RFIDMania - Extensible and Adaptable RFID Middleware and Specifications

By

Eyad Aboulouz

A thesis submitted to

the Faculty of Graduate Studies and Research

in partial fulfillment of

the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

September, 2009

Your file Votre référence
ISBN: 978-0-494-60239-3
Our file Notre référence
ISBN: 978-0-494-60239-3

Abstract

Radio Frequency Identification (RFID) technology is an emerging technology that allows objects to be electronically tagged and identified wirelessly. In recent years, many organizations have started to show interest in porting this technology to their existing business processes. With the increase in popularity of the technology, many vendor-specific RFID readers were manufactured and sold to interested organizations.

Deployment of such RFID readers to existing business processes became difficult as software developers needed to understand each vendor-specific RFID reader on its own, due to lack of standardization among RFID readers of different makes, before they were able to work with it and implement what is needed. In recognizing this, a scalable and adaptable middleware, called RFIDMania, was designed to allow software developers to interact with any RFID reader of their choice without having to know the specifics of the reader. RFIDMania provides an environment to process all data received from RFID tags, translates the data, decodes it, filters it and routes it to the application in the form of an event. RFIDMania is able to offer software developers an environment to deliver portable generic code that is not tightly coupled with hardware specific commands.

Acknowledgments

I would like to thank Dr. Dwight Deugo, of the School of Computer Science at Carleton University, for patiently supervising this thesis project and providing assistance and resources in all matters of the project.

I would also like to thank my mother for helping me stay focused during my graduate school years and for all the love and support.

Finally, I would like to thank my brother Emad Aboulouz for constantly encouraging and supporting me throughout my graduate years and for proof-reading this thesis.

Contents

List of Tables

List of Figures

Listings

Acronyms

MCF	Manufacturer Configuration File	Chapter 3, Section 3.4
PCF	Protocol Configuration File	Chapter 3, Section 3.2
RCF	Reader Configuration File	Chapter 3, Section 3.6
RFCF	Radio-Frequency Configuration File	Chapter 3, Section 3.3
TCF	Transponder Configuration File	Chapter 3, Section 3.5

1 Introduction

Radio Frequency Identification, RFID, is said to be the "technology of the future". It is already in use in various areas ranging from inventory management to the automated collection of road-tolls. The technology is known for its ability to auto-identify a tagged object using radio waves. Benefits of using the RFID technology includes the reduction of human errors, reduction of labor costs and an increase in accuracy and visibility.

The RFID technology is not new. It is thought that its roots can be traced back to World War II. However, in recent years, the technology has caught a lot of attention and current trends indicate that the RFID market will grow fast in the coming years, with "1.02 billion tags sold in 2006, the value of the market is expected to increase by a factor of six between 2007 and 2017" [3]. Tags are likely to replace Universal Tag Codes, UPCs, for many reasons. For example, barcodes are only capable of identifying the type of a bar-coded product whereas RFID tags can carry a lot more data. Bar-codes also require a line-of-sight for a successful read whereas an RFID reader just has to be within a certain range to pickup the transmitted radio waves of a tag. The ability to identify tagged objects that are not necessary in the line-of-sight of the reader has given the RFID technology a great edge over other identification technologies. This along with an increase in scanning accuracy and overall efficiency has made RFID a favorite among many supply-chains.

While RFID is mostly being used in the retail industry, it has the full potential of being used by just about any industry for reasons such as the ones mentioned previously. Usage

of RFID technology in the retail industry includes the ability to do an inventory count without any physical involvement and the automatic detection of expired items on shelves. In the medical field, RFID technology is used to track when a tagged medicine bottle enters or leaves a patient's room - thus ensuring that the patient is not accidentally given their medicine twice - and is used to track when a medicine cabinet runs out of a certain medicine and automatically puts through an order for more of that medicine.

A typical RFID system setup consists of a reader, which is a device that is used to interrogate an RFID tagged object, and one or more tagged objects. Tags do not have a specific look as they come in many different shapes, sizes and materials. This flexibility allows manufacturers to embed RFID tags into their products with ease. Tags also come in two types: passive and active. Active tags incorporate their own power source, which allows them to transmit radio frequency signals, while passive tags are only activated if they are within the response range of an RFID reader and are powered up from the radio-wave field that is emitted by the RFID reader. Lastly, tags can be restricted to data read-only or support both read and write. Figure 1.1 illustrates the main components of a typical RFID system [14].

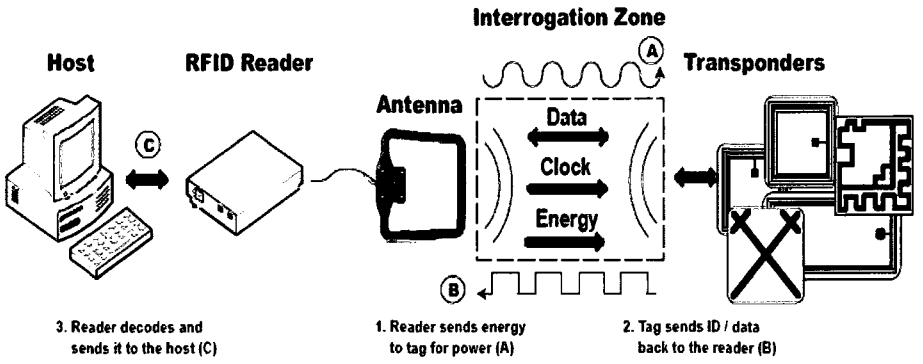

Figure 1.1: Components of a typical RFID system

RFID readers also come in a variety of shapes and sizes and operate on either a single or on multiple frequencies. In general, the more frequencies an RFID reader supports, the more tags it is compatible with. RFID readers also vary in hardware components, manufacturers and their ability to support other functionality such as anti-collision handling and accurate tag detection. An RFID reader is comprised of a box with a number of internal or external antennas whose purpose is to detect tags and read from or write to tags within their vicinity. RFID readers are often attached to one or more port interfaces such as *Serial (RS232)*, *USB*, *TCP* and *UDP*, thus allowing communication with the RFID reader directly from a computer.

The operations of a typical RFID reader are defined by the Specification for RFID Air Interface draft [24]. These operations include the *inventory* operation - where the reader returns a listing of all tags within its vicinity, the *read* operation - where data is read from a specific tag, the *write* operation - where data is written to a specific tag, the *kill* operation - where a tag is disabled until activated again and the *lock* operation - where a tag is locked with a password thus requiring all future reads and writes to provide a password. An RFID reader

can also implement other non-standard operations while following the previously mentioned specification. An example of a non-standard operation is the 'disable-anti-collision' operation, which disables the anti-collision capabilities of a reader.

There are regulations imposed on the frequencies that can be omitted by RFID readers and active RFID tags. The frequency of an RFID system often defines the read range and detection range of an RFID tag by an RFID reader. For instance, low frequency bands ranging from 100 to 500 kHz can cover a read area of up to 6 inches, while Ultra-High frequency bands ranging from 850 - 950 MHz can cover a read area of up to 20 feet. RFID frequency bands are often categorized in four distinct groups depending on the frequency omitted by the reader: Low, High, Ultra-High and Microwave. For each of these categories, there are one or more standard RFID air protocols that an RFID reader needs to understand in order to communicate with a tag that implements them. For instance, an RFID tag that implement the I-Code [42] protocol can only be detected properly by a compatible RFID reader that supports the I-Code protocol.

Aside from all the positive benefits that the RFID brings to the world, it has come under serious criticism in recent years for its inability to protect tags from being detected and read by unauthorized readers. Tag counterfeiting, cloning, eavesdropping by other readers and "replaying" - where a valid data transmission is fraudulently intercepted and retransmitted, are some other major security breaches in the technology. In spite of these concerns, the use of the technology continues to grow with more and more adoption in retail stores around the globe.

In recent years, there has been a lot of research in the development of secure, efficient and cost-effective RFID tags and readers but very little research has been done for the standardization of RFID readers. There have been a few attempts to create an application middleware to address the standardization problem, but none seem to be scalable enough to address the growing gap among RFID readers of different makes. In recent years, a non-profit organization called EPCglobal [25], has compiled a set of specifications drafts for standardizing the communication between RFID readers and RFID tags. This organization was founded by the Uniform Code Council and EAN International whom are responsible for maintaining bar-code standards in North America and internationally. A more elaborate description of EPCglobal's initiatives will be discussed in Chapter 2, section 2.1.

The rest of this chapter will introduce the problem of RFID standardization from a software engineering perspective and address the motivation for creating a solution to this problem. The goals for creating a feasible solution will then be highlighted along with the objective of how those goals will be met. Lastly, this chapter will provide an outline of all subsequent chapters in this thesis.

1.1 Problem

RFID systems have a big future ahead but the lack of standardization will likely result in reader-specific applications as opposed to having an application that works across all RFID readers available regardless of their hardware and architecture differences. From a software engineering perspective, developers should in theory spend their time fulfilling business requirements without having to understand the details of a specific RFID hardware. Without

a middleware solution, such as the one being explored in this thesis, developers' productivity is likely to decrease and their code will not likely be suitable to read and maintain in the future.

The problem arises from the lack of an access management model that developers can code against without having to write code that is tightly coupled to a specific RFID reader and for a scalable middleware that is able to communicate with new RFID readers without any changes to its core. Existing middleware solutions, which will be explored in chapter 2, do not fulfill the requirements of ease of code portability to work with other RFID readers without having to do changes to the core of the application and/or middleware or registering and de-registering of RFID device drivers. In addition, existing middleware solutions do not offer a way to define reader commands in a configuration file and therefore extensibility is not possible without writing additional code.

