

June 2006



DB2 Information Management Software

XML SCHEMA REGISTRATION AND VALIDATION

*By Aarti Dua
Software Developer
IBM Software Group*

Table of Contents

<i>XML Schema Support in DB2</i>	3
<i>XML Schema Repository</i>	3
<i>XML Schema Registration</i>	7
<i>XML Schema Validation in DB2</i>	10
XMLValidate Function	10
Explicit Validation	10
Implicit Validation.....	13
<i>XML Schema Management in DB2</i>	16
Querying the XML Schema Repository	16
Dropping a Registered Schema	17
<i>Recommended DB2 Settings on Win 32 Platform</i>	18
<i>Application Heap Size (APPLHEAPSZ)</i>	18
<i>Catalog Cache Size (CATALOGCACHE_SZ)</i>	19
<i>Agent Stack Size (AGENT_STACK_SZ)</i>	21
<i>Acknowledgements</i>	22
<i>References</i>	22

XML Schema Support in DB2

An XML schema defines the structure of an XML document by setting constraints on the data types of elements and attributes in the document. XML documents can be validated against the XML schema to ensure they conform to the constraints defined in the schema.

There are several schema languages that define the grammar for an XML document, for instance, Document-Type Definition (DTD) language, and XML Schema language. Of these languages, because of its extensive support for data types and namespaces, XML Schema is becoming the most widely used schema language; hence, it is the chosen language by DB2 9.

DB2 9 also supports DTDs and external entities; the support for DTDs and external entities is limited to entity resolution and does not include validation. Nevertheless, almost all DTDs can be converted into an equivalent XML Schema without loss of information. This XML Schema can be parsed and validated by DB2 9.

This paper explores DB2's support for XML schemas. In particular, it explains how to register XML schemas in DB2's built-in repository, how to validate XML documents against registered schemas and lastly, how to manage the registered XML schemas.

XML Schema Repository

Applications can choose to validate XML documents against an XML schema to ensure valid data is stored in the database. To validate documents, DB2 needs access to the appropriate schema information. For this reason, DB2 has an XML schema repository (XSR) that maintains a copy of the XML schemas that might be used during validation. The XSR consists of a set of new catalog tables together with commands, stored procedures and APIs to register and manage XML schemas.

An XML schema is a collection of one or more XML schema documents. A simple XML schema consists of only 1 schema document. However, if this schema document, lets call it *documentA.xsd*, imports and /or includes other XML schema documents, the XML schema is a collection of all schema documents imported and /or included by *documentA.xsd* and *documentA.xsd*. *DocumentA.xsd* is called the primary schema document as it is at the top of the hierarchy of the documents; it includes and /or imports the remaining documents in the XML schema.

All schema documents (.xsd files), DTDs and external entities need to be registered individually with the XSR before they can be used. Registered XML Schemas, and DTDs and external entities are called *XSR Objects*.

Let's review a few sample schema documents and explore how to register them in DB2

Examples of Schema Documents

We'll rely on a few sample schema documents in this tutorial. They're listed below: *order.xsd*, *header.xsd*, *product.xsd* and *customer.xsd*. These documents are shipped as samples with DB2 and can be found in the <db2_install_root> directory. They can also be downloaded from the attachments section of this paper.

order.xsd: Primary Schema Document

The Order schema document defines an XML element "PurchaseOrder" of type "potype". This type is defined as sequence of elements, "Header", "Items", and "Customer". The type of the "Header" and "Customer" elements are defined in other schema documents, which are imported. "Items" is defined as a sequence of two child elements, "ItemDescription" and "NumberOrdered". "ItemDescription" is of type "prodType", which is defined in the imported schema document "product.xsd" and "NumberOrdered" is defined as an integer type.

In order.xsd, the namespace attribute xmlns:xsd declares the namespace of order.xsd. The targetNamespace attribute value "<http://www.test.com/po>" defines how this document will be referenced by the outside world i.e. other schema documents and instance documents. Namespaces xmlns:head, xmlns:cust and xmlns:prod are declared to refer to types defined in other namespaces in other schema documents.

order.xsd imports schema documents, product.xsd, header.xsd and customer.xsd. To import an XML schema document two pieces of information are needed: the namespace of the schema document and the schema location, i.e. the URL of the where the schema document can be found.

