# BEA WebLogic RFID Edge Mobile SDK 1.0

## Programming with the BEA WebLogic RFID Edge Mobile SDK APIs

Version 1.0
Revised: December 2006

# Copyright



# Restricted Rights Legend



# Trademarks and Service Marks

# Contents

## 1. Introduction and Roadmap



## 2. BEA WebLogic RFID Edge Mobile SDK Overview

# 3. BEA WebLogic RFID Edge Mobile SDK APIs

# 4. Sample C# Applications

# Index

CHAPTER 1

# Introduction and Roadmap

The following sections describe the scope and organization of this document, *Programming with the BEA WebLogic RFID Edge Mobile SDK APIs*:



## Document Scope and Audience

This programming guide describes how to use the BEA WebLogic RFID Edge Mobile SDK (Software Development Kit) APIs to develop mobile applications.

The guide documents the SDK interfaces, classes, and a sample application built with the BEA WebLogic RFID Edge Mobile SDK.

## Guide to This Document

- This chapter, Introduction and Roadmap, describes the organization of this document.
- BEA WebLogic RFID Edge Mobile SDK Overview provides a general overview of the SDK and its components. The chapter introduces the concepts and terminology used throughout this guide.

- BEA WebLogic RFID Edge Mobile SDK APIs describes the SDK APIs available for mobile applications.
- Sample C# Application describes how to set up, run, and work with the bundles examples, which are a part of the installation process. You can use these applications as a starting point for developing your own applications.

# Related Documentation

This document is a part of the BEA WebLogic RFID Edge Mobile SDK documentation set. The other documents are:

- *Installing BEA WebLogic RFID Edge Mobile SDK* describes how to install the BEA WebLogic RFID Edge Mobile SDK.
- *BEA WebLogic RFID Edge Mobile SDK API HTML Reference* provides the HTML reference documentation for the APIs.
- *BEA WebLogic RFID Edge Mobile SDK 1.0.chm* is the Microsoft Windows Compiled Help Module for the APIs.
- *Microsoft MSDN Library: .NET Compact Framework 1.0* provides the documentation for the .NET Compact Framework and corresponding APIs.
- *Microsoft MSDN Library: Visual Studio .NET 2003* provides the documentation for the Microsoft Visual Studio .NET 2003 IDE.
- *Microsoft MSDN Library: Visual Studio 2005* provides the documentation for the Microsoft Visual Studio 2005 IDE.
- *Microsoft MSDN Library: Visual C#* provides the documentation for the C# language specifications.

# Using BEA WebLogic RFID Edge Mobile SDK Sample to Develop Applications

The following programming sample is installed by default in the `BEA_MOBILE_SDK_HOME/samples` directory, where `BEA_MOBILE_SDK_HOME` represents the product installation directory on the mobile device. You can modify the sample application and use it as a starting point for developing your own applications.

"Sample C# Application" on page 4-1 provides procedures for setting up your development environment, as well as instructions for compiling, running, and working with the sample application.

`BEAMobileTagRead`

This sample demonstrates how to use the BEA WebLogic Mobile SDK APIs for tag reading with EPCIS capture (send an EPCIS ObjectEvent for one or more EPC reads). The source code for this sample application is installed with the product to allow application developers to modify the code to meet their requirements and deployment environments.

CHAPTER 2

# BEA WebLogic RFID Edge Mobile SDK Overview

The following sections provide an overview of the BEA WebLogic RFID Edge Mobile SDK and describe how client applications use the SDK APIs to write mobile applications:

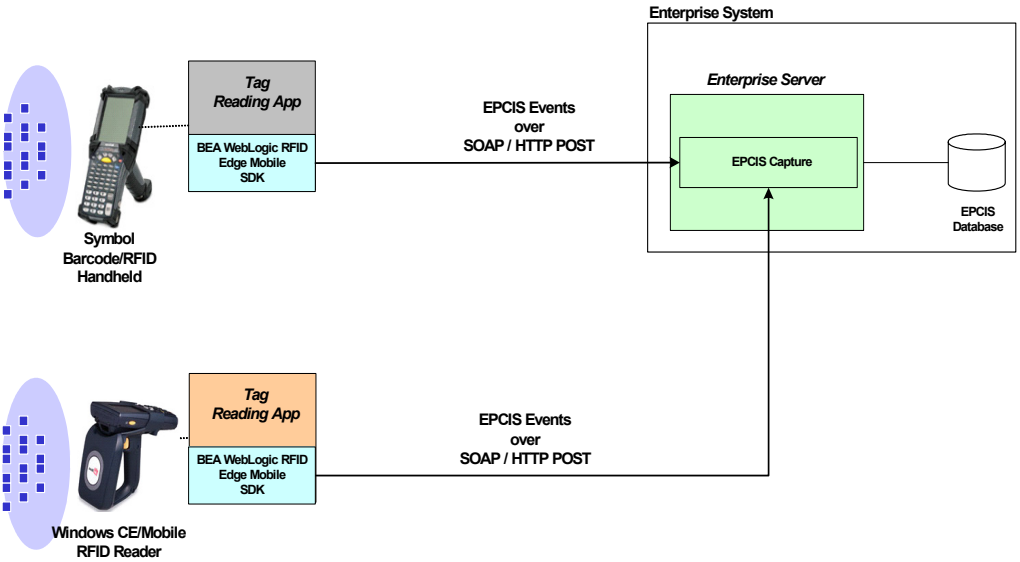


## Overview of the BEA WebLogic RFID Edge Mobile SDK

The BEA WebLogic RFID Edge Mobile SDK provides APIs and a set of libraries for application developers to write mobile applications. Mobile applications provide flexibility in scenarios where fixed reader portals are not well-suited. With mobile devices, data capture is brought to the target product as opposed to bringing the target product to the fixed reader location.

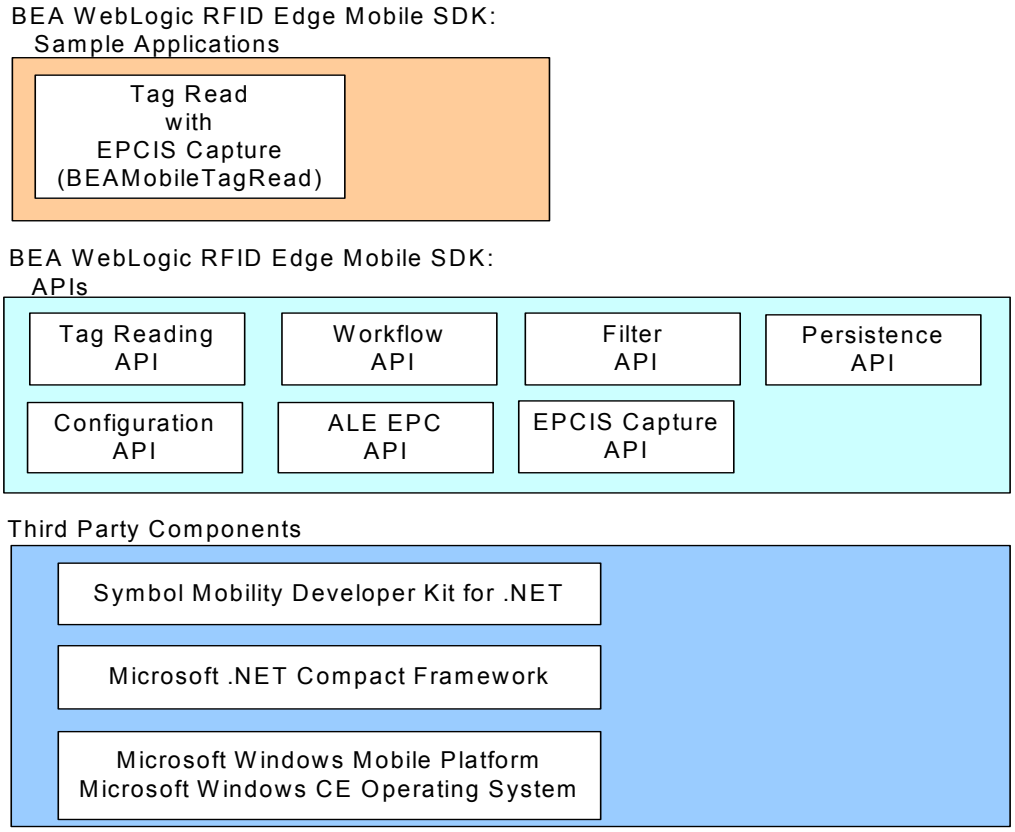
The SDK includes class libraries, associated documentation, and a sample application. Developers may utilize the sample application for proof-of-concepts or for prototyping. Developers may also extend the sample application and utilize the APIs for customization to meet desired requirements.

As illustrated below, the SDK provides support for creation of EPCIS messages which represent mobile device transactions and for transmission of the EPCIS messages over HTTP to configured destinations.


Enterprise System
Enterprise Server
Tag Reading App
BEA WebLogic RFID Edge Mobile SDK
EPCIS Events over SOAP / HTTP POST
EPCIS Capture
EPCIS Database
Symbol Barcode/RFID Handheld
Tag Reading App
BEA WebLogic RFID Edge Mobile SDK
EPCIS Events over SOAP / HTTP POST
Windows CE/Mobile RFID Reader

# BEA WebLogic RFID Edge Mobile SDK Components

The BEA WebLogic RFID Edge Mobile SDK consists of the following components which leverage the specified third party components:




# BEA WebLogic RFID Edge Mobile APIs

The APIs provided in this SDK include:

- Tag Reading API providing the following functions:
  - Create reader
  - Listen for trigger pull and release

  - Read tags
  - Close reader
- Workflow API that allows creation, starting, and stopping a workflow.

The Workflow API allows the developer to create, start, and stop a workflow.

- Filter API - for filtering tag reads in a workflow.
- Persistence API - to allow presisting EPCIS capture events to a persistence store (file system).
- Configuration API - retrieve configuration data and perform limited dynamic configuration.
- ALE EPC API - construct EPC objects utilizing EPC-standard formats.
- EPCIS Captue API - constructs EPCIS event messages (ObjectEvent) and send them to configured destinations.

# Sample Application

The sample application, BEA Mobile Tag Read,  is for reading one or more tags and sending an EPCIS ObjectEvent via the EPCIS Capture interface. The source code for the sample application is provided with the SDK to allow developers to modify the code to meet their requirements and deployment environments.

# Third Party Components

## Symbol Mobility Developer Kit for .NET

This component is the reader manufacturer-specific developer kit for .NET applications. The currently supported version is Symbol Mobility Developer Kit v1.4 for .NET.

## Microsoft .NET Compact Framework

This component is the framework for .NET applications on mobile devices. The currently supported versions are Microsoft .NET Compact Framework 1.0 SP3 and 2.0.

## Microsoft  Windows Mobile Platform

This component is the mobile platform for the device. The currently supported versions are Microsoft Windows Mobile 2003 (Pocket PC 2003) and Microsoft Windows Mobile 2005.

## Microsoft  Windows CE Operating System

This component is the operating system for the mobile device. The currently supported versions are Microsoft Windows CE .NET 4.2 and Microsoft Windows CE 5.0.

CHAPTER 3

# BEA WebLogic RFID Edge Mobile SDK APIs

The following sections describe the BEA WebLogic RFID Edge Mobile SDK API programming components that you use to read tags and send EPCIS event messages to configured destinations.



## Tag Reading API

The Tag Reading API consists of two packages:

- `com.bea.mobile.rfid.reader`
- `com.bea.mobile.rfid.readertypes`

# com.bea.mobile.rfid.reader Package

This package contains the classes and interfaces representing the physical and logical reader for the mobile device. The **PhysicalReader** interface represents the physical device, whereas the **LogicalReader** represents an antenna on the device. Currently, RFID handhelds are comprised of one antenna, but this API is designed to support multiple antennas for future scalability. The **LogicalReader** can subscribe and unsubscribe to various event types (represented by the **EventType** class) and process events accordingly. The **EventType** class resides in the `com.bea.mobile.rfid.eventhandler` package.

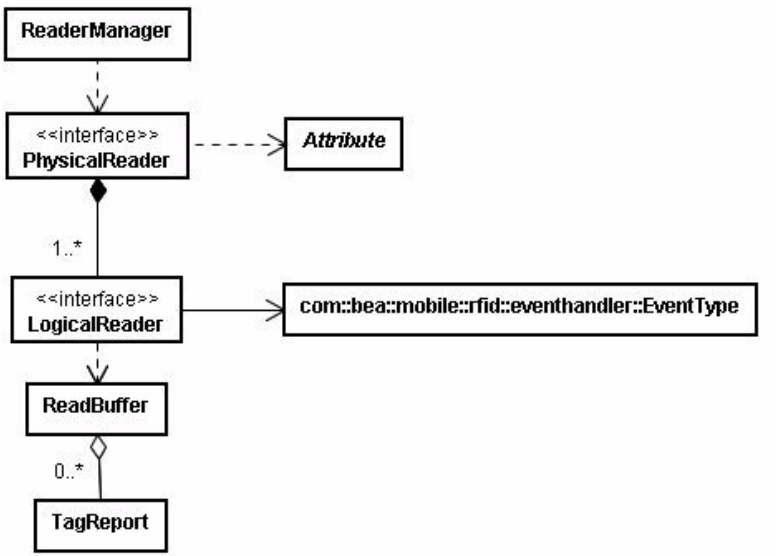
Implementations for the **PhysicalReader** and **LogicalReader** for the Symbol MC9000 RFID handheld are provided in the SDK.