1.2 Motivation

Developers who write code to interact with an RFID device will quickly realize that the majority of their code is focused on how to communicate with an RFID device and less to do with tackling their business requirements. This realization is what motivated the creation of an RFID reader specification and schema to cut-down substantially on the amount of time that developers would otherwise spend learning about the details of a reader before they are able to work with it. In addition, the lack of a non-commercialized solution that is optimized for extensibility was another factor. Similarly, cutting down on development

costs and quicker delivery of RFID applications were other factors for creating such a standardization. The core and architecture of the standardization is geared around helping the developer create portable code that is not tightly coupled with reader-specific commands from its initial phase.

1.3 Goals

In order to address the problems outlined in the *Problem* section, a standardization in the form of a middleware and specification needs to be implemented that provides the following properties:

a) provide a schema specification that allows users to describe RFID protocols in a configuration file.

b) provide a schema specification that allows users to describe RFID radio-frequencies in a configuration file.

c) provide a schema specification that allows users to describe RFID manufacturer in a configuration files.

d) provide a schema specification that allows users to describe RFID tags in a configuration file.

e) provide a schema specification that allows users to describe RFID readers in a configuration file.

f) provide users with a set of APIs that they can use to communicate with their RFID reader through the middleware.

g) encapsulate all communication details with an RFID reader from the application layer.

h) provide implementation of basic data transfer protocols such as *Serial*, *USB*, *TCP* and *UDP*.

i) provide data processing capabilities and present the application with application events instead of raw data.

To elaborate, the specification needs to provide a generic interface - API - that can be used to communicate with any RFID reader regardless of its hardware. It must provide a set of schemas that developers can follow to create configuration files for their RFID protocols, radio-frequencies, manufacturers, tags and readers. It must allow the user to specify all RFID-reader-specific settings in its own reader configuration file, which is used by the middleware to initiate and carry on future communications with the RFID reader without the user's intervention. The middleware must be able to provide complete encapsulation of all communication details with the RFID reader thus allowing the software developer to deliver faster, more reliable code while focusing primarily on the business requirements and not the communication details. The middleware must also provide data processing capabilities to decode and encode data as necessary and must deliver all captured data and reader generated errors in the form of application events and not in the raw form in which it is received from the reader.

As part of good software engineering design and development practices, the standardization should also provide means to log all events and errors that are captured by the middleware

system for future analysis. The standardization should also allow for future data transfer protocols to be added easily and provides means of filtering unwanted data.

The above goals will be addressed in more detail in chapter 3.

1.4 Objective

The objective of this thesis is to develop a standardization as outlined in the *Goals* section and then validate it against two different readers with a reference implementation.

1.5 Contributions

This thesis provides a solution for heterogeneous RFID readers by providing the following contributions:

a) A schema for describing RFID protocols along with an example of a Protocol Configuration File, as provided in Appendices A and B respectively.

b) A schema for describing RFID radio-frequencies along with an example of a Radio-Frequency Configuration File, as provided in Appendices C and D respectively.

c) A schema for describing RFID manufacturers along with an example of a Manufacturer Configuration File, as provided in Appendices E and F respectively.

d) A schema for describing RFID tags along with an example of a Transponder Configuration File, as provided in Appendices G and H respectively.

e) A schema for describing RFID readers along with three example of a Reader Configuration File for three different readers: FEIG MR200, PhidgetRFID 1023 and Intermec IF61, as provided in Appendices I, J and K respectively.

f) A middleware - called RFIDMania - that works with the above specifications and schema, as described in Chapter 3.

g) An API through an interface - ISimpleRFID - that applications can tap into to interact with RFIDMania, as described in Section 3.8.

h) Four coded examples for how to interact with RFIDMania using the FEIG MR200 and PhidgetRFID 1023 readers, as provided in Chapter 4.

These contributions will be addressed in Chapter 3 and Chapter 4.

1.6 Outline

Chapter 2, called *Background*, renders an overview of how communication with RFID readers are currently being handled from a software engineering perspective and the issues surrounding it. The chapter will also address why a new method was researched and what it can provide in contrast to existing solutions.

Chapter 3, called *Approach*, discusses a reference implementation of the RFIDMania middleware and standardization in the form of a schema specification to address the problems outlined in section 1.1. UML class diagrams, sequence diagrams and API calls are provided in this chapter.

Chapter 4, called *Results*, validates the outcome of the results against the problems and solution provided in earlier chapters.

Chapter 5, called *Conclusion*, provides a set of conclusions by reviewing goals and contributions made by the *Approach* and *Results* chapters. This chapter will also provide a few ideas for future work.

2 Background

This chapter reviews a number of existing RFID middleware solutions that attempt to provide solutions for working with heterogeneous RFID readers and the translation of raw data into application events. It is important to note that the term *'middleware'* means different things to different vendors. It can be loosely interpreted as a software layer that aims to fill the gap between RFID readers that collect data from tags and enterprise information systems. However, some vendors and analysts broaden the term to include a software that is also capable of filtering tag data and managing of RFID readers. Regardless of what an RFID middleware is meant to do, all middleware solutions share one thing in common in that they are all able to help organizations implement RFID devices and get meaningful data out of a tag easily [16]. Figure 2.1 depicts how an RFID middleware fits into an RFID system.

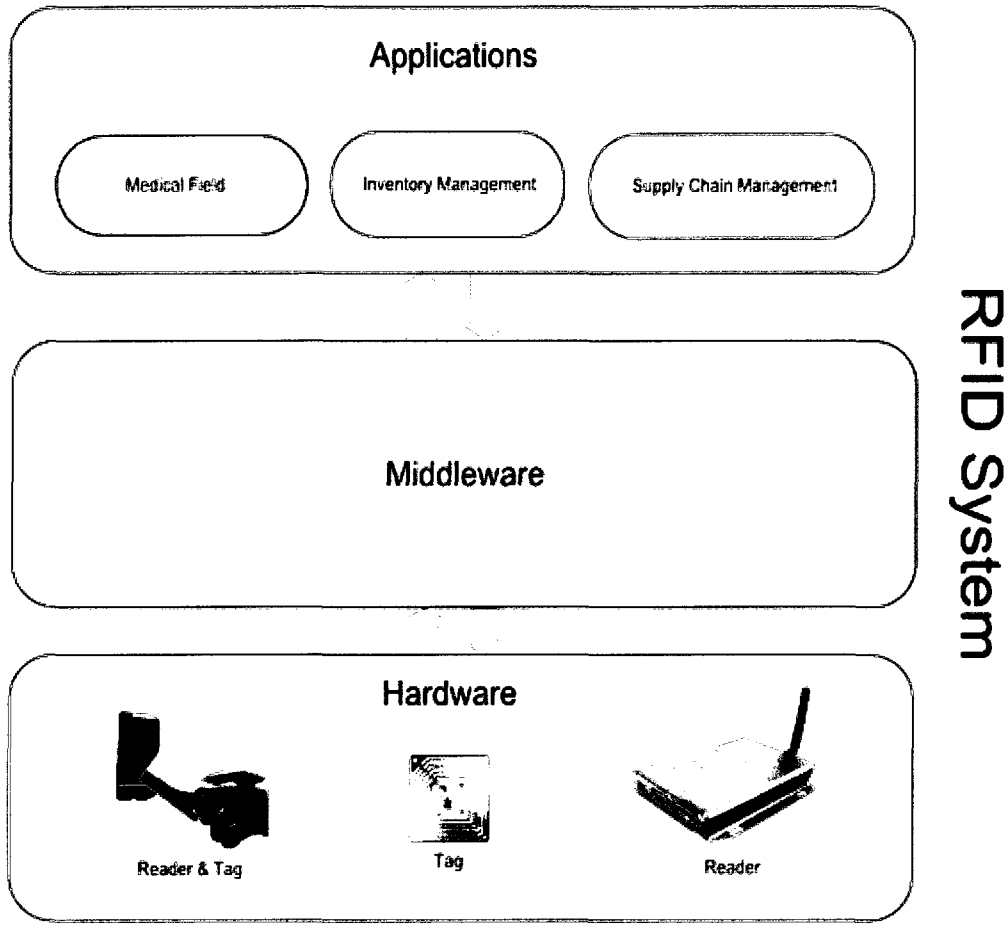

Figure 2.1: RFID middleware placement in an RFID system

The first wave of commercial RFID middleware was very expensive, averaging $125,000 or more per installed site [39]. Today, a more functional middleware can be purchased for as little as $5,000 to $20,000 [39]. Still, the scale of RFID middleware markets is estimated to be around 220 million dollars in 2011 [39]. In recognizing the potential to make a profit from RFID middleware solutions, software giants such as Microsoft, Sun Microsystems and Oracle are now coming on board with their own implementation of RFID middleware. Similarly, a number of Open Source RFID middleware solutions have emerged in the last few years [21] [37] [4].

The biggest challenge facing current middleware solutions is the diversity of RFID hardware. Inexpensive RFID reader models, such as the APSX RW-110 RFID Reader [2], do not have the capability of data translation or data processing and thus most of the work needs to be done by the middleware layer. More expensive RFID readers, such as the Motorola RD5000 Mobile RFID Reader [38], have built-in Operating Systems - such as Linux - and are able to perform data translation and processing. In such cases, the work that a typical RFID middleware does is no longer necessary. However, there still exists the problem of communicating with multiple heterogeneous RFID devices and converting raw data into application events in which a middleware can help with.

RFID readers can be used in many different environments to do various things. Readers can be classified into a two main categories: Fixed Readers and Portable Readers [44]. These classifications can further be broken down by how a reader receives its power (either through an A/C adapter or batteries) [44], what Communication Interface is supported by the reader (USB, TCP, UDP and Serial) [44], what Interrogation Protocol is supported by the reader (Passive Readers, which are limited to only "listening" and do not perform additional tag interrogations or Active Readers, which are full fledged interrogators) [44], what Frequency Spectrum is used by the reader (Unique Frequency Response Based Readers, which operate at a defined frequency range and use this frequency for both data transmission and reception or None-unique Frequency Response Based Readers which operate using one frequency for sending commands to tags) [44], what Data Encoding Protocols are supported by the reader (Simple RFID Readers, which use a unique protocol for communication and data transmission between transponders in the reader's interrogation zone or Agile RFID Readers, which can operate and perform interrogations and data transmission

with transponders using multiple protocols) [44] and the type of antenna that the reader uses (Fixed Beam, which are characterized with a unique and fixed beam radiation pattern or Scanned Array, which use smart antenna systems in order to reduce the number of transponders within their main lobe radiation zone thus reducing reading errors and collisions among tags) [44].

