Import - Immutable Tables

Table Import is a procedure modifying otherwise immutable tables. This procedure is not expected to deal with large quantities of records because it is primarily intended for small changes and updates in the following tables: plants, nearest bigger seats, phytochoria, territories, and metadata.

Introduction

Since the Import to Immutable tables was never meant to process excessive amounts of data, DOM4J can be safely used. It is not a problem to parse a file containing several thousands records. The framework resembles an "inverse" Builder pattern: instead of a Builder there is a Parser. The difference is that the Director asks the Parser to return reconstructed records, and stores them into the database. It's the data flow what is inversed.

The format of the file is based on XML and allows adding, deleting, and updating of records.

Participants

Data Holder

Data holder is designed for communication between the Parser and the Director (the Import task in this case). It stores the data retrieved from the file and the intention with this data.

The source is in the file net.sf.plantlore.client.tableimport.DataHolder.java.

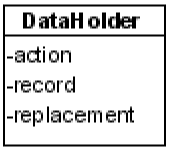
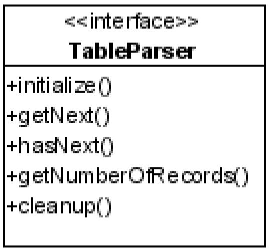


Table Parser

The Table Parser provides the interface for parsers retrieving data from tables. Table Parser must be able to fetch the next record if there is any. Implementation of other methods are optional.

The source is in the file net.sf.plantlore.client.tableimport.TableParser.java.

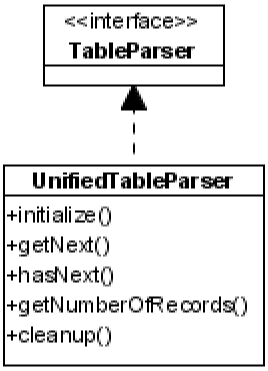


Unified Table Parser

Unified Table Parser can parse and reconstruct any record from the Immutable table. This parser can be used to parse the difference list of plants, as well as nearest bigger seats, territories, villages, and metadata. This is achieved because of the Java reflection API. The **initialize()** method returns the table into which the records will belong, but it is not necessary as the type of the record may be identified any time.

The Unified Table Parser uses DOM4J to parse the XML file which is why all methods **hasNext()**, **getNext()**, and **getNumberOfRecords()** are supported.

The source code is located in the file net.sf.plantlore.client.tableimport.parsers.UnifiedTableParser.java.



DBLayerUtils

The Database Layer Utils is a set of high-level methods incorporating all rules for records inserting, updating, and deleting posed by the Database Model. It operates with the Database Layer and simplifies and unites most common operations.

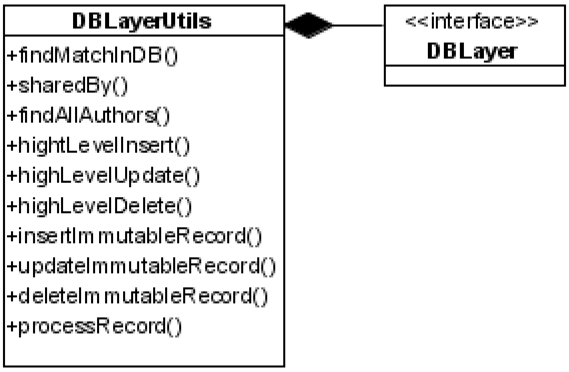


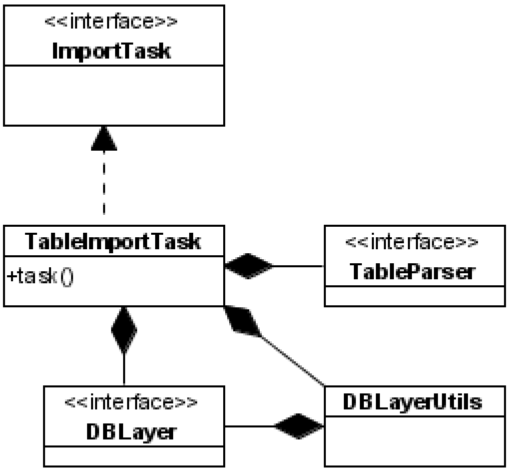
Table Import Task

Table Import task is a Plantlore Task that must be dispatched accordingly. To learn more about tasks see the Section **Tasks and Dispatcher**.