As the trigger is pressed on the mobile device, the **LogicalReader** stores tag read data in its **ReadBuffer**. The **ReadBuffer** contains a **TagReport** for each EPC read.

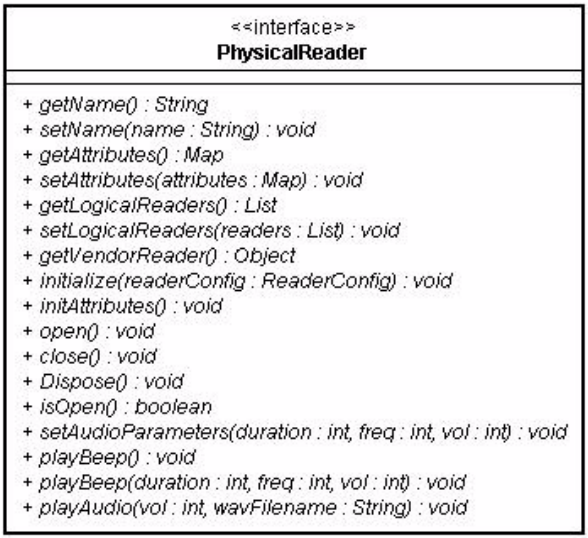
The **ReaderManager** class allows the developer to create, initialize, register, and unregister a **PhysicalReader**.

Figure 3-1 illustrates the relationships among the classes and interfaces.

**Figure 3-1 Tag Reading API UML Diagram**



## com.bea.mobile.rfid.reader.PhysicalReader Interface




The **PhysicalReader** interface represents the physcial device and contains the following methods:

**PhysicalReader**

```
getName() : String          // Returns the name of the physical reader

setName(name:String) : void // Sets the specified name of the physical
                            // reader

getAttributes() : Map       // Returns the name of the physical reader

setAttributes(attributes:Map) : void // Sets the specified map of
                            // attribute names to attribute values

getLogicalReaders() : List // Returns the list of logical readers (
// (antennas) for the mobile device)

setLogicalReaders(readers:List) : void // Sets the specified list of
                            // logical readers (antennas) for the
                            // mobile device
```

```
getVendorObject() : Object  // Returns the vendor reader object

initialize(readerConfig:ReaderConfig) : void // Performs initialization
                            // with the specified reader configuration
                            // data for the mobile device

initAttributes() : void    // Initializes the physical reader attributes

open() : void              // Enables the reader to start processing tags

close() : void              // Disables the reader from processing tags

Dispose() : void           // Frees up resources used by the reader (for
                           // when the reader is no longer needed by the
                            // application)

isOpen() : boolean          // Indicates whether or not the reader is
                            // enabled for processing tags

setAudioParameters(duration:int, frequency:int, volume:int) : void
                            // Sets the specified audio parameters

playBeep() : void          // Plays beep for default duration, frequency,
                            // and volume

playBeep(duration:int, frequency:int, volume:int) : void
                            // Plays beep for specified duration,
                            // frequency, and volume

playAudio(volume:int, wavFilename:String) : void
                            // Plays specified WAV file at specified
                            // volume
```

## com.bea.mobile.rfid.reader.LogicalReader Interface


The **LogicalReader** interface represents the logical reader and contains the following methods:

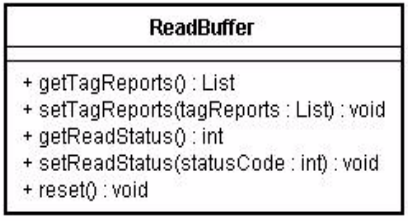
**LogicalReader**

```
getName() : String          // Returns the name of the logical reader

setName(name:String) : void // Sets the specified name of the logical
                            // reader

getAttributes() : Map       // Returns the name of the physical reader

setAttributes(attributes:Map) : void // Sets the specified map of
                            // attribute names to attribute values

setLogicalReaders(readers:List) : void // Sets the specified list of
                            // logical readers (antennas) for the
                            // mobile device
```

```
initialize(readerConfig:ReaderConfig) : void // Performs initialization
                              // with the specified reader configuration
                              // data for the mobile device

initAttributes() : void     // Initializes the logical reader attributes

supportsReading() : boolean // Indicates whether or not the logical
                              // reader supports tag reading

supportsWriting() : boolean // Indicates whether or not the logical
                              // reader supports tag writing

getReadBuffer() : ReadBuffer  // Returns the read buffer of the logical
                              // reader

resetReadBuffer() : void    // Resets the read buffer of the logical
                              // reader

powerUp() : void            // Powers up the reader

powerDown() : void          // Powers down the reader

startRead() : void         // Starts processing tag reads via read event
                              // handler

stopRead() : void           // Stops processing tag reads

flush() : void              // Flushes the logical reader (this call
                             // attempts to cancel pending reads or waits
                              // for read completion)

Dispose() : void            // Frees up resources used by the logical
                              // reader

isBusy() : boolean         // Indicates whether or not an RFID operation
                              // has been started
```

```
subscribe(type:EventType, handler:EventHandler) : void
                          // Sets subscriptions for the specified event
                            // type to the specified event handler

unsubscribe(type:EventType, handler:EventHandler) : void
                            // Unsets subscriptions for the specified
                          // event type from the specified event handler
```

## com.bea.mobile.rfid.reader.ReadBuffer Class

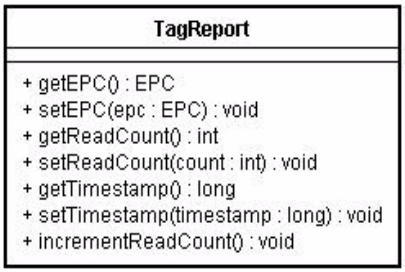



The **ReadBuffer** class contains the tag read data for the logical reader and consists of the following methods:

**ReadBuffer**

```
getTagReports() : List      // Returns the list of TagReport objects

setTagReports(tagReports:List) : void // Sets the specified list of
                            // TagReport objects

getReadStatus() : int       // Returns the read status code

setReadStatus(statusCode:int) : void // Sets the specified read status
                            // code

getLogicalReaders() : List  // Returns the list of logical readers
                            // (antennas) for the mobile device)

reset() : void              // Resets the accumulated tag read data
```

## com.bea.mobile.rfid.reader.TagReport Class

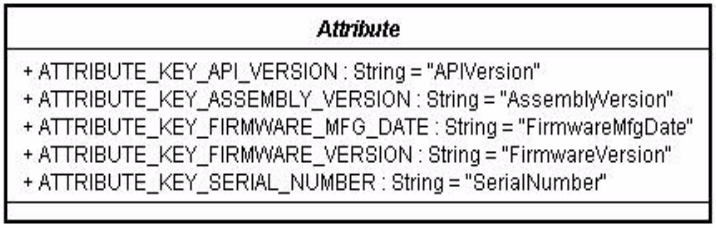



The **TagReport** class contains the individual tag read in the read buffer and consists of the following methods:

**TagReport**

```
getEPC() : EPC              // Returns the EPC object

setEPC(epc:EPC) : void      // Sets the specified EPC object

getReadCount() : int        // Returns the tag read count

setReadCount(count:int) : void // Sets the specified tag read count

getTimestamp() : long       // Returns the tag read timestamp

setTimestamp(timestamp:long) : void // Sets the specified tag read
                            // timestamp

incrementReadCount() : void // Increments the tag read count
```

## com.bea.mobile.rfid.reader.Attribute Class

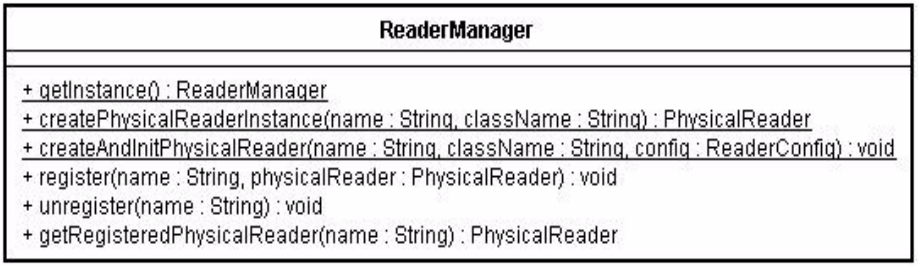



The **Attribute** abstract class represents attributes of the mobile device and contains the following static variables:

**Attribute**

```
ATTRIBUTE_KEY_API_VERSION : String // The reader API version

ATTRIBUTE_KEY_ASSEMBLY_VERSION : String // The assembly version

ATTRIBUTE_KEY_FIRMWARE_MFG_DATE : String // Manufacturer date

ATTRIBUTE_KEY_FIRMWARE_VERSION : String // Firmware version

ATTRIBUTE_KEY_SERIAL_NUMBER : String // Serial number
```

## com.bea.mobile.rfid.reader.ReaderManager Class



The **ReaderManager** singleton class supports the lifecycle management of a **PhysicalReader** and contains the following methods:

**ReaderManager**

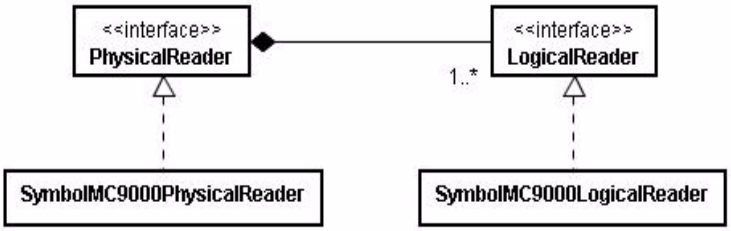
```
getInstance() : ReaderManager // Returns the ReaderManager instance

createPhysicalReaderInstance(name:String, className:String) :
PhysicalReader              // Creates the PhysicalReader instance for
                            // the specified name and class name

createAndInitPhysicalReader(name:String, className:String,
config:ReaderConfig) : PhysicalReader // Creates and initializes the
                           // PhysicalReader instance for the specified
                            // name, class name, and reader config

register(name:String, physicalReader:PhysicalReader) : void
                            // Registers the specified physical reader
                            // with the specified name

unregister(name:String) : void // Unregisters the physical reader with
                            // the specified name

getRegisteredPhysicalReader(name:String) : PhysicalReader // Returns the
                            // PhysicalReader instance registered with
                            // the specified name
```

# com.bea.mobile.rfid.readertypes Package

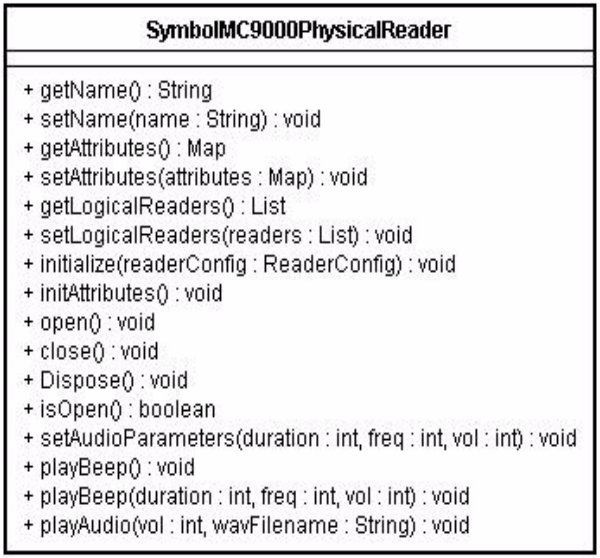
This package contains the implementation classes for the physical and logical readers. The **SymbolMC9000PhysicalReader** class provides the concrete implementation of the **PhysicalReader** interface for the Symbol MC9000 RFID handheld. The **SymbolMC9000LogicalReader** class provides the concrete implementation of the **LogicalReader** interface.

Figure 3-2 illustrates the relationships among the classes and interfaces

**Figure 3-2 Symbol Tag Reading API UML Diagram**

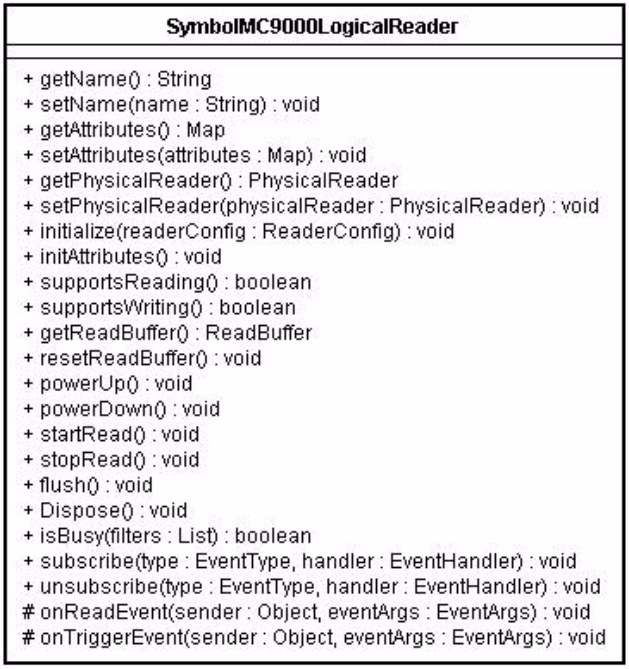



## com.bea.mobile.rfid.readertypes.SymbolMC9000PhysicalReader Class




The **SymbolMC9000PhysicalReader** class implements the **PhysicalReader** interface and provides concrete method implementations for all methods documented in "PhysicalReader" on page 3-3.