Fixed Readers are readers that are not meant to be moved while Portable ones are. Examples of Fixed Readers include the Dock Door or Portal Readers (which can capture bulk data from a large number of tagged items on a pallet as they are removed by a forklift or by hand [1] [9] - e.g., Symbol DC600 Portal System [10]), Overhead Readers (which work the same way as a Portal Reader but instead of capturing data as the pallet leaves the truck, they read tags that are oriented skyward inside the truck [1] [9] - e.g., Intermec F5 [6]), Stretch Wrap Station Reader (which identifies and categorizes items on pallets and associates them with RFID-enabled pallets [9] - e.g., Intermec F4 [7]), Printing and Encoding (which are printers that contain a reader module that allows them verify the data commissioned to the smart label insert at the time of printing [9] - e.g., Intermec PM4i [8]), Conveyor-belt Readers (which log information to a computer system as things move down a conveyor belt [1] [9] - e.g., Symbol XR440 [13]), Development-Kit Readers (which are inexpensive reader that can easily be used for rapid application development and testing before buying an expensive reader [29] - e.g., RFIDPhidget 1023 [41]) and Desktop Readers (which are perfect for office environments as they are small sized readers and can be placed anywhere on a computer desk [29] - e.g., RFID Internet Security Mouse [43]).

Unlike Fixed Readers, Mobile Readers are readers that are meant to be moved. Examples

of Mobile Readers include Forklift Readers (which mount on the top of a forklift and can read the pallets as they are being removed from a truck [1] - e.g., Symbol RD5000 [12]) and Handheld Readers (which are mobile readers that can be carried and operated by users as handheld units and are often running an OS for mobile devices such as Windows Mobile [1] - e.g., The Symbol MC9060-G [11]).

RFID middleware can aid developers in a number of areas. For instance, a middleware solution can support data filtering capabilities in which data can be filtered down before it reaches the application. To clarify, consider a case where an application is only interested in receiving events when tags of the ISO-15693 protocol are detected and is not interested in receiving events for tags of other protocols. If the middleware supports data filtering then the same application would require less code to determine if the detected tag is one that it is interested in.

Among the capabilities listed above that are found in a typical RFID middleware, it is also important to have middleware that is configureable and scalable to grow with the growing number of heterogeneous RFID readers. A configurable middleware is one that allows the application to define the properties of the reader that the application is working with. For instance, the application is able to define all commands that are understood by the reader and what data transfer protocol to use, i.e. *USB*, *Serial*, *TCP* or *UDP*. A scalable middleware is one that allows more data transfer protocols to be added without any changes to the middleware core. In addition, a scalable middleware must allow for more reader commands to be added to existing configurations and must allow more reader configurations to be added without any changes to the architecture of the middleware. Unfortunately, most

of today's middleware solutions do not address these important properties [21].

2.1 Related specifications

There are two organizations that are involved in drafting standards for the RFID technology: the International Organization for Standardization (ISO) [26] and EPCglobal [25]. ISO has published a number of standards for various tag communication protocols, such as the ISO 15693 [32], which are in use today by a large number of tags. EPCglobal's mission had started with the vision to identify "every item with a unique electronic product code (UPC)" [22]. It had recently published standards for newer tag communication protocols, such as the UHF Class 0/1 and UHF Class 1 Gen 2 protocols [24]. In addition, it published a Reader Protocol specification that specifies the interactions between an RFID reader device capable of reading/writing tags and application software [49]. The draft provides implementation detail for commonly used commands that can be performed on a tag, such as the ability to read, write and disable a tag. However, the draft falls short in providing a complete reader protocol specification that covers reader specific commands, such as switching the CPU of a reader on or off. Regardless, the Reader Protocol specification is in use by a number of RFID middleware today. Examples of such middleware that are compliant with the EPCglobal Reader Protocol specification are in the section to follow.

2.2 Related middleware

A number of middleware solutions were surveyed to determine what each provides in terms of data dissemination, data filtering, reliability, extensibility, support for heterogeneous

readers, support for multiple applications and the ability to perform basic operations on a tag such as disabling it or reading its data or writing data to it.

2.2.1 Fosstrak

Fosstrak is the new name for the original Open Source Java-based RFID middleware called Accada. This middleware was founded by ETH Zürich's Institute for Pervasive Computing using the Java programming language. It has a strong user community ranging from academia researchers to software developers. Fosstrak was built entirely based on the EPCglobal's EPC Network standards, which aim to formulate the basis for global adoption of RFID and the methodology for capturing, storing, sharing and viewing data [23]. Fosstrak implements most of the EPCglobal Reader Protocol, which is a standard that enables EPC-compatible readers to communicate with middleware in a standardized way [45]. Fosstrak is also able to provide data filtering functionality and relay it down to the reader level in terms of which reader is able to read a tag and what tags are filtered out. Figure 2.2 shows the different roles of the EPC Network and the interfaces involved in forwarding tag data as it arrives from a reader.

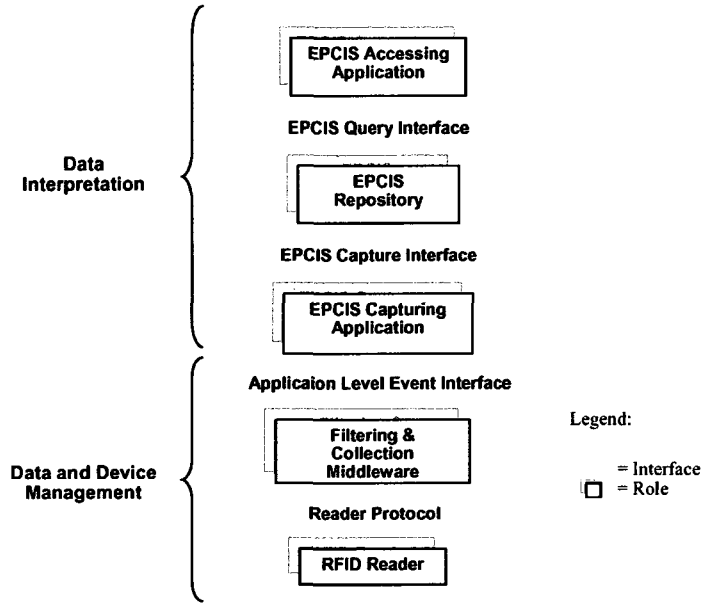

Figure 2.2: EPC Network roles and interfaces

Fosstrak comes bundled with a set of APIs to use with some of the commercially available readers such as Impinj and readers belonging to the FEIG OBID i-scan series [21]. The middleware uses a set of XML-based files to store all configurations pertaining to where an implementation of these APIs are found along with other configurable settings that can be altered by the user. The middleware allows extensibility in the sense that it allows other readers to be implemented by following a specific Hardware Abstraction Layer (HAL) interface that it provides. The middleware requires a full implementation of each new reader in a Java class files and does not aid in providing functionality to commonly used functionalities relating to data translation and filtering with ease. The middleware supports communication through *TCP* for both its FEIG and Impinj implementations but only provides *Serial* Port communication for its FEIG implementation. There is no support for other data transfer protocols such as USB for either of the two implementation.

The Fosstrak architecture consists of three modules: Reader, Filtering and Collection Middleware and EPC Information Services (EPCIS). The Reader module is mainly responsible for data filtering and data dissemination as specified in the EPCglobal Reader Protocol and EPCglobal Reader Management specifications. This module can perform in three distinct ways: by wrapping a proprietary RFID reader protocol, by using its built-in reader simulation facilities or by deploying its own reader implementation on an RFID reader to provide data filtering and dissemination capabilities [33]. The Filtering and Collection Middleware module is responsible for notifying subscribed applications when new tag data is detected by one or more readers. Data aggregation functionality is also provided by the middleware module with enhancements to discard redundant read events from different readers originating from the same location [33]. Figure 2.3 provides an illustration of this module [33]. Finally, the EPC Information Services module is mostly responsible for further processing of data received from the Filtering and Collection Middleware module by converting it into application events that an application can work with as illustrated by the EPCglobal Application Level Events (ALE) specification.

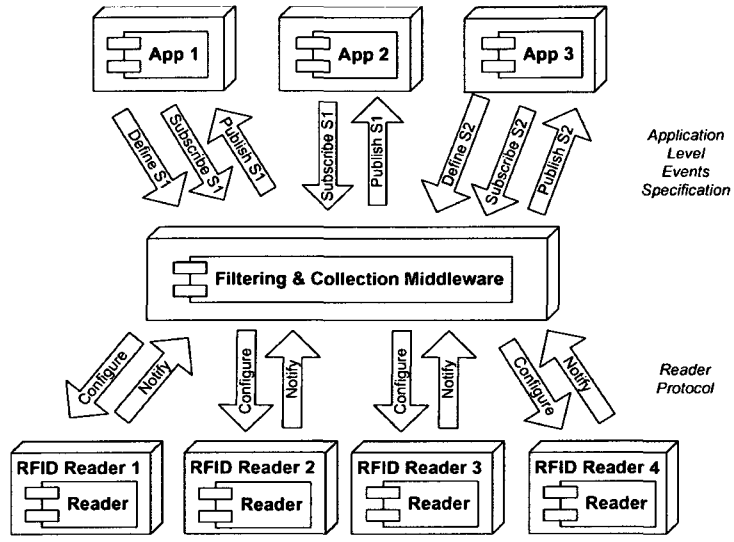

Figure 2.3: Filtering and collection middleware

In summary, the Fosstrak middleware provides an excellent implementation of the EPC-global's EPC Network standards but lacks extensibility on many grounds. It comes bundled with two basic implementations of the Impinj and FEIG OBID i-scan readers but allows implementation of other readers with little support for proprietary reader protocols. All implementations of other readers must be done entirely in code and must handle all data parsing, filtering and conversion into application events. In addition, standard data transfer protocols - such as *Serial, USB , TCP* and *UDP* - must be coded by the developer for each custom reader implementation. This quickly creates a non-standardization in how data transfer protocols are implemented for different reader modules that use the same data transfer protocol. Furthermore, any new commands that need to be added for each of the existing reader implementations would mean changing existing code and re-testing the implementation, which does not sit very well in terms of extensibility from a software engineering perspective.

