Project: Stage 2
Data Storage, Client-Server Architecture

Due: 11:59pm, Monday, November 15, 2010.

Outline

On this stage of the project, you will do the following:

- Implement the basic client-server architecture for MiniNXBase.
- Implement the algorithms for indexing a full XML document.
- Implement an XML generator.

Client-Server Architecture

MiniNXBase shall be implemented as a database server program on top of the storage facilities designed by you on Stage 1 of the project. The MiniNXBase server, upon being started, shall initialize its data structures and then start listening for commands sent to it from client programs. When a command comes, the server shall spawn a thread dedicated to processing the command.

In addition to the server, a simple command-line client program, XpathPlus shall be implemented to test MiniNXBase. XpathPlus shall allow the user to enter one of the supported commands. Once the command is entered, XpathPlus shall contact the MiniNXBase server, pass the command to it, wait until the answer comes from MiniNXBase and display the answer.

XpathPlus shall have two modes of operation, *interactive* and *batch*.

The *interactive mode* of XpathPlus shall be invoked by running XpathPlus without any additional parameters:

```
> java XpathPlus
```

In this mode, XpathPlus shall display a command prompt to the user, read commands, pass them to the MiniNXBase server and output the results. XpathPlus shall run in this mode until a QUIT command is entered.

The *batch mode* of XpathPlus shall be invoked by running XpathPlus with command line arguments representing a sepcific XpathPlus command:

```
>java XpathPlus CREATE NewRepository
```

In this case, XpathPlus upon startup shall parse and execute the command passed through the command-line parameters, outputs the results and stops execution.

We provide the following code for you:

- for MiniNXBase, we will provide the generic server code. This code, the `main` method for MiniNXBase, which includes the main server loop, as well as the code for spawning threads, will have to be integrated with your code from Stage 1, and the code for implementing data insertion and DOM operations.
- for XpathPlus, we will provide the generic client code. The code will start the client, wait for input, contact the server and pass the input to the server. You may still have to write some code for the client to make it do error-checking.

Stage 2 Functionality

At this stage, you will implement the methods for dealing with indexing XML documents and overall management of XML indexes. Please note, that you are expected to use the Stage 1 APIs to store and retrieve data. No calls to NEUSTORE API should be made from the classes/methods you develop for this stage. (If some functionality is missing from the API, add it to the Stage 1 layer and use in your new code).

Indexing XML documents

MiniNXBase shall support mulptiple XML repositories at the same time. Each XML repository has a unique, user-given name and can store one XML file. Via XpathPlus the user can create empty repositories, index XML documents into repositories, clear and delete repositories, output their full contents and query them.

For Stage 2 you have to implement all the functionality mentioned above, except for querying. All functionality implemented at this stage shall be invoked via XpathPlus commands. You will implement the following commands.

- CREATE command. The XpathPlus syntax of this command is

  CREATE RepositoryName;

This command should result in creation of an empty XML repository with the name RepositoryName. MiniNXBase shall keep track of the list of all XML repositories created. The name of the repository shall be unique, If and XML repository with the nameRepositoryName already exists, MiniNXBase shall notify the user of that.

- CLEAR command. The XpathPlus syntax of this command is

  CLEAR RepositoryName;

  This command removes all content from the repository RepositoryName, but keeps the repository itself. At the end of the command the repository must be empty. If the repository does not exist, return an error.

- DELETE command. The XpathPlus syntax of this command is

  DELETE RepositoryName;

  This command deletes the repository. If the repository does not exist, notify the user.

- LIST command. The XpathPlus syntax of this command is

  LIST;

  This command lists all repositories available on the MiniNXBase server. For each repository, the command shall list (a) its name and (b) its state.

  – If the repository is empty, return `"EMPTY"` as the state of the repository.
  – If a disk XML file is stored, return the path/filename of the stored file.
  – IF an XML stream is stored, return the word `"STREAM"` followed by the name of the root element.

  E.g.,

```
XpathPlus> LIST;

NewRepository  EMPTY
DBLP  ~/dekhtyar/dblp/dblp.xml
StreamTest   STREAM <stream>
```

  Please note, that the list of repositories needs to be stored *persistently* and should be easily accessible by the MiniNXBase server.

- INSERT FILE commad. The XpathPlus syntax of this command is

  INSERT FILE RepositoryName filename;

  This command inserts/indexes given file (filename) in the XML repository RepositoryName. This command shall detect and return the following errors:

- No Repository. Repository with the name RepositoryName does not exist on the server.
- Repository Full. Repository RepositoryName exists, but is already occupied by an index of a different XML document.
- File not found. Presented XML document does not exist on disk.
- Not an XML file/Parser Error. Presented file is either not an XML file, or is not a well-formed XML file.

Note, that the filename parameter may contain path information. E.g., the following command:

```
INSERT FILE NewRepository ../docs/myFile.xml
```

shall load into the repository `NewRepository` the content of the `myFile.xml` file located in the directory `docs` which is a sibling of the directory from which XpathPlus is running.

- INSERT XML commad. The XpathPlus syntax of this command is

  INSERT XML RepositoryName XMLData;

  This command inserts in the XML repository RepositoryName the xml data provided in the XMLData argument. This command shall detect and return the following errors:

  - No Repository. Repository with the name RepositoryName does not exist on the server.
  - Repository Full. Repository RepositoryName exists, but is already occupied by an index of a different XML document.
  - Not XML/Parser Error. Presented XMLData is not a well-formed XML file.

  For example, the following command

```
INSERT XML StreamTest <a><b>Hello, World!</b><c>Test</c></a>
```

  will index the XML document

```
<a>
  <b>Hello, World!</b>
  <c>Test</c>
</a>
```

  and store it to the repository `StreamTest`.