## com.bea.mobile.rfid.readertypes.SymbolMC9000LogicalReader Class




The **SymbolMC9000LogicalReader** class implements the **LogicalReader** interface and provides concrete method implementations for all methods documented in "LogicalReader" on page 3-5.

In addition to the interface methods, the class contains the following methods:

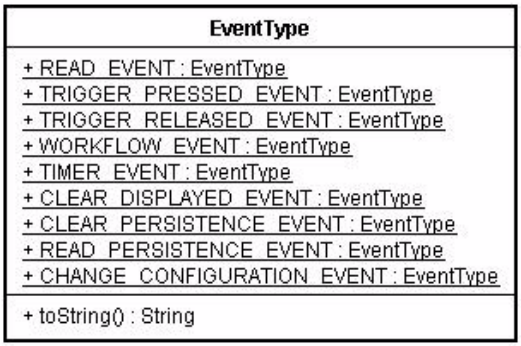
**SymbolMC9000LogicalReader - Additional Methods**

```
onReadEvent(sender:Object, eventArgs:EventArgs) : void // Processes the
                              // specified read event

onTriggerEvent(sender:Object, eventArgs:EventArgs) : void // Processes
                              // the specified trigger event
```

# com.bea.mobile.rfid.eventhandler Package

This package contains classes supporting event handling for mobile applications.

## com.bea.mobile.rfid.eventhandler.EventType Class




The **EventType** class is a type-safe enumeration class and represents supported event types. The class consists of the following attributes and methods:

**EventType**

```
READ_EVENT : EventType      // Read event

TRIGGER_PRESSED_EVENT : EventType // Trigger pressed event

TRIGGER_RELEASED_EVENT : EventType // Trigger released event

WORKFLOW_EVENT : EventType  // Workflow event

TIMER_EVENT : EventType     // Timer event

CLEAR_DISPLAYED_EVENT : EventType // Clear displayed event

CLEAR_PERSISTENCE_EVENT : EventType // Clear persistence event

READ_PERSISTENCE_EVENT : EventType // Read persistence event
```

```
CHANGE_CONFIGURATION_EVENT : EventType // Change configuration event

toString() : String          // Returns the string representation of the
                              // the event
```

# Workflow API

The Workflow API consists of the `com.bea.mobile.rfid.workflow` package.

## com.bea.mobile.rfid.workflow Package

This package contains the classes and interfaces representing the workflow framework for the mobile device. The **WorkflowFactory** interface provides the ability to create a **Workflow** instance.

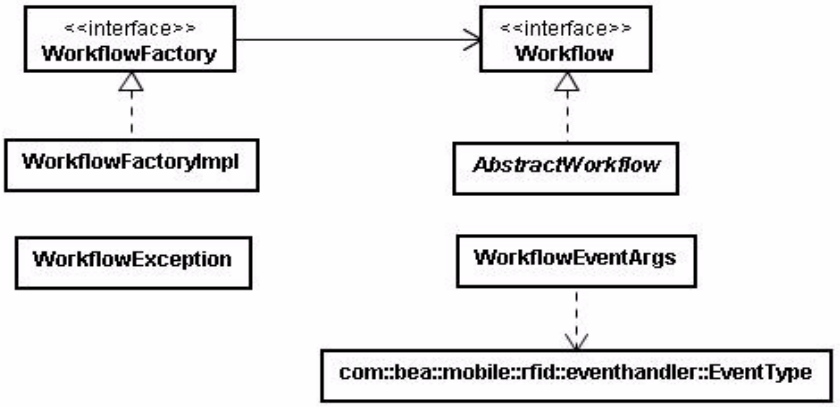
The **AbstractWorkflow** class provides an abstract implementation of the **Workflow** interface.

The **WorkflowEventArgs** class contains the event type and event info for workflow processing.

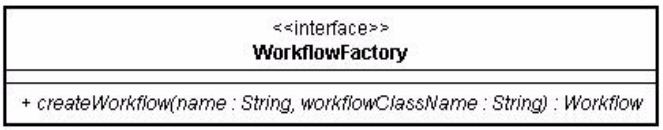
The **EventType** class resides in the `com.bea.mobile.rfid.eventhandler` package and represents the supported event types. See "com.bea.mobile.rfid.eventhandler Package" on page 3-13 for details on the **EventType** class.

Figure 3-3 illustrates the relationships among the classes and interfaces

**Figure 3-3 Workflow API UML Diagram**



## com.bea.mobile.rfid.workflow.WorkflowFactory Class

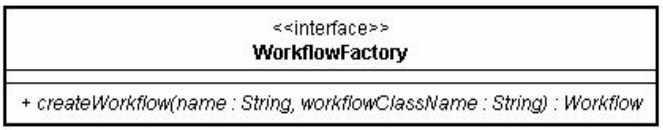



The **WorkflowFactory** interface represents a factory for creating workflow objects. This interface contains the following method:com.bea.mobile.rfid.workflow.WorkflowFactory Class

**WorkflowFactory**

```
createWorkflow(name:String, workflowClassName:String) : Workflow
                              // Returns an instance of the specified
                              // workflow name and workflow class name
```

## com.bea.rfid.workflow.WorkflowFactoryImpl Class



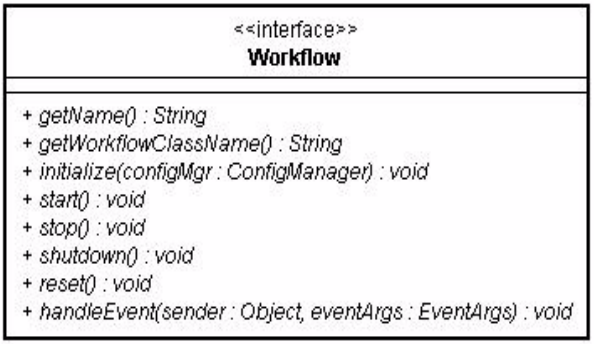

The **WorkflowFactoryImpl** class implements the **WorkflowFactory** interface and provides concrete method implementations for all methods documented in "WorkflowFactory" on page 3-16.

In addition to the interface methods, the class contains the following methods:

**WorkflowFactoryImpl - Additional Methods**

```
getInstance() : WorkflowFactory // Returns the WorkflowFactory singleton
                              // instance
```

## com.bea.mobile.rfid.workflow.Workflow Interface

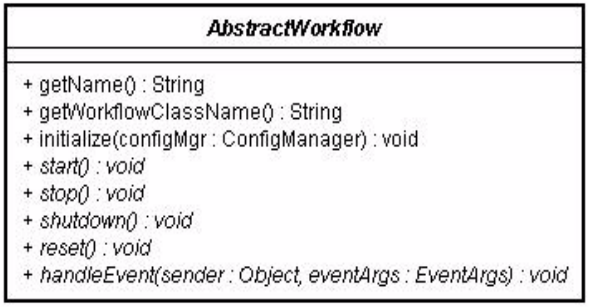



The **Workflow** interface represents a workflow within a mobile device application. This interface contains the following methods:

**Workflow**

```
getName() : String           // Returns the workflow name

getWorkflowName() : String   // Returns the workflow class name

initialize(configMgr:ConfigManager) : void // Performs initialization of
                             // the workflow based on the specified
                             // configuration manager

start() : void               // Starts the workflow

stop() : void                // Stops the workflow

shutdown() : void            // Shuts down the workflow

reset() : void               // Resets the workflow

handleEvent(sender:Object, eventArgs:EventArgs) : void // Processes the
                             // specified event
```

## com.bea.mobile.rfid.workflow.AbstractWorkflow Class




The **AbstractWorkflow** class is an abstract class that implements the **Workflow** interface. The interfacde represents a factory for creating workflow objects. This class contains the following implementation methods:

**AbstractWorkflow - Implementation Methods**

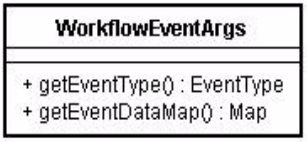
```
getName() : String           // Returns the workflow name

getWorkflowName() : String   // Returns the workflow class name

initialize(configMgr:ConfigManager) : void // Performs initialization of
                             // the workflow based on the specified
                             // configuration manager
```

The class contains the following abstract methods:

**AbstractWorkflow - Abstract Methods**

```
start() : void               // Starts the workflow

stop() : void                // Stops the workflow

shutdown() : void            // Shuts down the workflow

reset() : void               // Resets the workflow

handleEvent(sender:Object, eventArgs:EventArgs) : void // Processes the
                             // specified event
```

## com.bea.mobile.rfid.workflow.WorkflowEventArgs Class

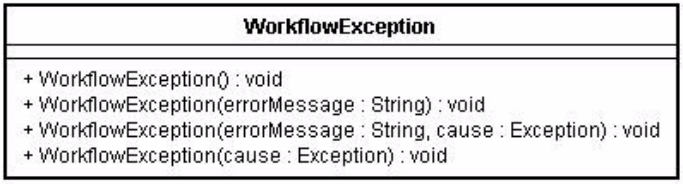



The **WorkflowEventArgs** class provides workflow-specific event data during event handling. This class contains the following methods:

**WorkflowEventArgs**

```
getEventType() : EventType  // Returns the event type object

getEventDataMap() : Map     // Returns the event data map of event names
                            // to event values
```

## com.bea.mobile.rfid.workflow.WorkflowException Class




The **WorkflowException** class represents an error during workflow processing. This class contains the following methods:

**WorkflowException**

```
WorkflowException() : void  // Default constructor

WorkflowException(errorMessage:String) : void // Constructor with error
                            // message parameter
```

```
WorkflowException(errorMessage:String, cause:Exception) : void
                              // Constructor with error message and cause
                              // exception parameters

WorkflowException(cause:Exception) : void  // Constructor with cause
                              // exception parameter
```

# Filter API

The Filter API consists of two packages:

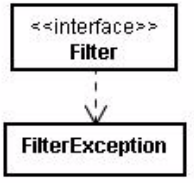
- `com.bea.mobile.rfid.filter`
- `com.bea.mobile.rfid.filtertypes`

## com.bea.mobile.rfid.filter Package

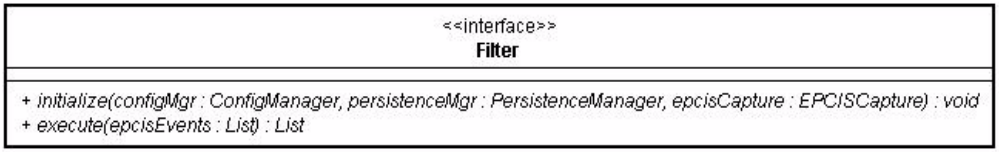
This package contains the classes and interfaces representing the filters for a workflow to apply on tag reads. The **Filter** interface provides the ability to refine the reported tag reads based on specific criteria.

Figure 3-4 illustrates the relationships among the classes and interfaces

**Figure 3-4 Filter API UML Diagram**



## com.bea.mobile.rfid.filter.Filter Interface

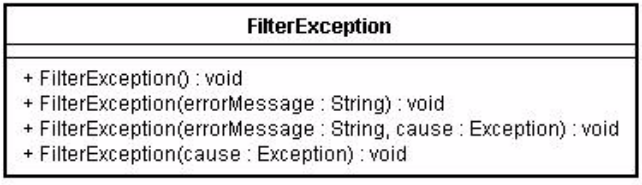



The **Filter** interface represents a workflow filter and contains the following methods:

**Filter**

```
initialize(configMgr:ConfigManager, persistenceMgr:PersistenceManager,
epcisCapture:EPCISCapture) : void  // Initializes the filter with the
                                   // specified objects

execute(epcisEvents:List) : List // Applies the filter to the specified
                                 // list of EPCISEvent objects and returns the
                                   // filtered list of EPCISEvent objects
```

## com.bea.mobile.rfid.filter.FilterException Class




The **FilterException** class represents an error during filter processing. This class contains the following methods:

**FilterException**

```
FilterException() : void  // Default constructor

FilterException(errorMessage:String) : void // Constructor with error
                                   // message parameter
```

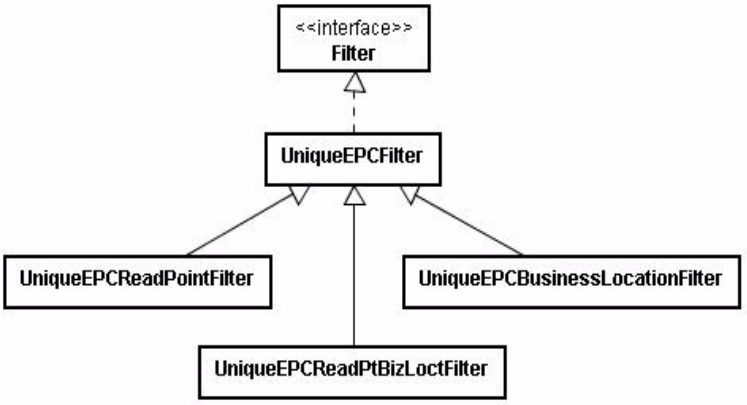
```
FilterException(errorMessage:String, cause:Exception) : void
                              // Constructor with error message and cause
                              // exception parameters

FilterException(cause:Exception) : void  // Constructor with cause
                              // exception parameter
```

# com.bea.mobile.rfid.filtertypes Package

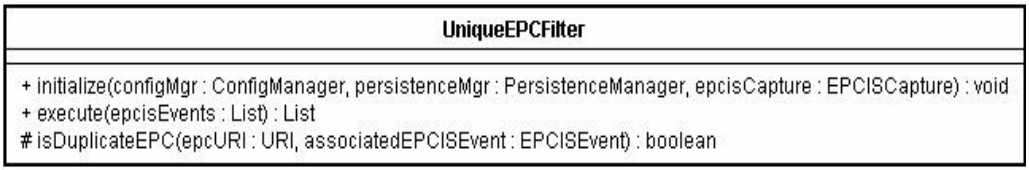
This package contains implementation classes for the **Filter** interface..

