DAD checker release notes

Release 1.3

Table of contents

I. Introduction

The Document Access Definition (DAD) file

The Document Access Definition (DAD) file is an XML file that is supported in DB2 XML Extender. The DAD associates XML documents to DB2 database tables through two alternative access and storage methods: XML columns and XML collections.
XML collection enables the decomposition (storage) of data from XML documents into DB2 relational tables and the compostion of XML documents from relational data. The DAD file is used to specify how XML documents are to be stored or composed.

The DAD checker can only be used to verify the validity of DAD files that use the XML collection storage method. In the DAD file, a mapping scheme specifies the relationship between the tables and the structure of the XML document. See the *XML Extender Administration and Programming Guide* for more information on the XML Extender and the creation of DAD files for XML collections.

Purpose and operation of the DAD checker

Much like Document Type Descriptions (DTDs) are used to validate the syntax of XML documents, the DAD checker is used to ensure that a DAD file is semantically correct. This validation can take place without connecting to a database. Use of the DAD checker can help minimize the number of errors that occur when submitting the file to the XML Extender for processing.

The DAD checker is a Java application that is called from the command line. When invoked, it produces a set of two output files that contain errors, warnings and success indicators. The two files are equivalent; one is a plain text file that you use to check for errors or warnings; the other is an XML file, 'errorsOutput.xml', which makes the results of the DAD checker application available to other applications (the output XML file is not provided when using the -xml option, see Part III). The name of the output text file is user-defined. If no name is specified, the standard output is used.

II. Installing the DAD checker

After you download the DADChecker.zip file, extract all files into a directory of your choice. Make sure you have a JRE or JDK Version 1.3.1 or later installed on your system.

III. Running the DAD checker

The DAD checker is a Java program, that can run on JDK version 1.3.1. and later. To run the DAD checker:

1. From a command line change to the **/bin** subdirectory in the directory where you installed the DAD checker
2. Execute the following commands:

```
setcp

java dadchecker.Check_dad_xml [-dad | -xml] [-all] [-dup dupName] [-enc encoding] [-dtd dtdPath]
      [-doc xmlDocument] [-db driver dbURL userID password] [-out outputFile] fileToCheck
```

Parameters

Here are the different parameters:

- ***fileToCheck***: specifies the path of the file to be checked. Required. It must be the final parameter.

- **-dad**: indicates that the file that is to be checked is a DAD file. This is the default option.

- **-xml**: indicates that the file that is to be checked is an XML document rather than a DAD file.
  For this kind of check, only the output text file is generated: no output XML file is provided.
  For large XML documents (several MB), the Java Virtual Machine might run out of memory, producing a java.lang.OutOfMemoryError exception. In such cases, the -Xmx option can be used to allocate more memory to the Java Virtual Machine. Refer to the JDK documentation for details.

- **-all**: the output will show all occurrences of tags that are in error.

- **-dup** *dupName*: For DAD files, only the duplicate tags whose *name* attribute values are *dupName* are displayed. For XML documents, only the duplicate tags or attributes whose names are *dupName* are displayed.

- **-enc** *encoding*: specifies the encoding of the file that is to be checked. *encoding* can be either a MIME encoding or a Java encoding. This option is used when the encoding declaration in the XML file does not match its actual encoding. It allows you to specify the actual encoding of the document and specifies that the XML parser should ignore the encoding declaration in the file.

- **-dtd** *dtdPath*: overrides the DTD declared in the XML document. The *dtdPath* parameter specifies the path for a DTD in the file system to be used to validate the DAD. The XML parser will then use *dtdPath* to locate the DTD instead of the DTD declared in the file.
  The validation against the DTD is dynamic: it will only be done if a DTD is declared in the file. Consequently, using this option has no effect when a DTD is not declared in the file.

- **-doc** *xmlDocument*: specifies an XML document whose structure is to be checked against the DAD's structure. The DAD checker will check that the attributes and leaf tags of *xmlDocument* are mapped in the DAD. This option cannot be used when checking an XML document with the -xml option.

- **-db** *driver dbURL userID password*: specifies the name of the driver to access DB2 with JDBC, the URL of the database to which the DAD checker connects, the user ID and the password needed to connect to the database. Here is an example of use of this option: -db COM.ibm.db2.jdbc.app.DB2Driver jdbc:db2:sales_db "" ""

- **-out** *outputFile*: specifies the output text file name. If omitted, the standard output is used.
  NOTE: A second output file, errorsOutput.xml is also created in the same directory as the DAD file, except when the -xml option is used. This file contains in XML form the same information as the output text file except all parser warnings and errors.

- **-help**: displays command line option information.

- **-version**: displays version information.

Sample files

The following sample files can be found in the 'samples' directory:

- bad_dad.dad: sample DAD file demonstrating all possible semantic errors.
- bad_dad.chk: output text file generated by the DAD checker for bad_dad.dad.
- errorsOutput.xml: output XML file generated by the DAD checker for bad_dad.dad.
- dup.xsl: XSL stylesheet used for tranforming the errorsOutput.xml file into an HTML file showing only the duplicate tags.
- dups.html: generated HTML file showing only the duplicate tags contained in bad_dad.dad.
- docWithDups.xml: sample XML document containing duplicates.