2.2.2 WinRFID

WinRFID is an RFID multi-layered middleware that was founded at the University of California using the .NET framework. Its architecture consists of five main layers: the Hardware Layer, Protocol Layer, Data Processing Layer, XML Framework Layer and Data Presentation Layer [33]. The Hardware Layer is responsible for the configuration and communication with hardware such as readers and tags. The Protocol Layer is responsible for abstracting some of the most commonly used RFID protocols such as ISO 15693, ISO 14443, ISO 18000, I-Code and EPC Class 0/1 by parsing and processing the raw tag data in these formats into a format that can be used in the Data Processing Layer. The Data Processing Layer is mainly responsible for removing duplicate or erroneous reads and verifying tag reads by means of validating checksums. The XML Framework Layer is responsible for converting the data it receives from the Data Processing Layer into a user-friendly XML format that applications can use. Lastly, the Data Presentation layer is mainly responsible for storing the data from the XML Framework Layer into any database for presentation and decision making by the application. Figure 2.4 shows the five main layers as described in the previous paragraph [14].

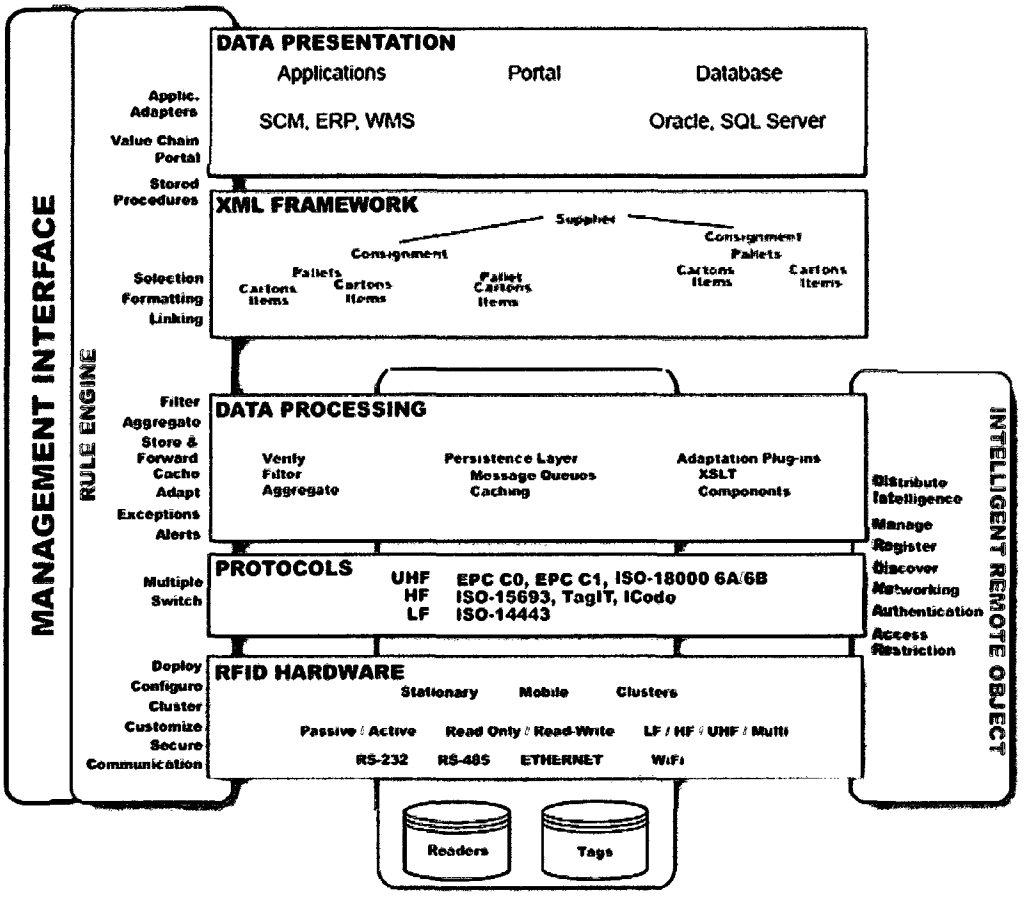

Figure 2.4: WinRFID architecture

Perhaps the most important of these layers is the Hardware Layer as it is mainly responsibly for interfacing directly with RFID readers. This layer is configurable per reader and allows the user to define their own custom implementations through the means of plug-ins. New data transfer protocols can also be added by means of plug-ins, which can be loaded or unloaded at runtime. In addition, the WinRFID middleware allows users to define their own data parsing and filtering rules in a plug-in, which is then picked up by the Data Processing Layer. Plug-ins allow the middleware to be extensible and scalable to support new hardware and new business rules.

The WinRFID middleware seems to offer a great range of extensibility options through means of plug-ins but is restricted to the .NET framework thus limiting its use to Windows/Linux based applications and Windows/Linux-compatible RFID readers only. Unlike the Fosstrak middleware, the WinRFID middleware does not implement any of the EPCglobal's EPC Network standards and does not use XML configuration files to allow changes to its various settings. Unfortunately, WinRFID shares a common element with the Fosstrak middleware in that both require the user to write code to support implementations of new readers, thus forcing the developer to have to learn how the middleware works before they are able to extend it with their own reader specific implementation.

2.2.3 Hybrid Middleware

Hybrid Middleware is another RFID middleware implementation, which is based on the idea of group communication in peer-to-peer (P2P) networks. This middleware was designed for an electronic parking management system (EPMS) to be used in a university campus setting and was supported by the Ministry of Commerce Industry and Energy of the Korean Government [17]. The middleware architecture is made up of four main groups of peer nodes: Peer Group Membership Manager, Group Communication Coordinator, Parking Event Manager and Service Connectors. The Group Membership Management node is mainly responsible for adding, changing and removing group memberships. The Group Communication Coordinator node is mainly responsible for coordinating all communication among member nodes to ensure that all parking status changes are conveyed quickly among the nodes. The Parking Event Manager is mainly responsible for relaying

all parking events to the database service where the events are stored for future processing and analysis. Lastly, the Service Connectors node is responsible for managing and controlling individual nodes within the distributed environment.

Currently, the Hybrid Middleware handles two types of parking events: ENTERING and LEAVING [33]. The ENTERING parking event is an event that is generated by the middleware when a vehicle enters a parking slot. Similarly, the LEAVING parking event is an event that is also generated by the middle when a vehicle departs from its parking slot.

Unlike the WinRFID and Fosstrak middleware, the Hybrid Middleware is geared to work in a distributed environment. Middleware that is setup to work in a distributed environment tends to have an advantage over middleware designed to work in a centralized environment because of its higher extensibility, resource availability and fault tolerance. On the downside, infrastructure costs for setting up a distributed middleware system is likely to be significantly higher than its centralized middleware system equivalent. Moreover, the Hybrid Middleware does not seem to provide a way to manage the different types of RFID readers available today and lacks the means to allow for new reader modules to be registered in its distributed architecture. Furthermore, the current implementation of the Hybrid Middleware is only suited for a parking management system type of environment and does not offer an adequate transition to other environments without changes to the core of the middleware itself.

2.2.4 LIT Middleware

Logistics Information Technology (LIT) Middleware was founded at the Research Institute of Logistics Information Technology in South Korea, which was designed to address current constraints of the RFID technology itself as well as the needs of a typical RFID application. It was written using the Java programming language and partially conforms to the EPC Network Architecture specification by implementing the Application Level Events (ALE) and EPC Information Services (EPCIS) components [33]. The architecture of the Application Level Event component is composed of four main layers: Application Abstraction, State-based Execution, Continuous Query and Reader Abstraction, while the EPC Information Services component is composed of three layers: Query Service Layer, Repository Layer and Capturing Service Layer [34].

Throughout these layers, the LIT Middleware hopes to deliver a high performance mechanism to manage large deployments of heterogeneous readers. The LIT Middleware directly benefits from features provided by the ALE and EPCIS specifications such as high performance, high extensibility, removal of duplicate reads, application abstraction and overall extensibility. Figure 2.5 shows where the LIT Middleware fits in on the EPC Network Architecture [34].

