Upskilled I.T

Software Design Document

for Wood Stocks Inventory Management System

Prepared by:	Darren Gansberg, Upskilled I.T
Prepared for:	Wood Stocks
Date Submitted:	17/09/2014
Version:	1.0
Date Created:	4/06/2013
Last Updated:	27/12/2014

Document Acceptance and Release

This document is version 1.0 (8/02/2015) of the Software Design Document for the Wood Stocks Stock Control System.

This Software Design Document is a managed document. For identification for changes to this document each page contains a release number and a page number. Changes will only be issued as a complete replacement. Recipients should remove superseded versions from circulation.

This document is authorised for release once all signatures, identified below, have been obtained.

Prepared by:	
Author: Signature: Date:	Darren Gansberg, Upskilled I.T
Approved by:	
Name: Title Signature: Date:	

Version History

Version	Date	Comments
Version 1.0	08/02/2015	

Table of Contents

1 Introduction 7
1.1 Purpose 7
1.2 Scope 7
1.3 Reference material 7
2 Use Cases 8
2.1 Use Case Diagram 8
2.2 Use cases 9
2.2.1 View current count of toys 9
2.2.2 Update current count for stock items 10
2.2.3 Export changes to the inventory data file 11
2.2.4 Sort stock items by Item Code 12
2.2.5 Sort stock items by Current Count 13
2.2.6 Sort stock items by On Order status 14
3 Application Architecture 15
3.1 Presentation Layer description 16
3.1.1 WoodstocksIMSForm object 16
3.1.2 WoodstocksIMSController object 17
3.2 Domain Layer description 17
3.2.1 WoodstocksIMS (Application Façade) 17
3.2.2 Business entities 18
3.3 Data Access Layer description 18
3.4 Sequence diagrams 19
3.4.1 Sequence Diagram for Use Case WIMS-01: View current count of toys 20
3.4.2 Sequence Diagram for Use Case WIMS-02: Importing – File not found 22
..... 22
3.4.3 Sequence Diagram for WIMS-03 andWIMS-04: Changing current count 23
..... 23

3.4.4 Sequence Diagram for WIMS-05 and WIMS-6: Export modified toy data 24
3.4.5 Sequence diagrams for Use Cases WIMS-07, WIMS-08 and WIMS-09: Sorting Toys 26
..... 26
3.4.6 Sequence Diagram for Use Cases WIMS-10, WIMS-11 and WIMS-12: Sorting Toys 27
4 Data Design 28
5 Component Design 30
5.1 Presentation Layer 30
5.1.1 Overview 30
5.1.2 IWoodstocksIMSView 31
5.2 Domain Layer 32
5.2.1 Overview 32
5.2.2 IWoodstocksIMSClient interface 33
Methods 33
Properties 33
Events 34
5.2.3 IWoodstocksIMS interface 35
Methods 35
Properties 37
Events 37
5.2.4 WoodstocksIMS class 38
Constructors 39
Methods 39
Fields 39
5.2.5 WoodstocksIMS State Design 41
5.3 Data Access Layer 43
5.3.1 Overview 43
5.3.2 IWoodstocksToyExporter and ToyExporterCSV 45
Methods 46
Properties 46

Events 46
Constructors 47
Methods 47
Fields 48
Properties 48
Events 48
5.3.3 IWoodstocksToyImporter and ToyImporterCSV 50
Methods 51
Events 51
Constructors 52
Methods 52
Fields 53
Events 53
5.3.4 CSVRecord, CSVHeader and CSVDataRecord 54
Classes 54
Constructors 55
Constructors 55
Methods 55
5.3.5 CSVWriter 56
Constructors 56
Methods 57
Fields 57
Properties 58
5.3.6 CSVReader 59
Constructors 59
Methods 60
Fields 60
Properties 61
5.3.7 CSVParser 62

Classes 62
Enumerations 62
Constructors 63
Methods 63
Fields 63
5.3.8 Data Access Layer Exceptions 65
6 Activity Diagram: Updating the current count of toys 66

1 Introduction

1.1 Purpose

This document describes the architecture and design for the development of the Wood Stocks Inventory Management System (WoodstocksIMS), a software application to be developed by Upskilled I.T, for Wood Stocks that Wood Stocks can use in automation of its daily stock control function.

1.2 Scope

The WoodstocksIMS is to be used by an office administrator at Wood Stocks as part of their stock control functions and enables them to:

- View information regarding stock, (i.e. wooden toys sold by Wood Stocks) including the item code, description, current count and order on order status of the stock.
- Update the current count of stock items.
- Sort stock according to item code, current count and on order status.

The use case requirements for WoodstocksIMS are discussed in more detail in the Business Requirements Document for the Wood Stocks Inventory Management System.

1.3 Reference material

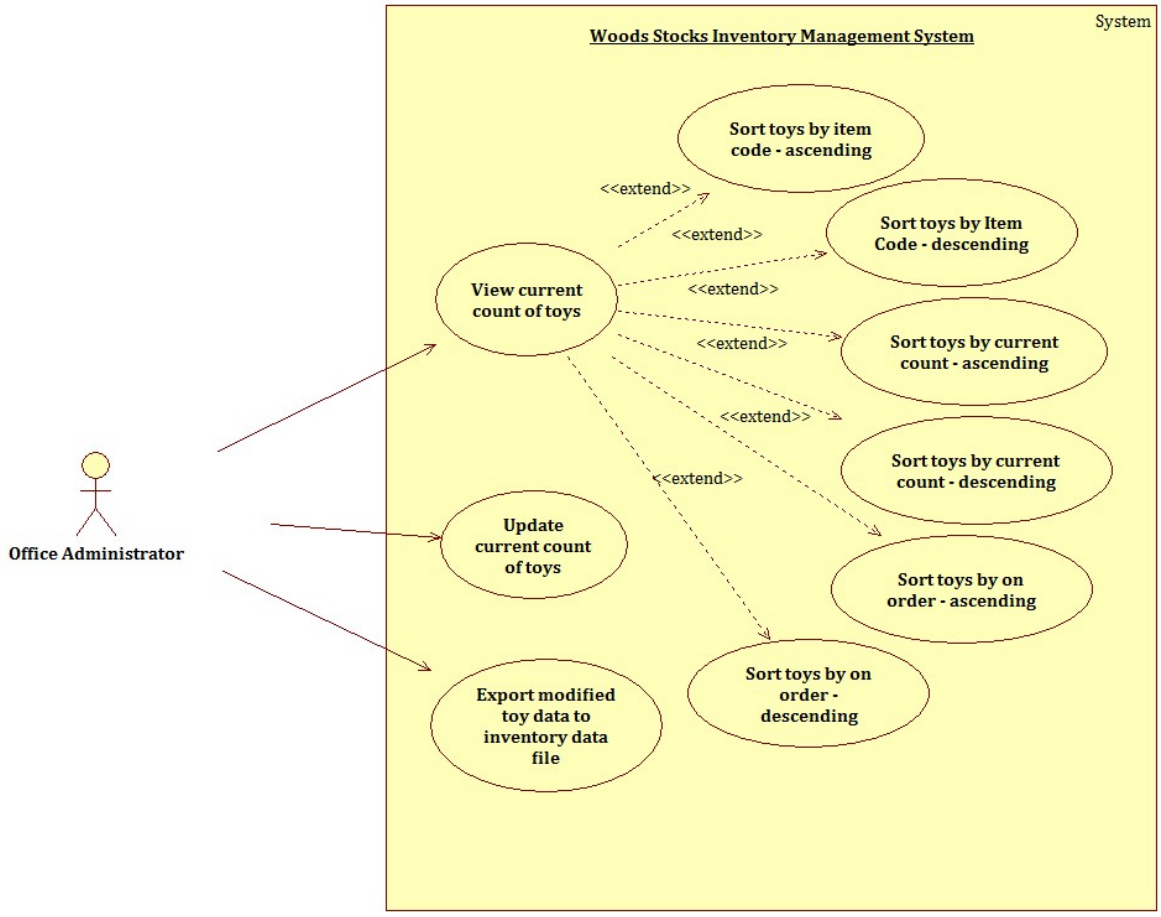
In reading this document regard should be had to:

> Version 1.2 (dated: 7/02/2015) of the Business Requirements Document for the Wood Stocks Inventory Management System.

2 Use Cases

2.1 Use Case Diagram

The following use case diagram presents a visual summary use case scenarios that the WoodstocksIMS must support. The diagram identifies the intended usage of the system by an Office Administrator.



2.2 Use cases

This section contains written uses cases that seek to capture the steps which an Office Administrator will go through in achieving their goals of using the system.

2.2.1 View current count of toys

Use Case ID:	**WIMS-01.**
Use Case Title:	View the current count of each toy in stock.
Actor:	Office Administrator (OA)
Scope:	Woods Stocks Inventory Management System.
Level:	User goal.
Description:	The OA starts the process to view the current count of each toy in Wood Stocks store stock. Toy data is imported and displayed to the OA.
Basic Flow:	1. The OA initiates the process to import toy data from the stock data file. 2. The application loads the data for each toy from the file. 3. The application displays the current count for each toy in stock, along with its item code, item description and on order status to the OA.
Extension cases:	2a: **WIMS-02** Notify OA (user) that file cannot be found.

Use Case ID:	**WIMS-02**
Use Case Title:	Notify Office Administrator (user) that file cannot be found.
Actor:	Office Administrator (OA)
Scope:	Wood Stocks Inventory Management System
Level:	User.
Description:	The OA starts the data importation process. The file containing import data cannot be found by the application. The OA is notified that the file cannot be found.
Preconditions:	1. The file containing inventory data is not the application folder/directory for the stock file.
Basic flow:	1. The OA starts the process to import toy data. 2. The application detects that the file cannot be found. 3. The application notifies the OA that the file cannot be found.

2.2.2 Update current count for stock items

Use Case ID:	**WIMS-03**
Use Case Title:	Update current count of toys.
Actor:	Office Administrator (OA)
Scope:	Wood Stocks Stock Control System
Description:	The OA enters an updated count for a toy item. The toy
Preconditions:	1. Toy data has been imported into the application.
Basic flow:	1. The OA enters new values for the current count of a toy in the store's stock. 2. The application ensures that the value entered by the OA is valid for the current count field. 3. The current count of the toy is changed to the value entered by the OA. (assumed to be valid). 4. The display is updated to show the new count as entered by the OA. 5. The export button is enabled by the application to allow the OA to export changes when they are finished entering data. 6. The application ensures that each new value entered for the current count is valid.
Extension Cases:	2a: **WIMS-04:** Notify OA that invalid value was entered for current count.

Use Case ID:	**WIMS-04**
Use Case Title:	Notify OA that invalid value was entered for current count.
Actor:	Office Administrator (OA)
Scope:	Wood Stocks Inventory Management System
Level:	User.
Description:	The OA has entered an invalid value for the current count, and the application responds by notifying the OA that the value is invalid.
Preconditions:	1. Toy data has been imported into the application.
Basic flow:	1. The OA enters a value for the current count of a toy. 2. The application detects the value entered is invalid. 3. The application notifies the OA that value entered for current count is invalid and reminds the OA of valid values for the current count of a toy.

2.2.3 Export changes to the inventory data file

Use Case ID:	**WIMS-05**
Use Case Title:	Export updated toy data to the inventory data file.
Actor:	Office Administrator (OA)
Level:	User.
Scope:	Wood Stocks Inventory Management System
Description:	The OA starts the exportation process and the inventory data file is updated with the modified data.
Preconditions:	The OA has modified the current count for at least one toy, as to as to enable the export function of the application.
Basic flow:	1. The OA starts the exportation process. 2. The application updates the inventory data file to reflect changes in current count as entered by the OA.
Alternate case:	**WIMS-06:** Notify the OA that an error occurred whilst exporting data to the inventory data file.

Use Case ID:	**WIMS-06**
Use Case Title:	Export updated toy data to the inventory data file.
Actor:	Office Administrator (OA)
Level:	User.
Scope:	Wood Stocks Inventory Management System
Description:	The OA starts the exportation process and the application encounters a problem exporting the changes to an updated inventory data file.
Preconditions:	The OA has modified the current count for at least one toy, as to as to enable the export function of the application.
Basic flow:	3. The OA starts the exportation process. 4. The application encounters a problem exporting the modified data to an updated inventory data file. 5. The application notifies the OA that the exportation was unsuccessful.
Alternate case:	**WIMS-05:** Export updated toy data to the inventory data file.

2.2.4 Sort stock items by Item Code

Use Case ID:	**WIMS-07**
Use Case Title:	Sort Toys by Item Code – **Ascending.**
Actor:	Office Administrator (OA).
Scope:	Wood Stocks Inventory Management System.
Description:	The OA selects the option to sort the display of toys by Item Code in ascending order. The user interface is updated showing **toys sorted by item code in ascending order.**
Preconditions:	1. The application has imported toy data.
Basic flow:	1. The OA selects the option to sort toys by item code in ascending order. 2. The view data is sorted so that toys are sorted by item code in ascending order. 3. The display is refreshed so that the OA can see the toys sorted by Item Code in descending order.
Alternate cases:	1a. **WIMS-07:** Sort stock items by Item Code – **Descending**

Use Case ID:	**WIMS-08**
Use Case Title:	Sort Toys by Item Code – **Descending**
Actor:	Office Administrator (OA)
Scope:	Wood Stocks Inventory Management System.
Description:	The OA selects the option to sort the display of toys by Item code in descending order. The user interface is updated showing **toys sorted by item code in descending order.**
Preconditions:	1. The application has loaded the stock item data.
Basic flow:	1. The OA selects the option to sort toys by item code in descending order. 2. The view data is sorted so that toys are sorted by item code in descending order. 3. The display is refreshed so that the OA can see the toys sorted by Item Code in descending order.
Alternate cases:	1a. **WIMS-06:** Sort stock items by Item Code – **Ascending.**

2.2.5 Sort stock items by Current Count

Use Case ID:	**WIMS-09**
Use Case Title:	Sort Toys items by Current Count – **Ascending.**
Actor:	Office Administrator (OA)
Scope:	Wood Stocks Inventory Management System
Description:	The OA selects the option to sort the display of toys by current count in ascending order. The user interface is updated **showing toys sorted by current count in ascending order.**
Preconditions:	1. The application has loaded the stock item data.
Basic flow:	1. The OA selects the option to sort toys by current count in ascending order. 2. The view data is sorted so that toys are sorted by current count in ascending order. 3. The display is refreshed so that the OA can see the toys sorted by current count in ascending order.
Alternate cases:	1a. **WIMS-09:** Sort stock items by Current Count – **Descending**

Use Case ID:	**WIMS-10**
Use Case Title:	Sort Toys by Current Count – **Descending**
Actor:	Office Administrator (OA)
Scope:	Wood Stocks Inventory Management System
Description:	The OA selects the option to sort the display of toys by current count in descending order. The user interface is updated showing **toys sorted by current count in descending order.**
Preconditions:	1. The application has loaded the stock item data.
Basic flow:	1. The OA selects the option to sort toys by current count in descending order. 2. The view data is sorted so that toys are sorted by current count in descending order. 3. The display is refreshed so that the OA can see the toys sorted by current count in descending order.
Alternate cases:	1a. **WIMS-08:** Sort stock items by Current Count – **Ascending.**