Figure 3-5 illustrates the relationships among the classes and interfaces

**Figure 3-5 EPC Filter APIs UML Diagram**




## com.bea.mobile.rfid.filtertypes.UniqueEPCFilter Class

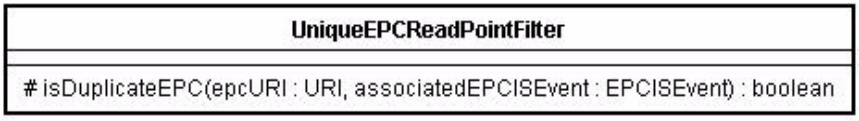


The **UniqueEPCFilter** class provides the ability to filter out duplicate EPC reads. This class contains the following methods:

**UniqueEPCFilter**

```
initialize(configMgr:ConfigManager, persistenceMgr:PersistenceManager,
epcisCapture:EPCISCapture) : void  // Initializes the filter with the
                                   // specified objects

execute(epcisEvents:List) : List // Applies a unique EPC filter to the
                                 // specified list of EPCISEvent objects and
                                 // returns the filtered list of EPCISEvent
                                 // objects

isDuplicateEPC(epcURI:URI, associatedEPCISEvent:EPCISEvent) : void
                                 // Indicates whether or not specified EPC
                                 // is a duplicate read
```

## com.bea.mobile.rfid.filtertypes.UniqueEPCReadPointFilter Class

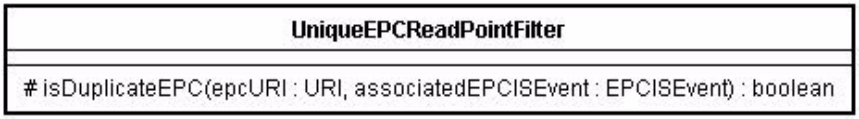



The **UniqueEPCReadPointFilter** class extends the **UniqueEPCFilter** class and overrrides the *isDuplicateEPC()* method to filter on EPC and read point. This class contains the following methods:

**UniqueEPCReadPointFilter**

```
isDuplicateEPC(epcURI:URI, associatedEPCISEvent:EPCISEvent) : void
                                   // Indicates whether or not specified EPC
                                   // is a duplicate read
```

## com.bea.mobile.rfid.filtertypes.UniqueEPCBusinessLocationFilter Class

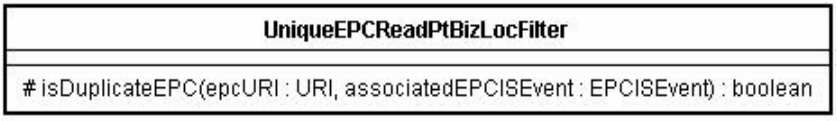



The **UniqueEPCBusinessLocationFilter** class extends the **UniqueEPCFilter** class and overrrides the *isDuplicateEPC()* method to filter on EPC and business location. This class contains the following methods:

**UniqueEPCBusinessLocationFilter**

```
isDuplicateEPC(epcURI:URI, associatedEPCISEvent:EPCISEvent) : void
                            // Indicates whether or not specified EPC
                            // is a duplicate read
```

## com.bea.mobile.rfid.filtertypes.UniqueEPCReadPtBizLocFilter Class




The **UniqueEPCReadPtBizLocFilter** class extends the **UniqueEPCFilter** class and overrrides the *isDuplicateEPC()* method to filter on EPC, read point, and business location. This class contains the following methods:

**UniqueEPCReadPtBizLocFilter**

```
isDuplicateEPC(epcURI:URI, associatedEPCISEvent:EPCISEvent) : void
                            // Indicates whether or not specified EPC
                            // is a duplicate read
```

# Configuration API

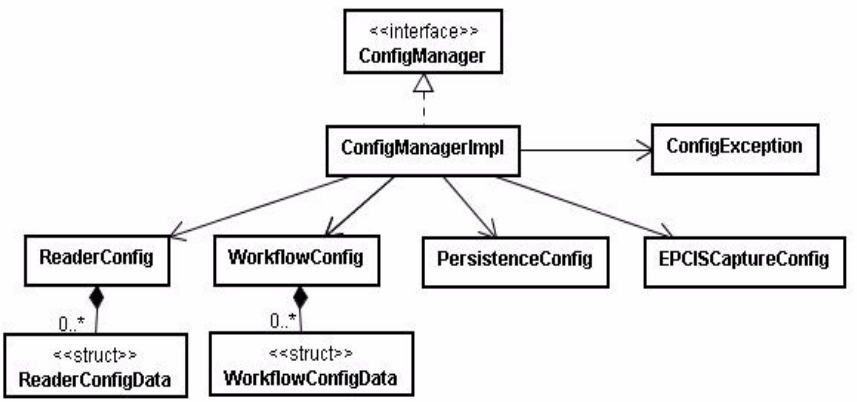
The Configuration API consists of the `com.bea.mobile.rfid.config` package.

## com.bea.mobile.rfid.config Package

This package contains the classes and interfaces for supporting configuration management on the mobile device. The Configuration API provides central management of reader, workflow, EPCIS event, and persistence configuration data.

Figure 3-6 illustrates the relationships among the classes and interfaces

**Figure 3-6 Configuration API UML Diagram**




## com.bea.mobile.rfid.config.ConfigManager Interface

The **ConfigManager** interface provides centralized configuration management for the mobile device application. This interface contains the following methods:

**ConfigManager**

```
loadConfig(appName:String) : void  // Loads a configuration file based on
                            // the specified application name
                            // (e.g., App1.config.xml)

loadConfig((appName:String, isResetAppConfig:boolean) : void
                             // The isResetAppConfig Boolean flag
                            // indicates wheter or not to reset the
                            // AppConfig object before loading the
                            // specified  configuration

getAppConfig : Map          // Returns a map of configuration property
                            // names to values

getValue(key:String) : String // Returns a value for the specified
                            // configuration property name
```

## com.bea.rfid.config.ConfigManagerImpl Class

The **ConfigManagerImpl** class implements the **ConfigManager** interface and provides concrete method implementations for all methods documented in Table "ConfigManager" on page 3-26.

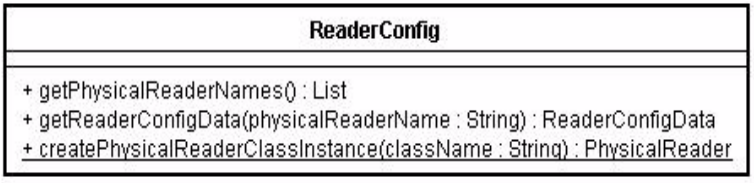
In addition to the interface methods, the class contains the following methods:

**ConfigManagerImpl - Additional Methods**

```
getInstance() : ConfigManagerImpl // Returns the ConfigManagerImpl
                                  // singleton instance

getWorkflowConfig() : WorkflowConfig // Returns the WorkflowConfig
                                     // instance

getEPCISCaptureConfig() : EPCISCaptureConfig // Returns the
                                     // EPCISCaptureConfig instance

getPersistenceConfig() : PersistenceConfig // Returns the
                                     // PersistenceConfig instance
```

```
getReaderConfig() : ReaderConfig // Returns the ReaderConfig
                                 // instance

updateConfig(key:String, value:String) : void // Updates the
                                 // configuration with the specified
                                 // key/value pair
```

## com.bea.mobile.rfid.config.ReaderConfig Class

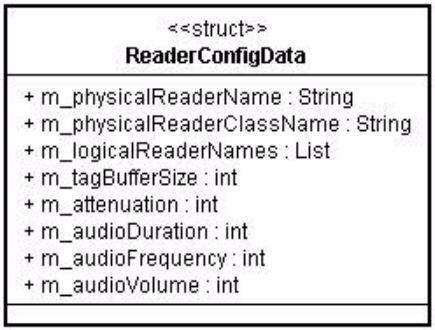



The **ReaderConfig** class manages the configuration data for the readers. This class contains the following methods:

**ReaderConfig**

```
getPhysicalReaderNames() : List // Returns the list of configured
                                // physical reader names

getReaderConfigData(physicalReaderName:String) :
ReaderConfig.ReaderConfigData // Returns the ReaderConfigData struct for
                                // the specified physical reader name

createPhysicalReaderClassInstance(physicalReaderClassName:String) :
PhysicalReader                  // Returns an instance of the specified
                                // physical reader class name
```

## com.bea.mobile.rfid.config.ReaderConfig.ReaderConfigData Struct

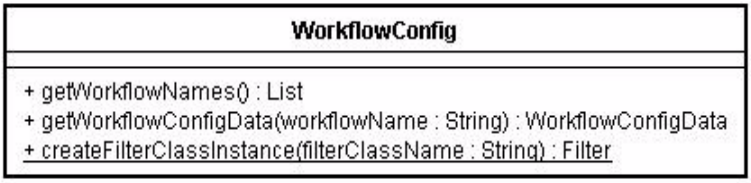



The **ReaderConfigData** struct contains the configuration data for a reader. This struct contains the following attributes:

**ReaderConfigData**

```
m_physicalReaderName : String // The physical reader name

m_physicalReaderClassName : String // The physical reader class name

m_logicalReaderNames : List // The list of configured logical reader
                            // names

m_tagBufferSize : int       // The tag buffer size for the reader

m_attenuation : int         // The attenuation level

m_audioDuration : int       // The audio duration

m_audioFrequency : int      // The audio frequency

m_audioVolume : int         // The audio volume
```

## com.bea.mobile.rfid.config.WorkflowConfig Class

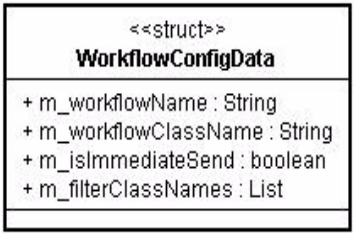



The **WorkflowConfig** class manages the configuration data for the workflow. This class contains the following methods:

**WorkflowConfig**

```
getWorkflowNames() : List   // Returns the list of configured workflow
                            // names

getWorkflowConfigData(workflowName:String) :
WorkflowConfig.WorkflowConfigData // Returns the WorkflowConfigData
                            // struct for the specified workflow name

createFilterClassInstance(filterClassName:String) : Filter
                            // Returns an instance of the specified
                            // filter class name
```

## com.bea.mobile.rfid.config.WorkflowConfig.WorkflowConfigData Struct

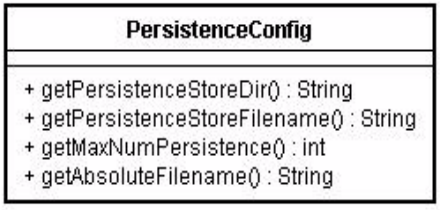


The **WorkflowConfigData** struct contains the configuration data for a workflow. This struct contains the following attributes:

**WorkflowConfigData**

```
m_workflowName : String     // The workflow name

m_workflowClassName : String // The workflow class name

m_isImmediateSend : boolean // Indicates whether or not the workflow is
                            // to immediately send the tag read as an
                           // EPCIS event to the configured destination

m_filterClassNames : List   // The list of configured filter class names
```

## com.bea.mobile.rfid.config.PersistenceConfig Class




The **PersistenceConfig** class manages the configuration data for the persistence store. This class contains the following methods:

**PersistenceConfig**

```
getPersistenceStoreDir() : String // Returns the persistence storage
                            // directory

getPersistenceStoreFilename() : String // Returns the persistence
                            // storage filename
```

```
getMaxNumPersistence() : int // Returns the maximum number of data
                             // objects to store

getAbsoluteFilename() : String // Returns the absolute filename
                               // for the persistence storage file
```

## com.bea.mobile.rfid.config.EPCISCaptureConfig Class

The **EPCISCaptureConfig** class represents the configuration data associated with EPCIS event messages sent from the mobile device. This class supports reading the configuration data as well as updating/adding configuration data. This class contains the following methods:

**EPCISCaptureConfig**

```
getReadPointURIs() : List   // Returns the list of available read point
                            // URI objects from which the user chooses

setReadPointURIs(uris:List) : void // Sets the specified list of
                            // available read point URI objects from
                            // which the user chooses

getBusinessLocationURIs() : List // Returns the list of available
                            // business location URI objects from which
                            // the user chooses

setBusinessLocationURIs(uris:List) : void // Sets the specified list of
                           // available business location URI objects
                            // from which the user chooses

getBusinessStepURIs() : List // Returns the list of available
                            // business step URI objects from which
                            // the user chooses

setBusinessStepURIs(uris:List) : void // Sets the specified list of
                            // available business step URI objects
                            // from which the user chooses

getDispositionURIs() : List // Returns the list of available
                            // disposition URI objects from which
                            // the user chooses

setDispositionURIs(uris:List) : void // Sets the specified list of
                            // available disposition URI objects
                            // from which the user chooses

getBusinessTransactions() : List // Returns the list of available
                            // business transaction objects from which
                            // the user chooses
```

