLOYEE><EMPLOYEE><PHONE_NUMBER>1.515.555.0101</PHONE_NUMBER><JOB_ID
>AD_VP</JOB_ID><SALARY>17000</SALARY><COMMISSION_PCT></COMMISSION_PCT><FIRST_NAME>Neena</FIRST
_NAME><EMPLOYEE_ID>101</EMPLOYEE_ID><EMAIL>NYANG</EMAIL><LAST_NAME>Yang</LAST_NAME><MANAGER_ID
>100</MANAGER_ID><DEPARTMENT_ID>90</DEPARTMENT_ID><HIRE_DATE>2015-09-
21T00:00:00</HIRE_DATE></EMPLOYEE><EMPLOYEE><PHONE_NUMBER>1.515.555.0102</PHONE_NUMBER><JOB_ID
>AD_VP</JOB_ID><SALARY>17000</SALARY><COMMISSION_PCT></COMMISSION_PCT><FIRST_NAME>Lex</FIRST_N
AME><EMPLOYEE_ID>102</EMPLOYEE_ID><EMAIL>LGARCIA</EMAIL><LAST_NAME>Garcia</LAST_NAME><MANAGER_
ID>100</MANAGER_ID><DEPARTMENT_ID>90</DEPARTMENT_ID><HIRE_DATE>2011-01-
13T00:00:00</HIRE_DATE></EMPLOYEE></EMP_90>
---- END OF XML DATA ----
```

Appendix B contains full list of content of DOC_CONV PL/SQL package

V. Appendix A: DocElement object type description

DocElement is the fundamental class used in the converter. It can be used directly and is also internally used by DOC_CONV PL/SQL package.

Note
This appendix describes only methods/constructors, which can be used programmatically directly. All the components not described in this appendix should not be used – they have been implemented for internal purposes.

Constructors

DocElement	Creates an empty DocElement object
DocElement(eVal clob)	Creates DocElement with single value set to eVal
DocElement(eKey clob, eVal clob)	Creates simple DocElement object with single key:value pair. Key is set to eKey, value to eVal
DocElement(xDoc XMLType)	Creates a DocElement object, which represents xDoc XML document
DocElement(jDoc JSON_ELEMENT_T)	Creates a DocElement object, which represents jDoc JSON document

High level methods, which can be used directly

member function getAsXML return XMLType	returns XML representation of the document stored in a DocElement object
member function getAsJSON return JSON_ELEMENT_T	returns JSON representation of the document stored in a DocElement object
member function getNoOfElements return integer	returns number of nested elements in a DocElement object
member function getElement(eKey clob) return DocElement	returns nested element with eKey name. If such element does not exist, returns null
member procedure addElement(eKey clob, eVal clob, nest boolean := false)	adds to a DocElement object new element parameters: eKey : name of the element eVal : value of the element nest boolean : used internally
member procedure delElement (eKey clob)	deletes from a DocElement object a nested element with eKey name
member function hasAttrs return Boolean	returns true if DocElement object has XML attributes, false otherwise
member function getNoOfAttributes	returns number of XML attributes stored in the DocElement object
member procedure addAttr(aName clob, aVal clob)	adds new XML attribute to the DocElement object; its name is set to aName, its value to aVal
member procedure delAttr(aName clob)	deletes from the DocElement object attribute with aName name
member procedure delAttrs	deletes all attributes from DocElement object
member function hasComments return boolean	returns true if the DocElement object has XML comments, false otherwise
member procedure addComment(comment clob)	adds new comment to the DocElement object
member procedure delComments	deletes all comments from the DocElement object
member function hasCData return boolean	returns true if the DocElement has CDATA XML element, false otherwise
member procedure addCData(ncdata clob)	adds CDATA data to the DocElement object
member procedure delCData	deletes CDATA XML element from the DocElement object
member procedure attr2element(ekey clob)	converts XML attribute identified by eKey to document element
member procedure element2attr(eKey clob)	Converts document element identified by eKey to XML attribute
member procedure setParameter(pName varchar2, pValue varchar2)	sets the parameter pName to the value of pValue at the level of specific object

member function toString(fmt integer) return clob	returns clob containing the document; fmt parameter is used to provide information about output format - it can be set to DOC_CONV.FMT_XML or DOC_CONV.FMT_JSON
member procedure aggregate(tName varchar2, tKey clob)	adds to given DocElement object nested array conaining set of child records taken from table with tName name; tKey parameter is used to provide the name of join key

VI. Appendix B: DOC_CONV PL/SQL package

Constants

FMT_XML	Used in some programs (example: DocElement.toString) to provide required information about the format
FMT_JSON	Used in some programs (example: DocElement.toString) to provide required information about the format

Subprograms

function xml2json_element_t(xDoc XMLType) return JSON_ELEMENT_T	converts XMLType value into JSON_ELEMENT_T value
function xml2json(xDoc XMLType) return JSON	converts XMLType value into JSON value
function json_element_t2xml (jDoc JSON_ELEMENT_T) return XMLType	converts JSON_ELEMENT_T value into XMLType value
function json2xml(jDoc JSON) return XMLType	converts JSON value into XMLType value
function get_param(p_name varchar2) return varchar2	returns value of a parameter; its name is provided in p_name parameter
procedure set_param(p_n varchar2,p_val varchar2,permanent boolean := false)	sets parameter p_n to value of p_val; if permanent is equal to TRUE, then this new value is written into parameters table, otherwise the new value will be used only in current session

VII. Appendix B: List of demos in /demo directory

Note

Current release of Document Conversion API requires running demo scripts in the schema, which owns the converter.
It is planned in future release to remove this limitation

- demo01.sql
  this demo demonstrates basic functionality of Document Conversion API
  - prerequisites: HR sample schema
  - creation of XML collection table
  - creation of JSON collection view on top of XML Collection table
  - exposing XML collection table to Oracle API for MongoDB

- demo02.sql
  basic document manipulation
  - prerequisites: HR sample schema
  - setting the new root element of a document
  - deleting and adding element from/to document

- demo03.sql
  operation on arrays of values/nested elements
  - prerequsites: HR sample schema
  - dynamic creation of an array of documents returned by a SQL query
  - dynamic creation of document reflecting parent-child relationship between relational tables (DEPARTMENTS – EMPLOYEES)

- demo04.sql
  this demo demonstrates XML comments and XML attributes support
  - prerequisites: HR sample schema
  - adding and removing comments and attributes
  - representation of XML comments and attributes in JSON documents (supplemental elements)
  - conversion of an XML attribute to a new element and an element to a new XML attribute