2.2.6 Sort stock items by On Order status

Use Case ID:	**WIMS-11**
Use Case Title:	Sort stock items by On Order status – **Ascending**
Actor:	Office Administrator (OA)
Scope:	Wood Stocks Inventory Management System.
Description:	The OA selects the option to sort the display of toys by on order in ascending order. The user interface is updated **showing toys sorted by on order in ascending order.**
Preconditions:	1. The application has loaded the stock item data.
Basic flow:	1. The OA selects the option to sort toys by on order in ascending order. (i.e. No before Yes) 2. The view data is sorted so that toys are sorted by on order in ascending order. 3. The display is refreshed so that the OA can see the toys sorted by on order in ascending order.
Alternate cases:	1a. **WSCS-11:** Sort stock items by On Order status – **Descending**

Use Case ID:	**WIMS-12**
Use Case Title:	Sort stock items by On Order status – **Descending**
Actor:	Office Administrator (OA)
Scope:	Wood Stocks Inventory Management System.
Description:	The OA selects the option to sort the display of toys by on order in descending order. The user interface is updated **showing toys sorted by on order in descending order.**
Preconditions:	1. The application has loaded the stock item data.
Basic flow:	1. The OA selects the option to sort toys by on order in ascending order. (i.e. Yes before No) 2. The view data is sorted so that toys are sorted by on order in descending order. 3. The display is refreshed so that the OA can see the toys sorted by on order in ascending order.
Alternate cases:	1a. **WIMS-10:** Sort stock items by On Order status – **Descending**

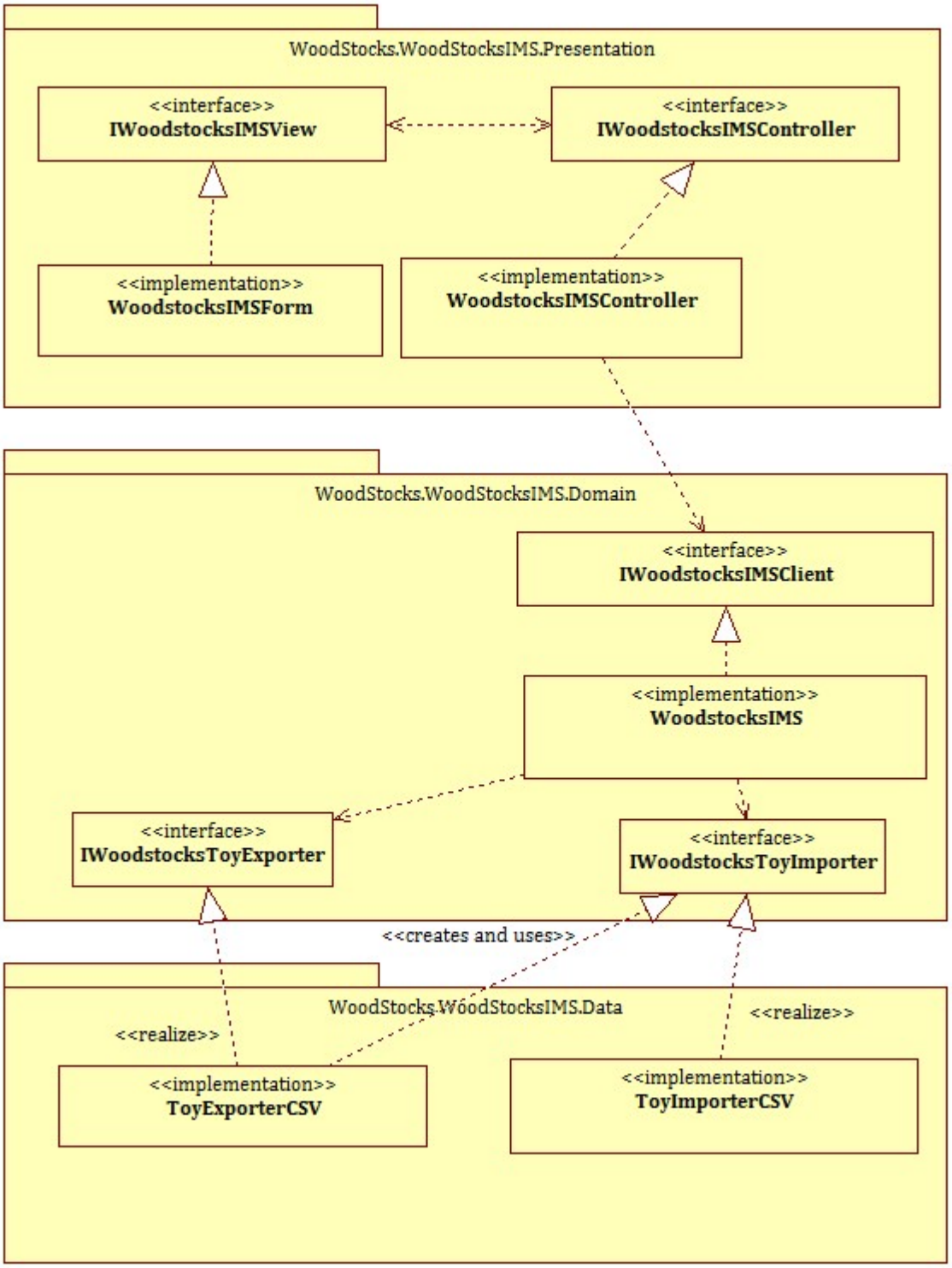
Note regarding Sorting toys by on order status:

Sorting in:

1. Ascending Order means sorting toys that are not on order (i.e. No) before those that are on order (i.e Yes).
2. Descending order means sorting toys that are on order (i.e Yes) before toys that are not on order (i.e No).

3 Application Architecture

The following diagram provides a high level architectural view for the WoodstocksIMS. As can be seen from the diagram the application will consist of three layers:



3.1 Presentation Layer description

The presentation layer will consist of objects that are responsible for providing the user interface for the WoodstocksIMS and for allowing the user to interact with the WoodstocksIMS.

The presentation layer of the application will consist of two different types of objects:

1. A WoodstocksIMSForm object.
2. A WoodstocksIMSController object.

The presentation layer will be contained in the Woodstocks.WoodstocksIMS.Presentation namespace.

3.1.1 WoodstocksIMSForm object

A WoodstocksIMSForm object is an object that is responsible for providing a Windows Form based user interface for the WoodstocksIMS .

An office administrator, or any other end user of the application, interacts directly with a WoodstocksIMSForm object via external resources such as the screen, keyboard and mouse, operating system and other systems software.

3.1.1.a Input and Output

A WoodstocksIMSForm object accepts input:

1. From office administrators and other end users and direct their output to a view controller or a view data object to create, modify/update, save or display inventory (application) data by calling methods of view controllers or methods of view data objects.
2. From WoodstocksIMSController object that directs output to the end user causing information to be displayed to the user.

3.1.1.b Dependencies

A WoodstocksIMSForm will be dependent upon a WoodstocksIMSController object to function correctly. A WoodstocksIMSForm will interact with a WoodstocksIMSController object via an IWoodstocksIMSController interface, that is implemented by a WoodstocksIMSController.

3.1.1.c Interface

A WoodstocksIMSForm will implement the IWoodstocksIMS View interface which defines methods for interacting with a WoodstocksIMSForm.

3.1.2 WoodstocksIMSController object

A WoodstocksIMSController object is an object that mediates actions between a WoodstocksIMSForm object and a WoodstocksIMS object. A WoostocksIMS object is an application façade, that will provided by the domain layer, to provides a unified service layer for the application.

3.1.2.a Input and Output

A WoodstocksIMSController will accept input from:

1. A WoodstocksIMSForm and direct output to a WoodstocksIMS object.
2. A WoodstocksIMS object and direct output to a WoodstocksIMSForm by calling its methods, via a IWoodstocksIMSView interface to present the user with results of user actions.

3.1.2.b Dependencies

As a result of its input and output flow a WoodstocksIMSController will be dependent upon:

1. An object that implements the IWoodstocksIMSView interface (i.e. a WoodstocksIMSForm object).
2. An object that implements the IWoodstocksIMSClient interface (i.e. a WoodstocksIMS object).

3.1.2.c Interface

A WoodstocksIMSController object will implement the IWoodstocksIMSController interface. The IWoodstocksIMSController interface defines methods for interacting with a WoodstocksIMSController.

3.2 Domain Layer description

The domain layer will consist of:

1. An application façade object, WoodstocksIMS, that provides a service layer for the application.

2. Domain objects, including business entities, that support realisation of the use cases by the WoodstocksIMS

3.2.1 WoodstocksIMS (Application Façade)

The WoodstocksIMS object will provide an application façade, for the WoodstocksIMS, and will provide a unified view of the application's services to a WoodstocksIMSController object.

The WoodstocksIMS will be primarily responsible for:

1. Managing the state of the application and co-ordinating application flow.
2. Servicing requests to import and export inventory (i.e toy) data.

3. Returning imported toy data as requested by an WoodstocksIMSController.

3.2.1.a Input and Output

The WoodstocksIMS object will accept input from:

1. A WoodstocksIMSController object and return output to the WoodstocksIMSController
2. Data access objects, in particular objects that implement the IWoodstocksToyImporter and IWoodstocksToyExporter interfaces.

3.2.1.b Dependencies

The WoodstocksIMS will be dependent upon Data Access Layer objects to import and export data from the WoodstocksIMS.

3.2.2 Business entities

In addition to the application façade, the domain layer will consist of domain objects, including business entity objects that are needed to realise the use case functionality of the application. The two main business entity objects for the application will be a Toys collection object and a Toy object. These two objects will be concerned with meeting the data needs of the application. As a result, the Toys and the Toy object are discussed below in section 3 Data Design.

3.3 Data Access Layer description

The data access layer will consist of data access objects that provide access to Wood Stocks inventory data.

As Wood Stocks inventory data is stored in a csv data file the data access layer will consist of two types of data access objects:

1. An object that is a ToyImporterCSV, that will implement the IWoodstocksToyImporter interface. A ToyImporterCSV will be responsible for importing toy data into the WoodstocksIMS from a csv file.
2. An object that is a ToyExporterCSV, that will implement the IWoodstocksToyExporter interface. A ToyExporterCSV object will be responsible for exporting toy data to a csv file.

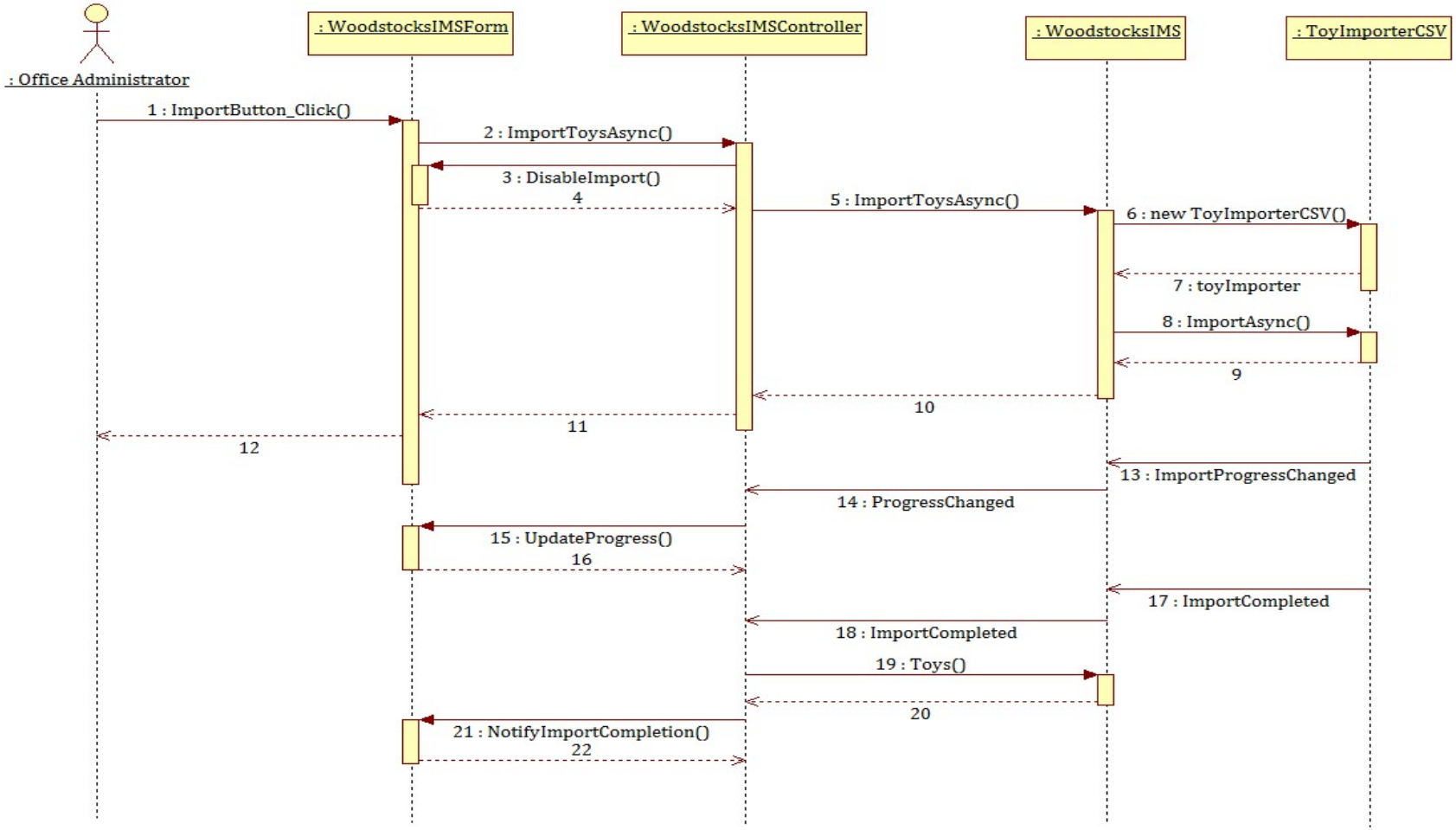
3.4 Sequence diagrams

In order to realise the use case functionality of the application the following sequence diagrams have been prepared.

