|  |
| --- |
| **FUNCTIONAL SPECIFICATION** |
| DATA GENERATOR v1.0 |
|  |
| *It contains functional details of project “Data Generator” done by project group SKNCOE-2.* |
|  |
| ***SKNCOE-2 Group*** |
| **11/10/2011** |
|  |

DATA GENEATOR

1.0

REVISION HISTORY

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Name | Date | Version | Remarks |
| 1 | SKNCOE2 | 10-Nov-2011 | 1.0 | Draft |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Table of Contents

[Problem Statement 5](#_Toc308790800)

[Functional Details 6](#_Toc308790801)

[Enhancements 10](#_Toc308790802)

[Limitations 11](#_Toc308790803)

[Appendix-A 12](#_Toc308790804)

# Problem Statement

**Design and implement a system to generate large amounts of artificial data for software testing.**

Data Generator is an automated testing and data generation tool, which enables you to create test data for software quality assurance testing (QA testing), performance testing, usability testing and database load testing. Data Generator is a computerized testing application specifically designed to generate random test data. It is a general tool to generate records for given data types in a given format.

The user will give input through xml file which has details of data type and format for each column. The tool will parse the xml file and start to generate the records.

Data Generator creates test data by generating random numbers, searching test data in built-in database, using user-defined custom lists or enhancing actual data.

Key points:

1. It will generate the records for different tables in parallel.

- It supports multithreading, hence it can create the data for different tables at a time.

2. It will manage the foreign key relationship between two tables.

- If the user requirement is to generate realistic data (not random) then the tool has to support

foreign key relationship between the two tables so as to generate the data from multiple tables.

3. Give option to upload the generated records into a data base using bulk load utility.

- The tool will support bulk loading to insert data from flat file to tables

4. Distributed data generator: Use nodes in a cluster to generate the records.

5. Supporting all possible data types for a column.

- The mapping of data types provided by the user into the data types supported by the database management system.

# Functional Details

**1. Preprocessing details:**

There should be a working database and we will need an [ODBC](http://www.easysoft.com/developer/interfaces/odbc/index.html) driver to connect to a database. ODBC drivers should work equally well on MS Windows and other operating systems with some minor alterations.

* **DSN (Data Source Name):**

The system DSN provides more tightened security and better performance. Data Source Name provides connectivity to a database through an ODBC driver. The DSN contains database name, directory, database driver, user ID, password, and other information. Once you create a DSN for a particular database, you can use the DSN in an application to call information from the database.

* There are **three types of DSN**:

(1) **System DSN** -- can be used by anyone who has access to the machine. DSN info is stored in the registry.  
(2) **User DSN** -- created for a specific user. Also stored in the registry.  
(3) **File DSN** -- DSN info is stored in a text file with .DSN extension.

* Using an ODBC DSN (Data Source Name) is a two step process. :

1) We must first create the DSN via the "ODBC Data Source Administrator" program found in our computer's Control Panel (or Administrative Tools menu in Windows 2000). We should make sure that we are creating a SYSTEM DSN (not a USER DSN) .

2) Then use the following connection string - with our own DSN name of course.

DSN:

oConn.Open "DSN=mySystemDSN;" & \_

"Uid=myUsername;" & \_

"Pwd=myPassword"

File DSN

oConn.Open "FILEDSN=c:\somepath\mydb.dsn;" & \_

"Uid=myUsername;" & \_

"Pwd=myPassword"

**2. Input details:**

Xml File: It table name and column names with their data type and format.

Datatypes supported:

| **Datatype** | **Description** |
| --- | --- |
| CHARACTER ARRAY, STRING | Names, phone numbers, email addresses, cities, states, provinces, counties, dates, street addresses, alphanumeric strings etc can be given by user with the data type as variable length character array (ex. VARCHAR (SIZE) in ORACLE) which will be stored in the STRING/ CHARACTER ARRAY in C++. |
| NUMBER,LONG,FLOAT,DOUBLE | Phone numbers, roll number, pin code such integer values can be given by user with the data type number(ex. NUMBER(SIZE) in ORACLE) which will also be stored in the STRING/ CHARACTER ARRAY in C++. |

**3. Processing Details:**

(1) The input is given by the user in XML file. The XML file is parsed so as to get the details of the table (table name, column name, data type and format).

(2) The data (random or user specific) is generated according to the details given in the xml file.

(3) Then the data generated is stored in the flat file format.

(4) The data stored in the flat file is then converted into tables.

**4. Output details:**

(1) After the generation of the data the output would be generated in the csv format where each entry will be separated with the comma in relationship with the input.

(2) The conversion of data in the bulk load would be an option to convert it into the table and it well structured form in correspondence with the Xml document.

**5. Post processing:**

Bulk Loading:

The Bulk Loader is a facility that allows you to populate database tables from flat files. To use this facility, one requires

(i) Flat file(s)

(ii) A control file that tells ORACLE how to "map" fields of the flat file to columns of an RDBMS table.