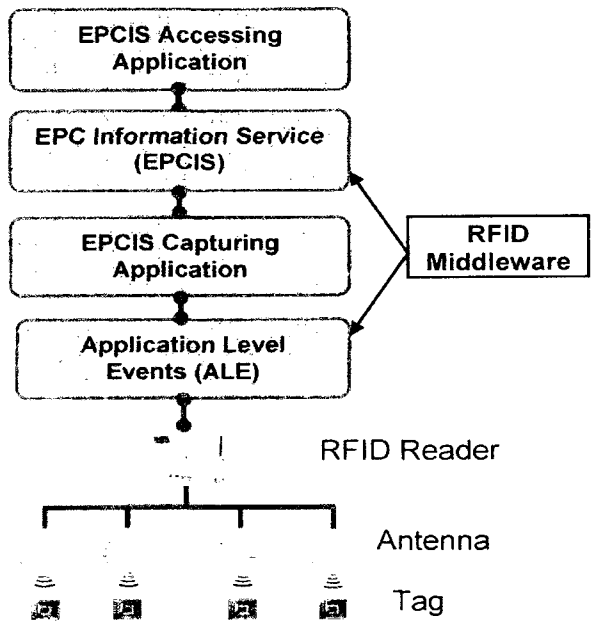

Figure 2.5: LIT Middleware on the EPC Network Architecture

2.2.5 Sun Java System RFID Software

Due to the sudden growth of RFID-related applications in recent years, software giants such as Microsoft, Oracle and Sun have come on board with their own RFID middleware implementations: Microsoft BizTalk RFID Server, Oracle Sensor Edge Server and Sun Java System RFID Software. The Sun Java System RFID Software is part of the Java Enterprise System (JES) and conforms to the EPCglobal standards. It is architected to provide high levels of reliability and extensibility for an Electronic Product Code (EPC) Network [27]. Its architecture also allows it to process EPC data and events between EPC readers and RFID software applications. The RFID Software is composed of two main components: RFID Event Manager and RFID Information Server. Both of these components are also built with Java and are available for download from Sun's website.

The RFID Event Manager is responsible for communicating with EPC readers and is optimized to process large amounts of data. It can seamlessly capture, filter and store EPC data and EPC events in an RFID Information Server that is available in the network. The main objective of this component is to provide an interface with RFID readers, gather EPC data and EPC events for processing and storage and filter out redundant data reads whenever possible. This component is used by many RFID middleware solutions today.

The RFID Information Server is mainly responsible for the storage of EPC data and EPC events. It is a J2EE application that can also be queried to obtain previously stored EPC data and EPC events, such as tag observation data that is captured by the RFID Event Manager. Another key feature of the RFID Information Server is to translate low level tag observation data into high level business functions. Figure 2.6 exemplifies how the RFID Event Manager and RFID Information Server fit into the EPCglobal network [27].

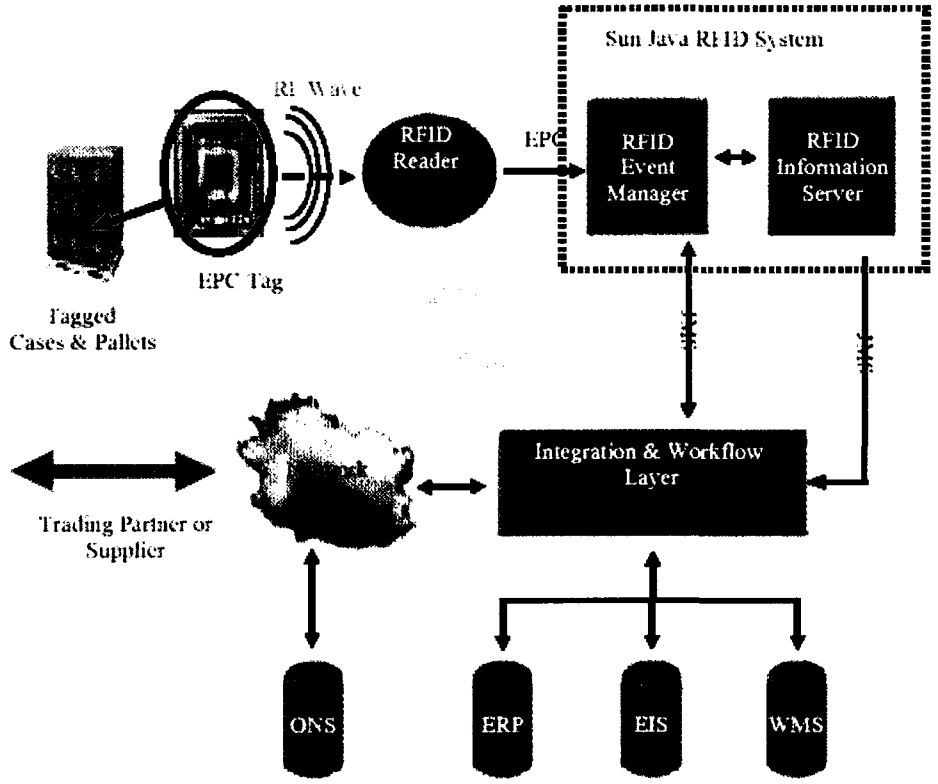

Figure 2.6: RFID Event Manager/Information Server in EPCglobal Network

2.2.6 Microsoft BizTalk RFID Server

The Microsoft BizTalk RFID Server provides a .NET middleware implementation that abstracts RFID devices and provides a plug-and-play mechanism to allow the addition and removal of standard and non-standard RFID devices to and from an RFID system on the fly. The middleware provides an event processing layer that is capable of managing events based a set of given business rules. It is designed to provide robustness and extensibility for development, deployment and management of RFID applications [36]. The middleware conforms to the EPCglobal standards such as EPC Information Services (EPCIS), Tag Data Translator (TDT) and Low Level Reader Protocol (LLRP) [36].

The middleware provides monitoring and troubleshooting capabilities of deployed RFID applications using its System Center Operations Manager. The middleware also provides data filtering configurations, which filter out non-business relevant data thus allowing the application to focus on relevant data events. Captured events are stored in a Microsoft SQL Server, which the BizTalk Server integrates well. RFID applications can query the BizTalk's Business Activity Monitoring (BAM) module at any time to obtain details of all previously captured events. The Microsoft BizTalk RFID Mobile provides an extension of all these features to mobile RFID applications that are created for the Windows Mobile OS or Windows CE OS. Microsoft has an advantage over its competitors in the RFID field by providing a middleware solution for mobile applications and thus providing mobile workers with real-time access to RFID information on the go. Figure 2.7 depicts an overview of the BizTalk RFID runtime architecture [36].

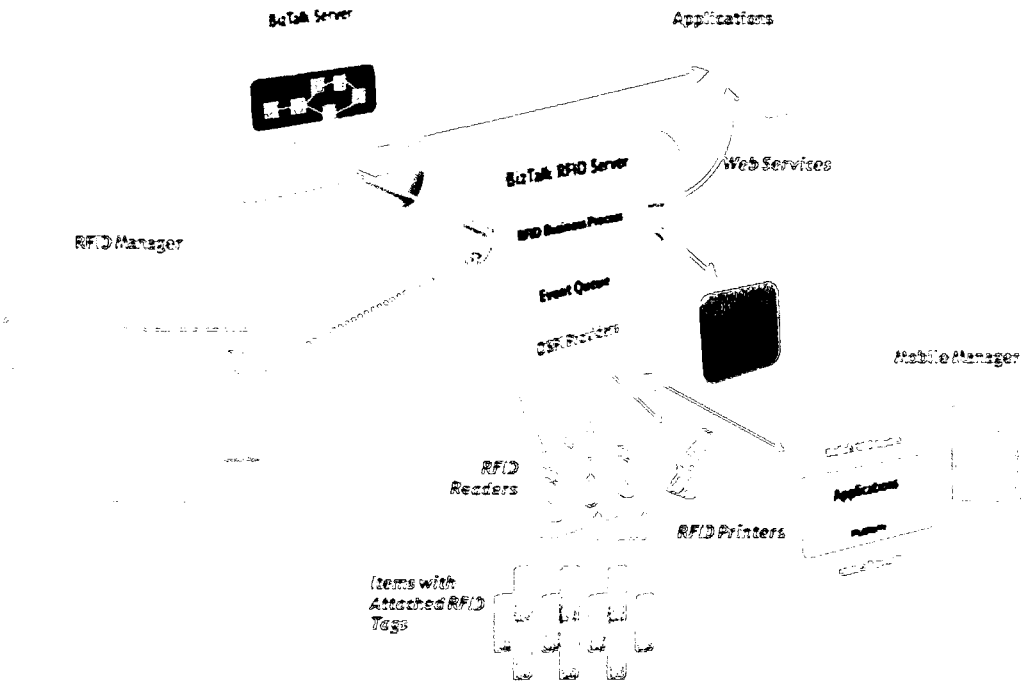

Figure 2.7: BizTalk RFID runtime architecture

2.2.7 Oracle Sensor Edge Server

The Oracle Sensor Edge Server brings time and cost benefits to companies that wish to integrate their existing IT solutions with the RFID technology. It provides out-of-the-box integration with a number of RFID devices, thus making it easy for small companies to adapt quickly to the RFID technology. The Oracle Sensor Edge Server is compliant with the EPCglobal Application Level Events specification. It hooks on to Oracle's Fusion Middleware [28] to provide RFID applications with management of RFID business processes. Applications can query the RFID Event Monitoring and the Business Activity Monitoring layers to obtain information on previously captured tag data. Captured tag data and events can be optionally stored in an Oracle database.

The Oracle Sensor Edge Server is composed of five main layers: Device Abstraction Layer, Dispatchers, Administration, Development Services and Sensor Data Management [40]. The Device Abstraction Layer abstracts RFID readers from subscribed applications and features a plug-and-play architecture for hot attaching and detaching of RFID readers. This layer is bundled with a few reader drivers from manufactures such as Alien, FEIG, EMS, Zebra, Matrics and Intermec. The Dispatcher layer provides different methods of communicating with applications through means of dispatching data via HTTP, Web Services and Streams/AQ, JMS [5]. The Administrator layer provides basic administrative functionalities to allow for the addition, configuration and removal of RFID devices, filters, dispatchers and the system itself [40]. The Development Services layer provides developers with a Web Services Interface and a HTTP Programming Interface to obtain server information, handle and process system events, encode and decode tag data and manage RFID devices. The Sensor Data Management layer provides means to archive system events to a database and allow stored events to be queried by the application. Figure 2.8 depicts an overview of the Oracle Sensor Edge Server Architecture [40].