  Note: this command is present to allow you to conduct simple tests of your DBMS without the need to create multiple small XML files.

- PRINT command. The XpathPlus syntax of this command is

  PRINT RepositoryName;

  The command retrieves the XML document from the specified repository and prints to to screen. The following errors shall be detected and returned:

  - No Repository. Repository with given name does not exist.
  - Repository Empty. Repository with the given name exists, but contains no XML file. Output `"Repository <RepositoryName> is EMPTY"` error message (replace `<RepositoryName>` with the actual name of the repository).

- STATS command. The XpathPlus syntax of this command is

  STATS RepositoryName;

  This command outputs the basic statistics about the repository. The following information shall be returned:

  1. The name of the repository and its status — same as in the LIST command. If the status of the repository is `EMPTY`, the this is the only information returned. Otherwise, `MiniXBase` shall provide the following information.
  2. Name of the root node (for STREAM repositories it will be a repeat of the name in the status line).
  3. Total number of unique element names. (i.e., the number of records in the top tier of the ElementIndex structure.)
  4. Total number of element nodes. (i.e., the number of records in the StructureIndex structure)
  5. Total number of text nodes (i.e., the number of records in the ContentIndex.
  6. Total number of attribute nodes (i.e., the number of records in the AttributeIndex structure.
  7. Number of disk blocks occupied by each index structure. This includes **all** disk blocks associated with the structure, from the file header page, and on.

  For example, the output of this command may be fomatted as follows:

```
XpathPlus> STATS MyRepository;

Repository MyRepository  test.xml

            Root : <root>
 Unique Elements : 5
   Element Nodes : 24
      Text Nodes : 12
```

```
Attribute Nodes : 5

Disk Pages:
   ElementIndex : 7 pages
 StructureIndex : 3 pages
   ContentIndex : 2 pages
 AttributeIndex : 2 pages
```

Note: STATS is supposed to be resolved quickly. You are not expected to scan the entire data structure each time STATS command is issued. Rather, feel free to precompute this information when creating XML documents and repositories and to store it keyed by repository name (or in the file header page/pages of the index structures).

- QUIT command. The XpathPlus syntax of this command is

  QUIT;

  Quit XpathPlus.

Your commands/keywords can be case-sensitive or case-insensitive – the decision is left up to each group. The command parameters, however, must be case-sensitive.

What to implement

Your implementation of XpathPlus must understand and properly pass to the MiniNXBase server all the commands above. MiniNXBase must contain implementations for each operation above.

CREATE command creates an empty repository — i.e., you can implement it as a sequence of calls to appropriate constructors for the data/index files you have implemented for the Stage 1.

INSERT commands are the most important on this list. You are given an XML document, either as an I/O stream or as a file, and are asked to completely index it. You shall use Java SAX parser functionality (e.g., org.xml.sax package) to parse incoming XML content. While parsing, your code shall properly handle incoming information, extract all data necessary for indexing the XML document, and insert records about individual nodes, attributes and portions of the content into the index structures. For INSERT XML commands you can "cheat" a bit by "sticking" the XML provided to you in the command into some form of an input stream (e.g., by saving it into a temporary file), and running the SAX parser on that stream.

PRINT command requires to you develop XML generation code during this stage of the project. Note, that in order to do this, you, essentially, need to conduct a full scan of the XML repository, and generate an XML string representing the data stored in the repository. You are not expected to replicate the carbon copy of the original file you were storing (NOR SHALL YOU simply print out the file itself): comments, DTD information and whitespace can/will be ignored. You are responsible for:

- proper XML structure (XML tree for your output must coincide with the XML tree for the original XML file inserted into the repository);
- meaningful indentation (make your output human-readable)

For example, the following command

```
PRINT StreamTest
```

can result (assuming the `INSERT XML` command from above) in the following output:

```
<a>
  <b>Hello, World!</b>
  <c>Test</c>
</a>
```

Note, that you will have to use XML generation code in stage 3 of the project to output results of your queries, so it makes sense to make your XML generation methods flexible enough to output the contents of *any node* in the XML tree. (For stage 3, special rules may apply to display of attribute nodes only, or just the text, but for this stage, you can simply implement XML generation for any record from `StructureIndex`.)

Grading

The work of indexing methods, as well as the overall work of the XpathPlus client and MiniNXBase server will be tested using scripts running XpathPlus in batch mode. Additionally, some "face-to-face" tests with XpathPlus in the interactive mode will be conducted.

Each test item will recieve a weight/percentage towards total grade for the stage. The grade for the stage is the sum of all test items correctly executed. (a test item may be a portion of the test, not the full test).

Coding and Submission

Since NEUStore encourages use of Eclipse as a programming environment, so do I, although, it is not necessary. All grading will take place on CSL machines so, please, make sure you test your code on one of the machines there. Your code will not have GUI components, so it can be run (outside of Eclipse, of course) from a command prompt via ssh.

While code is not graded, we reserve the right to examine it. Please comment adequately and make effort to make your code readable (your teammates will be the first to benefit from this).

If your submission requires any explanations (e.g., you have changed the indexing structure), put them in a README file, stored in the root of your submitted archive (see below). Also, make certain the name of the team and the names of all team members appear on every text file and any Java file submitted.

You should have one submission per team. Zip and gzipped tar are the allowed submission formats. Name your archive project-stage2-<team>.ext, where <team> is the name of your team and <ext> is either zip or tar.gz.

Submit your work using handin:

```
handin dekhtyar project02 <archive> README
```