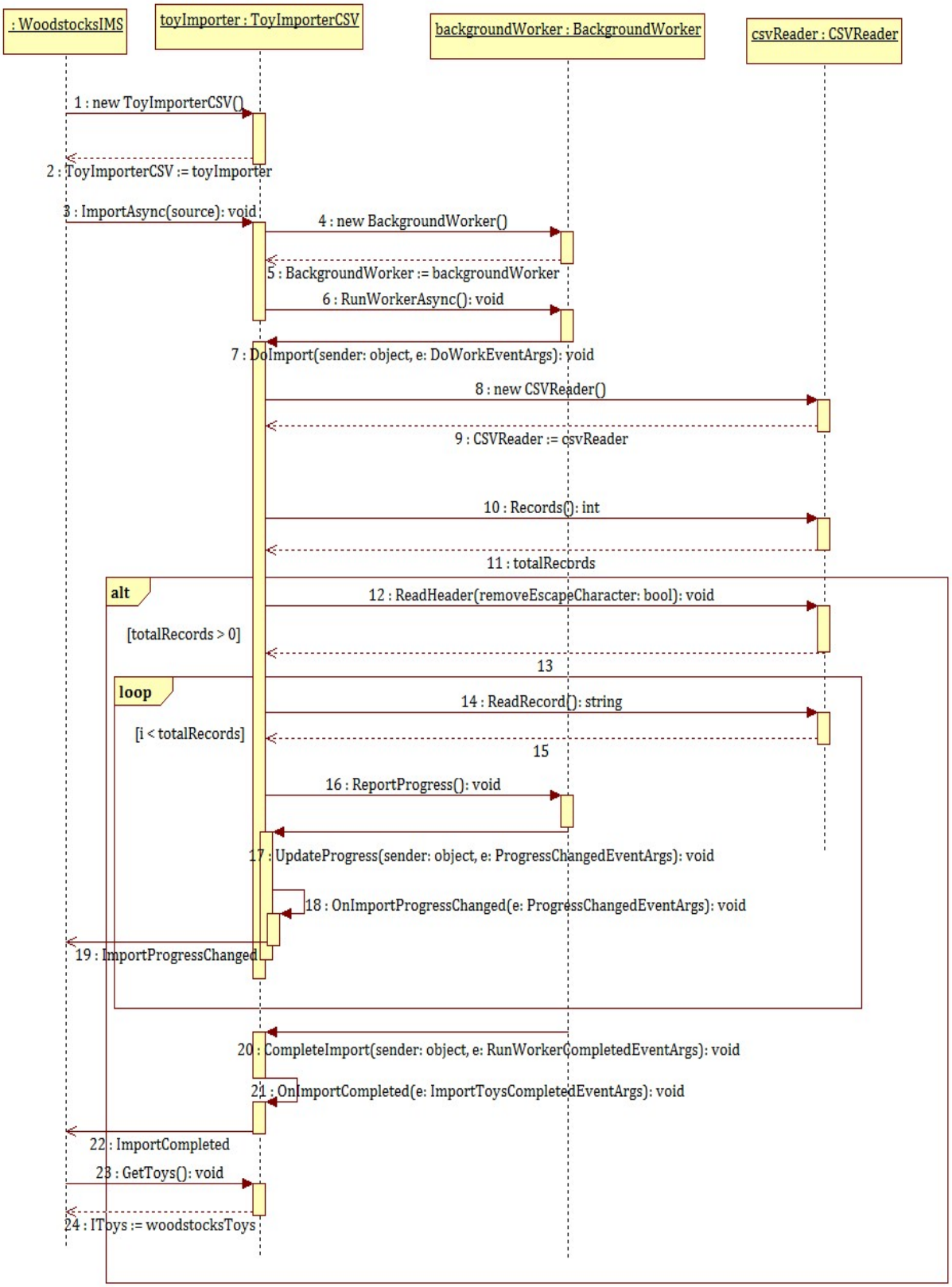
The sequence diagrams show the lifetime and interaction of objects from the various application layers, discussed previously, that are necessary to provide the use case functionality.

3.4.1 Sequence Diagram for Use Case WIMS-01: View current count of toys

3.4.1.a Part 1: Layer communication (assumes WoodstocksIMS is in IdleState and woodstocksToys field of WoodstocksIMS is null).

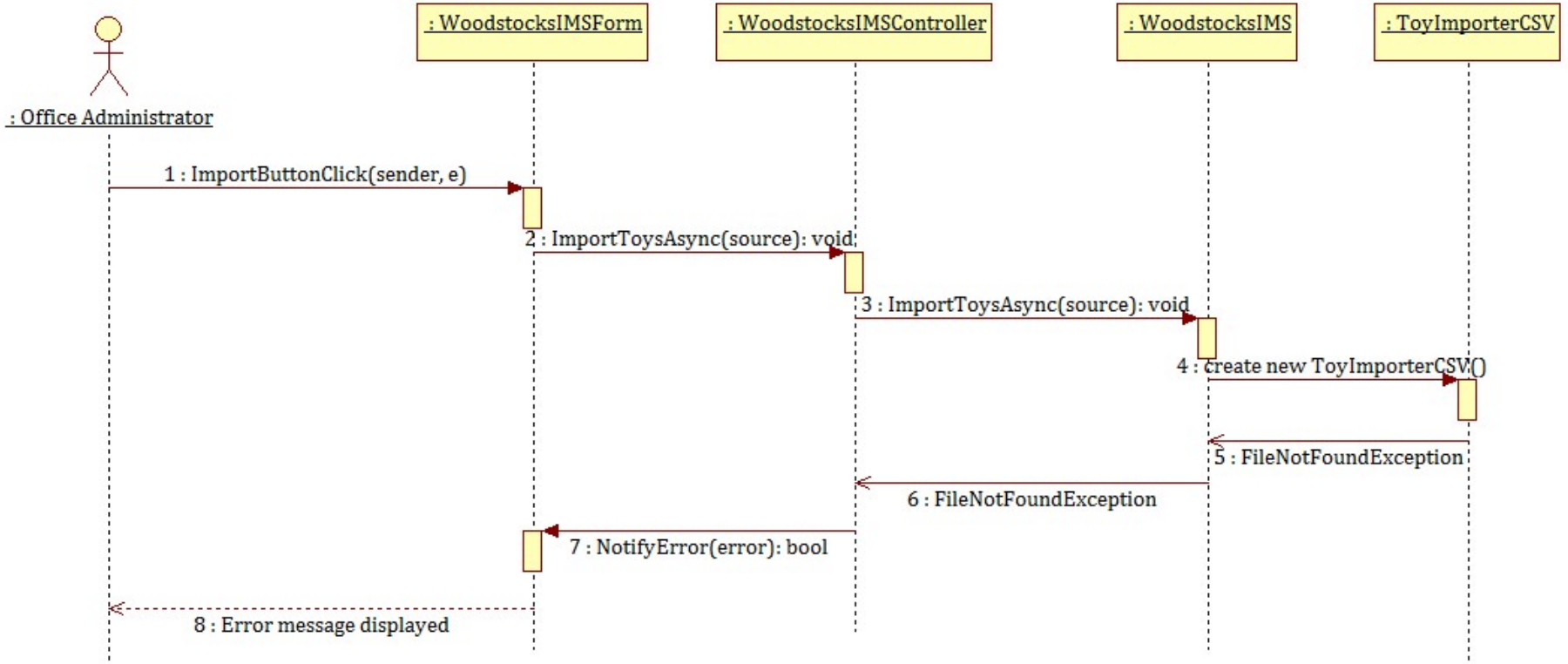


3.4.1.b Part 2: ToyImporterCSV view of importation

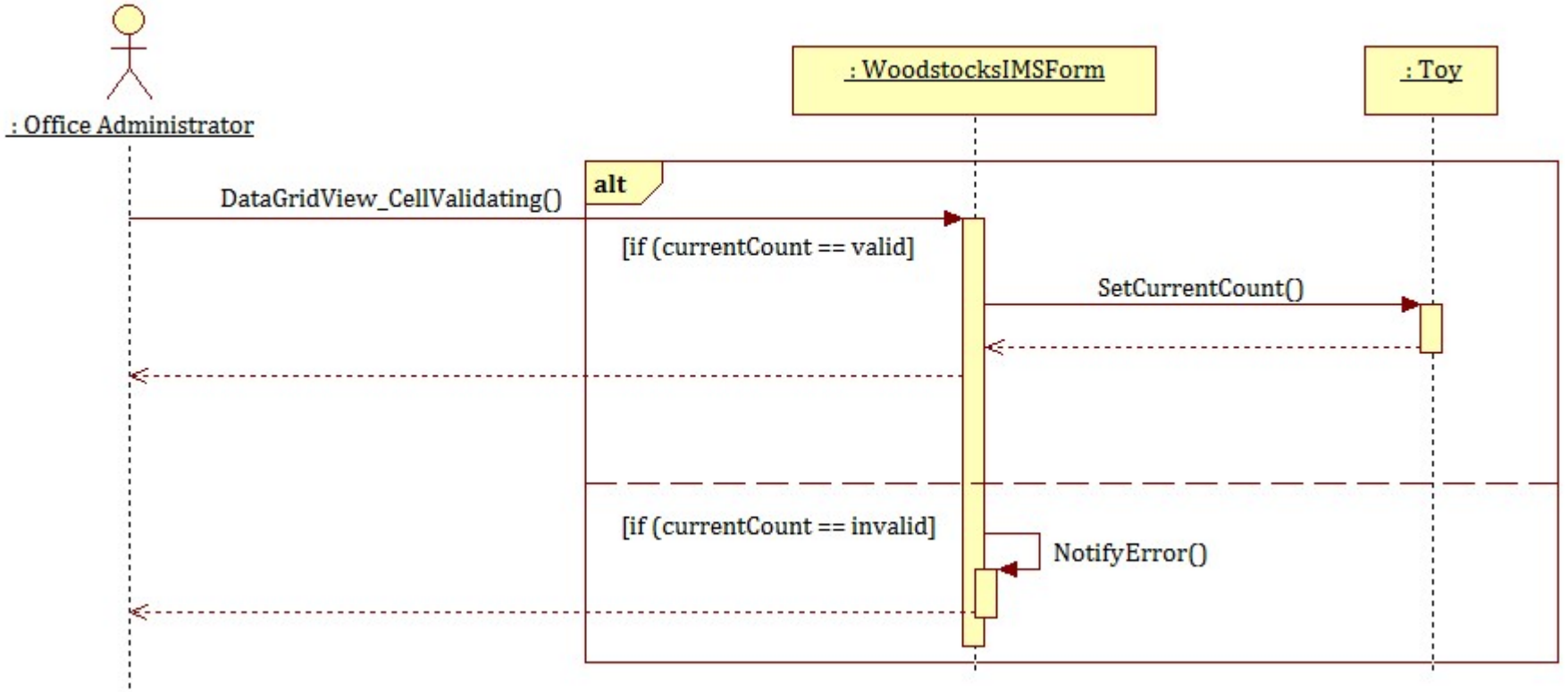


3.4.2 Sequence Diagram for Use Case WIMS-02: Importing – File not found

Assumes WoodstocksIMS is in IdleState.