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.test.com/po"
  xmlns:po="http://www.test.com/po"
  xmlns:head="http://www.test.com/header"
  xmlns:prod = "http://www.test.com/product"
  xmlns:cust = "http://www.test.com/customer">

<!-- Using multiple XML-Schema Docs-->
<xsd:import namespace="http://www.test.com/product" schemaLocation="product.xsd" />
<xsd:import namespace="http://www.test.com/customer" schemaLocation="customer.xsd" />
<xsd:import namespace="http://www.test.com/header" schemaLocation="header.xsd" />

<xsd:complexType name="itemType">
  <xsd:sequence>
    <xsd:element name="Item" minOccurs="1" maxOccurs="unbounded" >
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="ItemDescription" type="prod:prodType" />
          <xsd:element name="NumberOrdered" type="xsd:integer" />
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="potype">
  <xsd:sequence>
    <xsd:element name="Header" type="head:headerType" />
    <xsd:element name="Items" type="po:itemType" />
    <xsd:element name="Customer" type="cust:customerType" />
  </xsd:sequence>
</xsd:complexType>

<xsd:element name="PurchaseOrder" type="po:potype" />

</xsd:schema>

```

header.xsd: Imported by order.xsd

This schema document contains the definition of the “headerType”, which is used for the element “Header” in the order schema. The definitions of all the elements defined in this document are associated with a namespace “http://www.w3.org/2001/XMLSchema”. This document is referenced by other schema and instance documents using its targetNamespace “http://www.test.com/header” namespace.

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.test.com/header">

<xsd:complexType name="headerType">
  <xsd:sequence>
    <xsd:element name="id"           type="xsd:integer" />
    <xsd:element name="date"        type="xsd:date" />
    <xsd:element name="description" type="xsd:string" />
    <xsd:element name="value"       type="xsd:integer" />
    <xsd:element name="status"      type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>

</xsd:schema>

```

product.xsd: Imported by order.xsd

The product schema document contains the definition of “prodType”. In the Order schema document, “prodType” defines the type of the element “ItemDescription”.

```

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.test.com/product" >

<xsd:complexType name="prodType">
  <xsd:sequence>
    <xsd:element name="name"       type="xsd:string" />
    <xsd:element name="sku"        type="xsd:string" />
    <xsd:element name="price"      type="xsd:integer" />
    <xsd:element name="comment"    type="xsd:string" />
  </xsd:sequence>
  <xsd:attribute name="color"     type="xsd:string" />
  <xsd:attribute name="weight"    type="xsd:integer" />
</xsd:complexType>

</xsd:schema>

```

customer.xsd: Imported by order.xsd

This schema document defines the structure for an element of type “customerType”. The Order schema document imports the customer.xsd to define an element “Customer” of type “customerType”.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.test.com/customer" >

<xsd:complexType name="customerType">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string" />
    <xsd:element name="address" type="xsd:string" />
    <xsd:element name="phone" type="xsd:string" />
    <xsd:element name="email" type="xsd:string" />
  </xsd:sequence>
  <xsd:attribute name="type" type="xsd:string" />
</xsd:complexType>

</xsd:schema>
```

XML Schema Registration

The process of registering an XML Schema with the XSR involves three steps.

1. Register the Primary Schema Document

An XML schema is a collection of 1 or more XML schema documents. Of this collection, one document must be designated as the primary schema document. The primary schema document is at the top of the hierarchy with respect to the import and include dependencies. It includes and/or imports other documents in the schema, which might further include and/or import other schema documents.

In order to register the document, a couple of key pieces of information regarding the document are required. First, the fully-qualified file name for the schema document is needed. Fully-qualified name implies the file system path of where the document is stored. Second, an SQL identifier is to be provided so that the XML schema can later be identified and used for validation. The user can choose any valid SQL two-part name to identify the XML schema. Using meaningful names that aid in identifying the XML schema is recommended.

Additional information like the schema location of the XML schema can also be provided during registration. Schema location is a URL that identifies and indicates where the schema document is located. The schema location can, for instance, be an FTP address or the full-qualified schema file name on the local computer system. This information is stored in the catalog tables and will be used by DB2 to validate XML documents implicitly. Implicit validation is discussed in a later section.

Now, let's look at an example of how to register the primary schema document and explore the details.

