http://www.egovframe.go.kr/wiki/doku.php?id=egovframework:rte2:brte:batch_core:item_reader

ItemReader

Outline

ItemReader reads only one item per attempt and returns null when the items are read up.

Description

ItemReader supports various data types such as Flat File, XML and Database.

- Flat File: ItemReader for flat file reads the string of data identified by transfixed data field and special characters.
- XML : ItemReader for XML processes XML for isolated parsing, mapping and validation. The input data can effectively validate XML files against XSD schema.
- Database: You can return 'resultset' for access by way of mapping in the database resource, in the form of object ItemReader. The basic SQL ItemReaders calls RowMapper that returns objects.

See the following for how ItemReader interface is comprised of:

```
public interface ItemReader<T> {

    T read() throws Exception, UnexpectedInputException, ParseException;

}
```

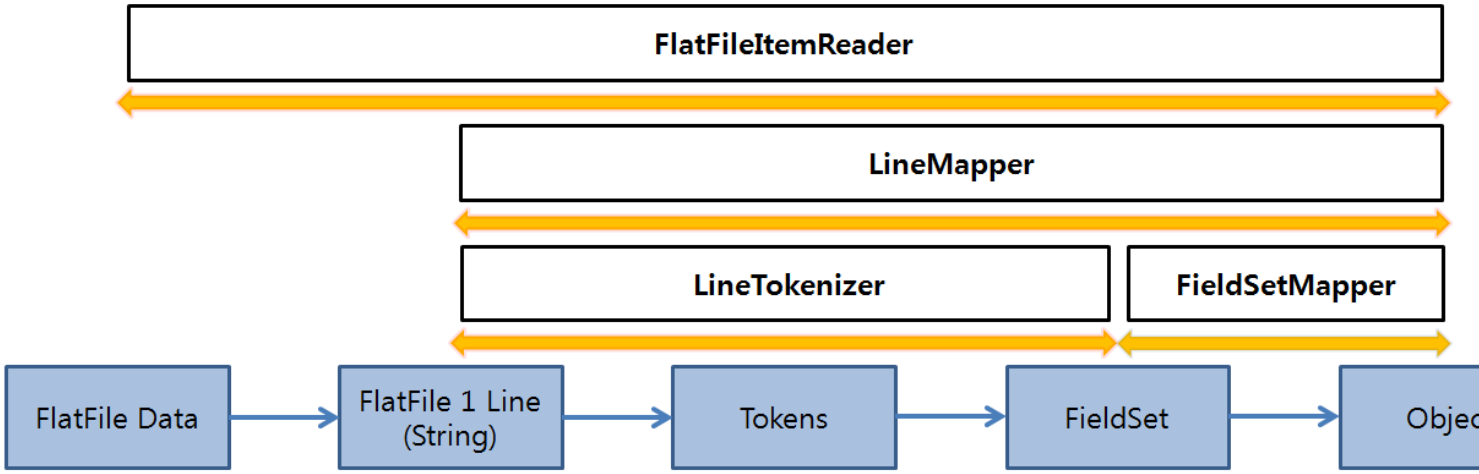
Being an essential method of ItemReader, the method 'read()' returns a single item per attempt and 'null' when all items are returned. For the purpose of this method, item refers to a line in a flat file, a line in a database and an element in an XML file.

FlatFile ItemReader

Comprising a 2-dimensional data, a flat file is read by way of the class FlatFileItemReader that offers reading and parsing in the Spring Batch Framework.

FlatFileItemReader

FlatFileItemReader is dependent on Resource, LineMapper, FieldSetMapper and LineTokenizer which defines FlatFileItemReader as either delimiter or fixed-length line.



Category	Data	Description
LineMapper	A line (String) in flat file → Object	Conversion of a line (String) read out of the flat line into an object (inclusive of LineTokenizer and FieldSetMapper)
LineTokenizer	String → Tokens → FieldSet	Tokenization of a line (String) read out of a flat file by way of delimiter or fixed length line, followed by conversion into FieldSet - **DelimitedLineTokenizer** : Tokenization of a line (String) segmented by unit of delimiter. - **FixedLengthTokenizer** : Tokenization of a line (String) segmented by unit of fixed length.
FieldSetMapper	FieldSet → Object	Conversion of FieldSet data into an object.