IV. Indicating errors and warnings in the output text file

Errors and warnings are indicated by tag occurrence. Two tags are considered as occurrences of the same tag if:

- Their name attributes have the same value
- They have the same number of ancestors

- The name attributes of their corresponding ancestor tags have the same value

Occurrences of the same tag could potentially have different children tags.

Tag occurrences that do not conform to the DAD semantic rules are indicated in the output text file in the following way:

- All ancestor tags and their attributes are displayed in sequence
- The tag that is in error is displayed, preceded by a number indicating its depth in the XML tree, and followed by a list of line numbers where all occurences of the tag appear in the DAD file.
  Alternatively, each occurrence can be displayed seperately, using the -all command line option.
- The direct children tags of the first tag occurrence are then displayed. For those children tags that specify a data mapping, the data mapping tags are also displayed.
  Again, using the -all command line option, this process can be repeated for all tag occurrences.

NOTE: The depth of the root element of the XML tree is 0.

Here is an example of indicating an error in the text output file:

```
<DAD>
  <Xcollection>
    <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
4          <element_node name="Password">  line(s): 49 75
            <element_node name="Pswd1">
              <element_node name="Pswd2">
```

In this example, the element_node tag whose name attribute has the value "Password" is in error. There are two occurrences of this tag in the DAD file, on lines 49 and 75.
The tag in error can be isolated from the list of ancestor and children tags easily by locating the tag's depth indicator (in this example 4). The list of ancestor and children tags help establish the context in which the error occurred.

For the same error, using the *-all* option, the output text file fragment would look as follows:

```
<DAD>
  <Xcollection>
    <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
4          <element_node name="Password">  line: 49
            <element_node name="Pswd1">
              <element_node name="Pswd2">

<DAD>
  <Xcollection>
    <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
4          <element_node name="Password">  line: 75
            <element_node name="Pswd1">
              <element_node name="Pswd3">
```

In this example, it is evident that the two occurrences have identical ancestors and name attribute values, but different children elements.

V. Checks performed by the DAD checker

On invocation, the DAD checker produces the following message

```
Checking DAD document: file_path
```

where *file_path* is the path to the DAD file being validated.

The DAD checker performs the following validation checks:

1. Well formedness checking and DTD validation
2. Duplicate <attribute_node> and leaf <element_node> detection (RDB_node mapping)
3. Missing type attribute detection
4. Missing table declaration detection
5. Missing <text_node> or <attribute_node> detection
6. <attribute_node> and <element_node> mapping order check (required for FIXPAK 3 and earlier)
7. Data mapping consistency check for tags with identical name attribute values
8. multi_occurrence attribute value checking (RDB_node mapping)
9. XML document structure check against the DAD structure
10. Column datatype checking against the database
11. Too big data detection in the XML document

These are described in detail below.

1. Well formedness checking and DTD validation

DAD files must be valid against the DAD DTD, located in DXX_INSTALL\dtd\dad.dtd, where DXX_INSTALL is the XML Extender installation path.

If the DAD file is not well-formed, a fatal error occurs that causes the DAD checker to terminate. This situation is indicated in the output text file.

Here is an example of a fatal error indicator:

```
org.xml.sax.SAXException: Stopping after fatal error, line 1, col 22. The XML declaration must end with "?>".
```

Validation against the DTD is dynamic: the validation is done only if a DTD is declared in the file. Validation errors and warnings are also reported in the output text file, but do not cause the DAD checker to terminate.

The following example is a fragment of an output text file showing two possible validation errors that can be encountered while parsing the DAD file:

```
** The document is not valid against the DTD, line 5, col 15. Element type "XCollection" must be declared.
** The document is not valid against the DTD, line 578, col 21. The content of element type "text_node" must match "(column|RDB_node)".
```

2. Duplicate <attribute_node> and leaf <element_node> detection (RDB_node mapping)

This check is relevant only to DAD files that use RDB_node mapping.

If the values of the *name* attributes of two or more <attribute_node> or leaf <element_node> tags that do **not** have the same ancestors in the DAD XML tree are identical, then the tags are considered to be duplicates.

Two or more tags are considered to have the same ancestors if the name attributes of their corresponding ancestor tags have the same value.

A leaf <element_node> is an <element_node> that is used to map a tag with no children in the XML document tree. It follows that leaf <element_node> tags must have one <text_node> tag as one of their direct children. No other <element_node> tags can have <text_node> tags as direct children.

This conflict may arise either between two or more leaf <element_node> tags, two or more <attribute_node> tags or between leaf <element_node> tags and <attribute_node> tags.

Here are some examples of such conflicts:

Example 1: leaf <element_node> conflict

```
<element_node name = "A1">
        <element_node name = "B">
                <element_node name = "C">
                        <text_node>
                                ...
        <element_node name = "A2">
                <element_node name = "B">
                        <element_node name = "C">
                                <text_node>
                                        ...
</element_node>
```

In this example, <element_node name = "C"> is duplicated, because it is mapped through two different paths: **A1**\B\C and **A2**\B\C. Note that <element_node name="B"> is not considered to be duplicated, because it is a non-leaf <element_node>.