```
REGISTER XMLSCHEMA http://joe FROM /db2/samples/schema/order.xsd AS joe.order;
```

The example above registers the primary schema document *order.xsd* with the XSR. The fully-qualified name of the *order.xsd* schema is */db2/samples/schema/order.xsd*. */db2/samples/schema* directory is the local directory where the *order.xsd* schema document is stored on the system. The schema location is specified using the URL *http://joe*. And last, but not the least, the XML schema is identified as *joe.order*.

2. Add Schema Documents (optional)

This step is required only when an XML schema involves multiple schema documents. To successfully add a schema document to the registered primary document, the schema location of the document must match the *schemaLocation* attribute referred in the import/include declaration in the importing/including document. No specific order is required when adding schema documents.

In our example schema documents, *header.xsd*, *product.xsd* and *customer.xsd* need to be registered in this step.

Let's look at an example of how to add a schema document.

```
ADD XMLSCHEMA DOCUMENT TO joe.order ADD header.xsd FROM /db2/samples/schema/header.xsd;
```

The above example adds the schema document *header.xsd* to the previously registered XML schema *joe.order*. The local directory in which the document *header.xsd* is saved is */db2/samples/schema*. Note that the schema location specified above is *header.xsd*, which matches the value of the *schemaLocation* attribute specified in the import tags for *header.xsd* in *order.xsd*.

3. Complete Registration

This step completes the registration process. The schema documents are analyzed and any mismatches between the schema locations provided in the 2nd step and *schemaLocation* values provided in the import and/or include tags in the schema documents are discovered. If the schema documents are not well-formed or there are

schema location mismatches, an error is issued to the user and the schema does not get registered with the XSR.

The example below shows how to complete the registration process for the XML schema *joe.order*.

```
COMPLETE XMLSCHEMA joe.order;
```

Here is a complete example of how to register the sample schema documents (order.xsd, header.xsd, product.xsd, and customer.xsd) using the CLP interface

Step 1:

```
REGISTER XMLSCHEMA http://joe FROM /db2/samples/schema/order.xsd AS joe.order;
```

Step 2:

```
ADD XMLSCHEMA DOCUMENT TO joe.order ADD header.xsd FROM  
/db2/samples/schema/header.xsd;
```

```
ADD XMLSCHEMA DOCUMENT TO joe.order ADD product.xsd FROM  
/db2/samples/schema/product.xsd;
```

```
ADD XMLSCHEMA DOCUMENT TO joe.order ADD customer.xsd FROM  
/db2/samples/schema/customer.xsd;
```

Step 3:

```
COMPLETE XMLSCHEMA joe.order;
```

XML Schema Validation in DB2

XML Schema Validation is optional and on a per-document basis, not per column. In DB2 an XML schema is not assigned to a table or a column. Validating XML documents is a good way to prevent junk data from entering the database. However, if the data comes from a trusted source, validation may not be needed.

XML Validate Function

In DB2, XML documents are validated using the XMLValidate function, which is part of the SQL/XML standard. This function checks whether an XML instance document satisfies the structural, data type and content constraints specified in the XML Schema. The validation process also stores the type information of the elements and attributes along with the XML document.

The function can perform explicit or implicit XML schema validation depending on whether the XML schema for validation is explicitly specified or deduced from the instance document.

Explicit Validation

Explicit Validation implies that in the XMLValidate function, information about which XML schema to use to validate an XML instance document is explicitly specified. This information can either be the SQL identifier or the namespace and the schema location of the XML Schema. If the namespace and the schema location are supplied, this information must be of the primary document of the XML Schema.

Let's look at the example instance document, order.xml that conforms to the registered XML schema joe.order.

order.xml has one root element PurchaseOrder with Header, Items and Customer as its child elements. The xmlns:po attribute declares the namespace for this instance document, which is the same as the namespace of XML schema joe.order.

order.xml

```

<po:PurchaseOrder xmlns:po="http://www.test.com/po">
  <Header>
    <id>1</id>
    <date>2004-01-29</date>
    <description>purchase order</description>
    <value>20</value>
    <status>shipped</status>
  </Header>
  <Items>
    <Item>
      <ItemDescription color="red" weight="5">
        <name>Widget C</name>
        <sku>1</sku>
        <price>30</price>
        <comment>no comment</comment>
      </ItemDescription>
      <NumberOrdered>1</NumberOrdered>
    </Item>
  </Items>
  <Customer type="regular">
    <name>Manoj K Sardana</name>
    <address>ring road, bangalore</address>
    <phone>918051055109</phone>
    <email>msardana@in.ibm.com</email>
  </Customer>
</po:PurchaseOrder>

```

Explicit validation can be done using SQL identifiers and URIs. The examples below, which show validation using SQL identifiers and URIs, require a table called purchaseOrder and the registered XML schema Joe.Order. Please follow the example in section *XML Registration* to register the XML schema Joe.Order. The command below creates the table purchaseOrder.