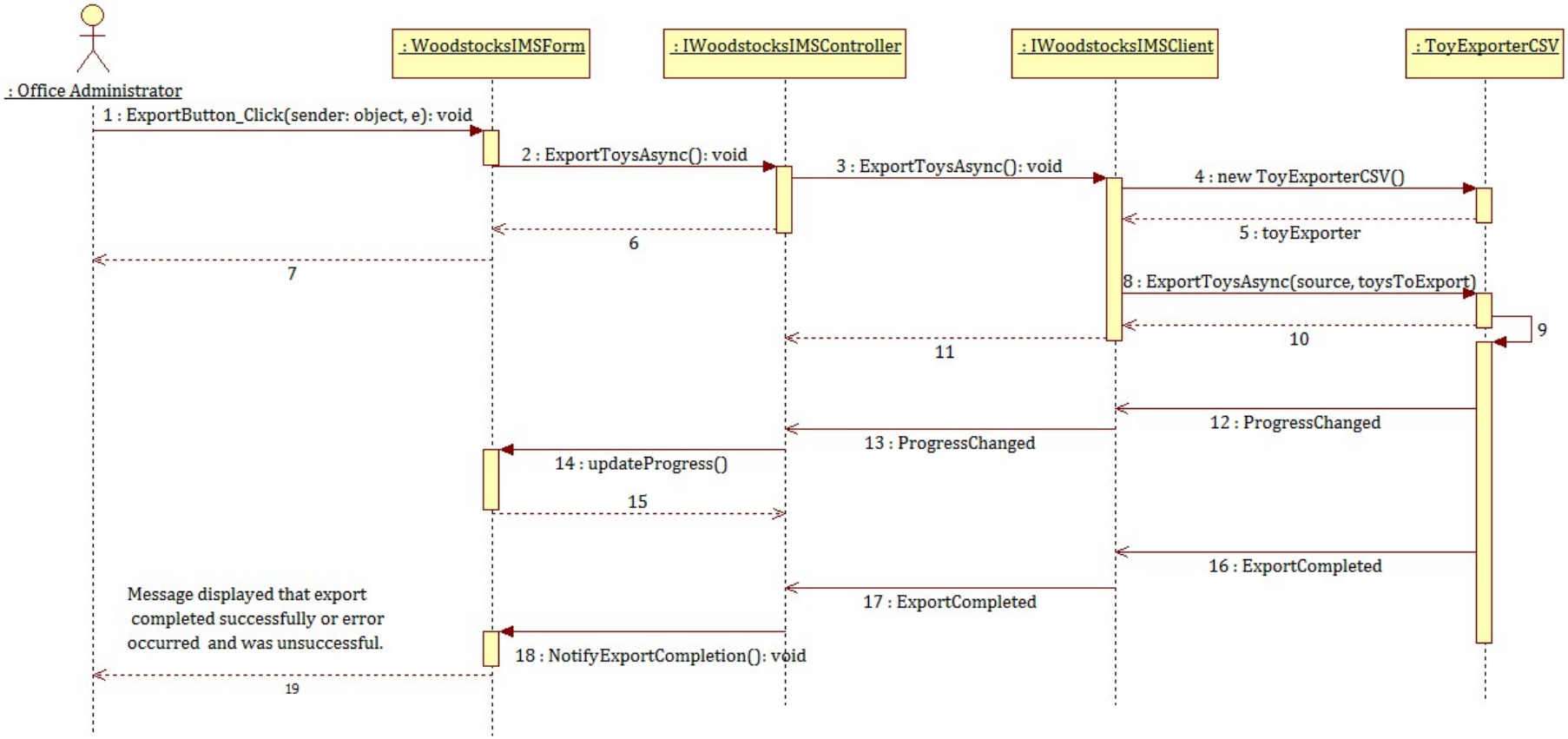


3.4.3 Sequence Diagram for WIMS-03 andWIMS-04: Changing current count

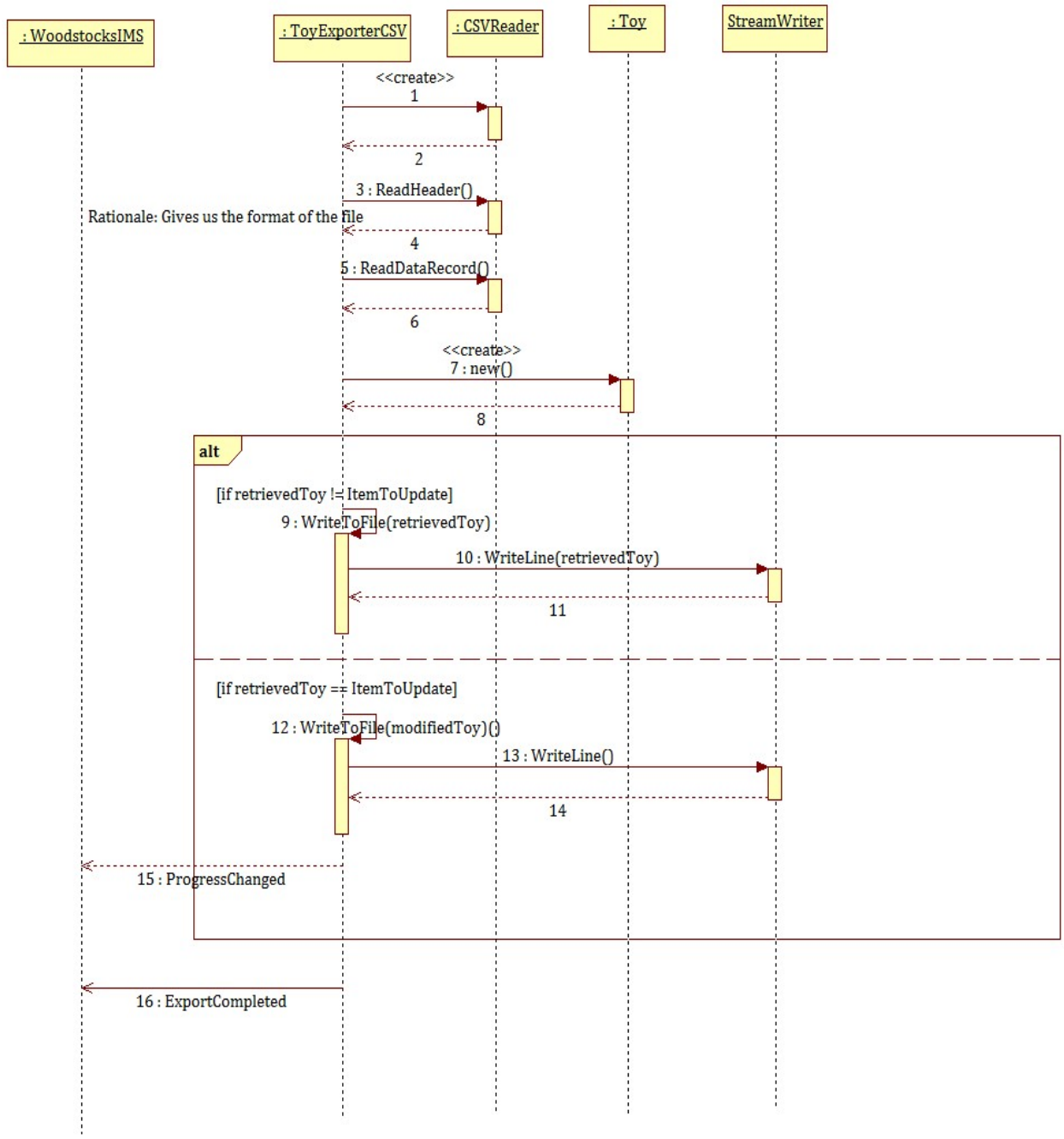


3.4.4 Sequence Diagram for WIMS-05 and WIMS-6: Export modified toy data

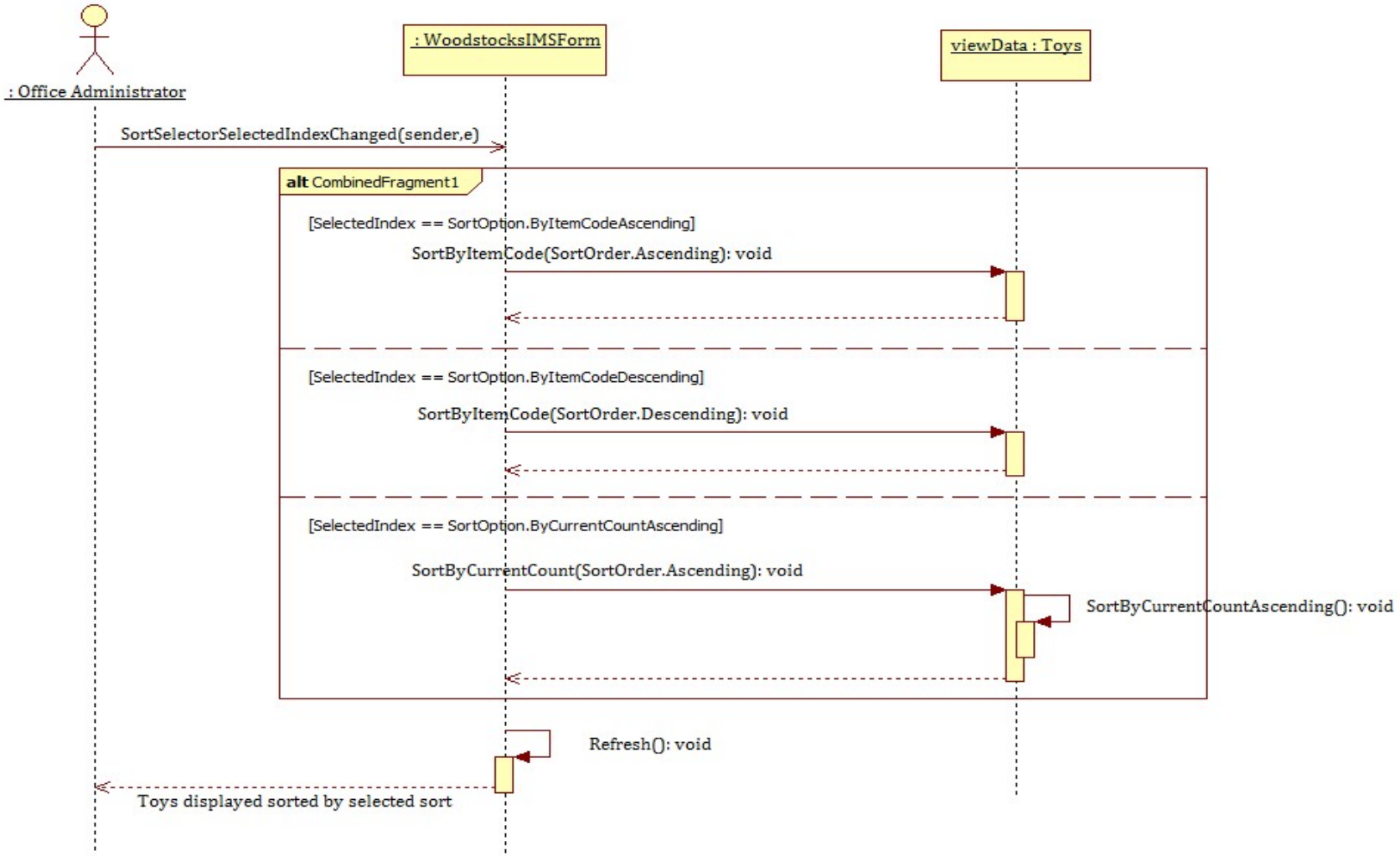
3.4.4.a Cross layer communication.



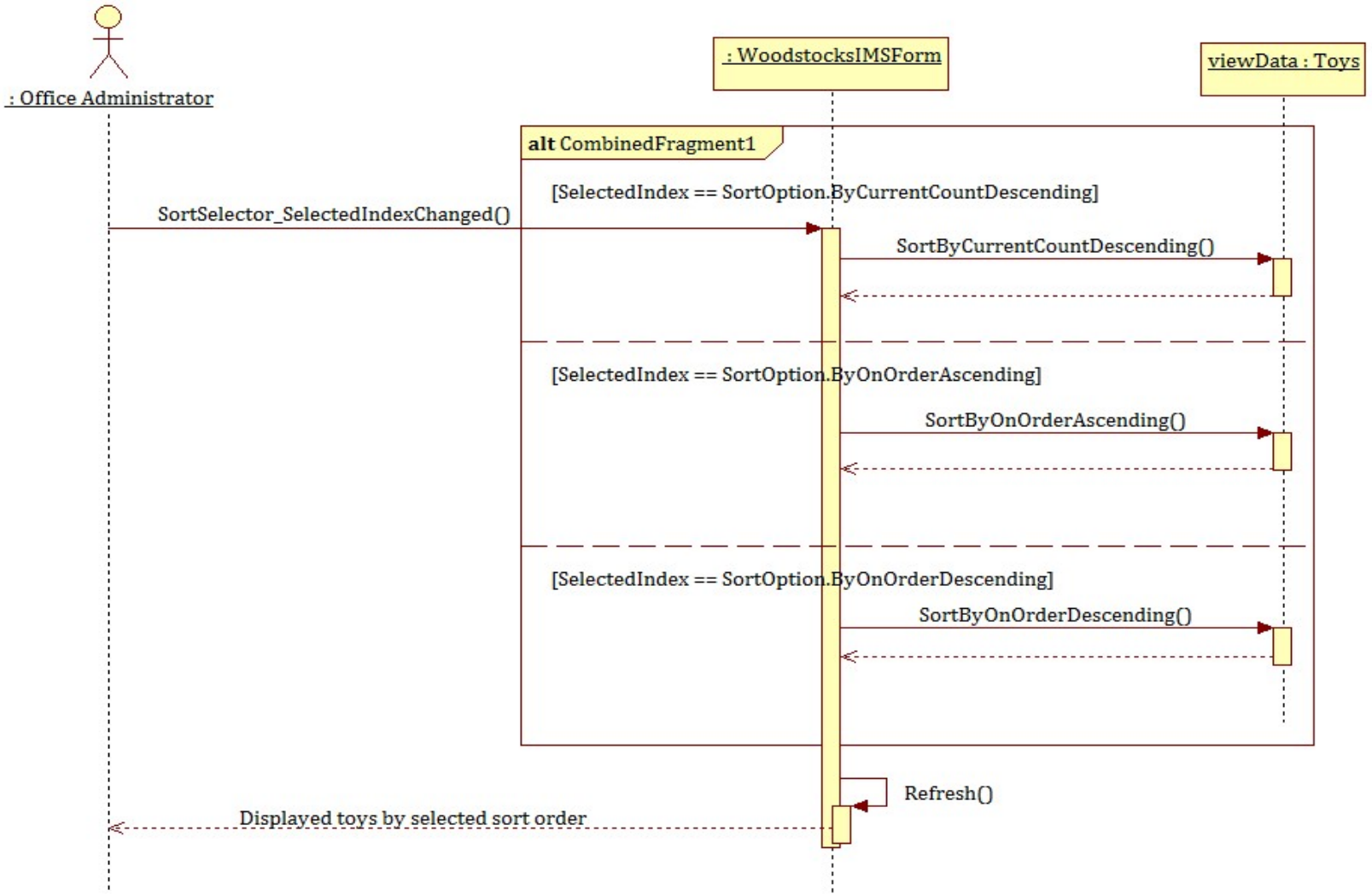
3.4.4.b ToyExporterCSV operation



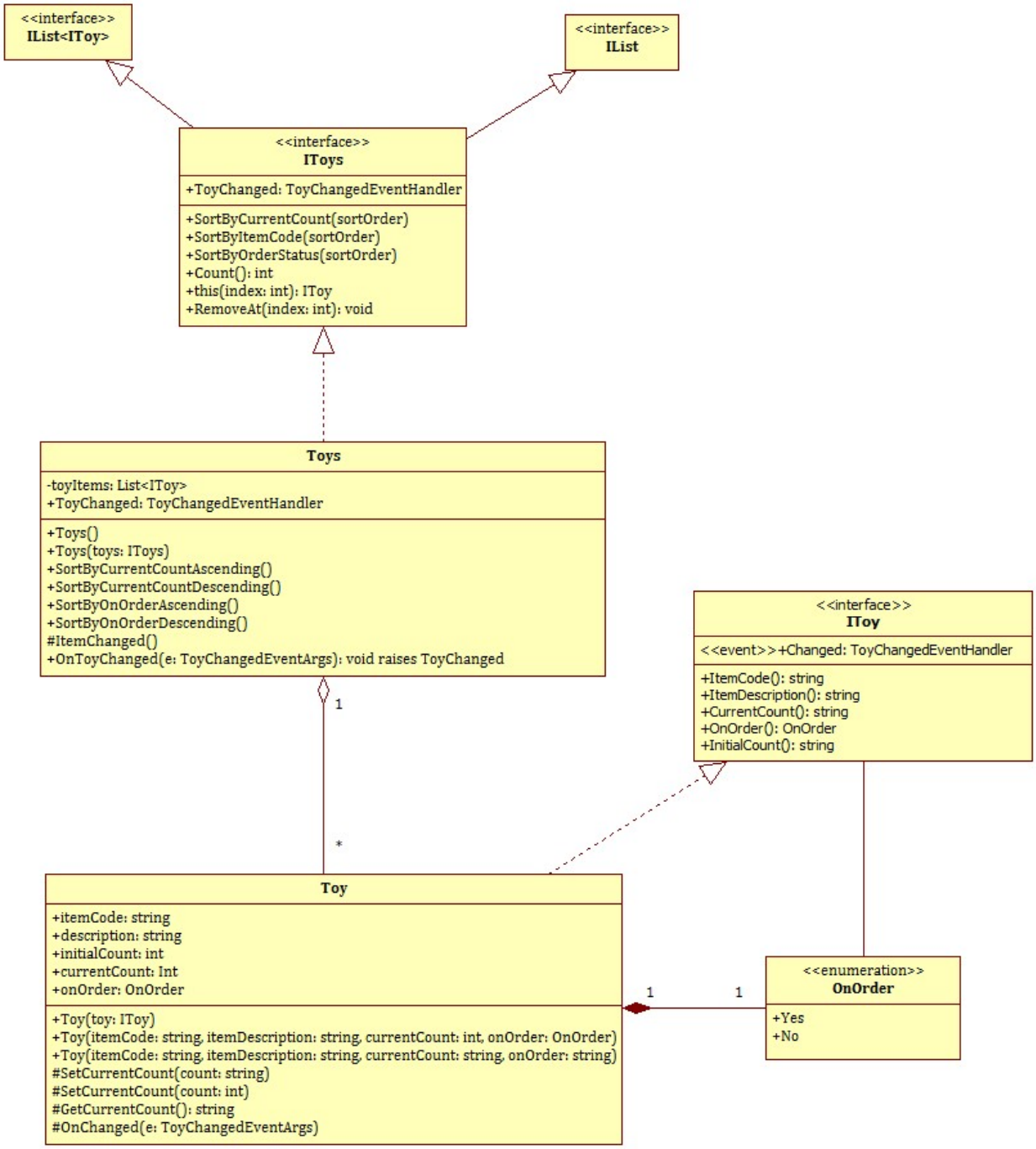
3.4.5 Sequence diagrams for Use Cases WIMS-07, WIMS-08 and WIMS-09: Sorting Toys



3.4.6 Sequence Diagram for Use Cases WIMS-10, WIMS-11 and WIMS-12: Sorting Toys



4 Data Design



The above class diagram presents a visual model of the business data requirements of the WoodstocksIMS.

Toys: The Toys class defines an object that represents a collection of Toys that is used to represent Wood Stocks total stock.

Toys is to be implemented using a generic collection of List<IToy>. This will allows the collection to support sorting of individual Toys by their item code, current count and on order status.

The sorting operations will be supported through implementation of the relevant sort methods defined on the IToys interface.

The IToys interface is defined such that if the implementation of Toys needs to change in the future then the changes to other components of the system will be minimal so long as the interface continues to be implemented.

It should be noted that the IToys interface defines a ToyChanged event. This event is to be raised whenever the data of a toy within the collection is changed. This will allow the application to identify changes to toy data, in particular changes to the current count of a Toy.

Toy: The Toy class defines an object that represents a Toy stocked and sold by Wood Stocks. It has fields and properties that support the item code, item description, current count, and on order status of a toy.

It should be noted that an initial count is defined for the object. The value of initial count should not change during the life of the object, and reflect the current count for the Toy item at the time the Toy object was created. This field, and its associated property, exist to allow the application to determine whether the current count has changed since it was instantiated.

Furthermore, the initial count and current count are to be defined using the int data type notwithstanding that the values of these fields are initially read from the inventory data file as strings. The choice of data type for their fields is premised upon the following:

1. The choice of an int data allows for the value of the fields to be valiated as numerical values.
2. The choice of an int supports both negative and positive values. This will allow the application to keep track of inventory level's in scenarios where Wood Stocks office has made sales directly throughout the day, without being aware of the actual stock level of a Toy in its stock room.