Example 2: <attribute_node> conflict

```
<element_node name = "A1">
        <attribute_node name = "B">
                ...
        <element_node name = "A2">
                <attribute_node name = "B">
                        ...
</element_node>
```

In this example, <attribute_node name = "B"> is duplicated, because it is mapped through two different paths: **A1**\B and **A2**\B.

Example 3: leaf <element_node> and <attribute_node> conflict.

```
<element_node name = "A">
                <element_node name = "B">
                                <text_node>
                                                ...
                </element_node>
</element_node>
...
<attribute_node name = "B">
...
<attribute_node name = "A">
...
```

In this example, <element_node name = "B"> conflicts with <attribute_node name = "B">. Note that <element_node name = "A"> and <attribute_node name = "A"> do **not** conflict, because <element_node name = "A"> is **not** a leaf <element_node>.

When such naming conflicts occur, the XML document DTD needs to be revised to eliminate the conflicts. The XML document and the DAD file need to be revised in turn, to reflect the DTD changes.

The following example is a fragment of an output text file showing duplication errors:

```
1 duplicate naming conflicts were found
A total of 2 tags are in error (total occurrences: 3)

The following tags are duplicates:

<DAD>
  <Xcollection>
    <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
4           <element_node name="Country">  line(s): 127 135
             <text_node>
               <RDB_node>
                 <table name="advertiser">
                 <column type="VARCHAR(63)" name="country">

----------------
<DAD>
  <Xcollection>
    <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
        <element_node name="Campaign" multi_occurrence="YES">
          <element_node name="Target" multi_occurrence="YES">
            <element_node name="Location" multi_occurrence="YES">
7                 <element_node name="Country">  line(s): 460
                   <text_node>
                     <RDB_node>
                       <table name="target_location">
                       <column type="VARCHAR(63)" name="country">

----------------
----------------------------------------------------
```

Tags that are in error are grouped by naming conflict. The groups are separated by long lines, while the tags are separated by short lines. Alternatively, for each tag in error, all the occurences can be displayed, by using the *-all* command line option.

If no duplicates exist in the DAD file, the following message is written in the output text file:

```
No duplicated tags were found.
```

3. Missing type attribute detection

When using a DAD file to enable a collection or for decomposition, the ***type*** attribute must be specified for each **<column>** tag.

For example:

```
<column name="email" type="varchar(20)">
```

The enable_collection command uses the column type specifications to create the tables in the collection if the tables do not exist. If the tables do exist, the type specified in the DAD must match the actual column type in the database..

The following example is a fragment of an output text file showing <column> tags that do not have the *type* attribute:

```
If this DAD is to be used for decomposition or for enabling a collection,
the type attributes are missing for the following <column> tag(s):

<DAD>
  <Xcollection>
    <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
        <element_node name="Address">
          <text_node>
            <RDB_node>
7                   <column name="address">  line: 86
```

If no *type* attributes are missing, the following message is written in the output text file:

```
No type attributes are missing for <column> tags.
```

4. Table declaration check

The first <RDB_node> tag in the DAD file must be a direct child of the first <element_node> tag. This <RDB_node> tag contains the <table> tags which declare the relational tables used for data mapping.

```
<root_node>
      <element_node name="Document">
            <RDB_node>
                  <table name="table1"/>
                  <table name="table2"/>
                  <table name="table3"/>
            </RDB_node>
            <element_node name="firstChild">
                  ...
```

All subsequent <RDB_node> tags must be enclosed in <text_node> or <attribute_node> tags. See Part V Section 5.

The following checks are performed:

* The checker checks that the first <element_node> tag (ie the only direct child of the <root_node> tag) has a <RDB_node> tag as a direct child. If it is not the case, it means that the table declaration is missing or that the table declaration is enclosed in a <text_node> tag.

The following example is a fragment of an output text file showing a missing or misplaced table declaration error:

```
line 9: The table declaration is missing (or abnormally enclosed in a <text_node> tag).
```

* The checker also checks that the table declaration does not wrongly contain a <column> tag.

The following example is a fragment of an output text file showing an incorrect table declaration:

```
line 25: The table declaration abnormally contains a <column> tag.
```

5. Missing <text_node> or <attribute_node> detection

Each **<RDB_node>** tag other than the first one, which is used for the table declaration, must be enclosed in an <attribute_node> or a <text_node> tag, as in the example below:

```
<element_node name ="amount">
      <text_node>
            <RDB_node>
                  <table name="fakebank.payments"/>
                  <column name="amount" type="decimal(8,2)"/>
            </RDB_node>
      </text_node>
</element_node>
```

The DAD checker points out all such <RDB_node></RDB_node> tags that are not enclosed in <text_node> or <attribute_node> tags.

The following example is a fragment of an output text file showing a missing <text_node> or <attribute_node> tag:

```
Each of the following <RDB_node> tags must be enclosed in a <text_node> or an <attribute_node> tag:

<DAD>
  <Xcollection>
    <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
        <element_node name="PostalCode">
5             <RDB_node>  line: 107
               <table name="advertiser">
```

```
            <column type="VARCHAR(10)" name="postal_code">
```

If all the <RDB_node> tags in the DAD are properly enclosed, the following message is written to the output text file:

```
All <RDB_node> tags are properly enclosed.
```

6. <*attribute_node*> and <*element_node*> mapping order check (required for FIXPAK 3 and earlier)

The following check is only applicable for versions of the XML Extender prior to and including FIXPAK 3.