2.2.8 CUHK RFID Middleware

The CUHK RFID Middleware is an Open Source Java-based RFID middleware. It was co-developed by the department of Electronic Engineering, Information Engineering and Systems Engineering & Engineering Management at the Chinese University of Hong Kong. The middleware complies with the EPCglobal version 1.0 specifications by following the architecture framework specification of EPCglobal and the Application Level Events (ALE)

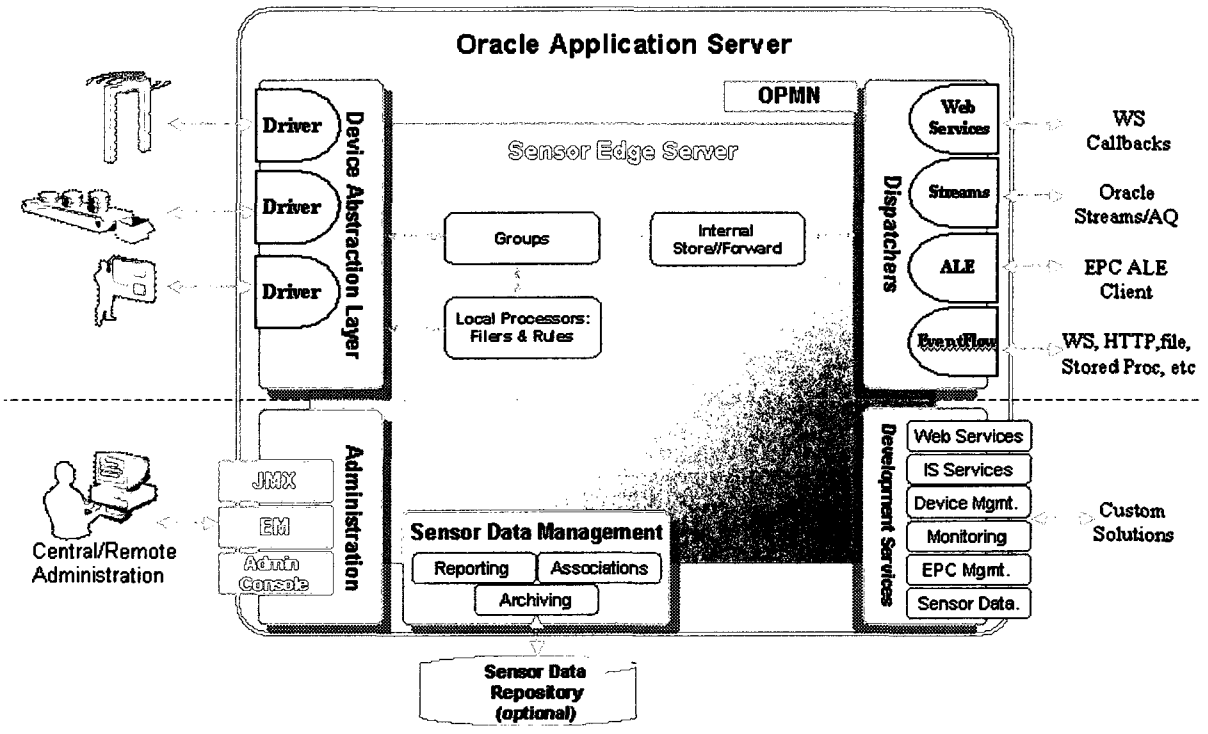

Figure 2.8: Oracle Sensor Edge Server Architecture

specification. Readers that support the TCP or Serial communication interfaces are supported by this middleware. In addition, the middleware supports EPC and ISO tag standards, such as the EPCglobal UHF Class 1 Gen 2 (ISO 18000-6C) as well as all earlier EPC tag types and ISO tags [37]. Furthermore, the middleware supports CUHK's own CuTag reader and CuBadge reader which were developed by the CUHK Electronic Engineering department in an attempt to support various kinds of sensing functionality. The middleware is extensible in itself as it provides means for new readers to be added via addition of more device adaptors. Figure 2.9 depicts an overview of the CUHK RFID Middleware architecture [35].

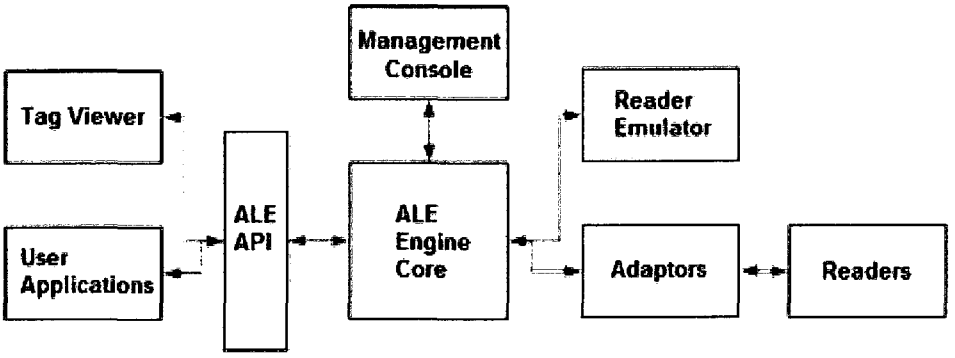

Figure 2.9: CUHK RFID Middleware architecture

The CUHK RFID Middleware is designed for easy deployment and configuration and comes bundled with a set of common filters, adaptors and processing modules. In addition, the solution provides a web-based management console for system administrator as well as simple test modules such as Reader Emulator and Tag Viewer. The bundle is available for download from CUHK's website [37].

2.2.9 Summary

The middleware solutions discussed previously all seem to share one common element in that none provides a specification or schema that applications can build against to interact with their RFID readers. Moreover, an implementation must be provided for each reader that the application wishes to work with for the middleware to work properly with the application. Lastly, these middleware solutions come-up short in providing the application with a tag object that it can use to obtain the different properties of a tag easily and to provide a complete abstraction from the tag hardware and its raw data. Chapter 3 addresses these shortcomings by proposing an extensible middleware solution - called RFIDMania -

that bridges the gap among heterogeneous readers by providing an adaptable reader specification with a schema specification for developers to code against. Table 2.1 provides a comparison among the middleware solutions discussed previously and the proposed RFID-Mania middleware solution.

Middleware	Extensible	Configurable	Data Management	Hot Plug	Portable Code	Requires Coding	Reader Specification & Schema	API	Reusable Data Transfer Layer	Tag Object
Fosstrack WinRFID	Class files Plug-ins	No No	EPCglobal ALE Data [illegible]	No Yes	Yes No	Yes Yes	No No	Yes [illegible]	No [illegible]	No [illegible]
Hybrid Middleware LIT Middleware	Distributed No	No	None EPCglobal ALE	No No	No Yes	Yes Yes	No No	No No	No No	No
Sun Java System RFID Software	Yes	Yes	EPCglobal ALE	Yes	Yes	Yes	No	Yes	No	No
Microsoft BizTalk	Yes	Yes	Microsoft SQL	Yes	No	Yes	No	Yes	No	No
Oracle Sensor CUHK RFID Middleware	Yes Class files	Yes Yes	Oracle DB EPCglobal ALE	Yes Yes	No No	Yes Yes	No No	Yes Yes	No No	No No
RFIDMania	PCF, RFCF, MCF, TCF & RCF	No	Application events	Yes *	Yes	No	Yes	Yes	TCP, UDP, Serial & USB	Yes

*= hot plug here means the ability to load new Reader Configuration Files or change existing ones on the fly, while the middleware is running.

Table 2.1: Middleware solutions comparison table

3 Approach

This chapter will discuss the reference implementation of a middleware, specifications and schemas that are proposed to achieve the goals and objectives as outlined in Sections 1.3 and 1.4. Three elements were implemented for this: a scalable RFID Middleware reference implementation, an API that applications can use to interact with the RFIDMania middleware and a specification and schema for standardizing the way RFID protocols/radio-frequencies/manufacturers/tags and readers are described in a configuration file. This chapter will discuss the RFIDMania middleware reference implementation in Section 3.1 and will address the schema specification that developers should follow to create configuration files that can be used with RFIDMania middleware in Sections 3.2-3.7. Lastly, the RFIDMania API will be introduced in Section 3.8 to allow applications to interact with RFIDMania through a common interface. Figure 3.1 shows how these three elements work together to provide standardization across RFID readers and to allow applications to interact indirectly with RFID readers through the RFIDMania middleware implementation to capture tag data and receive application events.

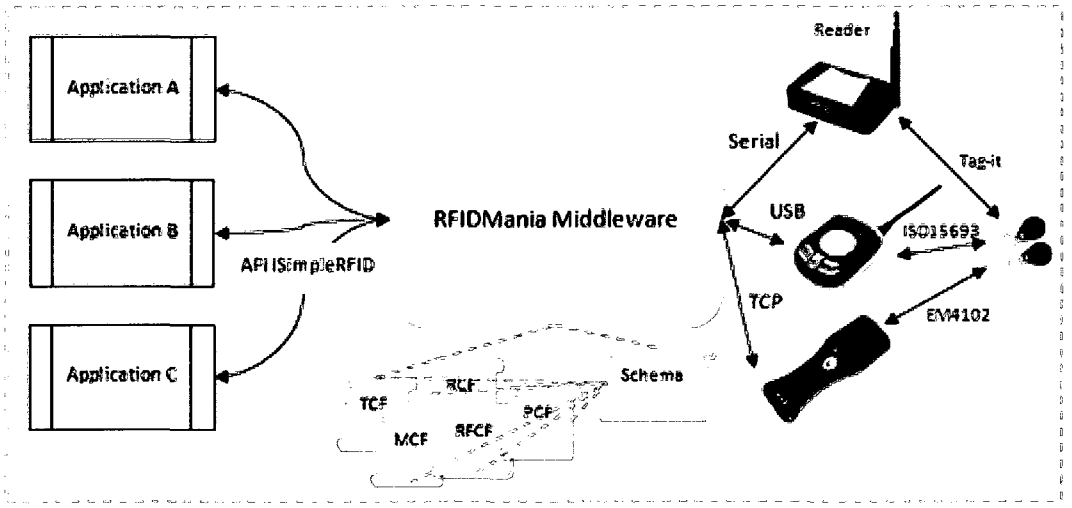

Figure 3.1: Applications, RFIDMania Middleware, Schema specifications, Readers and Tags interaction diagram

3.1 RFIDMania Reference Implementation

RFIDMania was written using the Java programming language. The implementation of RFIDMania is split into four main packages: *Definition*, *Implementation*, *Validation* and *Utilities*. The *Definition* package defines all the key classes that make up the core of the middleware and provides the dependencies needed by the *Implementation* package. The *Implementation* package, as its name suggests, provides the basic implementation of the middleware. The *Validation* package provides developers with a basic reader validation GUI, which the developer can use to quickly validate their implementation against an RFID reader without having to write any code. Finally, the *Utilities* package provides implementation of shared functionality that the middleware can use in any of its packages. The subsequent paragraphs will explore each of these packages. Figure 3.2 illustrates the RFIDMania middleware layer relative to other layers of a typical RFID system [15].