The Table Import task acts as a Director in the Builder Design pattern but it asks the Parser for records instead of passing them to it. The Table Import task then processes these records accordingly. It can add, delete and update them, while maintaining the reference integrity of the database.

It also very carefully monitors the state of the heap, because DOM4J can be very memory hungry.

The source is in the net.sf.plantlore.client.tableimport.TableImportTask.java.

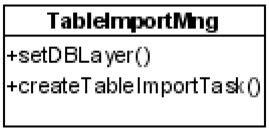


After the Table Import task finishes, the Application is notified so that changes made to the

Immutable tables are backpropagated to the rest of the Application, so that the User is presented with up-to-date data. To study the propagation of changes see the Section **GUI Communication Layer**.

Table Import Manager

The Table Import Manager is the Table Import Task factory. It is used to store partial information about the future Table Import Task and once all the information is supplied, it can create a new Table Parser and Table Import Task.

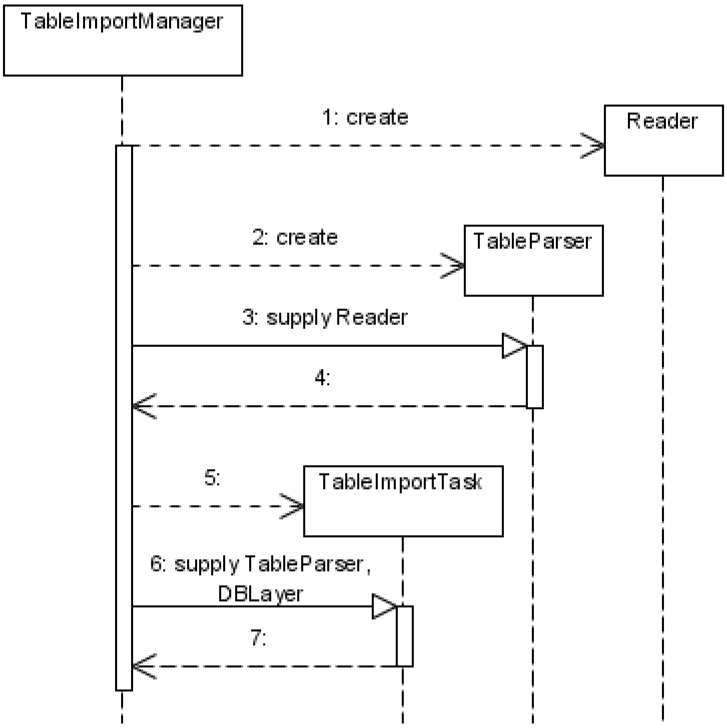


Interaction

In this section we describe the creation of a new Table Import Task and how the Table Import works.

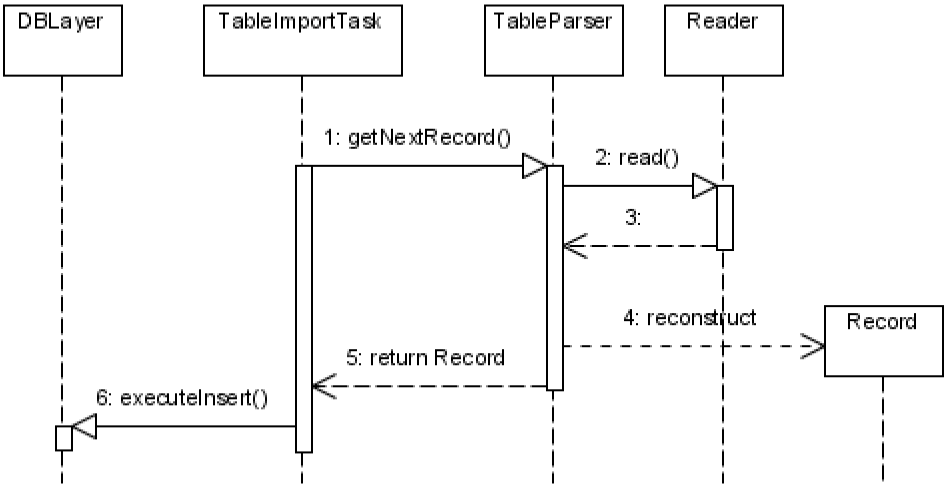
Creation of a New Table Import Task

Creation of all participants.



Start of the Import Task

The next picture depicts the process of obtaining, reconstructing, and processing of a record.


DBLayer
TableImportTask
TableParser
Reader
1: getNextRecord()
2: read()
3:
4: reconstruct
Record
5: return Record
6: executeInsert()