OnOrder: Defines an enumeration that represents the On Order Status of a toy stocked by Wood Stocks. The literal values for OnOrder are:

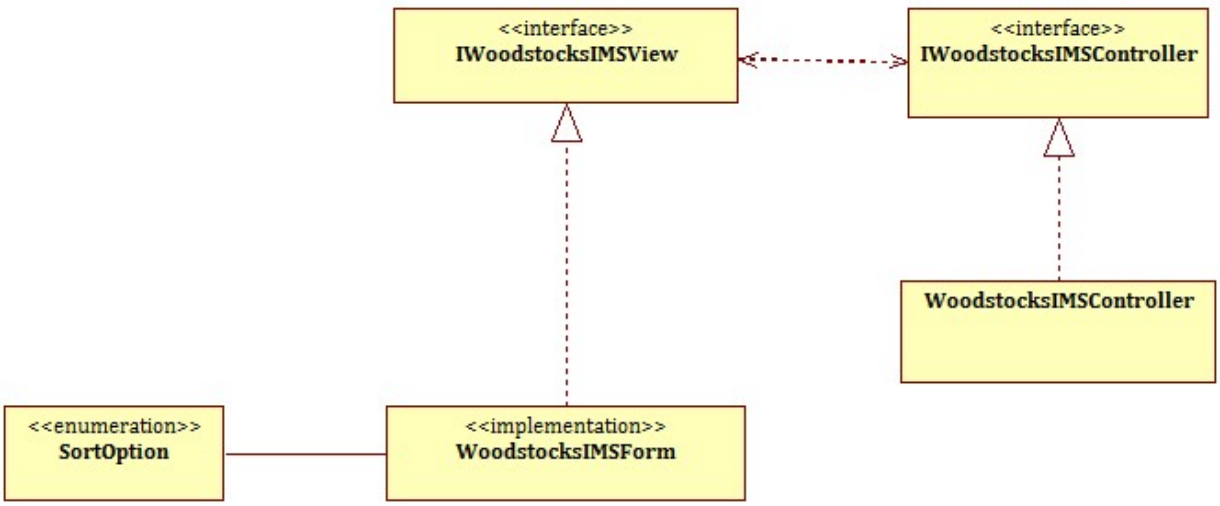
- **No:** will represent the toy is not on order. A value of 1.
- **Yes:** will represent the toy is on order. A value of 2 (or any value greater than 1).

The values have been chosen to ensure items not on order appear in ascending order before those that are not on order.

5 Component Design

5.1 Presentation Layer

5.1.1 Overview



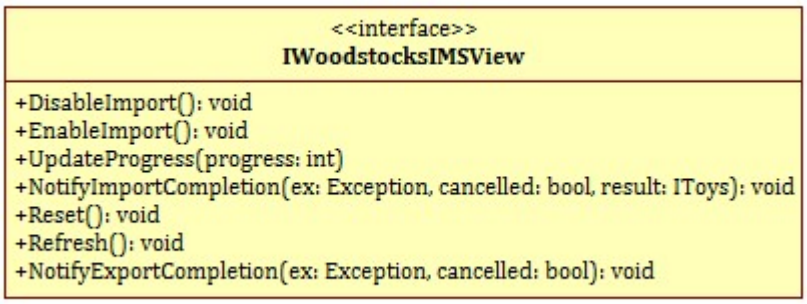
5.1.1.a Classes

Class	Description
ProgressDialog	
WoodstocksIMSController	A controller for the WoodstocksIMS.
WoodstocksIMSForm	A view for the WoodstocksIMS.

5.1.1.b Interfaces

Interface	Description
IWoodstocksIMSController	Defines the interface of a controller for aWoodstocksIMS.
IWoodstocksIMSView	An interface for a View within the Wood Stocks Inventory Management System.

5.1.2 IWoodstocksIMSView



An interface for a View within the Wood Stocks Inventory Management System.

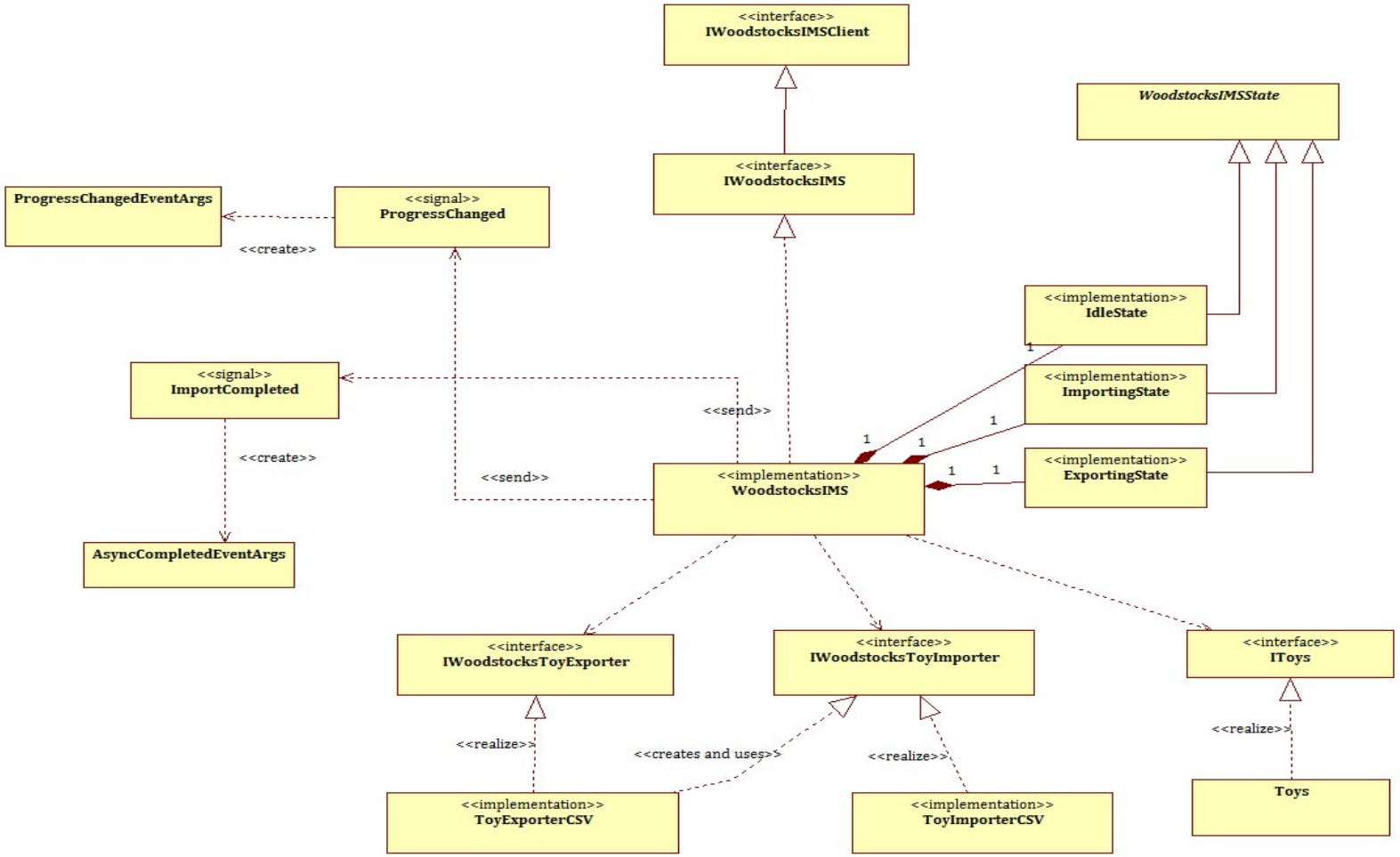
The **IWoodstocksIMSView** is to expose the following members.

Methods

Name	Description
DisableImport	Disables import option of the View. **Rationale:** Disabling the import option of the view is desirable to prevent the user activating the import functionality at times that importation cannot be supported. For example, the user cannot both export and import simulataneously.
EnableImport	Enables import option of the View. **Rationale:** To allow the user to invoke the import functionality From the view, especially after it has been disabled.
NotifyExportCompletion	Notifies the user that exportation has completed.
NotifyImportCompletion	Notifies the user that importation has completed.
Reset	Resets the view.
UpdateProgress	Notifies the user of progress of a task.

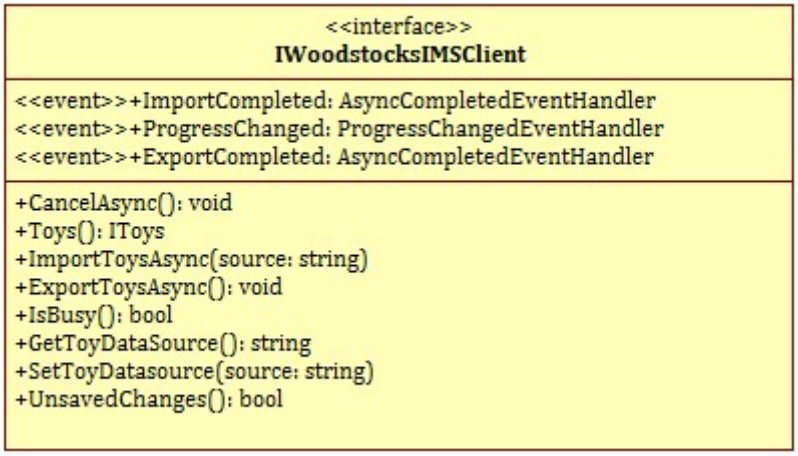
5.2 Domain Layer

5.2.1 Overview



5.2.2 IWoodstocksIMSClient interface

Defines a client interface for the WoodstocksIMS.



The **IWoodstocksIMSClient** interface will define following members:

Methods

Name	Description
CancelAsync	Cancels an asynchronous operation.
ExportToysAsync	Exports modified toy data from the system.
ImportToysAsync	Imports toy data into the WoodstocksIMS for use by the system.
IsBusy	Indicates if the WoodstocksIMS is busy carrying out an asynchronous operation.
UnsavedChanges	Gets whether the toy data contains unsaved changes.

Properties

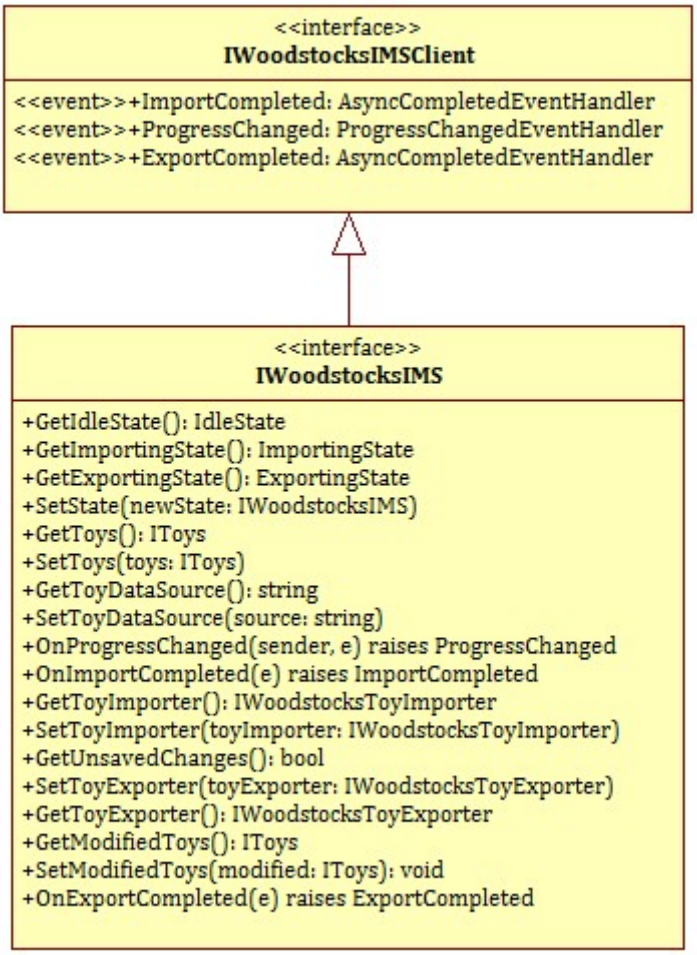
Name	Description
ToyDataSource	Gets and Sets the data source from which toy data should be imported.
Toys	Retrieves IToys which references Wood Stocks toy data after importation.

Events

Name	Description
ExportCompleted	Raised when an asynchronous export completes.
ImportCompleted	Raised when an asynchronous import completes.
ProgressChanged	Raised when progress on an asynchronous operation is made.

5.2.3 IWoodstocksIMS interface

Defines an interface to the WoodstocksIMS.



The **IWoodstocksIMS** interface will define the following members:

Methods

Name	Description
CancelAsync	Cancels an asynchronous operation. (Inherited from IWoodstocksIMSClient.)
ExportToysAsync	Exports modified toy data from the system. (Inherited from IWoodstocksIMSClient.)
GetExportingState	Get the Exporting state of the system

GetIdleState	Get the Idle state of the system.
GetImportingState	Get the Importing state of the system.
GetModifiedToys	Gets the toy data that has been modified and has not been saved.
GetToyDataSource	Gets the source from which the system will, or has, imported toy data.
GetToyExporter	Gets the toy exporter used for exporting data.
GetToyImporter	Gets the toy importer of the system.
GetToys	Gets the toy data currently imported into the WoodstocksIMS.
GetUnsavedChanges	Gets whether the **IWoodstocksIMS** has imported stock data that has been modified but has not been saved.
ImportToysAsync	Imports toy data into the WoodstocksIMS for use by the system. (Inherited from IWoodstocksIMSClient.)
IsBusy	Indicates if the WoodstocksIMS is busy carrying out an asynchronous operation. (Inherited from IWoodstocksIMSClient.)
OnExportCompleted	Raises the ExportCompleted event of the **IWoodstocksIMS**.
OnImportCompleted	Raises the ImportCompleted event of the **IWoodstocksIMS**.
OnProgressChanged	Raises the ProgressChanged event of the **IWoodstocksIMS** to indicate that progress of an asynchronous operation.
SetModifiedToys	Sets the toy data that has been modified and has not been saved.
SetState	Set the current state of the system.
SetToyDataSource	Sets the source from which the system will, or has, imported toy data.
SetToyExporter	Sets the toy exporter used for exporting data.

SetToyImporter	Sets the toy importer of the system.
SetToys	Sets the toy data in use by the WoodstocksIMS.
UnsavedChanges	Gets whether the toy data contains unsaved changes. (Inherited from IWoodstocksIMSClient.)

Properties

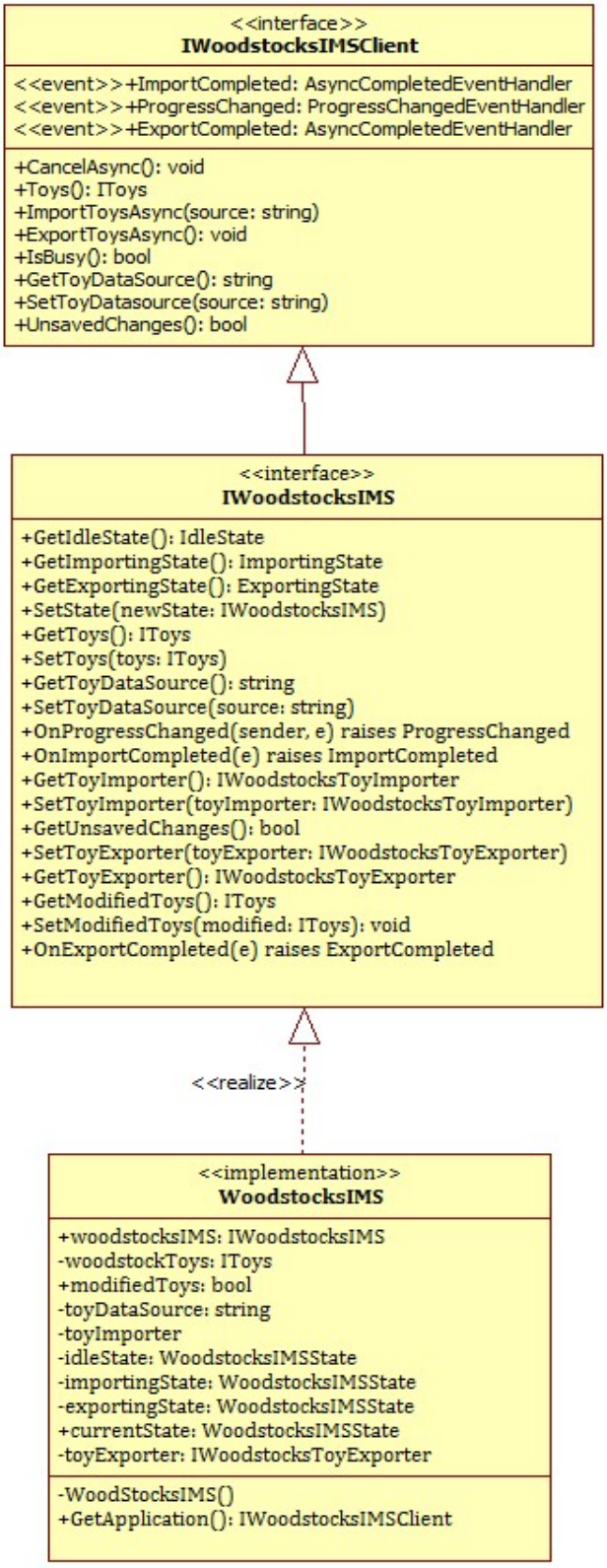
Name	Description
ToyDataSource	Gets and Sets the data source from which toy data should be imported. (Inherited from IWoodstocksIMSClient.)
ToyImporter	Gets and Sets the IWoodstocksToyImporter to be used by the system to import toy data.
Toys	Retrieves IToys which references Wood Stocks toy data after importation. (Inherited from IWoodstocksIMSClient.)