See the following examples for how FlatFileItemReader, segmented by either delimiter or fixed length, is dependent on FlatFileItemReader, LineMapper, LineTokenizer and FieldSetMapper:

Tokenization	Settings
Delimiter	<bean id="itemReader" class="org.springframework.batch.item.file.FlatFileItemReader" scope="step"> <property name="resource" value="#{jobParameters[inputFile]}" /> <property name="lineMapper"> <bean class="org.springframework.batch.item.file.mapping.DefaultLineMapper"> <property name="lineTokenizer"> <bean class="org.springframework.batch.item.file.transform.DelimitedLineTokenizer"> <property name="delimiter" value=","/> <property name="names" value="name,credit" /> </bean> </property> <property name="fieldSetMapper"> <bean class="org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper"> <property name="targetType" value="org.springframework.batch.CustomerCredit" /> </bean> </property> </bean> </property> </bean>
Fixed Length	<bean id="itemReader" class="org.springframework.batch.item.file.FlatFileItemReader" scope="step"> <property name="resource" value="#{jobParameters[inputFile]}" /> <property name="lineMapper"> <bean class="org.springframework.batch.item.file.mapping.DefaultLineMapper"> <property name="lineTokenizer"> <bean class="org.springframework.batch.item.file.transform.FixedLengthTokenizer"> <property name="columns" value="1-9,10-11" /> <property name="names" value="name,credit" /> </bean> </property> <property name="fieldSetMapper"> <bean class="org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper"> <property name="targetType" value="org.springframework.batch.CustomerCredit" /> </bean> </property> </bean> </property> </bean>

You need to make sure you configure LineTokenizer and FieldSetMapper as follows:

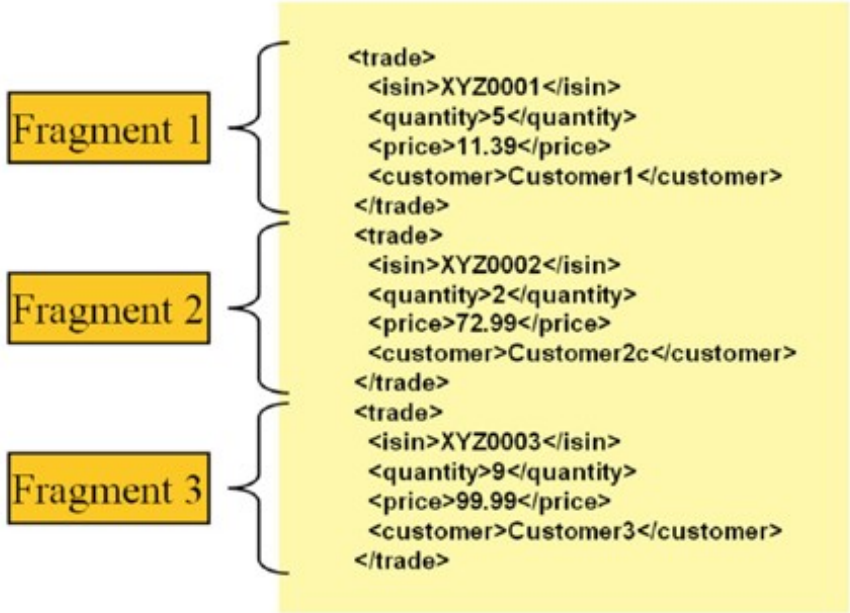
Items Configured	Description
targetType	Represents class VO
names	Represents fields of class VO

✔ Particular attention is needed for configuration to fit the concerned LineTokenizer.