```
CREATE TABLE purchaseOrder (xmldoc XML);
```

The following example validates order.xml using SQL Identifiers.

The command below validates order.xml against the registered schema *joe.order* and if the document is valid, inserts it into purchaseOrder. If the document is not valid, or is not well-formed, an error is raised and the insert fails.

```
INSERT INTO purchaseOrder
```

```
VALUES XMLVALIDATE (XMLPARSE( DOCUMENT '

```

```
<po:PurchaseOrder xmlns:po="http://www.test.com/po">
  <Header>
    <id>1</id>
    <date>2004-01-29</date>
    <description>purchase order</description>
    <value>20</value>
    <status>shipped</status>
  </Header>
  <Items>
    <Item>
      <ItemDescription color = "red" weight = "5">
        <name>Widget C</name>
        <sku>1</sku>
        <price>30</price>
        <comment>no comment</comment>
      </ItemDescription>
      <NumberOrdered>1</NumberOrdered>
    </Item>
  </Items>
  <Customer type = "regular">
    <name>Manoj K Sardana</name>
    <address>ring road, bangalore</address>
    <phone>918051055109</phone>
    <email>msardana@in.ibm.com</email>
  </Customer>
</po:PurchaseOrder> ' PRESERVE WHITESPACE)

```

ACCORDING TO XMLSCHEMA ID joe.order);

The following example validates order.xml using URIs

In this command, the schema for validation is identified by the target namespace and the schema location of the primary schema document, *order.xsd*.

```
INSERT INTO purchaseOrder

```

```
VALUES XMLVALIDATE (XMLPARSE( DOCUMENT '

```

```
<po:PurchaseOrder xmlns:po="http://www.test.com/po">
  <Header>
    <id>1</id>
    <date>2004-01-29</date>
    <description>purchase order</description>
    <value>20</value>
    <status>shipped</status>
  </Header>
  <Items>
    <Item>
      <ItemDescription color = "red" weight = "5">

```

```

    <name>Widget C</name>
    <sku>1</sku>
    <price>30</price>
    <comment>no comment</comment>
  </ItemDescription>
  <NumberOrdered>1</NumberOrdered>
</Item>
</Items>
<Customer type = "regular">
  <name>Manoj K Sardana</name>
  <address>ring road, bangalore</address>
  <phone>918051055109</phone>
  <email>msardana@in.ibm.com</email>
</Customer>
</po:PurchaseOrder> ' PRESERVE WHITESPACE)

```

ACCORDING TO XMLSCHEMA URI '<http://www.test.com/po>'

LOCATION '<http://joe>');

Validating using SQL Identifier Vs URIs

Validating XML documents using SQL identifiers or URIs for the XML Schema, both provide the same functionality and performance. Depending on the application and the user's preference, one method can be favored over the other. SQL identifiers are database centric. Thus, if the user or the application requires a database oriented method to validate XML schemas, validation using SQL identifiers should be the chosen approach.

Following is an example of when SQL identifiers might not be the ideal way to validate XML documents. Two partner companies work together on a project and they share their XML schemas and their applications refer to the schemas. Each company registers the schemas in its own database. Hence, each schema has two SQL identifiers, one from each database. In this case, if the SQL identifiers were used for validation, the companies would need two sets of applications to validate their schemas. However, if URIs are used to validate the XML schemas, the two companies can share one application. Thus, in this case, using URIs to validate the XML schemas provide a better solution.

Implicit Validation

Implicit validation means that no SQL Identifier or URI information are provided during schema validation. The schema hints in the XML instance document are used to locate the XML schema, which is then used to validate the document.

Schema hints are special attributes in the XML instance documents called either *xsi:schemaLocation* or *xsi:noNamespaceSchemaLocation*. The *xsi:schemaLocation* attribute has a pair value that comprises of the namespace and its corresponding schema location. If there is no namespace for a schema, the schema hint attribute

xsi:noNamespaceSchemaLocation is used. Its value, as the name suggests, consists of only the schema location of the XML Schema.

DB2 searches the catalog tables using the pair values specified in the schema hints and locates the XML schema needed to validate the XML document.

Let's look at a couple of XML instance documents.

Client.xml document shows the use of *xsi:schemaLocation* attribute and *Sales.xml* document shows the use of *xsi:noNamespaceSchemaLocation* attribute.

Client.xml

This instance document has a root element "Client", which has "name" and "address" as its child elements. The schemaLocation attribute is referred as the schema hint because its value serves as a hint in locating the schema associated with this instance document. The value of the schemaLocation is a pair value; the first value is the namespace "http://client" and the second value is the location of the schema, http://client.xsd. schemaLocation attribute is specified using the XML namespace prefix xsi. Before, the xsi namespace is used, it should be declared. In this document, it is declared as <http://www.w3.org/2001/XMLSchema-instance> namespace.