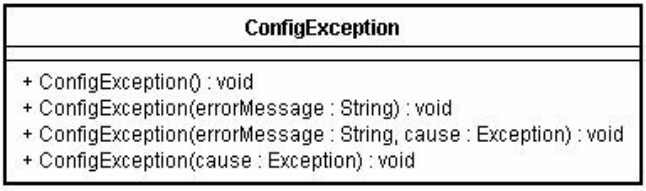
```
setBusinessTransactions(uris:List) : void // Sets the specified list of
                            // available business transaction objects
                            // from which the user chooses

getDestinationURIs() : List // Returns the list of available
                            // destination URI objects from which
                            // the user chooses

setDestinationURIs(uris:List) : void // Sets the specified list of
                            // available destination URI objects
                            // from which the user chooses

getAction() : EPCISAction   // Returns the EPCISAction object indicating
                            // the EPCIS event action (ADD,
                            // OBSERVE, DELETE)

setAction(action:EPCISAction) : void // Sets the specified EPCISAction
                            // object indicating the EPCIS event action

getTransport() : EPCISTransport // Returns the EPCISTransport object
                            // indicating the transport (HTTP_POST,
                            // SOAP) to deliver the EPCIS message

setTransport(transport:EPCISTransport) : void // Sets the specified
                            // EPCISTransport object indicating the
                            // transport to deliver the EPCIS message

getSocketTimeout() : long   // Returns the socket timeout value for
                            // the HTTP POST transport

setSocketTimeout(timeout:long) : void // Sets the specified socket
                           // timeout value for  the HTTP POST transport

getCurrentReadPointURI() : URI // Returns the current read point
                            // URI chosen by the user

setCurrentReadPointURI(uri:URI) : void // Sets the specified read point
                            // URI as the current read point URI chosen
                            // by the user
```

```
addReadPointURI(uri:URI) : void // Adds the specified read point
                            // URI to the list of available read point
                            // URI objects

deleteReadPointURI(uri:URI) : void // Deletes the specified read point
                           // URI from the list of available read point
                            // URI objects

getCurrentBusinessLocationURI() : URI // Returns the current business
                            // location URI chosen by the user

setCurrentBusinessLocationURI(uri:URI) : void // Sets the specified
                            // business location URI as the current
                            // business location URI chosen by the user

addBusinessLocationURI(uri:URI) : void // Adds the specified business
                            // location URI to the list of available
                            // business location URI objects

deleteBusinessLocationURI(uri:URI) : void // Deletes the specified
                            // business location URI from the list of
                            // available business location URI objects

getCurrentBusinessStepURI() : URI // Returns the current business
                            // step URI chosen by the user

setCurrentBusinessStepURI(uri:URI) : void // Sets the specified
                           // business step URI as the current business
                            // step URI chosen by the user

addBusinessStepURI(uri:URI) : void // Adds the specified business
                            // step URI to the list of available
                            // business step URI objects

deleteBusinessStepURI(uri:URI) : void // Deletes the specified
                            // business step URI from the list of
                            // available business step URI objects

getCurrentDispositionURI() : URI // Returns the current disposition
                            // URI chosen by the user
```

```
setCurrentDispositionURI(uri:URI) : void // Sets the specified
                            // disposition URI as the current
                            // disposition URI chosen by the user

addDispositionURI(uri:URI) : void // Adds the specified disposition
                            // URI to the list of available
                            // disposition URI objects

deleteDispositionURI(uri:URI) : void // Deletes the specified
                            // disposition URI from the list of
                            // available disposition URI objects

getCurrentBusinessTransaction() : BusinessTransaction // Returns the
                           // current business transaction chosen by the
                            // user

setCurrentBusinessTransaction(bt:BusinessTransaction) : void // Sets the
                            // specified business transaction as the
                           // current business transaction chosen by the
                            // user

addBusinessTransaction(bt:BusinessTransaction) : void // Adds the
                          // specified business transaction to the list
                            // of available business transaction objects

deleteBusinessTransaction(bt:BusinessTransaction) : void // Deletes the
                            // specified business transaction from the
                            // list of available business transaction
                            // objects

getCurrentDestinationURI() : URI // Returns the current destination
                            // URI chosen by the user

setCurrentDestinationURI(uri:URI) : void // Sets the specified
                            // destination URI as the current
                            // destination URI chosen by the user
```

```
addDestinationURI(uri:URI) : void // Adds the specified destination
                              // URI to the list of available
                              // destination URI objects

deleteDestinationURI(uri:URI) : void // Deletes the specified
                              // destination URI from the list of
                              // available destination URI objects
```

## com.bea.mobile.rfid.config.ConfigException Class




The **ConfigException** class represents an error during configuration processing. This class contains the following methods:

**ConfigException**

```
ConfigException() : void    // Default constructor

ConfigException(errorMessage:String) : void // Constructor with
                              // error message parameter

ConfigException(errorMessage:String, cause:Exception) : void
                              // Constructor with error message and cause
                              // exception parameters

ConfigException(cause:Exception) : void  // Constructor with cause
                              // exception parameter
```

# Persistence API

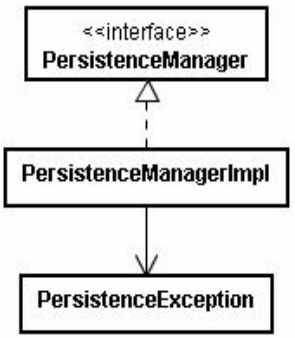
The Persistence API consists of the `com.bea.mobile.rfid.persistence` package.

## com.bea.mobile.rfid.persistence Package

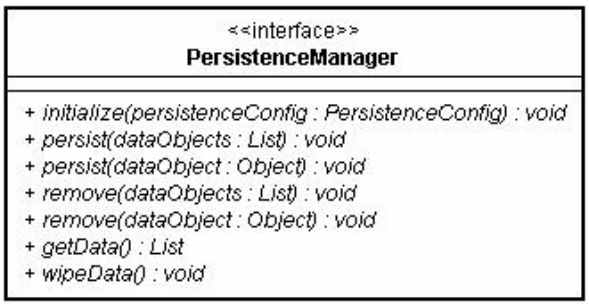
This package contains the classes and interfaces for supporting the persistence of tag reads onto the filesystem of the mobile device. Note that location data and EPCIS event data will be persisted for each tag read.

Figure 3-7 illustrates the relationships among the classes and interfaces

**Figure 3-7 Persistence API UML Diagram**




### com.bea.mobile.rfid.persistence.PersistenceManager Interface




The **PersistenceManager** interface provides centralized persistence management and contains the following methods:

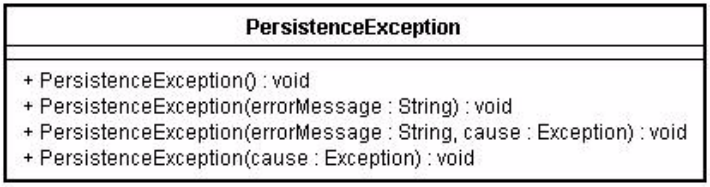
**PersistenceManager**

```
initialize(persistenceConfig:PersistenceConfig) : void // Performs
                            // initialization based on specified
                            // persistence configuration

persist(dataObjects:List) : void // Saves the specified list of data
                          // objects to the persistence store and memory

persist(dataObject:Object) : void // Saves the specified data object to
                                  // the persistence store and memory

remove(dataObjects:List) : void // Removes the specified list of data
                            // objects from memory

remove(dataObject:Object) : void  // Removes the specified data object
                                  // from memory

getData() : List            // Returns the cloded list of of persisted
                            // data objects in memory

wipeData() : void          // Wipes the currently persisted list of data
                         // objects from the persistence store and memory
```

## com.bea.mobile.rfid.persistence.PersistenceManagerImpl Class

The **PersistenceManagerImpl** class provides the concrete implementation of the **PersistenceManager** interface.

### com.bea.mobile.rfid.persistence.PersistenceException Class




The **PersistenceException** class represents an error during persistence processing. This class contains the following methods:

**PersistenceException**

```
PersistenceException() : void  // Default constructor

PersistenceException(errorMessage:String) : void // Constructor with
                              // error message parameter

PersistenceException(errorMessage:String, cause:Exception) : void
                              // Constructor with error message and cause
                              // exception parameters

PersistenceException(cause:Exception) : void  // Constructor with cause
                              // exception parameter
```

# EPCIS Capture API

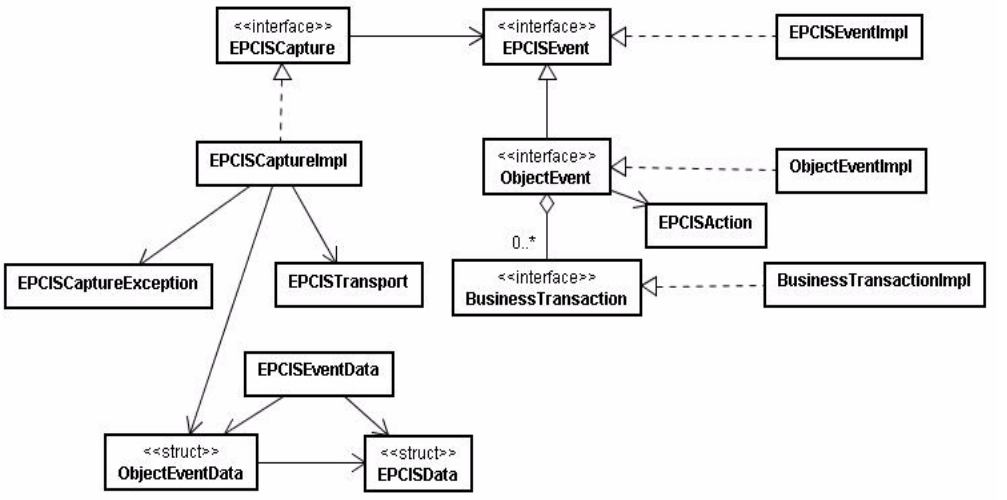
The EPCIS Capture API consists of the `com.bea.mobile.rfid.epcis` package.

## com.bea.mobile.rfid.epcis Package

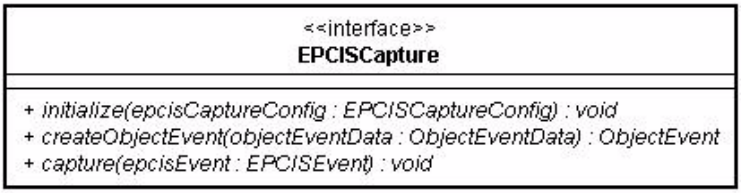
This package contains the classes and interfaces for creating, serializing/deserializing, and sending EPCIS events.

Figure 3-8 illustrates the relationships among the classes and interfaces

**Figure 3-8 EPCIS Capture API UML Diagram**




## com.bea.mobile.rfid.epcis.EPCISCapture Interface



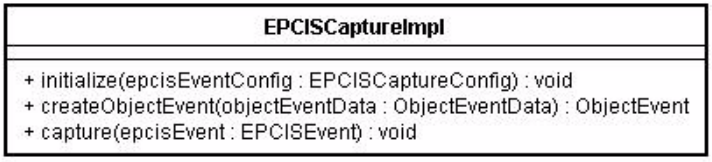

The **EPCISCapture** interface is utilized to create an EPCIS event object and to send it to configured destinations. This interface contains the following methods:

**EPCISCapture**

```
initialize(epcisEventConfig:EPCISEventConfig) : void // Performs
                              // initialization based on specified
                              // EPCIS event configuration
```

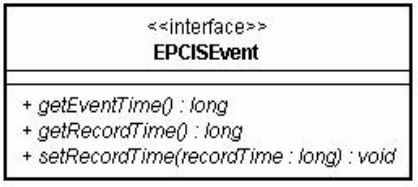
```
createObjectEvent(objectEventData:EPCISEventData.ObjectEventData) :
void
                              // Returns an ObjectEvent object based the
                              // specified object event data

capture(epcisEvent:EPCISEvent) : void // Sends the specified EPCIS event
                              // to configured destinations
```

## com.bea.mobile.rfid.epcis.EPCISCaptureImpl Class




The **EPCISCaptureImpl** class provides the concrete implementation of the **EPCISCapture** interface.

## com.bea.mobile.rfid.epcis.EPCISEvent Interface




The **EPCISEvent** interface represents the base interface for EPCIS events and contains the following methods:

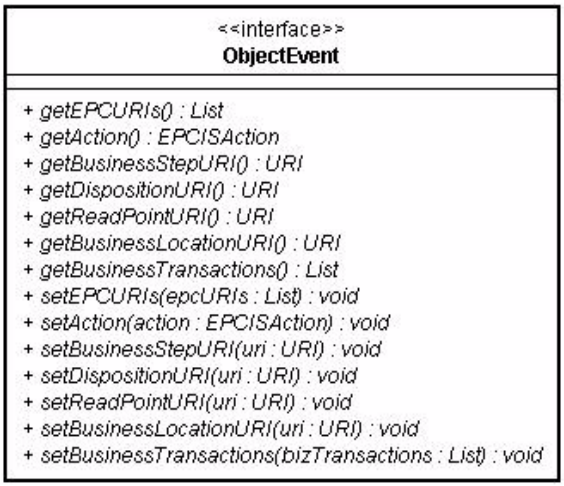
**EPCISEvent**

```
getEventTime() : long       // Returns the time when the EPCIS event
                            // occurred

getRecordTime() : long      // Returns the time when the EPCIS event
                            // was recorded

setRecordTime(recordTime:long) : void // Sets the specified time when
                            // the EPCIS event was recorded
```

## com.bea.mobile.rfid.epcis.EPCISEventImpl Class


The **EPCISEventImpl** class provides the concrete implementation of the **EPCISEvent** interface.

