Mapper

Implement the following interface. Optimise implementation for performance. Invalid operations (based on contract definition) should result in an exception.

Write unit tests.

```
/// <summary>
/// Defines mapping contract between objects.
/// Objects are identified by an integer identifier.
/// Valid identifiers are in range [1..131072]
/// A parent may have multiple children.
/// A child may have only one parent.
/// </summary>
public interface IOneToManyMapper {
    /// <summary>
    /// Defines a new mapping between parent and child
    /// </summary>
    /// <param name="parent">Parent identifier</param>
    /// <param name="child">Child identifier</param>
    void Add (int parent, int child);

    /// <summary>
    /// Removes all mappings for a valid parent
    /// </summary>
    /// <param name="parent">Parent identifier</param>
    void RemoveParent (int parent);

    /// <summary>
    /// Removes a mapping for a valid child
    /// </summary>
    /// <param name="child">Child identifier</param>
    void RemoveChild (int child);

    /// <summary>
    /// Returns all (immediate) children for a given parent.
    /// If there are no mappings for the parent, empty set is returned
```

```
        /// </summary>
        /// <param name="parent">Parent identifier</param>
        /// <returns>Children identifiers</returns>
        IEnumerable<int> GetChildren (int parent);

        /// <summary>
        /// Returns a parent for a given child.
        /// If there is no mapping for a child, returns 0
        /// </summary>
        /// <param name="child">Child identifier</param>
        /// <returns>Parent identifier</returns>
        int GetParent (int child);

        /// <summary>
        /// Changes the valid object identifier for a given parent.
        /// All mappings between old identifier should be migrated to the new identifier
        /// </summary>
        /// <param name="oldParent">Current parent identifier</param>
        /// <param name="newParent">New parent identifier</param>
        // void UpdateParent (int oldParent, int newParent);

        /// <summary>
        /// Changes the valid object identifier for a given child.
        /// The parent for the child should be maintained.
        /// </summary>
        /// <param name="oldChild">Current child identifier</param>
        /// <param name="newChild">New child identifier</param>
        // void UpdateChild (int oldChild, int newChild);
    }
```

PaperRulez

PaperRulez Ltd is a company that handles paper documents for its clients and processes them as contracted. The typical contracts require company to:

- Report if documents contain certain phrases
- Convert document to certain formats
- Send documents to certain recipients
- Etc

At the moment, the processing is done manually. However, with the significantly increasing number of clients and documents, company had to invest in automated solution. And this is where your help is needed.

Design and implement the part of the system that will (for a predefined list of clients) perform following steps:

- Load a document – they will be already scanned and prepared for further processing
- Process the document as required – in future, a document may be processed in multiple ways but for now only one process per document
- Store the processing results – each result may be stored in a different way
- Remove successfully processed file – in future, the common final step may be more complex

The documents are stored in folders. Each client has its own folder. The clients are known upfront. Each document starts with a text line, ended with single '\n' character, followed by the content of certain type (text, binary, etc).

Each document will have specific naming convention: **<document-id>_<document-name>.<content-type>**

The first line in each document defines what must be done with it. The line has following structure:

<type-of-processing>|<parameters>

Implement only one type of processing: **lookup**. Its parameters are words, comma-separated, that must be looked up in the document (full words, case insensitive). It is assumed that all such documents will be text files.

Once file is processed, the following interface should be used to record the processing result:

```
  public interface ILookupStore {
        /// <summary>
        /// Records set of keywords identified in the given document for a given client
        /// </summary>
        /// <param name="client">Client identifier</param>
        /// <param name="documentId">Document identifier</param>
        /// <param name="keywords">Enumeration of unique keywords found in the document, in any
order</param>
        void Record (string client, string documentId, IEnumerable<string> keywords);
    }
```

Try to implement as much as possible but code does not need to run or be complete. Instead, focus on good coding practices, ability to enhance the future functionality and code testability.