```
<po:client xmlns:po="http://client"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://client http://client" >

  <name>Kathy Smith</name>
  <address> 25 EastCreek</address>

</po:client>
```

Sales.xml

This document has a root element "Sales" that has one child element called "Item". Sales.xml does not have a namespace; thus, the schema hint is given by the attribute noNamespaceSchemaLocation. The value of this attribute, "sales.xsd", is the location of the XML schema that will be used to validate the document.

```

<Sales xmlns="http://www.w3.org/2001/XMLSchema "
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="sales.xsd">

  <Item>
    <itemDescription color="red" weight="5">
      <name>Widget C</name>
      <price>30</price>
      <comment>no comment</comment>
    </itemDescription>
    <numOrdered>1</numOrdered>
  </Item>
</Sales>

```

The following example for implicit validation requires a table called purchaseOrder and a registered client.xsd schema document. The schema document client.xsd can be downloaded from the attachments section of this paper.

The commands below create the table and register the schema.

```
CREATE TABLE purchaseOrder (xmldoc XML);
```

```
REGISTER XMLSCHEMA http://client FROM /db2/samples/schema/client.xsd AS joe.client;
```

```
COMPLETE XMLSCHEMA joe.client;
```

Example of Implicit validation

The command below will implicitly validate the client.xml document and if it is valid, it will insert the document into the table purchaseOrder.

Note that when validating using schema hints, no 'ACCORDING TO XMLSCHEMA' clause is required. If the clause is present, then the document is explicitly validated.

To locate the appropriate schema, DB2 searches the catalog tables for the pair value specified in the schemaLocation attribute. In this case, DB2 searches the catalog tables for the XML schema *Joe.Client* using the pair value <http://client> and <http://client>, which are its namespace and schema location values respectively.

```

INSERT INTO purchaseOrder VALUES XMLVALIDATE (XMLPARSE( DOCUMENT '
<po:client xmlns:po="http://client"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://client http://client" >

  <name>Kathy Smith</name>
  <address> 25 EastCreek</address>

</po:client>'
PRESERVE WHITESPACE));

```

Explicit Vs Implicit Validation

Depending on the type and nature of the application, implicit and explicit validation can be used. Implicit validation relies on the XML instance documents to provide accurate information regarding the XML schema in the schema hints. When the XML documents come from a trusted source, explicit validation is not necessary. However, if the contents of an XML document are unknown, explicit validation is preferable.

When explicitly validating a document, only one XML schema can be specified. Implicit validation, on the other hand, does not have such restrictions. Thus, implicit validation is preferable in applications that require two or more XML schemas to validate an XML document. A SOAP message is an example for an XML document that can require two XML schemas. A SOAP message consists of a header and a body. The header can have one schema and the body can have another schema associated with it. Implicit validation can be used to validate the header and the body of the message.

In terms of performance, explicit validation can be faster than implicit validation. This is because in explicit validation, unlike implicit validation, DB2 does not have to search the catalog tables to locate the XML schema that will be used to validate the documents.

XML Schema Management in DB2

Querying the XML Schema Repository

DB2 stores information about XML Schemas in a system catalog view `SYSCAT.XSROBJECTS`. This view can be queried to retrieve information like the object

schema, object name, target namespace, and schema location. The actual schema documents are stored in catalog `SYSCAT.XSROBJECTCOMPONENT`.

The `SYSCAT.XSROBJECTS` catalog view contains 1 row per XML Schema. On the other hand, `SYSCAT.XSROBJECTCOMPONENT` catalog has 1 row per schema document.

Examples