## com.bea.mobile.rfid.epcis.ObjectEvent Interface



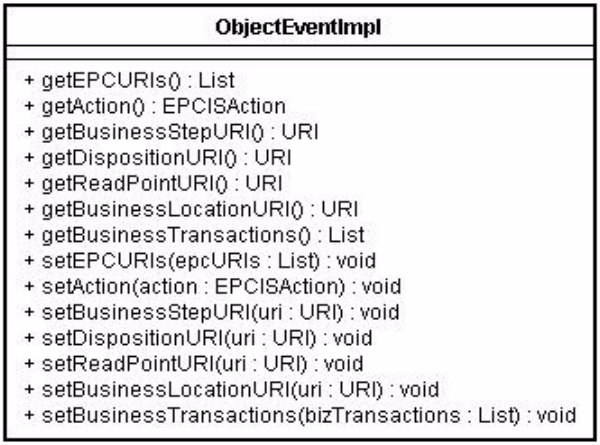

The **ObjectEvent** interface implements the **EPCISEvent** inteface and represents an EPCIS Object event. This interface contains the following methods:

**ObjectEvent**

```
getEPCURIs() : List          // Returns the list of EPC URI objects

getAction() : EPCISAction    // Returns the EPCIS action (ADD,
                             // OBSERVE, DELETE)

getBusinessStepURI() : URI   // Returns the business step URI

getDispositionURI() : URI    // Returns the disposition URI

getBusinessLocationURI() : URI // Returns the business location URI

getBusinessTransactions() : URI // Returns the list of
                             // BusinessTransaction objects

setEPCURIs(epcURIs:List) : void  // Sets the specified list of EPC URI
                             // objects
```

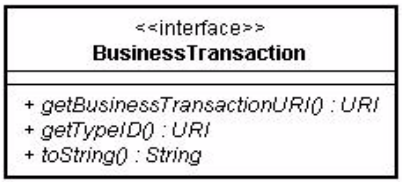
```
setAction(action:EPCISAction) : void // Sets the specified EPCIS action
                                     // (ADD, OBSERVE, DELETE)

setBusinessStepURI(uri:URI) : void // Sets the specified business step
                                   // URI

setDispositionURI(uri:URI) : void // Sets the specified disposition URI

setReadPointURI(uri:URI) : void // Sets the specified read point URI

setBusinessLocationURI(uri:URI) : void // Sets the specified business
                                       // location URI

setBusinessTransactions(bizTransactions:List) : void // Sets the
                                       // specified list of BusinessTransaction
                                       // objects
```

## com.bea.mobile.rfid.epcis.ObjectEventImpl Class




The **ObjectEventImpl** class provides the concrete implementation of the **ObjectEvent** interface.

## com.bea.mobile.rfid.epcis.BusinessTransaction Interface

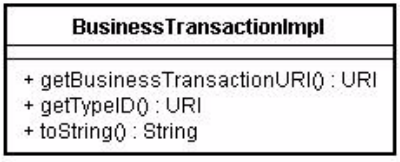



The **BusinessTransaction** interface represents the business transaction (order ID) of an EPCIS event. This interface contains the following methods:

**BusinessTransaction**

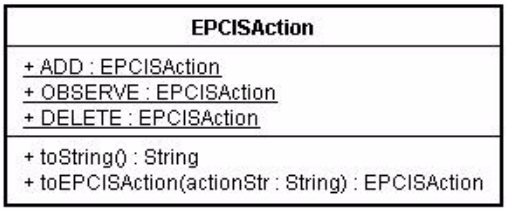
```
getBusinessTransactionURI() : URI // Returns the business transaction URI

getTypeID() : URI             // Returns the type ID

toString() : String           // Returns the string representation of the
                              // business transaction
```

## com.bea.mobile.rfid.epcis.BusinessTransactionImpl Class




The **BusinessTransactionImpl** class provides the concrete implementation of the **BusinessTransaction** interface.

## com.bea.mobile.rfid.epcis.EPCISAction Class

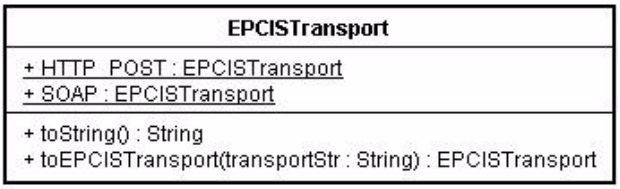



The **EPCISAction** class is a type-safe enumeration class and represents the action of an EPCIS event (ADD, OBSERVE, DELETE). This class contains the following attributes and methods:

**EPCISAction**

```
ADD : EPCISAction            // Attribute representing ADD action

OBSERVE : EPCISAction        // Attribute representing OBSERVE action

DELETE : EPCISAction         // Attribute representing DELETE action

toString() : String          // Returns string representation of EPCIS
                             // action

toEPCISAction(actionStr:String) : EPCISAction // Returns EPCISAction
                             // object for specified string
```

## com.bea.mobile.rfid.epcis.EPCISTransport Class

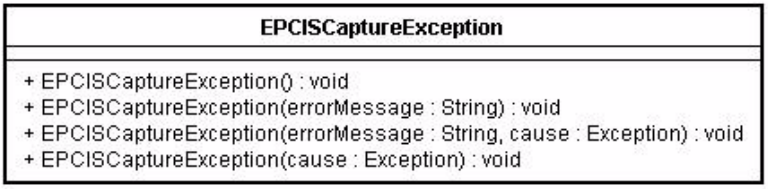



The **EPCISTransport** class is a type-safe enumeration class and represents the transport for sending an EPCIS event (HTTP_POST, SOAP). This class contains the following attributes and methods:

**EPCISTransport**

```
HTTP_POST : EPCISTransport  // Attribute representing HTTP POST transport

SOAP : EPCISTransport       // Attribute representing SOAP transport

toString() : String         // Returns string representation of EPCIS
                            // action

toEPCISTransport(transportStr:String) : EPCISTransport // Returns
                          // EPCISTransport object for specified string
```

## com.bea.mobile.rfid.epcis.EPCISCaptureException Class




The **EPCISCaptureException** class represents an error during EPCIS capture processing. This class contains the following methods:

**EPCISCaptureException**

```
EPCISCaptureException() : void // Default constructor

EPCISCaptureException(errorMessage:String) : void // Constructor with
                            // error message parameter

EPCISCaptureException(errorMessage:String, cause:Exception) : void
                            // Constructor with error message and cause
                            // exception parameters

EPCISCaptureException(cause:Exception) : void  // Constructor with cause
                            // exception parameter
```

# ALE EPC API

The ALE EPC API consists of the following packages:

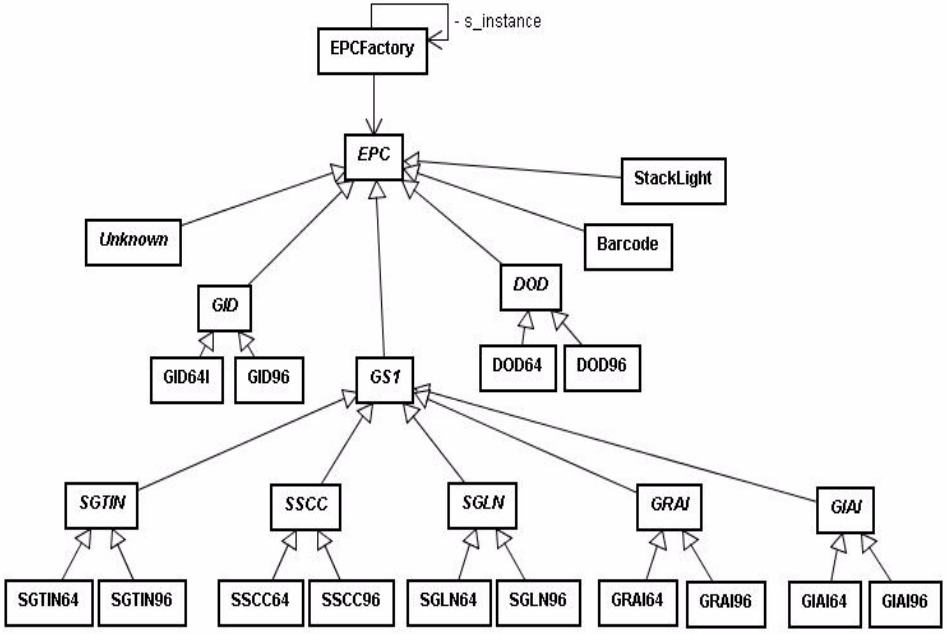
- `com.connecterra.ale.epc`
- `com.connecterra.ale.epchelpers`

## com.connecterra.ale.epc Package

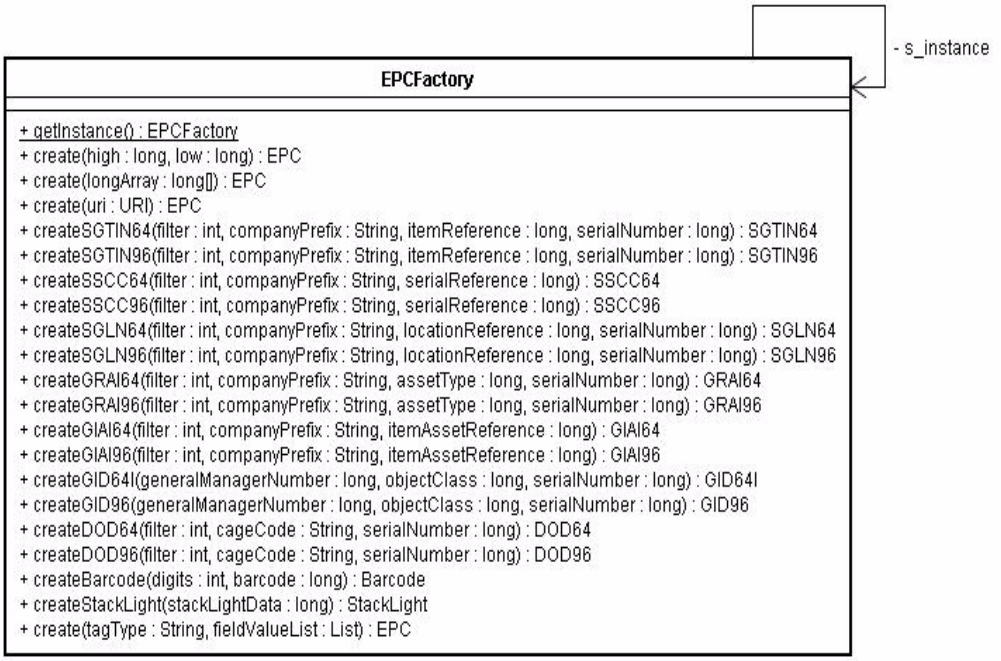
This package contains the classes and interfaces for creating and processing EPC objects.

Figure 3-9 illustrates the relationships among the classes and interfaces

**Figure 3-9 ALE EPC API UML Diagram**



## com.connecterra.ale.epc.EPCFactory Class




The **EPCFactory** class is a singleton class for creating various types of EPC objects. This class contains the following methods:

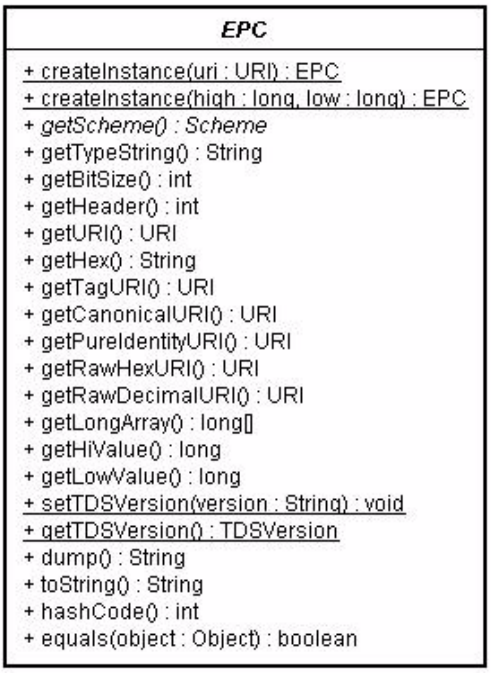
**EPCFactory**

```
getInstance() : EPCFactory  // Returns the singleton instance of the
                            // EPCFactory

create(high:long, low:long) : EPC // Returns an EPC object for the
                            // specified high and low long values

create(longArray: long[]) : EPC // Returns an EPC object for the
                            // specified array of long values

create(uri:URI) : EPC       // Returns an EPC object for the specified
                            // URI
```

```
createSGTIN64(filter:int, companyPrefix:String, itemReference:long,
serialNumber:long) : SGTIN64
                          // Returns an SGTIN64 object for the specified
                            // EPC parameters

createSGTIN96(filter:int, companyPrefix:String, itemReference:long,
serialNumber:long) : SGTIN96
                          // Returns an SGTIN96 object for the specified
                            // EPC parameters

createSSSCC64(filter:int, companyPrefix:String, serialNumber:long) :
SSCC64
                           // Returns an SSCC64 object for the specified
                            // EPC parameters

createSSSCC96(filter:int, companyPrefix:String, serialNumber:long) :
SSCC96
                           // Returns an SSCC96 object for the specified
                            // EPC parameters

createSGLN64(filter:int, companyPrefix:String, locationReference:long,
serialNumber:long) : SGLN64
                          // Returns an SGLN64 object for the specified
                            // EPC parameters

createSGLN96(filter:int, companyPrefix:String, locationReference:long,
serialNumber:long) : SGLN96
                          // Returns an SGLN96 object for the specified
                            // EPC parameters

createGRAI64(filter:int, companyPrefix:String, assetType:long,
serialNumber:long) : GRAI64
                          // Returns an GRAI64 object for the specified
                            // EPC parameters

createGRAI96(filter:int, companyPrefix:String, assetType:long,
serialNumber:long) : GRAI96
                          // Returns an GRAI96 object for the specified
                            // EPC parameters
```

```
createGIAI64(filter:int, companyPrefix:String, itemAssetReference:long)
: GIAI64
                          // Returns an GIAI64 object for the specified
                            // EPC parameters

createGIAI96(filter:int, companyPrefix:String, itemAssetReference:long)
: GIAI96
                          // Returns an GIAI96 object for the specified
                            // EPC parameters

createGID64I(generalManagerNumber:long, objectClass:long,
serialNumber:long) : GID64I
                          // Returns an GID64I object for the specified
                            // EPC parameters

createGID96(generalManagerNumber:long, objectClass:long,
serialNumber:long) : GID96
                           // Returns an GID96 object for the specified
                            // EPC parameters

createDOD64(filter:int, cageCode:String, serialNumber:long) : DOD64
                          // Returns an DOD64 object for the specified
                            // EPC parameters

createDOD96(filter:int, cageCode:String, serialNumber:long) : DOD96
                          // Returns an DOD96 object for the specified
                            // EPC parameters

createBarcode(digits:int, barcode:long) : Barcode
                          // Returns a Barcode object for the specified
                            // barcode parameters

createStackLight(stackLightData:long) : StackLight
                            // Returns a StackLight object for the
                            // specified stacklight parameters

create(tagType:String, fieldValueList:List) : EPC
                            // Returns an EPC object for the specified
                            // tag type and list of field values
```