Many a time the source XML document can be large, the entire document is not read into memory for bulk load processing. Instead, they interpret the XML data as a stream and read it. As the system reads the data, it identifies the database table(s), generates the appropriate record(s) from the XML data source, and then sends the records to SQL Server for insertion.

For example, the following source XML document consists of <Customer> elements and <Order> child elements:

<Customer ...>

<Order.../>

<Order .../>

...

</Customer>

...

As XML Bulk load reads the <Customer> element, it generates a record for the **Customer**table. When it reads the </Customer> end tag, XML Bulk Load inserts that record into the table in SQL Server. In the same manner, when it reads the <Order> element, XML Bulk Load generates a record for the **Order**, and then inserts that record into the SQL Server table upon reading the </Order> end tag.

**6. Unit testing:**

**Unit testing** is a test validates that individual units of source code are working properly. A unit is the smallest testable part of an application. In procedural programming a unit may be an individual program, function, procedure, etc., while in object-oriented programming, the smallest unit is a method, which may belong to a base/super class, abstract class or derived/child class.

Ideally, each test case is independent from the others.

|  |  |
| --- | --- |
| Purpose | Test the parsed xml file |
| Pre-requisite | The input xml file is provided by the user |
| Test Data | 1. To test if the table name is provided in the xml file.  2. To test if the column names with the data type and format are provided.  3. To test if parsed data is stored in the proper data structure. |

Test case 1:

Test case 2:

|  |  |
| --- | --- |
| Purpose | To test the generated data |
| Pre-requisite | 1. The xml file is provided by the user.  2. The xml file is parsed.  3. The data is generated randomly or by some programming logic |
| Test Data | 1. To test whether data is generated as per the input given by the user.  2. To test if parsed data is stored in the proper data structure. |

Test case 3:

|  |  |
| --- | --- |
| Purpose | To test the database(tables) created |
| Pre-requisite | 1. The xml file is provided by the user.  2. The xml file is parsed.  3. The data is generated randomly or by some programming logic.  4. The data is inserted into the table. |
| Test Data | 1. To test if the database created is in the format given by the user. |

# Enhancements

(1) These tool may come with a pre-built library of the most commonly used data generation rules by the user, but also will allow the flexibility to create, modify, save and share the Auto Generator rule library. This Tool will enable us to meet our specific software demands and save significant development time.

(2) The parallel load feature would allow generating data into several tables simultaneously or loading data into a single table via several parallel threads. Depending on the complexity of the data, this feature will increase performance of the data generation process up to specified times.

(3) Further bulk load would increase the speed of the data load process and would generate the data in table format.

# Limitations

(1) Some of the problems with proper data generation include the ability to generate proper sampling of data from a variety of sources, ability to generate large data.

(2) Maintaining the relationship between two tables would be a challenging task and hence if not handled properly the system will generate vague output

(3) The Data Generator Tool supports multithreading for parallel processing of data but it does have limitations. Multiple threads can interfere with each other when sharing hardware resources.

(4) The Data Generator Tool does not support Unicode which is used for encoding, to represent text in other forms. The output data cannot be encoded using Unicode and will remain in the form it is.

(5)The constraint for the columns have to be followed else the data generation fails.

# 

# Appendix-A

Source and binary distributions installation instructions are available for the following platforms:

* [Windows](http://xerces.apache.org/xerces-c/install-3.html" \l "Windows)
* [UNIX/Linux/Mac OS X](http://xerces.apache.org/xerces-c/install-3.html#Unix)
* [Cygwin](http://xerces.apache.org/xerces-c/install-3.html#Cygwin)

WINDOWS:

The Xerces-C++ source is available in the source distribution: xerces-c-3.1.1.zip.

Install the Xerces-C++ source distribution by using unzip on the xerces-c-3.1.1.zip archive in the Windows environment. You can use WinZip, or any other UnZip utility.

This creates the 'xerces-c-3.1.1' sub-directory containing the Xerces-C++ source distribution.

If we need to build the Xerces-C++ source after installation we follow the [Build Instructions](http://xerces.apache.org/xerces-c/build-3.html).

Xerces-C++ source distribution comes with Microsoft Visual C++ projects and solutions. The following describes the steps you need to build with this compiler.

To build Xerces-C++ from the source distribution you will need to open the solution containing the project. The solutions containing the Xerces-C++ project files are in the following sub-directories in the xerces-c-3.1.1 directory:

Once you have the solution open, you need to build the project named XercesLib. You can select Debug/Release, Static/DLL, and, for VC8, VC9, and VC10, 32/64 bit builds using the Configuration Manager dialog. You can also select whether the Xerces-C++ library should use ICU for transcoding.

When building your own applications you need to make sure that you are linking your application with the xerces-c\_3.lib (Release) and/or xerces-c\_3D.lib (Debug) libraries (or the static versions of them) and also that the associated DLLs are somewhere in the executable/DLL search path (PATH).