In these versions of the XML Extender, <attribute_node> tags need to be mapped to a table before any <element_node> tags are mapped to the same table.

The following example is a fragment of a DAD file demonstrating this problem:

```
<element_node name="payment-request" multi_occurrence="YES">
   <element_node name="payment-request-id">
      <text_node>
         <RDB_node>
            <table name="fakebank.payments"/>
            <column name="statement_id" type="varchar(30)"/>
         ...
   <element_node name="bank-customer-info">
      <element_node name="account">
         <attribute_node name="type">
            <text_node>
               <RDB_node>
                  <table name="fakebank.payments"/>
                  <column name="payor_account" type="char(6)"/>
               ...
```

In this example, <attribute_node name="type"> is mapped to the same table, 'fakebank.payments', as <element_node name = "payment-request-id">; the mapping of the <attribute_node> must precede the mapping of the <element_node>.

The best solution to this problem is to use a later version of the XML Extender.

The DAD checker locates and reports all such <attribute_node> mapping order problems.

The following example is a fragment of an output text file showing a mapping order warning:

```
FIXPAK 3 or earlier only:
each of the following <attribute_node> tags in the hierarchy must be mapped before all <element_node> tags mapped to the same table.

<DAD>
 <Xcollection>
  <root_node>
   <element_node name="Advertiser" multi_occurrence="YES">
    <element_node name="Password">
     <attribute_node name="Psw_att">
      <RDB_node>
6             <table name="advertiser">  line(s): 56
```

If no such mapping order problems exist, the following message is written to the output text file:

```
FIXPAK 3 or earlier only:
No <attribute_node> tag mapping order problems were found.
```

7. Data mapping consistency check for tags with identical name attribute values

Within the DAD file, all <element_node> tags which are mapped as well as all <attribute_node> tags, identified by distinct *name* attribute values, should be mapped exactly once. It follows that if two or more occurrences of an <element_node> tag or <attribute_node> tag are mapped to different columns, their *name* attributes should be assigned different values.

The following cases are therefore illegal:

- <element_node> tags with the same *name* attribute values, that are mapped to different columns or tables
- <attribute_node> tags with the same *name* attribute values, that are mapped to different columns or tables

In the following example the second occurrence of the <element_node name="type"> tag has a different mapping than the first occurence.

```
<element_node name="bank-customer-info">
   <element_node name="account">
      <element_node name="type">
         <text_node>
```

```
                <RDB_node>
                    <table name="fakebank.payments"/>
                    <column name="payor_account" type="char(20)"/>
                </RDB_node>
            </text_node>
        </element_node>
    </element_node>
</element_node>
<element_node name="bank-customer-info">
    <element_node name="account">
        <element_node name="type">
            <text_node>
                <RDB_node>
                    <table name="fakebank.payments"/>
                    <column name="payto_account" type="char(20)"/>
                </RDB_node>
            </text_node>
        </element_node>
    </element_node>
</element_node>
```

A solution to this problem, is to create a new element to be used with the second mapping. This would require changes to the DTD, the XML document and the DAD file.

The DAD checker detects all such data mapping inconsistencies, but does not display duplicate <attribute_node> and duplicate leaf <element_node> tags as part of this check.

The following example is a fragment of an output text file indicating the inconsistent mapping problem:

```
The following <element_node> tags have the same names and ancestors but they do not have the same mappings:

<DAD>
  <Xcollection>
    <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
4           <element_node name="PostalCode">  line(s): 127
             <text_node>
               <RDB_node>
                 <table name="advertiser">
                 <column type="VARCHAR(10)" name="postal_code">

----------------
<DAD>
  <Xcollection>
    <root_node>
      <element_node name="Advertiser" multi_occurrence="YES">
4           <element_node name="PostalCode">  line(s): 135 143
             <text_node>
               <RDB_node>
                 <table name="advertiser">
                 <column type="VARCHAR(10)" name="postal_code2">

----------------

----------------------------------------------------
...
```

In this example, one occurrence of the <element_node name="PostalCode"> on line 127 is mapped to the 'postal_code' column, whereas two other occurrences of the same tag, on lines 135 and 143, are mapped to the 'postal_code2' column.

If no data mapping inconsistency problems exist, the following text is written to the output text file:

```
No <element_node> tags have been found with the same names and different mappings.

No <attribute_node> tags have been found with the same names and different mappings.
```

8. multi_occurrence attribute value checking (RDB_node mapping)

This check is relevant only to DAD files that use RDB_node mapping. It looks for the lowest common ancestor tags and checks that their *multi_occurrence* attributes are set to "YES".

For decomposition, useful terms in the discussion of *multi_occurrence* attributes are **wrappers** and **lowest common ancestor**.