## com.connecterra.ale.epc.EPC Class




The **EPC** class is a base abstract class for all EPC derivations. This class contains the following methods:

**EPC**

```
createInstance(uri:URI) : EPC // Returns an EPC object for the specified
                              // URI

createInstance(high:long, low:long) : EPC // Returns an EPC object for
                              // the specified high and low long values

getScheme() : Scheme          // Returns the Scheme object for this EPC

getTypeString() : String      // Returns the type string of this EPC

getBitSize() : int            // Returns the bit size of this EPC
```

```
getHeader() : int            // Returns the bit size of this EPC

getURI() : URI               // Returns the URI of this EPC

getHex() : String            // Returns the hex string of this EPC

getTagURI() : URI            // Returns the tag URI of this EPC

getCanonicalURI() : URI      // Returns the canonical URI of this EPC

getPureIdentityURI() : URI  // Returns the pure identity URI of this EPC

getRawHexURI() : URI         // Returns the raw hex URI of this EPC

getRawDecimalURI() : URI     // Returns the raw decimal URI of this EPC

getLongArray() : long[]      // Returns the long array for this EPC

getHighValue() : long        // Returns the high long value of this EPC

getLowValue() : long         // Returns the low long value of this EPC

setTDSVersion(version:String) : void // Sets the specified TDS (Tag Data
                             // Standard) version

getTDSVersion() : TDSVersion // Returns the TDS version

dump() : String              // Returns a dump of this EPC object

toString() : String        // Returns a string representation of this EPC

hashCode() : int             // Returns the hashcode for this EPC

equals(object:Object) : boolean // Indicates whether or not the specified
                             // object equals this EPC
```

### EPC Derivation Classes

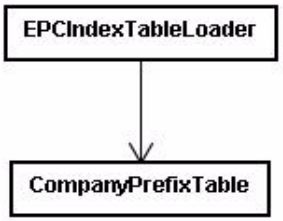
For details on the EPC derivation classes, see the API references listed in "Related Documentation" on page 1-2.

## com.connecterra.ale.epchelpers Package

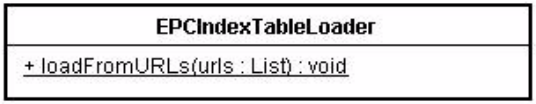
This package contains the helper classes used with EPC processing.

Figure 3-10 illustrates the relationships among the classes and interfaces

**Figure 3-10 EPC Helper API UML Diagram**




## com.connecterra.ale.epchelpers.EPCIndexTableLoader Class

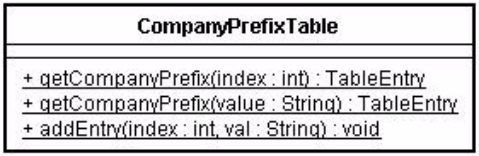



The **EPCIndexTableLoader** class provides the ability to load EPC index entries into the company prefix table. This class contains the following methods:

**EPCIndexTableLoader**

```
loadFromURLs(urls:List)     // Loads EPC index entries from the specified
                             // list of URLs
```

## com.connecterra.ale.epchelpers.CompanyPrefixTable Class



The **CompanyPrefixTable** class provides the ability to get and add company prefix table entries from the company prefix table. This class contains the following methods:

**CompanyPrefixTable**

```
getCompanyPrefix(index:int) : TableEntry // Returns the company prefix
                              // table entry for the specified index

getCompanyPrefix(value:String) : TableEntry // Returns the company
                             // prefix table entry for the specified value

addEntry(index:int, val:String) : void // Adds the specified value at the
                              // specified index
```

CHAPTER 4

# Sample C# Application

The following sections describe how to use the sample C# application provided with the SDK. The sample application is free for you to use and modify for your own purposes. You can use this application as a starting point for developing your own applications.



## Overview of Sample C# Applications

The following sample is provided with BEA WebLogic RFID Edge Mobile SDK:

- `BEAMobileTagRead` — Shows how to use the BEA WebLogic RFID Edge Mobile SDK API for tag reading with EPCIS capture (send an EPCIS ObjectEvent for one or more EPC reads).

# Setting Up Your Development Environment

To compile and run the sample application, you need to install the Microsoft Visual Studio .NET 2003 or Microsoft Visual Studio 2005, and the BEA WebLogic RFID Edge Mobile SDK software.

For detailed information about system requirements, prerequisite software, and how to install BEA WebLogic RFID Edge Mobile SDK, see *Installing BEA WebLogic RFID Edge Mobile SDK*.

# Compiling and Running the Samples

Use Microsoft Visual Studio to open the sample solution and associated projects to build the sample application.

For example, for the BEA Mobile Tag Read sample application, perform the following:

1. Use the Windows File Explorer to go to the following directory:

   ```
   BEA_MOBILE_SDK_HOME\samples\BEAMobileTagRead
   ```

2. Double-click on the `BEAMobileTagRead`.sln solution file to bring up the BEA Mobile Tag Read solution, associated project, and source code files in Microsoft Visual Studio.
3. Make the desired changes to the source code.
4. In the main menu of Microsoft Visual Studio, select **Build** -> **Rebuild Solution** to compile the BEA Mobile Tag Read sample application.

For details on running the BEAMobileTagRead sample application, see *Configuring and Running the BEA Mobile Tag Read Sample Application* in *Installing BEA WebLogic RFID Edge Mobile SDK*.

For a tutorial walk through of a sample, see:

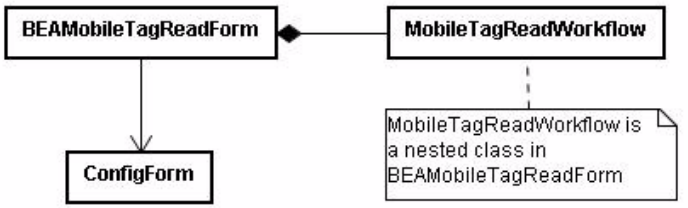
"BEAMobileTagRead: UML Diagrams" on page 4-3

# BEAMobileTagRead: UML Diagrams

## Class Diagrams

Figure 4-1 illustrates the relationships among the classes of the BEA Mobile Tag Read sample application.

**Figure 4-1 BEA Mobile Tag Read Sample Application UML Diagram**



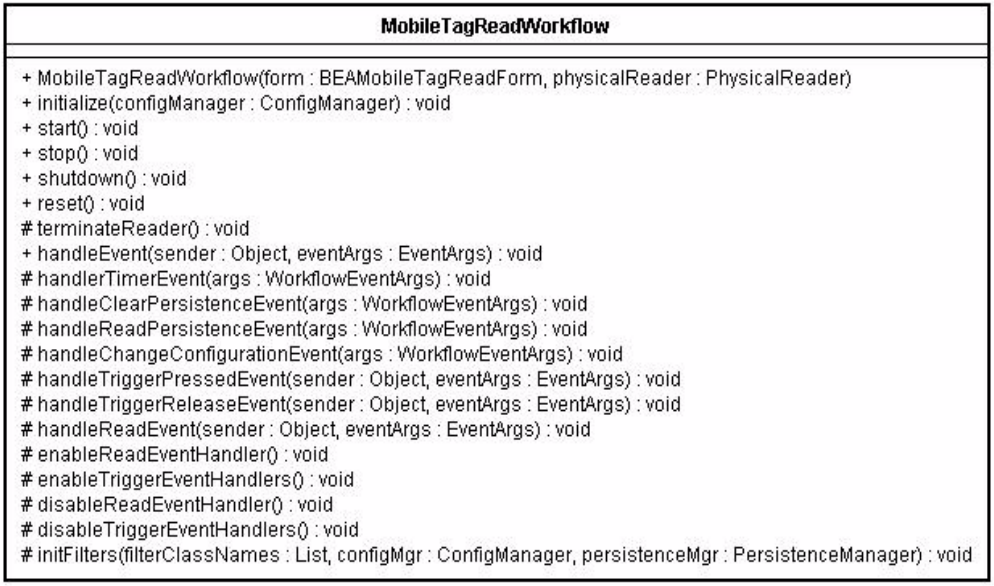

### com.bea.mobile.rfid.samples.tagread.BEAMobileTagReadForm Class

The **BEAMobileTagReadForm** class is the main application class supporting the user interface and application logic. This class contains the following methods:

**BEAMobileTagReadForm**

```
BEAMobileTagReadForm()      // Default constructor

Main() : void               // Main program method

BEAMobileTagReadForm_Load(sender:Object, eventArgs:EventArgs) : void
                           // Processes BEAMobileTagReadForm load event

ReaderTimer_Tick(sender:Object, eventArgs:EventArgs) : void
                           // Processes timer tick event

OnClosing(CancelEventArgs:EventArgs) : void // Processes cancel event

Dispose(disposing:boolean) : void // Frees up resources

initApp() : void            // Initializes the application

ExitMenu_Click(sender:Object, eventArgs:EventArgs) : void
                           // Processes Exit menu event

AboutMenu_Click(sender:Object, eventArgs:EventArgs) : void
                           // Processes About menu event

SendPersistedMenu_Click(sender:Object, eventArgs:EventArgs) : void
                           // Processes Send Persisted menu event

ClearDisplayedMenu_Click(sender:Object, eventArgs:EventArgs) : void
                           // Processes Clear Displayed menu event

ClearPersistedMenu_Click(sender:Object, eventArgs:EventArgs) : void
                           // Processes Clear Persisted menu event

EditConfigMenu_Click(sender:Object, eventArgs:EventArgs) : void
                           // Processes Edit Config menu event

ReadImmediatelyMenu_Click(sender:Object, eventArgs:EventArgs) : void
                           // Processes Read Immediately menu event

DisplayMenuBox(text:String, caption:String) : void
                           // Displays menu box with specified text
```

## com.bea.mobile.rfid.samples.tagread.BEAMobileTagReadForm.MobileTagReadWorkflow Nested Class



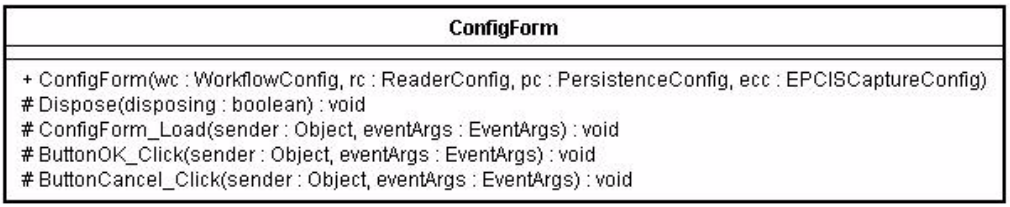

The **BEAMobileTagReadForm.MobileTagReadWorkflow** nested class provides the concrete implementation of the **AbstractWorkflow** class and encapsulates the business logic for the tag read workflow. This class contains the following methods:

**BEAMobileTagReadForm.MobileTagReadWorkflow**

```
MobileTagReadWorkflow(form:BEAMobileTagReadForm,
physicalReader:PhysicalReader) // Constructor with form and physical
                               // reader parameters

initialize(configManager:ConfigManager) : void // Initializes the
                              // workflow based on specified ConfigManager

start() : void                // Starts the workflow
```

```
stop() : void                // Stops the workflow

shutdown() : void            // Shuts down the workflow

reset() : void               // Resets the workflow

terminateReader() : void     // Terminates the reader

handleEvent(sender:Object, eventArgs:EventArgs) : void
                             // Processes the event

handleTimerEvent(args:WorkflowEventArgs) : void
                             // Processes the timer event

handleClearPersistenceEvent(args:WorkflowEventArgs) : void
                             // Processes the clear persistence event

handleReadPersistenceEvent(args:WorkflowEventArgs) : void
                             // Processes the read persistence event

handleChangeConfigurationEvent(args:WorkflowEventArgs) : void
                             // Processes the change configuration event

handleTriggerPressedEvent(sender:Object, eventArgs:EventArgs) : void
                             // Processes the trigger pressed event

handleTriggerReleasedEvent(sender:Object, eventArgs:EventArgs) : void
                             // Processes the trigger released event

handleReadEvent(sender:Object, eventArgs:EventArgs) : void
                             // Processes the read event

enableReadEventHandler() : void // Enables the read event handler

enableTriggerEventHandlers() : void // Enables the trigger event handlers

disableReadEventHandler() : void // Disables the read event handler

disableTriggerEventHandlers() : void // Disables the trigger event
                             // handlers

initFilters(filterClassNames:List, configMgr:ConfigManager,
persistenceMgr:PersistenceManager) : void // Initializes the configured
                             // filters
```