LineTokenizer	Items Configured	Description	Configuration
DelimitedLineTokenizer	delimiter	Base delimiter for tokenization	,
FixedLengthTokenizer	columns	Base length for tokenization	1-9,10-11

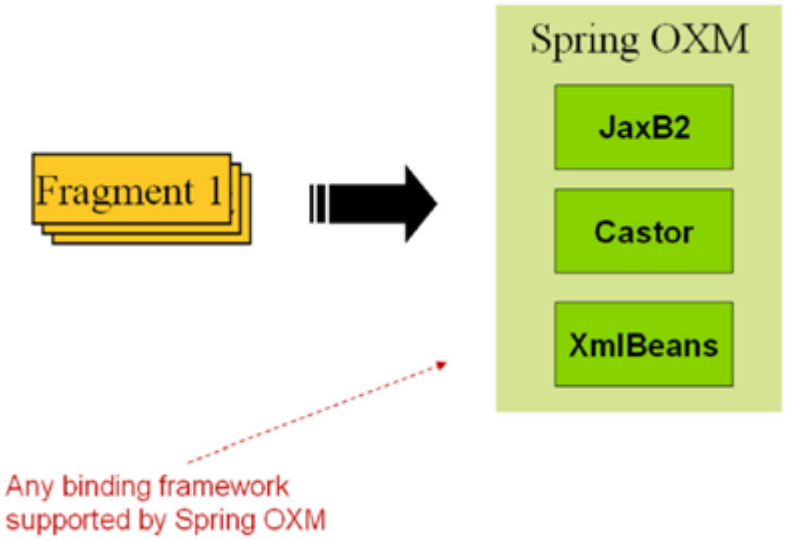
XML ItemReader

In Spring, batch provides the user with transaction infrastructure for mapping in Java object from the XML record read. Though input and output of XML vary by how the file is read and written, the Spring batch XML processing applied are very same. Instead of FieldSets requiring tokenizing in XML processing, Spring assumes a collection of fragments that are equivalent to the individual records.



Note that the tag 'trade' is defined as a 'root element' where the contents between '<trade>' and '</trade>' are deemed a single fragment. While using Object/XML Mapping (OXM) to bind fragments into an object, Spring does not stick to the specific XML binding technology. For instance, Spring delegates binding works to Spring OXM that offers the uniform abstraction to the common OXm technologies. As the dependency upon Spring OXM is optional, you may

choose the Spring batch implement the specific interface. See the following for how XML support is made:



StaxEventItemReader

You can configure how records are to be processed in XML input stream by working StaxEventItemReader. See the following for a set of XML records that can be processed by StaxEventItemReader:

```
<?xml version="1.0" encoding="UTF-8"?>
<records>
    <trade xmlns="http://springframework.org/batch/sample/io/oxm/domain">
        <isin>XYZ0001</isin>
        <quantity>5</quantity>
        <price>11.39</price>
        <customer>Customer1</customer>
    </trade>
    <trade xmlns="http://springframework.org/batch/sample/io/oxm/domain">
        <isin>XYZ0002</isin>
        <quantity>2</quantity>
        <price>72.99</price>
        <customer>Customer2c</customer>
    </trade>
    <trade xmlns="http://springframework.org/batch/sample/io/oxm/domain">
        <isin>XYZ0003</isin>
        <quantity>9</quantity>
        <price>99.99</price>
        <customer>Customer3</customer>
    </trade>
</records>
```

Requirements for XML Record processing are as follows:

- Title of Root Element: The title of root element of the fragment comprising the object to be mapped. The title of root element in the foregoing example is ‘Trade’.
- Resource : The directory (or URL) of data to be read.
- FragmentDeserializer : Unmarshaling provided by Spring OXM that maps XML fragment object.

```
<bean id="itemReader" class="org.springframework.batch.item.xml.StaxEventItemReader">
    <property name="fragmentRootElementName" value="trade" />
    <property name="resource" value="data/iosample/input/input.xml" />
    <property name="unmarshaller" ref="tradeMarshaller" />
</bean>
```

See the following for how XStreamMarshaller can be used. With XStreamMarshaller intended to submit the alias in the form of a map comprising keys and values which are referred to when defining the title and type of element, Spring configuration utility is made available to refer to the alias in the configuration file as follows:

```
<bean id="tradeMarshaller" class="org.springframework.oxm.xstream.XStreamMarshaller">
    <property name="aliases">
        <util:map id="aliases">
            <entry key="trade" value="org.springframework.batch.sample.domain.Trade" />
            <entry key="price" value="java.math.BigDecimal" />
            <entry key="name" value="java.lang.String" />
        </util:map>
    </property>
</bean>
```