- a wrapper is an <element_node> tag that has:
    - no direct child <attribute_node> or <text_node> tag
    - one or more child that is an <element_node> tag
- the lowest common ancestor (LCA) is the lowest (in terms of tree level) <element_node> tag such that all <element_node> and

<attribute_node> that map to the same table have the LCA as their ancestor.

If the LCA has a child <attribute_node> or <text_node>, then it is not a wrapper and does not require its *multi_occurrence* attribute to be set to "YES".

If the LCA has no child <attribute_node> or <text_node>, then it is a wrapper and its *multi_occurrence* attribute can be set to "YES". If so, the values of the mapped elements wrapped by the LCA will populate only one row in the table with which the LCA is associated.

The default value for the *multi_occurrence* attribute is "NO".

In the following example, *payment-request* is the LCA for table *fakebank.payments*. It has no <attribute_node> and no <text_node> as direct children, so it is a wrapper. Then it is possible to set its *multi_occurrence* attribute to "YES". If it is set to "YES", the values of the mapped elements <element_node name="payment-request-id">, <element_node name ="amount"> and <element_node name ="sender"> will populate the same row in the *fakebank.payments* table.

```
<element_node name="payment-request" multi_occurrence="YES">
    <element_node name="payment-request-id">
        <text_node>
            <RDB_node>
                <table name="fakebank.payments"/>
                <column name="statement_id" type="varchar(30)"/>
            </RDB_node>
        </text_node>
    </element_node>
    <element_node name ="amount">
        <text_node>
            <RDB_node>
                <table name="fakebank.payments"/>
                <column name="amount" type="decimal(8,2)"/>
            </RDB_node>
        </text_node>
    </element_node>
    <element_node name ="sender">
        <attribute_node name ="ID">
            <RDB_node>
                <table name="fakebank.payments"/>
                <column name="sender_ID" type="decimal(8,2)"/>
            </RDB_node>
        </attribute_node>
    </element_node>
</element_node>
```

The DAD checker indicates all <element_node> tags whith *multi_occurrence* attributes that can be set to "YES".

The following example is a fragment of an output text file demonstrating an <element_node> tag with *multi_occurrence* attribute that can be set to "YES". If it is set to "YES", the values of its descendants will be decomposed into one row.

```
The multi_occurrence attributes of the following tags can possibly be set to "YES":

<DAD>
  <Xcollection>
    <root_node>
      <element_node name="Advertiser">
4         <element_node name="Password">  line(s): 49
           <element_node name="Pswd1">
           <element_node name="Pswd2">
------------------------------------------------------
```

NOTE: This version of the DAD checker never says that a multi_occurrence attribute must be set to "YES". However, in some particular cases, the decomposition may fail if one or several multi_occurrence attributes are not set to "YES".

9. XML document structure check against the database

The DAD checker can be used to check that the structure of an XML document matches the structure of the DAD file. The structure of the DAD file and the structure of the XML documents must match. This means that the attributes and leaf tags of an XML document that is to decomposed must be mapped in the DAD file using exactly the same names (with the same case) and the same paths.

The following example shows mismatch errors between the XML document and the DAD file.
Here is a fragment of the XML document:

```
<Employee ID = "000">
    <PersonalData>
        <Address>New York</Address>
        <phone>(000) 000-0000</phone>
    ...
```

And here is its corresponding fragment in the DAD:

```
<element_node name = "Employee">
     <attribute_node name = "id">
          <RDB_node>
               ...
          </RDB_node>
     </attribute_node>
     <element_node name = "Personal">
          <element_node name = "Address">
               <text_node>
                    <RDB_node>
                         ...
                    </RDB_node>
               </text_node>
          </element_node>
     ...
```

In the first mismatch, the *ID* attribute of the <Employee> tag in the XML document is not properly spelled in the DAD file. Consequently the value of this attribute will not be decomposed in the database.

In the second mismatch, the <Address> tag does not have the same path in the XML document and in the DAD: its parent tag is <PersonalData> in the document, whereas its parent tag in the DAD is a tag whose name is *Personal*. Therefore, the value of the <Address> tag will not be stored in the database either.

The following output text file is generated by checking the DAD shown in the example above and by using the doc option to check that the structure of the XML document matches the structure of the DAD:

```
** Ensure that the following attributes have the correct case:

attribute in error: ID

0     <Employee ID="000">  line(s): 2
       <PersonalData>

----------------------------------------------------

** For each of the following elements, at least one ancestor element was not found in the DAD:

<Employee>
  <PersonalData>
2       <Address>  line(s): 4

----------------------------------------------------
```

The DAD checker will also detect all the attributes or leaf tags of the XML document that are not mapped in the DAD file, and will issue the following kind of message:

```
** The following elements are not mapped in the DAD:

<Employee>
  <PersonalData>
2       <phone>  line(s): 5

----------------------------------------------------
```

10. Column datatype checking against the database

In the DAD, the **<column>** tags may contain a ***type*** attribute. This attribute is mandatory if the DAD is used for decomposition or is to be enabled as a collection (see Part V Section 3). The value of the ***type*** attribute must match the type of the column in the database.