Events

Name	Description
ExportCompleted	Raised when an asynchronous export completes. (Inherited from IWoodstocksIMSClient.)
ImportCompleted	Raised when an asynchronous import completes. (Inherited from IWoodstocksIMSClient.)
ProgressChanged	Raised when progress on an asynchronous operation is made. (Inherited from IWoodstocksIMSClient.)

5.2.4 WoodstocksIMS class

Implementation of the Wood Stocks Inventory Management System.



Refer to the IWoodstocksIMSClient and IWoodstocksIMS interfaces for descriptions of the methods defined by the respective interfaces.

The **WoodstocksIMS** will consist of the following additional members:

Constructors

Name	Description
WoodstocksIMS	Initializes a new instance of the **WoodstocksIMS** class

Methods

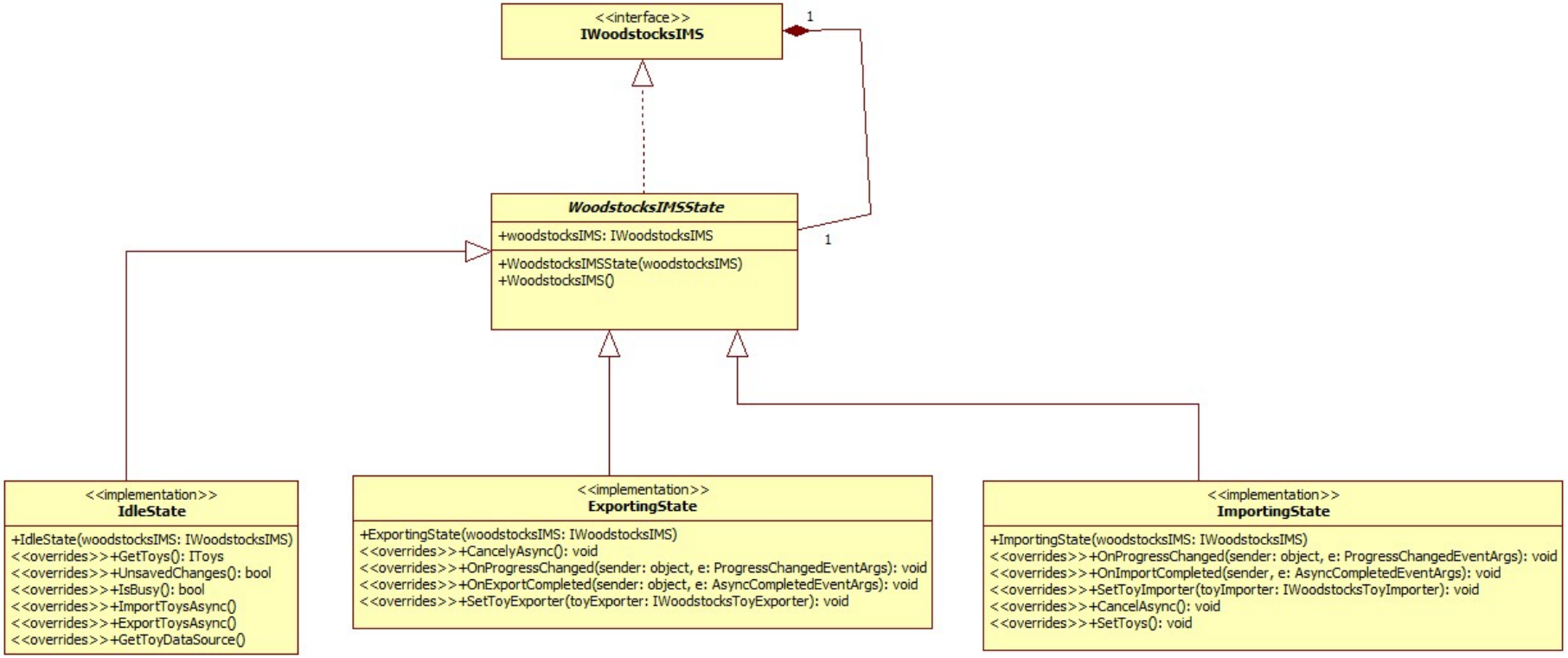
Name	Description
GetApplication	Creates the **WoodstocksIMS** for the application when called if not null and returns a client interface reference to the system.

Fields

Name	Description
currentState	The current state of the WoodstocksIMS.
exportingState	The exporting state of the WoodstocksIMS.
idleState	The idle state of the WoodstocksIMS.
importingState	The importing state of the WoodstocksIMS.
modifiedToys	A collection of the toys that have modified data.
toyDataSource	The source from which the system should, or has, retrieved toy stock data.
toyExporter	A reference to the importer used by the system for importing toy data.
toyImporter	A reference to the importer used by the system for importing toy data.

woodstocksIMS	A singleton instance of the **WoodstocksIMS**.
woodstocksToys	Represents Wood Stocks toy stock.

5.2.5 WoodstocksIMS State Design



The WoodstocksIMSState is to be an abstract class that defining states of the WoodstocksIMS. The WoodstocksIMSState will realise the IWoodstocksIMS interface by providing a default implementation for the bulk of the methods such that they throw an InvalidOperationException . This is to ensure that if a particular state is to provide functionality for the method, the method must be overridden by the concrete defining state.

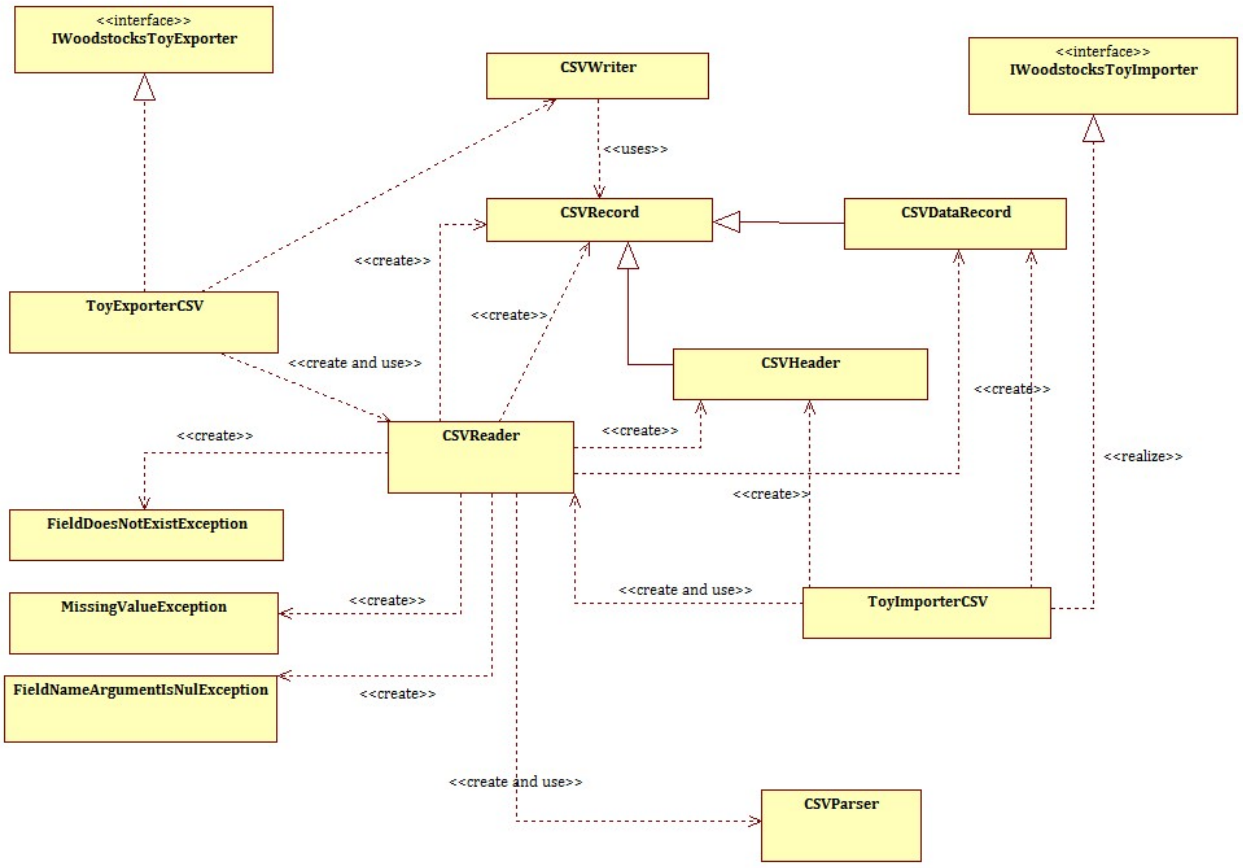
Additionally the WoodstocksIMSState abstract class will be defined with an IWoodstocksIMS interface reference, woodstocksIMS. This member is to be initialised when a concrete state is instantiated with a reference to the WoodstocksIMS. This is to ensure that each state can access resources and functionality that are provided for by the implementation of the WoodstocksIMS class.

The importation or exportation operations of the system can only be commenced from the IdleState of the system as indicated by the above class diagram, as the IdleState overrides the default implementation of the WoodstocksIMSState class to enable import and export functionality.

Once an importation or exportation is commenced from the system's idle state, the state of the system will be set to the respective ImportingState or ExportingState. The respective states will handle the progress and completion events of an importer or exporter as required.

5.3 Data Access Layer

5.3.1 Overview



5.3.1.a Classes

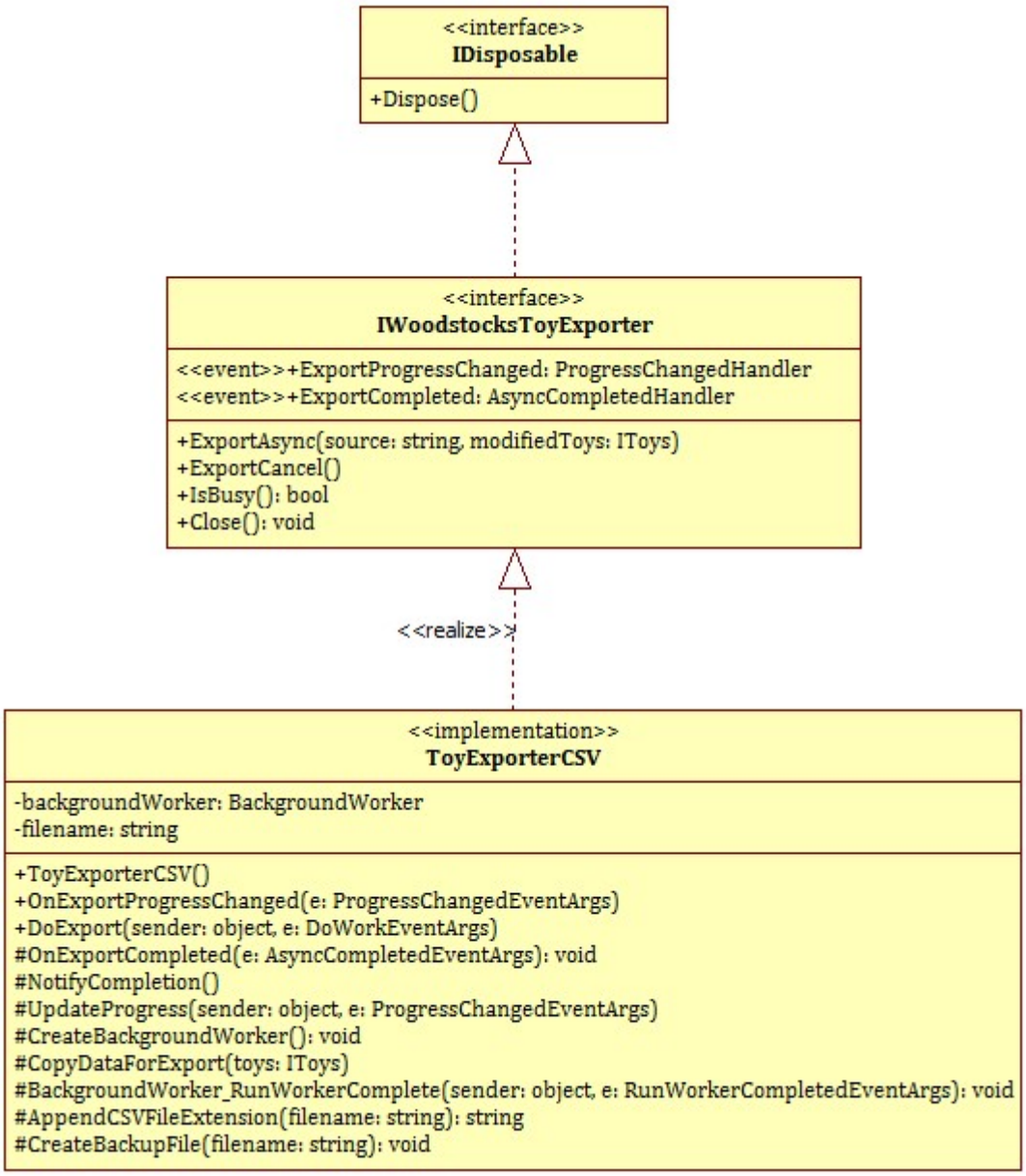
Class	Description
CSVDataRecord	Represents a csv data record.
CSVHeader	Represents the header record from a csv file.
CSVParser	A CSV Parser that converts a csv string into a List(T).
CSVReader	A CSVReader that is used to read records from a csv file.
CSVRecord	A base abstract class for CSV Records.

CSVWriter	A CSVWriter that is used to write CSVRecords to a file.
FieldDoesNotExistException	The exception that is raised when an attempt is made to access a named field that does not exist.
MissingValueException	The exception that is raised when a value is missing from a CSVRecord.
ToyExporterCSV	An exporter to export Wood Stocks Toy stock data to a csv data file.
ToyImporterCSV	An importer to toy data for Wood Stock toys from a csv data file.

5.3.1.b Enumerations

Enumeration	Description
CSVParser.TrimOption	An enumeration that defines the values of trimming options.

5.3.2 IWoodstocksToyExporter and ToyExporterCSV



Note: BackgroundWorker class to be supplied by .NET Framework 3.5.