ItemReader continues reading XML resources until reaching the new fragment (recognized by the tag identity), generates an XML document independent to the fragment and submits the XML document generated in deserializer. See the following for the relevant coding:

```
StaxEventItemReader xmlStaxEventItemReader = new StaxEventItemReader()
Resource resource = new ByteArrayResource(xmlResource.getBytes())

Map aliases = new HashMap();
aliases.put("trade","org.springframework.batch.sample.domain.Trade");
aliases.put("price","java.math.BigDecimal");
aliases.put("customer","java.lang.String");
Marshaller marshaller = new XStreamMarshaller();
marshaller.setAliases(aliases);
xmlStaxEventItemReader.setUnmarshaller(marshaller);
xmlStaxEventItemReader.setResource(resource);
xmlStaxEventItemReader.setFragmentRootElementName("trade");
xmlStaxEventItemReader.open(new ExecutionContext());

boolean hasNext = true

CustomerCredit credit = null;

while (hasNext) {
    credit = xmlStaxEventItemReader.read();
    if (credit == null) {
        hasNext = false;
    }
    else {
        System.out.println(credit);
    }
}
```

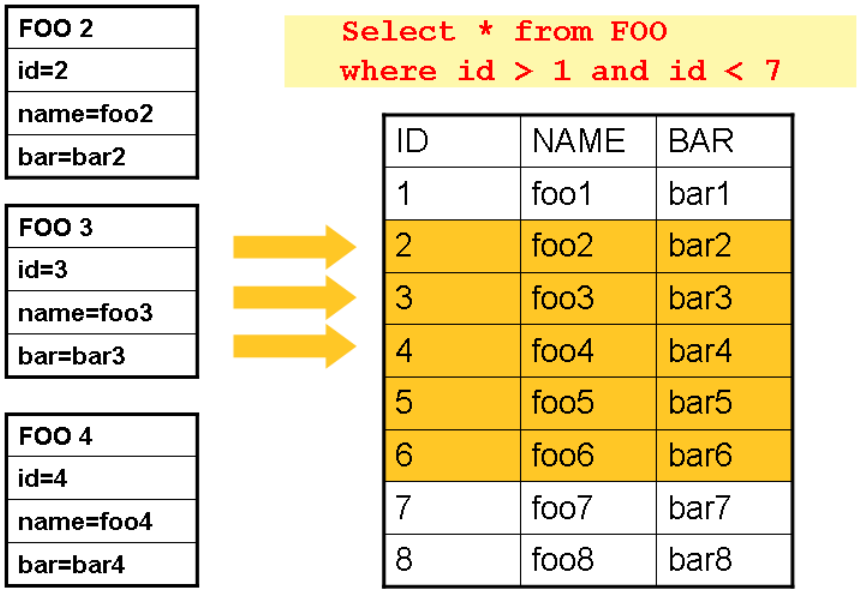
Database ItemReader

Being the core of batch repository mechanism in the greater part of enterprise applications, Database varies batch according to their style. Assuming an SQL statement returns a million lines, a set of results are retained in the memory, whereby Spring batch provides you with ItemReader, cursor-based and paging-based.

Cursor-based ItemReader

Being the very basic of database approach fitting ‘streaming' relational data, Cursor-based ItemReader moves a cursor onto the ensuing line after which ResultSet, required for object-oriented mechanism to work the cursors, transfixes the cursor .
In Spring, ItemReader initializes the cursor, moves the cursor onto the ensuing life when 'read’ is called and returns the mapped object used while processing is underway.

Refer to the following figure for how ItemReader operates. With the table 'FOO' featuring a total of three columns, ID, NAME and BAR, the SQL statement refers to the result of a line where ID is somewhere between 1 and 7. Commenced at ID 2, a cursor is mapped to the object FOO when read() is called and moves onto the ensuing line.