For example, in the DAD if an element_node or attribute_node is mapped to column2 of table1 and if the datatype for this column is VARCHAR(12), then the *type* attribute for the corresponding <column> tag must be VARCHAR(12) too:

```
<RDB_node>
     <table name="table1"/>
     <column name="column2" type="varchar(12)"/>
</RDB_node>
```

The DAD checker checks that the *type* attributes values of <column> tags match the actual datatype of the corresponding columns in the database. Because the checker needs to connect to the database to perform this check, it is done only when the db option is used.

The following example is a fragment of an output text file showing mismatch errors between type attributes of <column> tags and the actual type of the corresponding columns in the database:

```
WARNING: The type attributes of the following <column> tags do not match the types of the columns in the database:

<DAD>
 <Xcollection>
  <root_node>
   <element_node name="Order">
    <element_node name="Part" multi_occurrence="YES">
     <element_node name="ExtendedPrice">
      <text_node>
       <RDB_node>
8                <column type="numeric(1, 2)" name="PRICE">  line: 78

In the database, the type found for column PRICE in table PART_TAB is: DECIMAL(10,2)

----------------------------------------------------
<DAD>
 <Xcollection>
  <root_node>
   <element_node name="Order">
    <element_node name="Customer">
     <element_node name="Phone">
      <text_node>
       <RDB_node>
8                <column type="clob(1)" name="CUSTOMER_PHONE">  line: 46

In the database, the type found for column CUSTOMER_PHONE in table ORDER_TAB is: VARCHAR(16)

----------------------------------------------------
```

11. Data exceeding the maximum allowed size in the XML document

When decomposing an XML document in a database, it is important to ensure that the size of the data that are to be inserted is not greater than the maximum size that can contain the target column.

The following example is a fragment of a DAD mapping element *Name* to column *CUSTOMER_NAME* in table *ORDER_TAB*:

```
<element_node name="Customer">
   <element_node name="Name">
      <text_node>
         <RDB_node>
            <table name="ORDER_TAB"/>
            <column name="CUSTOMER_NAME" type="VarChar(16)"/>
         </RDB_node>
      </text_node>
   </element_node>
   ...
```

And here is element *Name* as it appears in the XML document:

```
<Customer>
   <Name>American Diesel Motor Company</Name>
   ...
```

In this example, element *Name* contains a string with too many characters to be inserted in column *CUSTOMER_NAME*. The decomposition would fail.

The DAD checker looks at the size of data contained in the tags of the XML document and compares it to the size of the data that can be stored in the corresponding target columns according to their datatypes. In this version of the checker, all the datatypes are checked except BLOB, DATE, TIME and TIMESTAMP.

To perform this check, the XML document must be passed to the checker using the doc option.
If the db option is used, then the datatypes of the columns where the data will be inserted will be checked in the catalog of the database. If this option is not used, then the datatypes of the columns are read in the DAD itself thanks to the *type* attributes of the <column> tags.

The following example is a fragment of an output text file showing the data found in the XML document that are too large to be inserted in the corresponding columns:

```
The following elements in the XML document contain values that exceed the maximum allowed size:

0    <Order key="11222222222222222222222222222222222221111">  line: 3
      <Customer>
      <Part color="black">
      <Part color="red">

In the DAD, element "key" is mapped to: <column type="Integer" name="ORDER_KEY">, line: 22 (type found in database: INTEGER).
The length of the data in this element is: > 4 bytes.
```

```
----------------------------------------------------
<Order>
  <Customer>
2       <Name>  line: 5

In the DAD, element "Name" is mapped to: <column type="VarChar(16)" name="CUSTOMER_NAME">, line: 30 (type found in database: VARCHAR(16)).
The length of the data in this element is: 29 bytes.

----------------------------------------------------
```

12. Empty name attributes for <table> and <column> tags

The DAD checker checks that the ***name*** attributes for ***table*** and ***column*** tags in the DAD are not empty or missing.

The following example is a fragment of an output text file showing *table* and *column* tags with *name* attributes that are missing or empty:

```
The following elements must not contain an empty 'name' attribute:

<DAD>
  <Xcollection>
    <root_node>
      <element_node name="Order">
        <attribute_node name="key">
          <RDB_node>
6               <column type="Integer" name="">  line: 22

----------------------------------------------------
<DAD>
  <Xcollection>
    <root_node>
      <element_node name="Order">
        <element_node name="Part" multi_occurrence="YES">
          <element_node name="key">
            <text_node>
              <RDB_node>
8                   <table>  line: 61

----------------------------------------------------
```

13. multi_occurrence attribute value checking for Xcolumn DAD

In Xcolumn DADs, **<column>** tags may have their **multi_occurrence** attribute set to "YES". But one **<table>** tag can enclose only one <column> tag if its multi_occurrence attribute is set to "YES".

The following example is a fragment of an Xcolumn DAD. The <column> tag with name *price* cannot have siblings because its multi_occurrence attribute is set to "YES":

```
<Xcolumn>
      <table name="order_side_tab">
            <column name="order_key" type="integer" path="/Order/@key" multi_occurrence="NO"/>
            <column name="customer" type="varchar(50)" path="/Order/Customer/Name" multi_occurrence="NO"/>
      </table>
      <table name="part_side_tab">
            <column name="price" type="decimal(10,2)" path="/Order/Part/ExtendedPrice" multi_occurrence="YES"/>
      </table>
      ...
```

If the DAD checker detects that the DAD is an Xcolumn DAD, then it checks the multi_occurrence attributes of the <column> tags.