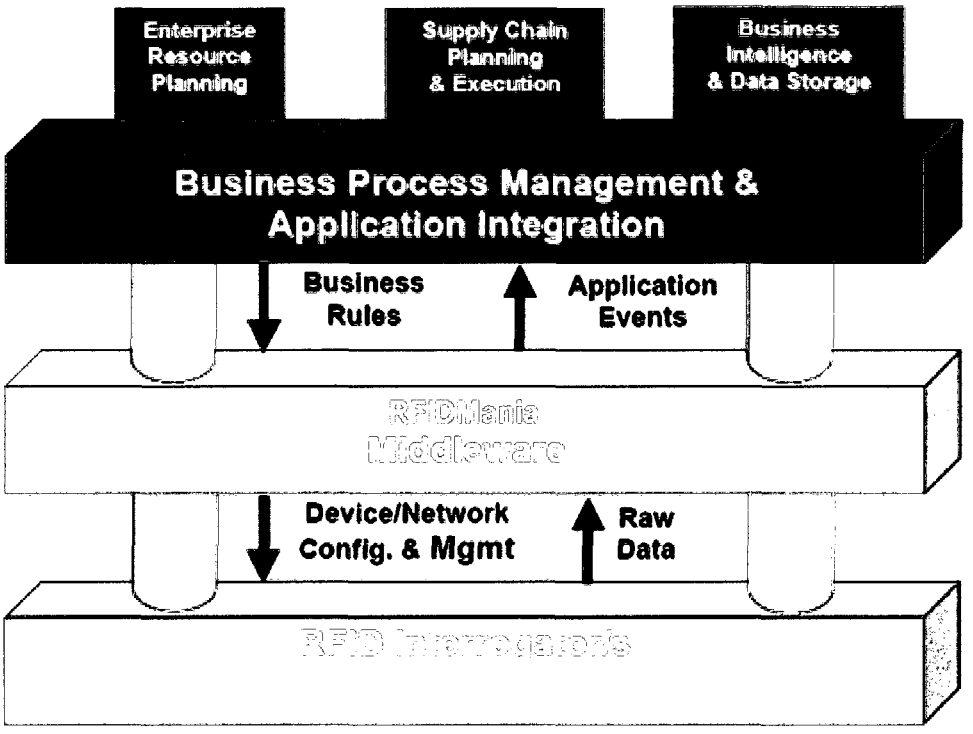

Figure 3.2: RFIDMania middleware layer in a typical RFID system

The *Definition* package includes classes that makeup the core of the middleware. Examples of such classes include a set of XML parser classes, a *Filter* class that the application can make use of to specify what information it is interested in receiving, the *Manufacturer* class, which represents a tag's manufacturer, the *Memory* class, which represents memory segments in a tag, the *Protocol* class, which represents a tag's protocol, the RFIDCommand, which represents a reader command, the *Transponder* class, which represent a tag and a few others of lesser significance. The *Definition* package also contains a set of enumerations that are used throughout the system, such as *TransmissionProtocol* enumeration that lists all possible data transfer protocols: *USB*, *Serial*, *TCP* and *UDP*. Furthermore, the *Definition* package contains a number of interfaces that are used by the system, such as the

ISimpleRFID interface that defines all the methods that an application can use with RFID-Mania. Lastly, the *Definition* package includes a number of resource files that are used by the classes defined above, such as the *Manufacturer.xml* resource file, which includes information regarding all known RFID tag manufacturers. Figures 3.4 and 3.5 provides a UML representations of all enumeration and interface classes respectively. Figure 3.3 provides a UML sequence diagram of a FEIG reader validation class and the *ISimpleRFID* interface.

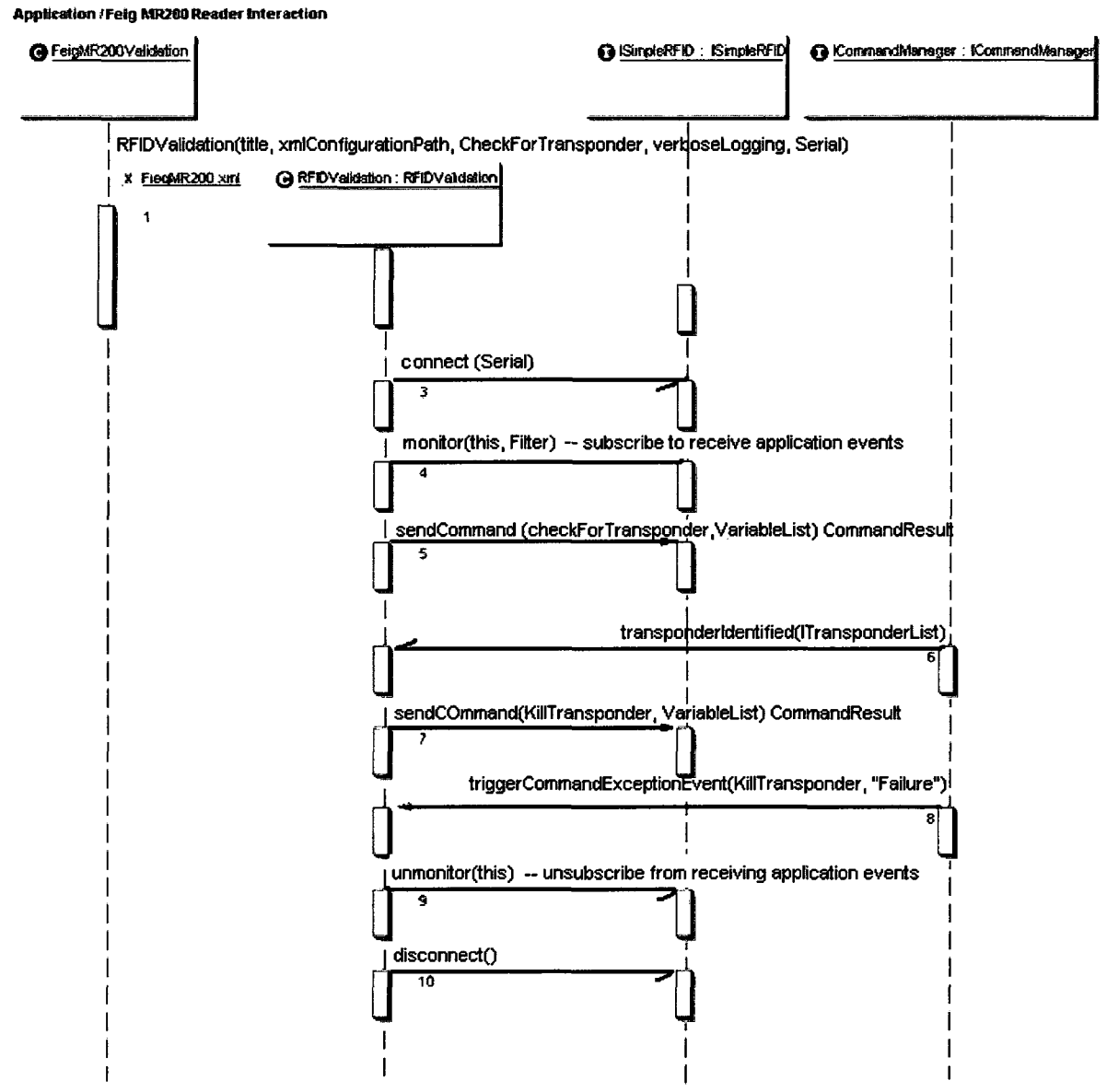

Figure 3.3: UML interaction diagram of a FEIG reader validation class and RFIDMania