JdbcCursorItemReader

JdbcCursorItemReader is the specific type of ItemReader implementing cursor-based JDBC. Refer to the following example for how JdbcCursorItemReader can be configured. All you need to do is to designate dataSource able to load DB connection and map the queries for the SQL properties and the object ResultSet in rowMapper and the class able to implement rowMapper interface:

```
<bean id="itemReader" class="org.springframework.batch.item.database.JdbcCursorItemReader">
    <property name="dataSource" ref="dataSource"/>
    <property name="sql" value="select ID, NAME, CREDIT from CUSTOMER"/>
    <property name="rowMapper">
        <bean class="egovframework.brte.sample.domain.trade.CustomerCreditRowMapper"/>
    </property>
</bean>
```

Paging-based ItemReader

A query loads the designated size of pages while the database cursor executes a multitude of queries.
Assign the line where the cursor starts and the number of lines to be returned to page are all you need to do to execute queries.

JdbcPagingItemReader

JdbcPagingItemReader is an implementor of Paging-based ItemReader and requires the PagingQueryProvider interface responsible for providing the user with SQL queries to return the lines comprising a page. While implementing OraclePagingQueryProvider, HsqlPagingQueryProvider, MySqlPagingQueryProvider, SqlServerPagingQueryProvider and SybasePagingQueryProvider, it is advised to use SqlPagingQueryProviderFactoryBean that automatically identifies the relevant database and applies the most optimal implementor of PagingQueryProvider, simplifying the configuration.

Refer to the following for how JdbcCursorItemReader can be coded:

```
<bean id="itemReader" class="org.springframework.batch.item.database.JdbcPagingItemReader">
    <property name="dataSource" ref="dataSource"/>
    <property name="queryProvider">
        <bean class="org.springframework.batch.item.database.support.SqlPagingQueryProviderFactoryBean">
```

```
            <property name="selectClause" value="select id, name, credit"/>
            <property name="fromClause" value="from customer"/>
            <property name="whereClause" value="where status=:status"/>
            <property name="sortKey" value="id"/>
        </bean>
    </property>
    <property name="parameterValues">
        <map>
            <entry key="status" value="NEW"/>
        </map>
    </property>
    <property name="pageSize" value="1000"/>
    <property name="rowMapper" ref="customerMapper"/>
</bean>
```

IbatisPagingItemReader

IbatisPagingItemReader is available when iBatis comes into play for data accessing.
While being unable to read the lines in a page, iBatis is capable of adding queries using the standardized variables:

Refer to the following example for how IbatisPagingItemReader can be coded:

```
<bean id="itemReader" class="org.springframework.batch.item.database.IbatisPagingItemReader">
    <property name="sqlMapClient" ref="sqlMapClient"/>
    <property name="queryId" value="getPagedCustomerCredits"/>
    <property name="pageSize" value="1000"/>
</bean>
```

“getPagedCustomerCredits” featuring the attribute queryId used in IbatisPagingItemReader comprises as follows:
ex) MySQL

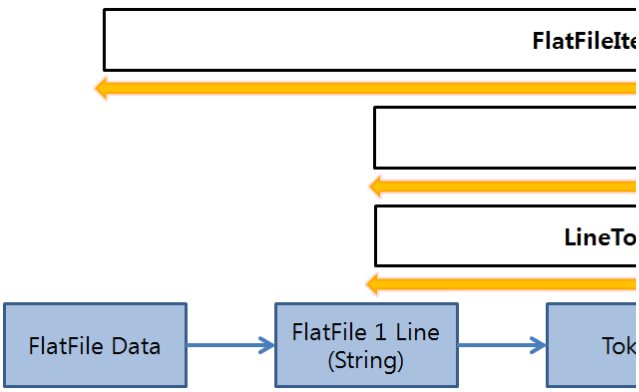
```
<select id="getPagedCustomerCredits" resultMap="customerCreditResult">
    select id, name, credit from customer order by id asc LIMIT #_skiprows#, #_pagesize#
</select>
```