### com.bea.mobile.rfid.samples.tagread.ConfigForm Class




The **ConfigForm** class provides support for the Edit Config dialog. This class contains the following methods:

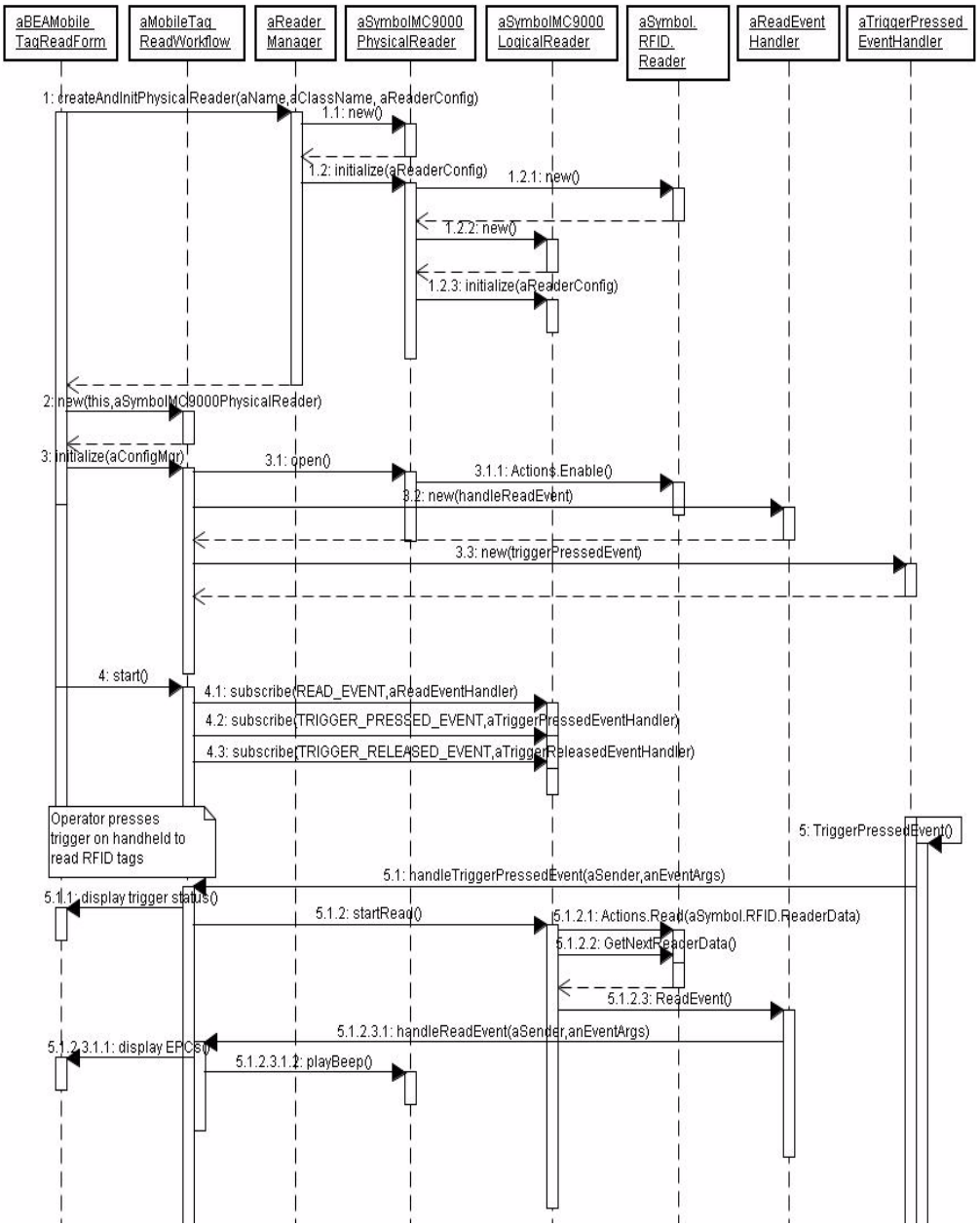
**ConfigForm**

```
ConfigForm(wc:WorkflowConfig, rc:ReaderConfig, pc:PersistenceConfig,
ecc:EPCISCaptureConfig)     // Constructor with config parameters

Dispose(disposing:boolean) : void // Cleans up any resources being used

ConfigForm_Load(sender:Object, eventArgs:EventArgs) : void
                            // Processes ConfigForm load event

ButtonOK_Click(sender:Object, eventArgs:EventArgs) : void
                            // Processes OK click event

ButtonCancel_Click(sender:Object, eventArgs:EventArgs) : void
                            // Processes Cancel click event
```

## Sequence Diagram

Figure 4-2 illustrates the messages involved from workflow initialization to tag read processing in the BEA Mobile Tag Read sample application.

**Figure 4-2 BEA Mobile Tag Read Sample Application - Sequence Diagram**



In the above sequence diagram, the BEA Mobile Tag Read sample application creates the device specific physical reader and the applicable workflow instance. The application calls *initialize()* on the workflow instance to perform initialization. The application calls *start()* on the workflow instance to enable the mobile device and setup the appropriate event handlers. The application processes events (read events, trigger events) and displays the desired data and status in the user interface.

# BEAMobileTagRead: How to Use the BEA WebLogic RFID Edge Mobile SDK APIs to Read Tags and Send EPCIS Events

The `BEAMobileTagRead` sample application shows how to do the following:

- Use the **Tag Reading API** to create a reader, listen for trigger press and release events, obtain tag read data, and close a reader.
- Use the **Workflow API** to manage the lifecycle (creation, start, stop) of the application workflow.
- Use the **Filter API** to filter tag reads based on configured criteria.
- Use the **Persistence API** to persist EPCIS capture events to a configured persistence store.
- Use the **Configuration API** to process static configuration and dynamic configuration for specific sets of configuration data.
- Use the **ALE EPC API** to perform EPC processing.
- Use the **EPCIS Capture API** to construct EPCIS event messages (ObjectEvent) and send the EPCIS event messages to configured destinations.

## Tag Reading API Usage

The `MobileTagReadWorkflow` in the `BEAMobileTagRead` sample application uses read event and trigger event handlers to start, process, and stop tag reads. In `MobileTagReadWorkflow.initialize()`, the following C# lines of code instantiate the required event handlers:

```
// Create handler for read event notifications

m_readEventHandler = new EventHandler(handleReadEvent);
```

```
// Create handler for trigger pressed event notifications

m_triggerPressedEventHandler = new
EventHandler(handleTriggerPressedEvent);

// Create handler for trigger released event notifications

m_triggerReleasedEventHandler = new
EventHandler(handleTriggerReleasedEvent);
```

`MobileTagReadWorkflow.start()` enables the event handlers via `LogicalReader.subscribe()` for the specific event type and event handler. `MobileTagReadWorkflow.stop()` disables the event handlers via `LogicalReader.unsubscribe()` for the specific event type and event handler.

When the trigger is pressed on the mobile device, the `MobileTagReadWorkflow.handleTriggerPressedEvent()` method is invoked, which then starts tag reads via `MobileTagReadWorkflow.startRead()`. The internal implementation of `MobileTagReadWorkflow.startRead()` publishes a `ReadEvent` event.

The `MobileTagReadWorkflow.handleReadEvent()` method is invoked to handle the `ReadEvent`. The implementation of `MobileTagReadWorkflow.handleReadEvent()` obtains the read buffer via the `LogicalReader.ReadBuffer` property and displays the EPC-formatted tag reads in the main UI. A beep is played for each tag read via `PhysicalReader.playBeep()`.

When the trigger is released on the mobile device, the `MobileTagReadWorkflow.handleTriggerReleasedEvent()` method is invoked, which then disables the tag reads.

# Workflow API Usage

In `BEAMobileTagReadForm.initApp()`, the `MobileTagReadWorkflow` is instantiated, initialized and started by the following C# lines of code:

```
// Create workflow

m_workflow = new MobileTagReadWorkflow(this, physicalReader);

// Initialize workflow

m_workflow.initialize(ConfigManagerImpl.Instance);

// Start workflow

m_workflow.start();
```

`Events are handled in BEAMobileTagReadForm` through delegation to the workflow. For example, when the user selects "Send Persisted Events" on the main UI, the `BEAMobileTagReadForm.SendPersistedMenu_Click()` method is invoked, which then processes the event via `Workflow.handleEvent()`.

The `BEAMobileTagReadForm.OnClosing()` method is called when the application is closing. The implementation of `BEAMobileTagReadForm.OnClosing()` calls `Workflow.stop()` and `Workflow.shutdown()` to stop and shutdown the workflow, respectively.

# Filter API Usage

In `MobileTagReadWorkflow.initFilters()`, each configured filter is created, initialized, and added to the filter list by the following C# lines of code:

```
foreach (string filterClassName in filterClassNames)
{
    // Create filter
    Filter filter = WorkflowConfig.createFilterClassInstance(
filterClassName);
    // Initialize filter
       filter.initialize(configManager, persistenceManager, null);
       m_filters.Add(filter);
    };
```

In `MobileTagReadWorkflow.applyFilters()`, the configured filters are applied to the specified `ObjectEvent` by the following lines:

```
IList objectEvents = new ArrayList();
objectEvents.Add(objectEvent);
foreach (Filter filter in m_filters)
{
    objectEvents = filter.execute(objectEvents);
}
```

## Persistence API Usage

In `MobileTagReadWorkflow.processReadData()`, each `ObjectEvent` is persisted to the configured persistence store by using the `PersistenceManager` class as follows:

```
m_persistenceManager.persist(objectEvent);
```

To process a "Send Persisted Reads" action from the main UI, the `MobileTagReadWorkflow.handleReadPersistenceEvent()` method is invoked to read persisted EPCIS events into memory and send them to configured EPCIS capture destinations via the following C# lines of code:

```
IList epcisEvents = m_persistenceManager.Data;

foreach (EPCISEvent epcisEvent in epcisEvents)

{

    try

    {

        // Send EPCIS capture event

        m_epcisCapture.capture(epcisEvent);

    }

    catch(EPCISCaptureException ece)

    {

        ...

    }

    };
```

## Configuration API Usage

In `MobileTagReadWorkflow.initialize()`, workflow-specific configuration data is obtained by using the `ConfigManagerImpl` class as illustrated in the following C# lines of code:

```
// Get workflow config data

IList workflowNames =
configManagerImpl.WorkflowConfig.getWorkflowNames();
```

```
m_workflowConfigData =
configManagerImpl.WorkflowConfig.getWorkflowConfigData((string)workflow
Names[0]);
```

## ALE EPC API Usage

In `MobileTagReadWorkflow.processReadData()`, the EPC tag URI string is obtained by using the `EPC` class as follows in C#:

```
tagReport.EPC.TagURI.ToString();
```

## EPCIS Capture API Usage

In `MobileTagReadWorkflow.createObjectEvent()`, an `ObjectEvent` instance is created by the following C# lines of code:

```
EPCISEventData.ObjectEventData objectEventData = new
EPCISEventData.ObjectEventData();

// Set event time

objectEventData.m_eventTime = ((TagReport)tagReports[0]).Timestamp;

// Create list of EPC URIs

IList epcURIs = new ArrayList();

for (int i=0; i<tagReports.Count; i++)

{

    TagReport tagReport = (TagReport)tagReports[i];

    epcURIs.Add(tagReport.EPC.TagURI);

}

objectEventData.m_epcURIs = epcURIs;

// Set EPCIS data

EPCISEventData.EPCISData epcisData = new EPCISEventData.EPCISData();

epcisData.m_action = m_epcisCaptureConfig.EPCISAction;

epcisData.m_readPointURI = m_epcisCaptureConfig.CurrentReadPointURI;
```

```
epcisData.m_businessLocationURI =
m_epcisCaptureConfig.CurrentBusinessLocationURI;
epcisData.m_businessStepURI = m_epcisCaptureConfig.CurrentBusinessStepURI;
epcisData.m_dispositionURI = m_epcisCaptureConfig.CurrentDispositionURI;
// Set business transaction data
IList businessTransactions = null;
if (m_epcisCaptureConfig.CurrentBusinessTransactions != null)
{
    businessTransactions = new ArrayList();
    businessTransactions.Add(m_epcisCaptureConfig.
CurrentBusinessTrasnaction);
}
epcisData.m_businessTransactions = businessTransactions;
objectEventData.m_epcisData = epcisData;
return m_epcisCapture.createObjectEvent(objectEventData);
```

In `MobileTagReadWorkflow.processReadData()`, an EPCIS event is sent to configured destinations using the `EPCISCapture` interface as follows in C#:

```
try
{
    // Send EPCIS capture event
    m_epcisCapture.capture(epcisEvent);
}
catch(EPCISCaptureException ece)
{
    ...
   }
```

# Index