5.3.2.a IWoodstocksToyExporter interface

Defines methods for an exporter to export toy data..

The **IWoodstocksToyExporter** interface will expose the following members.

Methods

Name	Description
Close	Closes the exporter.
ExportAsync	Exports toy data to the specified destination.
ExportCancel	Cancels an asynchronous exportation of toy data.

Properties

Name	Description
IsBusy	Indicates if the exporter is busy carrying out an exportation.

Events

Name	Description
ExportCompleted	Event that is raised upon completion of exportation.
ExportProgressChanged	Event that is raised upon progress of exportation.

5.3.2.b ToyExporter Class

An exporter to export Wood Stocks Toy stock data to a csv data file.

The **ToyExporterCSV** class will expose the following members:

Constructors

Name	Description
ToyExporterCSV	Initialises a **ToyExporterCSV**.

Methods

Name	Description
AppendCSVFileExtension	Checks whehter a file name, assumed to include, the path of the file has a .csv extension. Appends the .csv extension if the filename string does not have the .csv extension.
BackgroundWorker_RunWorkerCompleted	Handles the RunWorkerCompleted event of the BackgroundWorker used to carry out the exportation.
Close	Closes the exporter.
CopyDataForExport	Creates a copy of the data that is to be exported to the csv file.
CreateBackgroundWorker	Creates a BackgroundWorker to be used to export data asynchronously.
CreateBackupFile	Creates a backup file for the file to which data is to be exported.
DoExport	The method that is called by a BackgroundWorker to perform an asynchronous exportation of IToy data to a

	csv file.
ExportAsync	Exports data asynchronously to a csv data file.
ExportCancel	Cancels an asynchronous export of toy data by the exporter.
OnExportCompleted	Handles the completion event of the BackgroundWorker by raising the ExportCompleted event.
OnExportProgressChanged	Raises the ExportProgressChanged event.
UpdateProgress	Handles the progress changed event of the background worker asynchronously exporting data.

Fields

Name	Description
backgroundWorker	A BackgroundWorker to carry out an asynchronous exportation.
filename	The file name, including the path, that the exporter should export data to.

Properties

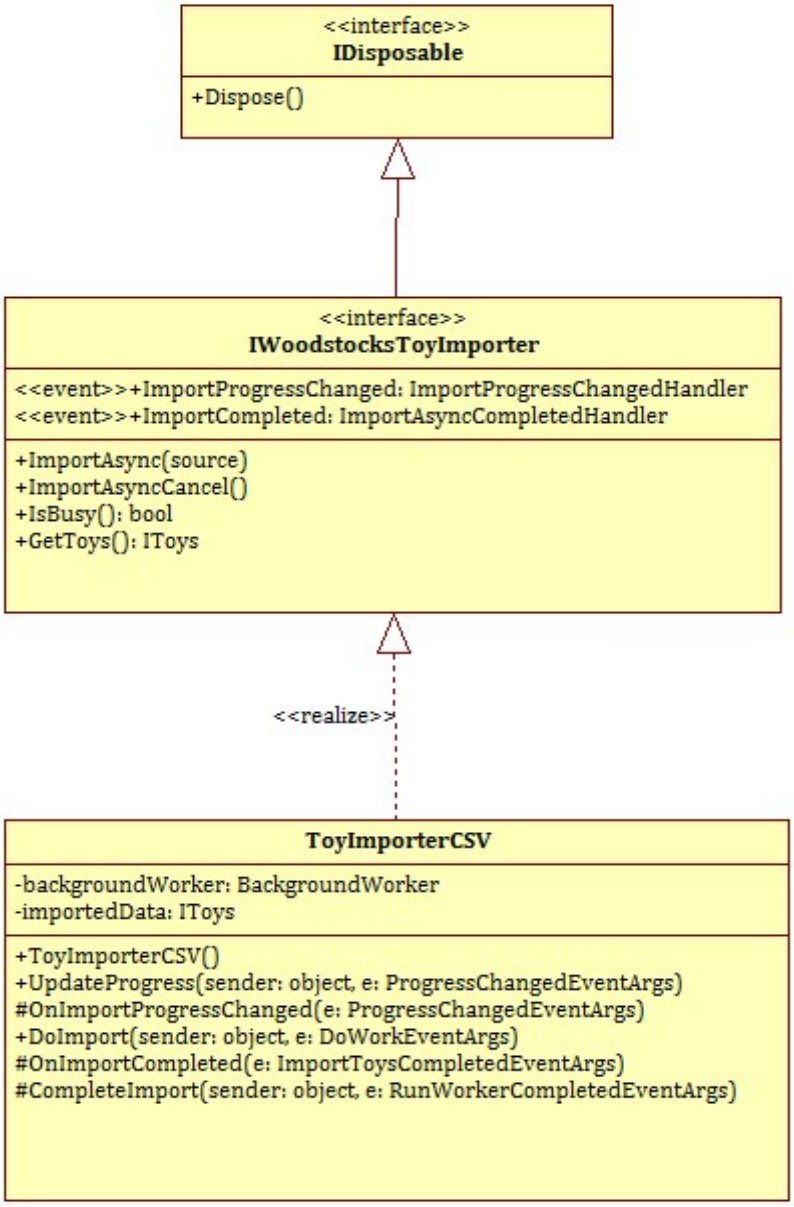
Name	Description
IsBusy	Indicates if the exporter is busy carrying out an asynchronous exportation of toy data.

Events

Name	Description
ExportCompleted	Event that is raised upon completion of exportation.

ExportProgressChanged	Event that is raised upon progress of exportation.

5.3.3 IWoodstocksToyImporter and ToyImporterCSV



Note: BackgroundWorker class to be supplied by .NET Framework 3.5.

Class	Description
ToyImporterCSV	An importer to toy data for Wood Stock toys from a csv data file.

5.3.3.a IWoodstocksToyImporter interface

Defines an interface to an importer to import toy data.

The **IWoodstocksToyImporter** interface will expose the following members:

Methods

Name	Description
Close	Closes the importer.
Dispose	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources. (Inherited from IDisposable.)
GetToys	Gets the data for toys that are imported by the importer.
ImportAsync	Imports toy data asynchronously.
ImportCancel	Cancels an asynchronous import.
IsBusy	Returns whether the importer is busy wilst carrying out an asynhronous import.

Events

Name	Description
ImportCompleted	Raised upon completion of an asynchronous operation.
ImportProgressChanged	The event when progress is made on an asynchronous import.

5.3.3.b ToyImporterCSV

An importer to toy data for Wood Stock toys from a csv data file.

The **ToyImporterCSV** type exposes the following members.

Constructors

Name	Description
ToyImporterCSV	Initialises a **ToyImporterCSV**.

Methods

Name	Description
Close	Closes the importer.
CompleteImport	Handles the RunbackgroundWorkerCompleted event of the BackgroundbackgroundWorker performing an asynchronous import.
Dispose()	Performs application-defined tasks associated with freeing, releasing, or resetting unmanaged resources.
Dispose(Boolean)	Implements the IDisposable.Dispose() method.
DoImport	Performs an asynchronous importation of toy data from a csv data file.
Equals	Determines whether the specified Object is equal to the current Object. (Inherited from Object.)
Finalize	Allows an object to try to free resources and perform other cleanup operations before it is reclaimed by garbage collection. (Inherited from Object.)
GetHashCode	Serves as a hash function for a particular type. (Inherited from Object.)
GetToys	Retrieves the imported data from the **ToyImporterCSV**.

GetType	Gets the Type of the current instance. (Inherited from Object.)
ImportAsync	Imports toy data asynchronously.
ImportCancel	Cancels an asynchronous import.
IsBusy	Returns whether the importer is busy wilst carrying out an asynhronous import.
MemberwiseClone	Creates a shallow copy of the current Object. (Inherited from Object.)
OnImportCompleted	Raises the ImportCompleted event of the **ToyImporterCSV**.
OnImportProgressChanged	Raises the ImportProgressChanged event of the **ToyImporterCSV**.
ToString	Returns a string that represents the current object. (Inherited from Object.)
UpdateProgress	Handles the ProgressChanged event raised by the BackgroundbackgroundWorker carrying out an asynchronous import.

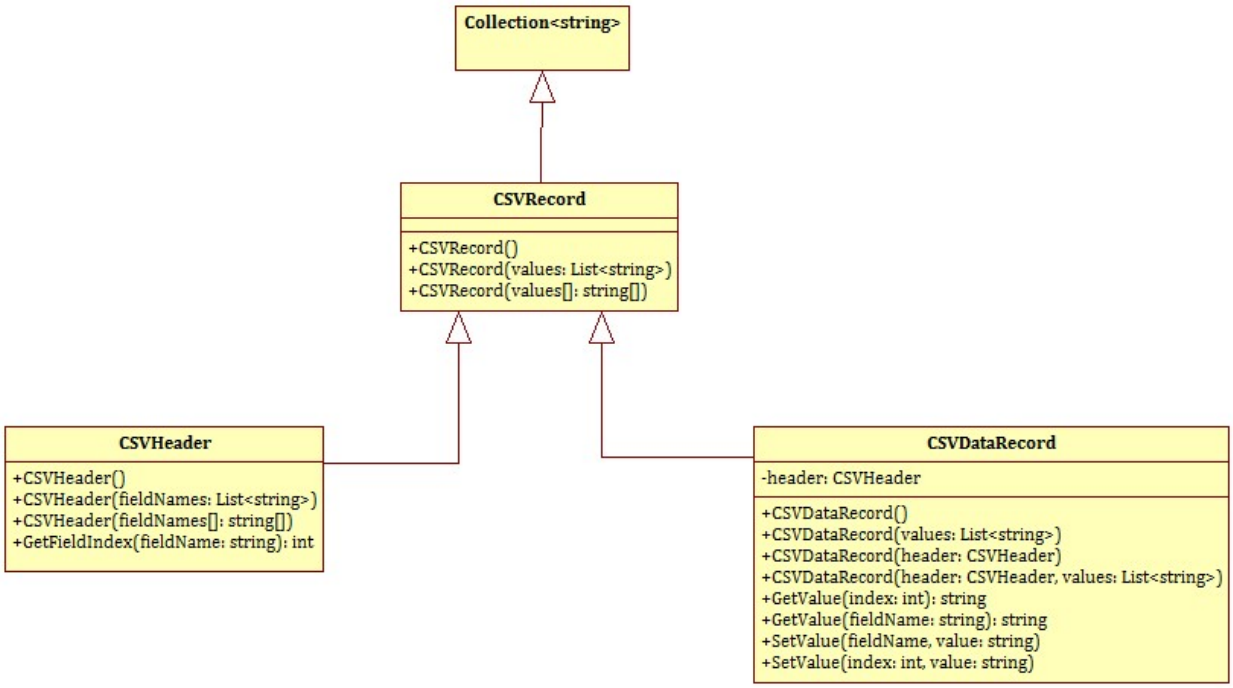
Fields

Name	Description
backgroundWorker	A BackgroundWorker to carry out an asynchronous importation.
importedData	Toy data imported by the importer.

Events

Name	Description
ImportCompleted	Raised upon completion of an asynchronous operation.
ImportProgressChanged	The event when progress is made on an asynchronous import.

5.3.4 CSVRecord, CSVHeader and CSVDataRecord



Classes

Class	Description
CSVDataRecord	Represents a csv data record.
CSVHeader	Represents the header record from a csv file.
CSVRecord	A base abstract class for CSV Records.

5.3.4.a CSVRecord

A base abstract class for CSV Records.

The **CSVRecord** will expose the following members:

Constructors

Name	Description
CSVRecord()	Initialises a new instance of the **CSVRecord**.
CSVRecord(List(String))	Initialises a new instance of the **CSVRecord**.
CSVRecord(String[])	Initialises a new instance of a **CSVRecord**.

5.3.4.b CSVHeader

Represents the header record from a csv file.

The **CSVHeader** will expose the following members:

Constructors

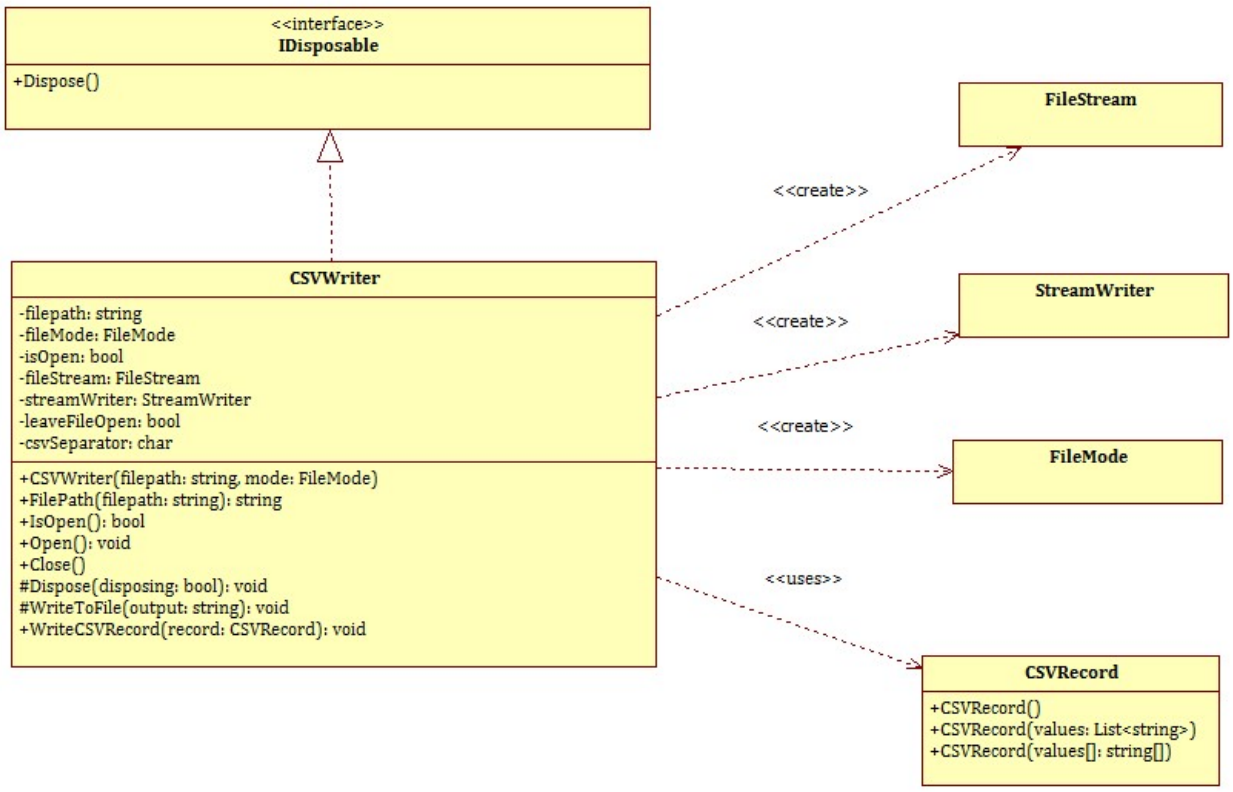
Name	Description
CSVHeader()	Initialises a **CSVHeader**.
CSVHeader(List(String))	Initialises a **CSVHeader**.
CSVHeader(String[])	Initialises a **CSVHeader**.

Methods

Name	Description
GetFieldIndex(string fieldname)	Gets the index of a field name.

5.3.5 CSVWriter

A CSVWriter to be used to write CSVRecords to a file.



Note: IDisposable, StreamWriter, FileMode and FileStream are to be supplied by .NET Framework 3.5.