eGovFlatFileItemReader, specific to eGovFramework

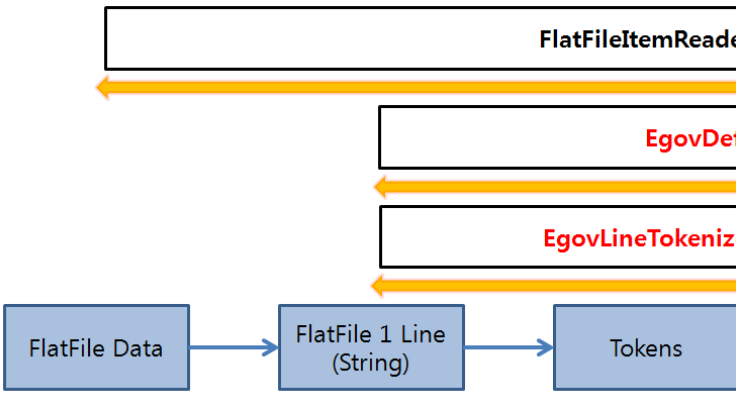
For improved performance in file-based configuration in Spring, eGovFramework offers eGovFlatFileItemReader that has no LineMapper, the largest contributor to latency.

Refer to the following figure to note eGovFramework does not use FieldSet, thus omitting conversion process from Tokens to FieldSet.
When using EgovDefaultLineMapper, EgovLineTokenizer and EgovObjectMapper offered by eGovFramework, you can map Tokens directly into Object.

Structure of Spring's FlatFileItemReader



Structure of eGovFramework's eGovFlatFileItemReader



Improvements	Description
EgovDefaultLineMapper	EgovLineTokenizer and EgovObjectMapper have been altered accordingly to stay in line with the LineMapper procedure controlled. DefaultLineMapper has thus been altered to offer EgovDefaultLineMapper.
EgovLineTokenizer	The interface LineTokenizer has been altered to return FieldSet that is no longer used in eGovFramework.
EgovAbstractLineTokenizer	Being an abstract class, EgovAbstractLineTokenizer solely involves in tokenization as the interface LineTokenizer has been altered into EgovLineTokenizer.
EgovDelimitedLineTokenizer	Features much improved functionality from the Spring's conventional DelimitedLineTokenizer.
EgovObjectMapper	With EgovObjectMapper, you can map the tokenized values to the object without using FieldSet.

Refer to the following XML configurations for how FlatFileItemReader using DefaultLineMapper and FlatFileItemReader using eGov's EgovDefaultLineMapper can be compared:

✔ **Caution!** When using EgovDefaultLineMapper, make sure EgovTokenizer(EgovFixedLengthTokenizer, EgovByteLengthTokenizer, EgovDelimitedTokenizer) and EgovObjectMapper should be applied.
✔ **Caution!** When using EgovObjectMapper, the VO field types are limited to String, int, double, float, long, char, boolean, short and BigDecimal.
✔ **Caution!** Contrary to the Spring's DefaultLineMapper where the attribute ‘names’ in Tokenizer is configured, EgovDefaultLineMapper requires the attribute **’names’ in EgovObjectMapper.**

How to configure EgovObjectMapper

How to configure Delimiter types

You can segregate the strings using the delimiter as a threshold.

Category	Settings
Spring's FlatFileItemReader	<bean id="itemReader" class="org.springframework.batch.item.file.FlatFileItemReader" scope="step"> <property name="resource" value="#{jobParameters[inputFile]}" /> <property name="lineMapper"> <bean class="org.springframework.batch.item.file.mapping.DefaultLineMapper"> <property name="lineTokenizer"> <bean class="org.springframework.batch.item.file.transform.DelimitedLineTokenizer"> <property name="delimiter" value=","/> <property name="names" value="name,credit" /> </bean> </property> <property name="fieldSetMapper"> <bean class="org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper"> <property name="targetType" value="egovframework.brte.sample.domain.trade.CustomerCredit" /> </bean> </property> </bean> </property> </bean>
eGovFramework's eGovFlatFileItemReader	<bean id="itemReader" class="org.springframework.batch.item.file.FlatFileItemReader" scope="step"> <property name="resource" value="#{jobParameters[inputFile]}" /> <property name="lineMapper">

```
            <bean
class="egovframework.brte.core.item.file.mapping.EgovDefaultLineMapper">
                <property name="lineTokenizer">
                    <bean
class="egovframework.brte.core.item.file.transform.EgovDelimitedLineTokenizer">
                        <property name="delimiter" value=","/>
                    </bean>
                </property>
                <property name="objectMapper">
                    <bean
class="egovframework.brte.core.item.file.mapping.EgovObjectMapper">
                        <property name="type"
value="egovframework.brte.sample.domain.trade.CustomerCredit" />
                        <property name="names" value="name,credit" />
                    </bean>
                </property>
            </bean>
        </property>
</bean>
```

How to configure EgovFlatFileItemReader	Description	Example
delimiter	Thresholds of field	, (comma)
type	Represents class VO	org.springframework.batch.CustomerCredit
names	Represents fields of class VO	name,credit

Fixed Length Configuration

Determines the boundary of fields in the strings read and segregates the fields accordingly.