The following example is a fragment of an output text file showing an incorrect multi_occurrence attribute:

```
The following <table> tags have at least one <column> tag as a child with multi_occurrence="YES".
Therefore they must have only one <column> tag as a child:

<DAD>
  <Xcolumn>
2       <table name="part_side_tab">  line: 17
         <column type="decimal(10,2)" path="/Order/Part/ExtendedPrice" name="price" multi_occurrence="YES">
         <column type="real" path="/Order/Part/Tax" name="tax">

----------------------------------------------------
```

14. Attribute and element potential naming conflict check (XML documents)

In XML documents, elements with the same name can appear in different contexts, i.e. having different ancestor elements. Also, attributes and elements can have identical names. The XML Extender however, is unable to resolve these naming conflicts, as they result in duplicate tags in the DAD file.

Therefore all attributes and all elements with the same ancestors that are to be mapped, must have unique names.

As a convenience, the DAD checker can be used to check XML documents for such naming conflicts. If more than one of the conflicting elements or attributes need to be mapped, then naming changes will have to be made to the document and the DTD if it exists.

It is best to perform this check on the XML document before the DAD file is created. Please note that the DAD checker does not validate the XML document against its DTD.

The following example is a fragment of an XML document where such naming conflicts occur:

```
<A1>
    <B>
        <C>
            ...
<A2>
    <B>
        <C>
            ...
        <D C = "attValue">
            ...
```

If both <C> elements and the C attribute in the example above are to be mapped, then the resulting DAD file would have the following duplicate conflicts:

```
<element_node name = "A1">
    <element_node name = "B">
        <element_node name = "C">
            <text_node>
                ...
<element_node name = "A2">
    <element_node name = "B">
        <element_node name = "C">
            <text_node>
                ...
        <element_node name = "D">
            <attribute_node name = "C">
                ...
</element_node>
```

The two <element_node name = "C"> tags and the <attribute_node name = "C"> tag are duplicates in the DAD.

Here is an excerpt of the output text file obtained by checking an XML document containing duplicated tags:

```
1 duplicate naming conflicts were found

A total of 2 tags are in error (total occurrences: 2)

The following tags are duplicates or they contain attributes that are duplicates:

<Advertiser>
1     <LoginName>  line(s): 4


----------------
<Advertiser>
1     <AdvertiserID LoginName="ok">  line(s): 3


----------------

----------------------------------------------------
...
```

The XML document used in this example, docWithDups.xml, can be found in the 'samples' folder.
If your browser supports XML, you can access the file directly by following this link: docWithDups.xml

VI. Enhancements and bug fixes

release 1.3

- The ***-db*** option was added, see -db option in Part III for details.
- Added check of the columns datatypes against the database, see Part V Section 10.
- Added detection of the data in the XML document exceeding the maximum allowed size, see Part V Section 11.

- Added detection of empty name attributes for <table> and <column> tags, see Part V Section 12.
- Added check of the multi_occurrence attributes values in an Xcolumn DAD, see Part V Section 13.

release 1.2

- Enhanced the checking of the multi_occurrence attribute value, see Part V Section 8.
- The ***-doc*** option was added, see Part V Section 9.
- The ***-dtd*** option was added, see Part III for details.
- Enhanced the checking of the table declaration. See Part V Section 4.
- Added support for table and column names starting and ending by *&quot;*.
- Use of version 2.0.1 of the Xerces parser.

release 1.1

- The ***-enc*** option was added, see Part III for details.
- Use of Version 2.0.0 of the Xerces parser, instead of Version 1.1.3.
- The total occurrences of all duplicated tags is now added to the output text files. See Part V Section 2.
- Added support for relative paths with no parent directory for the *fileToCheck* parameter.
- Corrected method of determining the line numbers in a file that has a different encoding than the default encoding of the machine.

release 1.0

This is the first release.

License Information

IBM defines use for the Program for charging purposes and specifies it in the Proof of Entitlement. Charges are based on extent of use authorized. If you wish to increase the extent of use, notify IBM or its reseller and pay any applicable charges. IBM does not give refunds or credits for charges already due or paid. If any authority imposes a duty, tax, levy or fee, excluding those based on IBM's net income, upon the Program supplied by IBM under this Agreement, then you agree to pay that amount as IBM specifies or supply exemption documentation.

4. No Warranty

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CAN NOT BE EXCLUDED, IBM MAKES NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, THE WARRANTY OF NON-INFRINGEMENT AND THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY. IBM MAKES NO WARRANTY REGARDING THE CAPABILITY OF THE PROGRAM TO CORRECTLY PROCESS, PROVIDE AND/OR RECEIVE DATE DATA WITHIN AND BETWEEN THE 20TH AND 21ST CENTURIES. The exclusion also applies to any of IBM's subcontractors, suppliers, or program developers (collectively called "Suppliers"). Manufacturers, suppliers, or publishers of non-IBM Programs may provide their own warranties.