```
SELECT objectschema, objectname, targetnamespace, schemalocation FROM syscat.xsrobjects
```

Query Returns:

OBJECTSCHEMA	OBJECTNAME	TARGETNAMESPACE	SCHEMALOCATION
JOE	ORDER	http://www.test.com/po	http://joe

```
SELECT objectschema, objectname, targetnamespace, schemalocation from syscat.xsrobjectcomponents
```

Query Returns:

OBJECTSCHEMA	OBJECTNAME	TARGETNAMESPACE	SCHEMALOCATION
JOE	ORDER	http://www.test.com/po	http://joe
JOE	ORDER	http://www.test.com/header	header.xsd
JOE	ORDER	http://www.test.com/product	product.xsd
JOE	ORDER	http://www.test.com/customer	customer.xsd

Dropping a Registered Schema

All XML schemas registered with the XML Schema Repository are referred to as XSR Objects. If an XSR Object is no longer needed, it can be removed from the repository using the Drop XSRObjct command.

The schema repository does not have a notion of individual schema documents. It only has a notion of an XML Schema, which is a collection of 1 or more schema documents. Thus, to drop an XSR Object implies deleting all schema documents associated with that XSR Object. XML schemas can be dropped irrespective of whether they have been completed or not.

Examples

The following removes all documents associated with `joe.order` from the XSR.

```
DROP XSROBJECT joe.order
```

The following command will result in a SQL0204N error code: header.xsd is undefined

```
DROP XSROBJECT header.xsd
```

Recommended DB2 Settings on Win 32 Platform

For a given a database, its current settings can be viewed using the following CLP command:

```
DB2 GET DB CFG FOR <database-name>
```

Two configuration parameters -- application heap size (APPLHEAPSZ) and catalog cache size (CATALOGCACHE_SZ) -- are particularly worth noting.

When working with XML Schemas on Win 32 platform, the values for these configuration parameters often need to be increased beyond their default settings. The details of these DB2 settings are below.

Application Heap Size (APPLHEAPSZ)

For large or complex XML Schemas, schema registration to succeed, the APPLHEAPSZ may need to be increased in order to successfully register the schemas within DB2. If you encounter an *SQLCODE: -954* with *SQLSTATE: 57011*, it implies that the current setting of the APPLHEAPSZ is too small for the application.

If the application you are using does not show the detailed error message, it can be obtained from the DB2 command line. Below are the command and the detailed error message.

```
C:\>db2 ? SQL00954N
```

```
SQL0954C Not enough storage is available in the application heap to process the
```

statement.

Explanation:

All available memory for the application has been used.

The statement cannot be processed.

User Response:

Terminate the application on receipt of this message. Increase the database configuration parameter (APPLHEAPSZ) to allow a larger application heap.

sqlcode: -954

sqlstate: 57011

The above error can be fixed by increasing the APPLHEAPSZ setting using the following CLP command:

```
DB2 UPDATE DB CFG FOR <database-name> USING APPLHEAPSZ 32000
```

The above command sets the application heap size to 32000. The maximum application heap size is 60000.

Catalog Cache Size (CATALOGCACHE_SZ)

Sometimes, the catalog cache size (CATALOGCACHE_SZ) may need to be increased in order to enable DB2 to validate XML documents against registered schemas during INSERT or IMPORT. This can occur when there are too many XML Schemas or other objects in the system catalog.

If you encounter **SQLCODE: -973**, **SQLSTATE: 57011** and **SQLERRMC: CATALOGCACHE_SZ**, it implies that the catalog cache size is too small for the application.

If the application you are using does not show detailed error message, it can be obtained from db2 command line. Below are the command and the detailed error message.