Category	Settings

Spring's FlatFileItemReader	`<bean id="itemReader" class="org.springframework.batch.item.file.FlatFileItemReader" scope="step">`
	`  <property name="resource" value="#{jobParameters[inputFile]}" />`
	`  <property name="lineMapper">`
	`  <bean class="org.springframework.batch.item.file.mapping.DefaultLineMapper">`
	`  <property name="lineTokenizer">`
	`  <bean class="org.springframework.batch.item.file.transform.FixedLengthTokenizer">`
	`  <property name="columns" value="1-9,10-11" />`
	`  <property name="names" value="name,credit" />`
	`  </bean>`
	`  </property>`
	`  <property name="fieldSetMapper">`
	`  <bean class="org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper">`
	`  <property name="targetType" value="egovframework.brte.sample.domain.trade.CustomerCredit" />`
	`  </bean>`
	`  </property>`
	`  </bean>`
	`  </property>`
	`</bean>`

eGovFramework's eGovFlatFileItemReader	`<bean id="itemReader" class="org.springframework.batch.item.file.FlatFileItemReader" scope="step">`
	`  <property name="resource" value="#{jobParameters[inputFile]}" />`
	`  <property name="lineMapper">`

```
<bean
class="egovframework.brte.core.item.file.mapping.EgovDefaultLineMapper">
    <property name="lineTokenizer">
        <bean
class="egovframework.brte.core.item.file.transform.EgovFixedLengthTokenizer">
            <property name="columns" value="1-9,10-11" />
        </bean>
    </property>
    <property name="objectMapper">
        <bean
class="egovframework.brte.core.item.file.mapping.EgovObjectMapper">
            <property name="type"
value="egovframework.brte.sample.domain.trade.CustomerCredit" />
            <property name="names" value="name,credit" />
        </bean>
    </property>
</bean>
</property>
</bean>
```

How to configure EgovFlatFileItemReader	Description	Example
column	Represents the scope of field	1-9,10-11
type	Represents class VO	org.springframework.batch.CustomerCredit
names	Represents fields of class VO	name,credit

How to configure ByteLength

eGovFramework offers EgovFixedByteLengthTokenizer that calculates thresholds of the fields based upon byte strings, with other things being equal to FixedLengthTokenizer.

Category	Settings
eGovFramework's eGovFlatFileItemReader	

```
<bean id="itemReader" class="org.springframework.batch.item.file.FlatFileItemReader"
scope="step">
    <property name="resource" value="#{jobParameters[inputFile]}" />
    <property name="lineMapper">
        <bean
class="egovframework.brte.core.item.file.mapping.EgovDefaultLineMapper">
            <property name="lineTokenizer">
                <bean
class="egovframework.brte.core.item.file.transform.EgovFixedByteLengthTokenizer">
                    <property name="byteEncoding" value="utf-8"/>
                    <property name="columns" value="1-9,10-11" />
                </bean>
            </property>
            <property name="objectMapper">
                <bean
class="egovframework.brte.core.item.file.file.mapping.EgovObjectMapper">
                    <property name="type"
value="egovframework.brte.sample.domain.trade.CustomerCredit" />
                    <property name="names" value="name,credit" />
                </bean>
            </property>
        </bean>
    </property>
</bean>
```

How to configure EgovFlatFileItemReader	Description	Example

column	Represents the length of threshold of field	1-9,10-11
byteEncoding	Represents the type of encoding of byte strings	utf-8
type	Represents class VO	org.springframework.batch.CustomerCredit
names	Represents fields of class VO	name,credit

References

- http://static.springsource.org/spring-batch/reference/html/readersAndWriters.html