5. Limitation of Liability

NEITHER IBM NOR ITS SUPPLIERS WILL BE LIABLE FOR ANY DIRECT OR INDIRECT DAMAGES, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST SAVINGS, OR ANY INCIDENTAL, SPECIAL, OR OTHER ECONOMIC CONSEQUENTIAL DAMAGES, EVEN IF IBM IS INFORMED OF THEIR POSSIBILITY. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE EXCLUSION OR LIMITATION MAY NOT APPLY TO YOU.

6. General

Nothing in this Agreement affects any statutory rights of consumers that cannot be waived or limited by contract. IBM may terminate your license if you fail to comply with the terms of this Agreement. If IBM does so, you must immediately destroy the Program and all copies you made of it. You agree to comply with applicable export laws and regulations. Neither you nor IBM will bring a legal action under this Agreement more than two years after the cause of action arose unless otherwise provided by local law without the possibility of contractual waiver or limitation. Neither you nor IBM is responsible for failure to fulfill any obligations due to causes beyond its control. IBM does not provide program services or technical support, unless IBM specifies otherwise. The laws of the country in which you acquire the Program govern this Agreement, except 1) in Australia, the laws of the State or Territory in which the transaction is performed govern this Agreement; 2) in Albania, Armenia, Belarus, Bosnia/Herzegovina, Bulgaria, Croatia, Czech Republic, Georgia, Hungary, Kazakhstan, Kirghizia, Former Yugoslav Republic of Macedonia (FYROM), Moldova, Poland, Romania, Russia, Slovak Republic, Slovenia, Ukraine, and Federal Republic of Yugoslavia, the laws of Austria govern this Agreement; 3) in the United Kingdom, all disputes relating to this Agreement will be governed by English Law and will be submitted to the exclusive jurisdiction of the English courts; 4) in Canada, the laws in the Province of Ontario govern this Agreement; and 5) in the United States and Puerto Rico, and People's Republic of China, the laws of the State of New York govern this Agreement.

Part 2 - Country-unique Terms

AUSTRALIA:

No Warranty (Section 4): The following paragraph is added to this Section: Although IBM specifies that there are no warranties, you may have certain rights under the Trade Practices Act 1974 or other legislation and are only limited to the extent permitted by the applicable legislation. Limitation of Liability (Section 5): The following paragraph is added to this Section: Where IBM is in breach of a condition or warranty implied by the Trade Practices Act 1974, IBM's liability is limited to the repair or replacement of the goods, or the supply of equivalent goods. Where that condition or warranty relates to right to sell, quiet possession or clear title, or the goods are of a kind ordinarily acquired for personal, domestic or household use or consumption, then none of the limitations in this paragraph apply.

GERMANY:

No Warranty (Section 4): The following paragraphs are added to this Section: The minimum warranty period for Programs is six months. In case a Program is delivered without Specifications, we will only warrant that the Program information correctly describes the Program and that the Program can be used according to the Program information. You have to check the usability according to the Program information within the "money-back guaranty" period. Limitation of Liability (Section 5): The following paragraph is added to this Section: The limitations and exclusions specified in the Agreement will not apply to damages caused by IBM with fraud or gross negligence, and for express warranty.

INDIA:

General (Section 6): The following replaces the fourth paragraph of this Section: If no suit or other legal action is brought, within two years after the cause of action arose, in respect of any claim that either party may have against the other, the rights of the concerned party in respect of such claim will be forfeited and the other party will stand released from its obligations in respect of such claim.

IRELAND:

No Warranty (Section 4): The following paragraph is added to this Section: Except as expressly provided in these terms and conditions, all statutory conditions, including all warranties implied, but without prejudice to the generality of the foregoing, all warranties implied by the Sale of Goods Act 1893 or the Sale of Goods and Supply of Services Act 1980 are hereby excluded.

ITALY:

Limitation of Liability (Section 5): This Section is replaced by the following: Unless otherwise provided by mandatory law, IBM is not liable for any damages which might arise.

NEW ZEALAND:

No Warranty (Section 4): The following paragraph is added to this Section: Although IBM specifies that there are no warranties, you may have certain rights under the Consumer Guarantees Act 1993 or other legislation which cannot be excluded or limited. The Consumer Guarantees Act 1993 will not apply in respect of any goods or services which IBM provides, if you require the goods and services for the purposes of a business as defined in that Act. Limitation of Liability (Section 5): The following paragraph is added to this Section: Where Programs are not acquired for the purposes of a business as defined in the Consumer Guarantees Act 1993, the limitations in this Section are subject to the limitations in that Act.

PEOPLE'S REPUBLIC OF CHINA:

Charges (Section 3): The following paragraph is added to the Section: All banking charges incurred in the People's Republic of China will be borne by you and those incurred outside the People's Republic of China will be borne by IBM.

UNITED KINGDOM:

Limitation of Liability (Section 5): The following paragraph is added to this Section at the end of the first paragraph: The limitation of liability will not apply to any breach of IBM's obligations implied by Section 12 of the Sales of Goods Act 1979 or Section 2 of the Supply of Goods and Services Act 1982.

Z125-5589-01 (10/97)