```
C:\>db2 ? SQL00973N
```

```
SQL0973N Not enough storage is available in the "<heap-name>" heap to process the statement.
```

Explanation:

All available memory for this heap has been used. The statement cannot be processed.

User Response:

Terminate the application on receipt of this message (SQLCODE). Modify the "<heap-name>" configuration parameter to increase the heap size.

For example, to update a database configuration parameter, issue the following command:

```
db2 update db cfg
for "<db-name>"
using "<heap-name>" "<heap-size>"
```

To view a list of the database configuration parameters, use the GET DATABASE CONFIGURATION command.

To update a database manager configuration parameter, issue the following command:

```
db2 update dbm cfg
for "<db-name>"
using "<heap-name>" "<heap-size>"
```

To view a list of the database manager configuration parameters, use the GET DATABASE MANAGER CONFIGURATION command.

For application group shared heap size, the following three database configuration parameters control its size and usage:

APPGROUP_MEM_SZ, GROUPHEAP_RATIO, and APP_CTL_HEAP_SZ.

The number of applications in one application group is calculated by: $\text{APPGROUP_MEM_SZ} / \text{APP_CTL_HEAP_SZ}$. The application group shared heap size is calculated by: $\text{APPGROUP_MEM_SZ} * \text{GROUPHEAP_RATIO} / 100$.

sqlcode : -973

sqlstate : 57011

The above error can be fixed by increasing the CATALOGCACHE_SZ setting using the following CLP command:

```
DB2 UPDATE DB CFG FOR <database-name> USING CATALOGCACHE_SZ 32000
```

The above command sets the catalog cache size to 32000. The maximum catalog cache size is 524288.

Agent Stack Size (AGENT_STACK_SZ)

Sometimes, due to the recursive nature of the XML parser, the agent stack size (AGENT_STACK_SZ) may need to be increased when validating XML documents against registered schemas so that the XML parser does not hit its size limits.

If XML parser hits the stack size limitations and DB2 runs out of stack for an agent, it cannot safely de-allocate resources (such as locks) that it had allocated on the way. This could affect other threads; therefore DB2 terminates itself in this situation. An error message will not be issued and further connection attempts will fail until a subsequent db2start.

To mitigate such risk, sample XML instances should be tested ahead of production and stack size should be increased to a satisfactory level.

Caution should be maintained when increasing the stack size of an agent. The side-effects of increasing the agent stack size can be detrimental for DB2's performance. The AGENT_STACK_SZ parameter applies to *every* agent in the database so its effect on the overall memory consumption gets multiplied accordingly. Thus, increase the agent stack size **only** when needed and with caution.

The command below lists the current settings for a database manager.

```
C:\> db2 get dbm cfg
```

AGENT_STACK_SZ is one of the database manager settings. The maximum agent size on Win 32 platform is 1000. The command below is used to change the agent stack size.

```
C:\> db2 update dbm cfg using AGENT_STACK_SZ 256
```

For further information on the XML schema repository, validation and schema management, please refer to the **DB2 Information Center** or to the **References** section below.

Acknowledgements

I would like to thank *Suavi Ali Demir* (sdemir@us.ibm.com) for his contribution to the *Recommended DB2 Settings on Win 32 Platform* section of the paper.

References

XML Schema Tutorial

http://www.w3schools.com/schema/schema_intro.asp

W3 XML Schema Specifications Part 1 - Structures

<http://www.w3.org/TR/xmlschema-1/>

W3 XML Schema Specifications Part 2 – Data types

<http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/>

W3C XML Schema Primer

<http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/>

IBM DB2 Viper Release Candidate Information Center

<http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.doc/welcome.htm>

IBM DB2 Viper Release Candidate – hot link to XML Validation documentation

<http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.apdv.embed.doc/doc/c0023539.htm>

IBM DB2 Viper Release Candidate – hot link to XML schema management using XSR

<http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.apdv.embed.doc/doc/c0023539.htm>

Attachments

order.xsd,
header.xsd,
customer.xsd,
product.xsd,
order.xml and
client.xsd



© Copyright IBM Corporation, 2006
IBM Canada
8200 Warden Avenue
Markham, ON
L6G 1C7
Canada

All Rights Reserved.

Neither this documentation nor any part of it may be copied or reproduced in any form or by any means or translated into another language, without the prior consent of the IBM Corporation.

DB2, DB2 Universal Database, IBM, and the IBM logo are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries or both.

Other company, product and service names may be trademarks or service marks of others.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

The information contained in this document is subject to change without any notice. IBM reserves the right to make any such changes without obligation to notify any person of such revision or changes. IBM makes no commitment to keep the information contained herein up to date.

References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.