The **CSVWriter** class will consist of the following members:

Constructors

Name	Description
CSVWriter	Initialises a **CSVWriter**.

Methods

Name	Description
Close	Closes the file the CSVWriter, and its associated file.
Dispose()	Implements the IDisposable.Dispose() method.
Dispose(Boolean)	Disposes of the resources that are utilised by the **CSVWriter**.
Open	Opens the file to which data is to be written.
WriteCSVRecord	Writes a CSVRecord to the file associated with this **CSVWriter**.
WriteToFile	Writes the output string to the associated file.

Fields

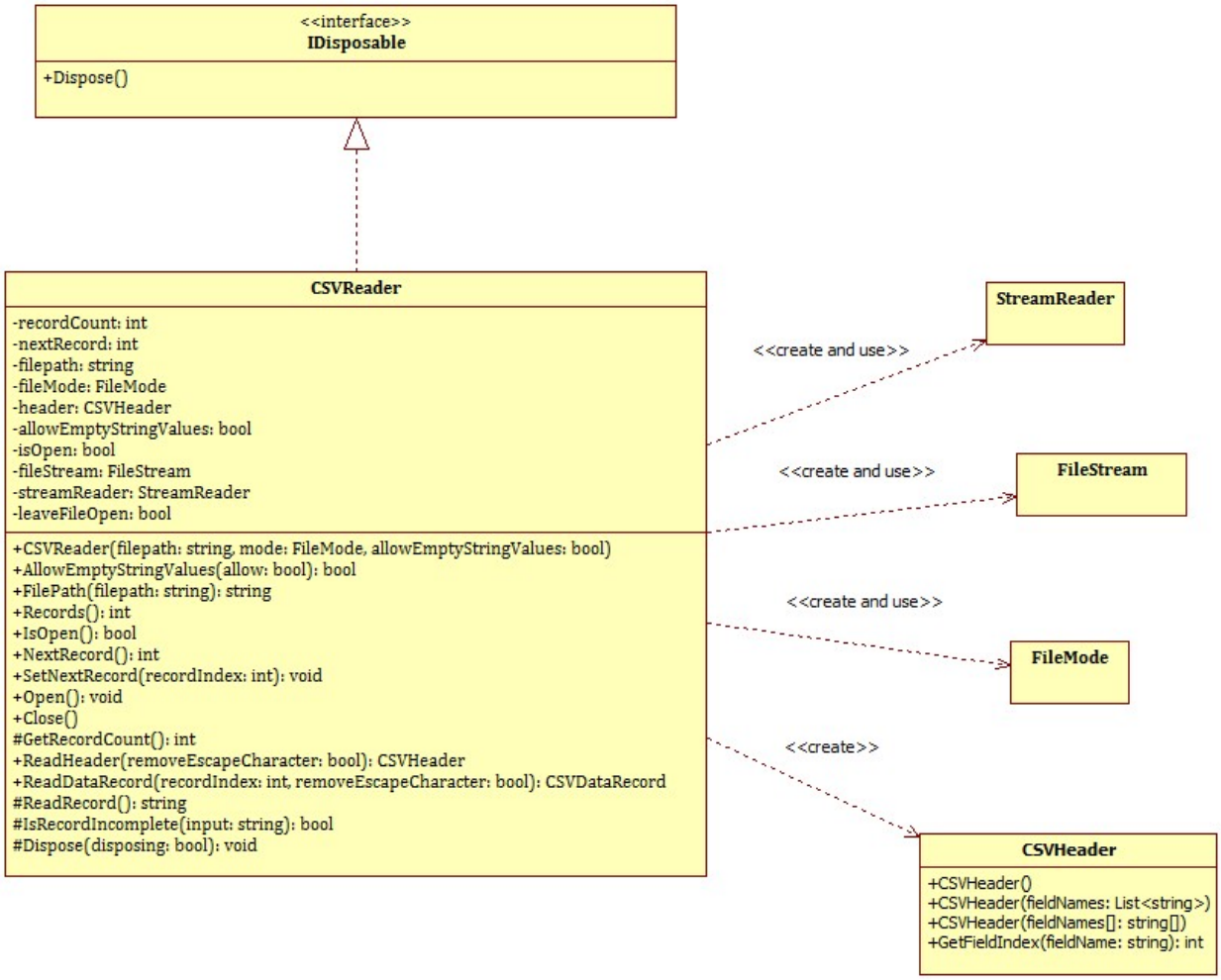
Name	Description
csvSeparator	The character that is used to separate values in a csv value string. By default it is a comma (,).
fileMode	The file mode for accessing the file.
filepath	The file path of the file, that includes the name of the file.
fileStream	A FileStream for accessing the file.
isOpen	Whether the file from which the reader reads is open.
leaveFileOpen	Whether file should be left open by the CSVReader.
streamWriter	A StreamWriter for writing to the file.

Properties

Name	Description
Filepath	Gets the filepath for the file that the CSVWriter should write data to.
IsOpen	Gets the status of the file. Returns true if the file is open.

5.3.6 CSVReader

A CSVReader that is used to read records from a csv file.



Note: IDisposable, StreamReader, FileMode and FileStream are to be supplied by .NET Framework 3.5.

The **CSVReader** class will consist of the following members:

Constructors

Name	Description
CSVReader	Initialises a CSVReader for reading from a file containing csv records.

Methods

Name	Description
Close	Closes the file that the CSVReader has open.
Dispose()	Implements IDisposable.Dispose() for the **CSVReader**.
Dispose(Boolean)	Disposes of the resources of the **CSVReader**.
GetRecordCount	Gets the number of csv records that are contained in the file.
IsRecordIncomplete	Tests if the input string is a complete csv record.
Open	Opens the file that the CSVReader should read from.
ReadDataRecord	Reads a data record from a csv file and returns the data as a CSVDataRecord.
ReadHeader	Reads the header record from a file containing csv records.
ReadRecord	Reads a record from a file and returns the record as a string.
SetNextRecord	Sets the value of the nextRecord field.
ToString	Returns a string that represents the current object. (Inherited from Object.)

Fields

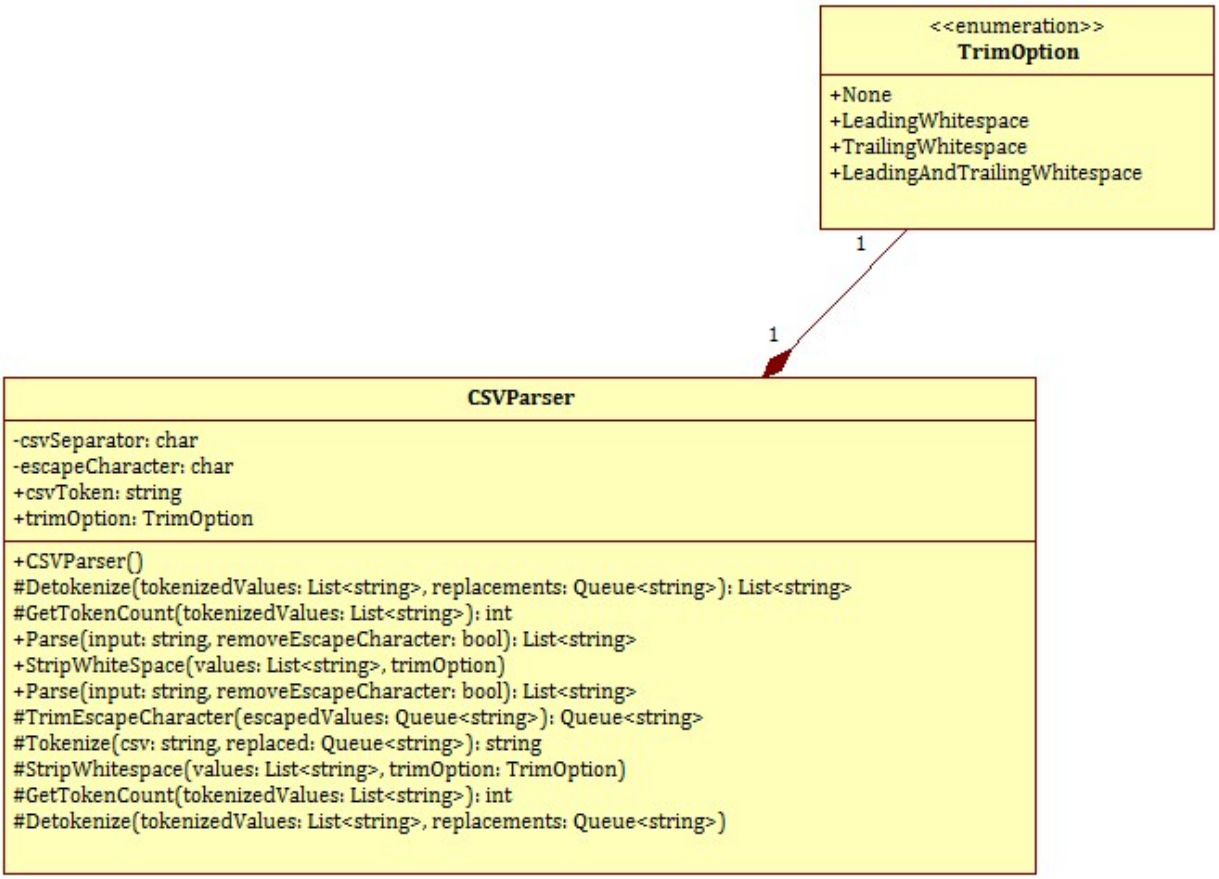
Name	Description
allowEmptyStringValues	Specifies whether empty strings are allowed as values in a csv string.
fileMode	The file mode for accessing the file.
filepath	The file path of the file, that includes the name of the file.
fileStream	A FileStream for accessing the file.

header	A csv header that contains the names of fields for records in the file.
isOpen	Whether the file from which the reader reads is open.
leaveFileOpen	Whether file should be left open by the CSVReader.
nextRecord	The index of the next record in the file.
recordCount	The number of records that can be read from the file.
streamReader	A StreamReader for reading the content of the file.

Properties

Name	Description
AllowEmptyStringValues	Gets or Sets whether the **CSVReader** should allow empty string values in a CSVRecord (either CSVHeader or CSVDataRecord
FilePath	Gets the path of the file that the CSVReader should read data from.
IsOpen	Gets whether the file is open or not.
NextRecord	Gets the zero-based index for the next record to be read. A value of -1 is returned if the current record is the last readable record.
Records	Returns the number of records available to be read by the **CSVReader**.

5.3.7 CSVParser



Classes

Class	Description
CSVParser	A CSV Parser that converts a csv string into a List(T).

Enumerations

Enumeration	Description
CSVParser.TrimOption	An enumeration that defines the values of trimming options.

5.3.7.a CSVParser class

A CSV Parser that converts a csv string into a List(T).

The **CSVParser** class will consist of the following members:

Constructors

Name	Description
CSVParser	Initializes a new instance of the **CSVParser** class

Methods

Name	Description
Detokenize	Detokenises a set of decomposed csv values.
GetTokenCount	Gets the number of tokenised values within List{T} of decomposed csv values.
Parse	Parses a csv string into a List(T) of strings that contain the component values of the csv string.
StripWhitespace	Removes the white space of values contained in values according to a specified trimming option.
Tokenize	Replaces an escaped csv value with a csv token.
TrimEscapeCharacter	Removes the escape character from a set of values from a csv value string.

Fields

Name	Description
csvSeparator	The character that is used to separate values in a csv value string. By default it is a comma (,).

csvToken	A value used as a token in tokenizing a csv string.
escapeCharacter	The character that is used to escape values in a csv value string. By default it is the double quotation (") character.
trimOption	A trim option that specifies whether the parser should remove whitespace from values contained in a csv value string. By default the parser removes both leading and trailing white space from any value within a csv value string.

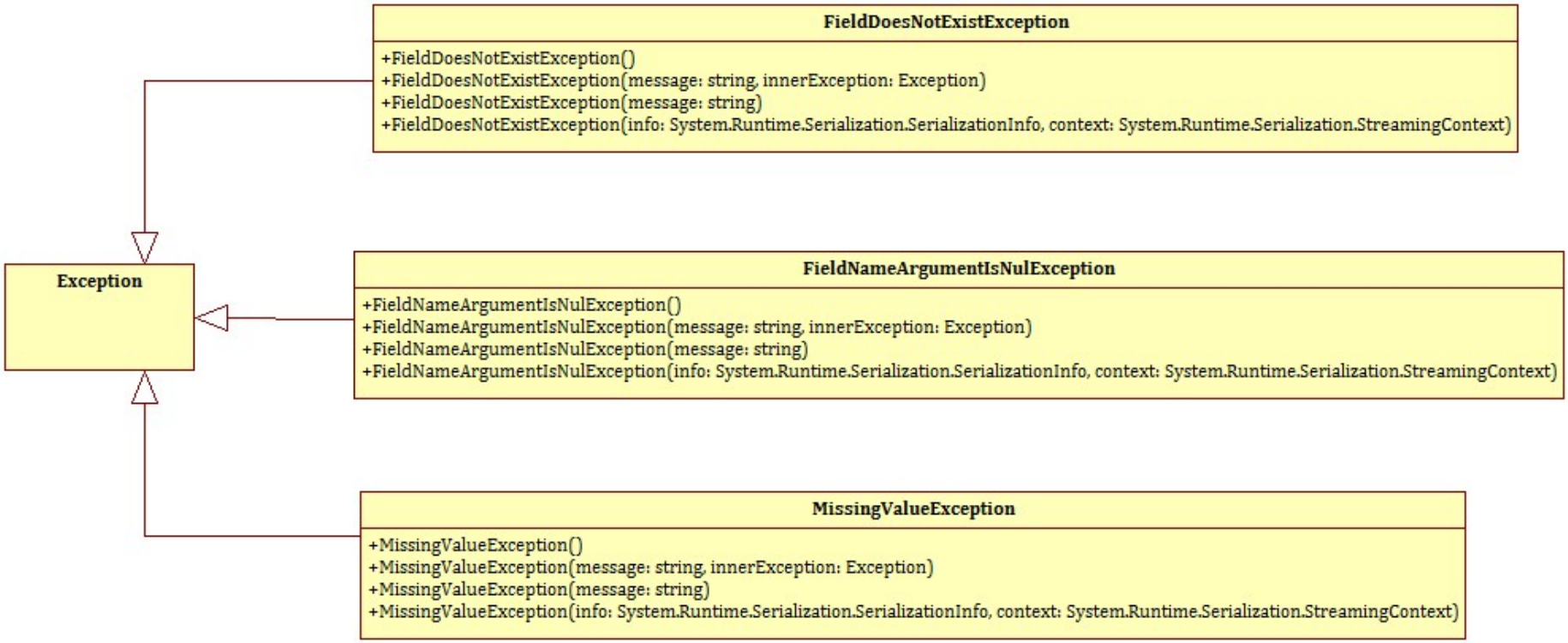
5.3.7.b TrimOption enumeration

An enumeration that defines the values of trimming options.

The TrimOption enumeration will define the following literal values:

Member name	Value	Description
None	0	Used to indicate no trimming should occur.
LeadingWhitespace	1	Used to indicate that only leading white space should be trimmed.
TrailiingWhitespace	2	Used to indicate that only trailing white space should be trimmed.
LeadingAndTrailingWhitespace	3	Used to indicate that both leading and trailing white space should be trimmed.

5.3.8 Data Access Layer Exceptions



Note: Exception class to be supplied by .NET Framework 3.5

6 Activity Diagram: Updating the current count of toys

The following activity diagram represents the workflow for using the WoodstocksIMS to update the current count of toys and forwarding the result to its stock room.