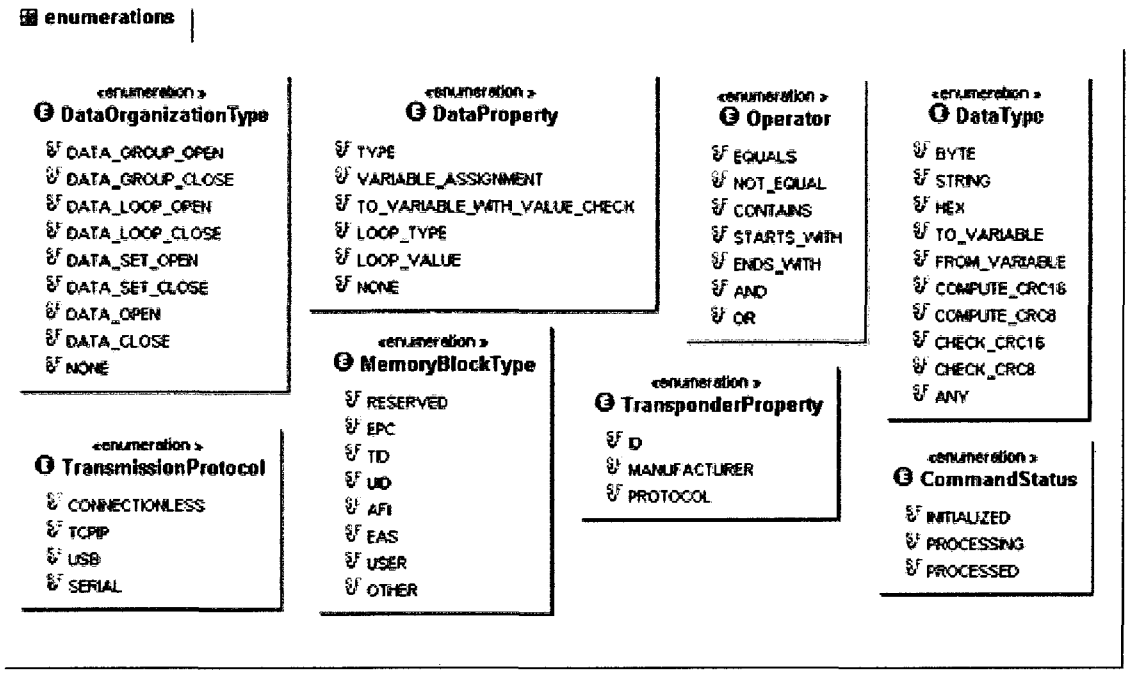

Figure 3.4: Enumeration classes

The *Implementation* package ties together the classes in the *Definition* package to create the middleware. The package contains a key class called *Initializer*, which takes on the role of initializing and preparing the middleware for use by the application. In addition, this class is the one that the application communicates with indirectly through the *ISimpleRFID* interface. All reader command requests by the application pass through this class and get forwarded to another class in this package called *CommandManager*. The *CommandManager* takes on the responsibility of managing the life cycle of a command request. The *Communicator* class is another key class of the *Implementation* package that provides a shell for data transfer protocol classes (Serial, USB, TCP and UDP) to send and receive data as needed. Figure 3.6 provides a UML representation of the Serial, USB, TCP and UDP data transfer protocol classes.

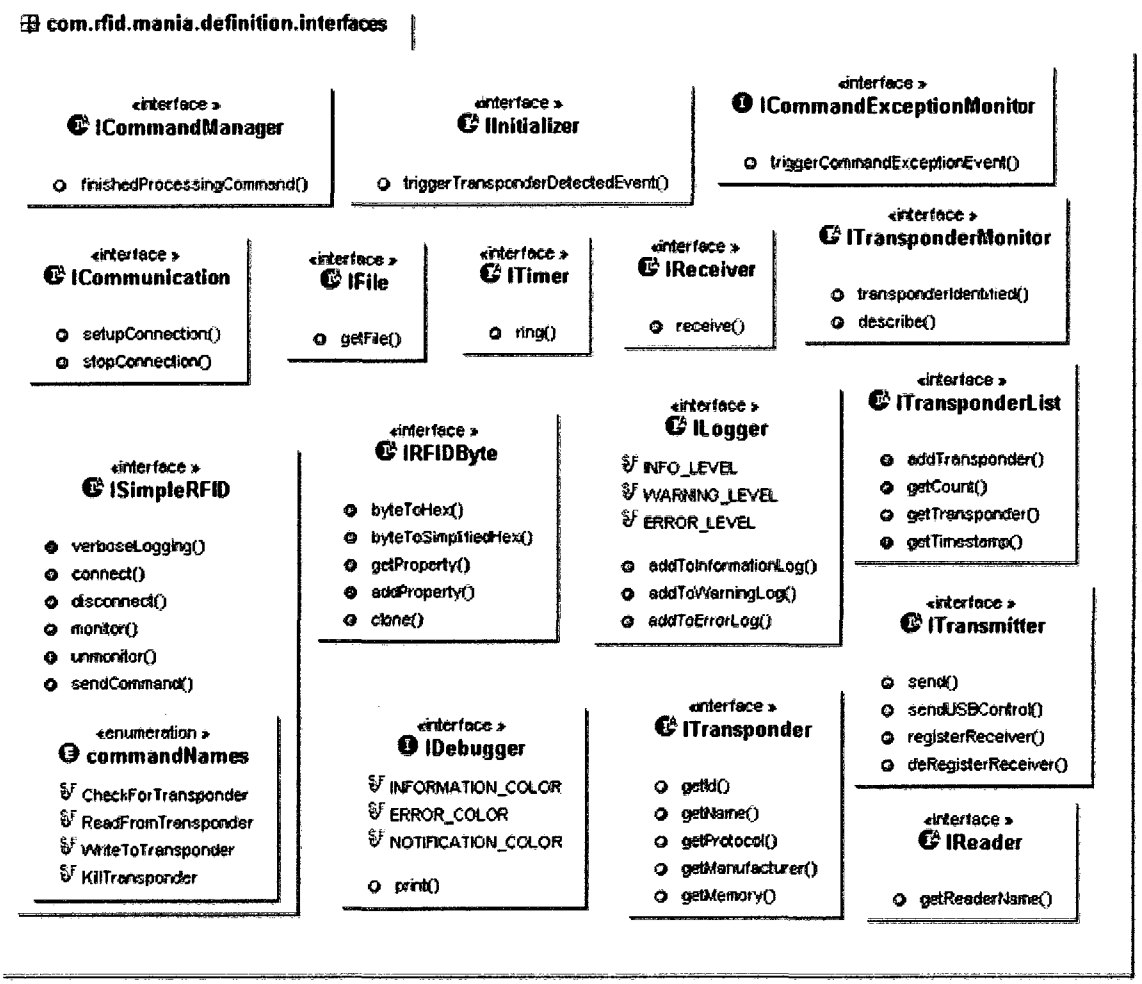

Figure 3.5: Interface classes

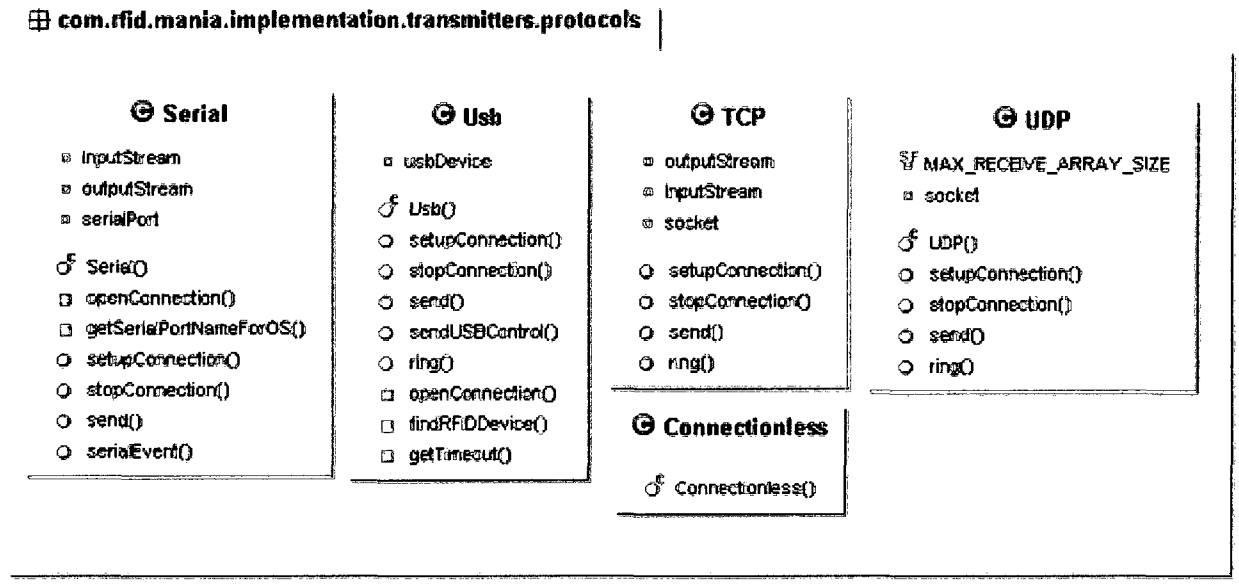

Figure 3.6: Transmission protocol classes

The *Validation* package provides a class that allows users to validate their Reader Configuration Files - discussed below - against their readers through an interactive GUI without having to write any code. The package contains a single class called *RFIDValidation*, which an application can easily extend to validate their own set of commands with a reader. The *RFIDValidation* GUI provides the user with a drop-down list of all possible commands to send to the reader for validation and uses a text area to log all events as they occur in the system. Figure 3.7 shows the RFIDValidation GUI with a drop-down list of possible reader commands to validate with. Listing 3.1 shows an example of a validation application that makes use of the *RFIDValidation* class by extending it.

Listing 3.1: Example validation application that extends the *RFIDValidation* class

```
1  //FEIG MR 200 CPU reset command validation class
2  public class FeigMR200Validation extends RFIDValidation {
3
4      public FeigMR200Validation()
5      {
6          //instantiate RFIDValidation using the 'FiegMR200.xml' Reader Configuration File
7          //only allow the CPUReset command to be available to the user for validation
8          super("FEIG MR200", "FiegMR200.xml", new String[] {"CPUReset"}, true, null);
9      }
10
11     //method triggered when user selects a command from GUI list
12     protected void commandListChanged(String commandName)
13     {
14         try
15         {
16             //if the user selected the 'CPU reset' command from the list
17             if (commandName.equals("CPUReset"))
18             {
19                 //send CPU reset command
20                 CommandResult CPUResetCommand = getInitializer().sendCommand("CPUReset", ↩
                       null);
21             }
22
23         } catch (RFIDException e) {
24             //log errors
25         }
26     }
27 }
```

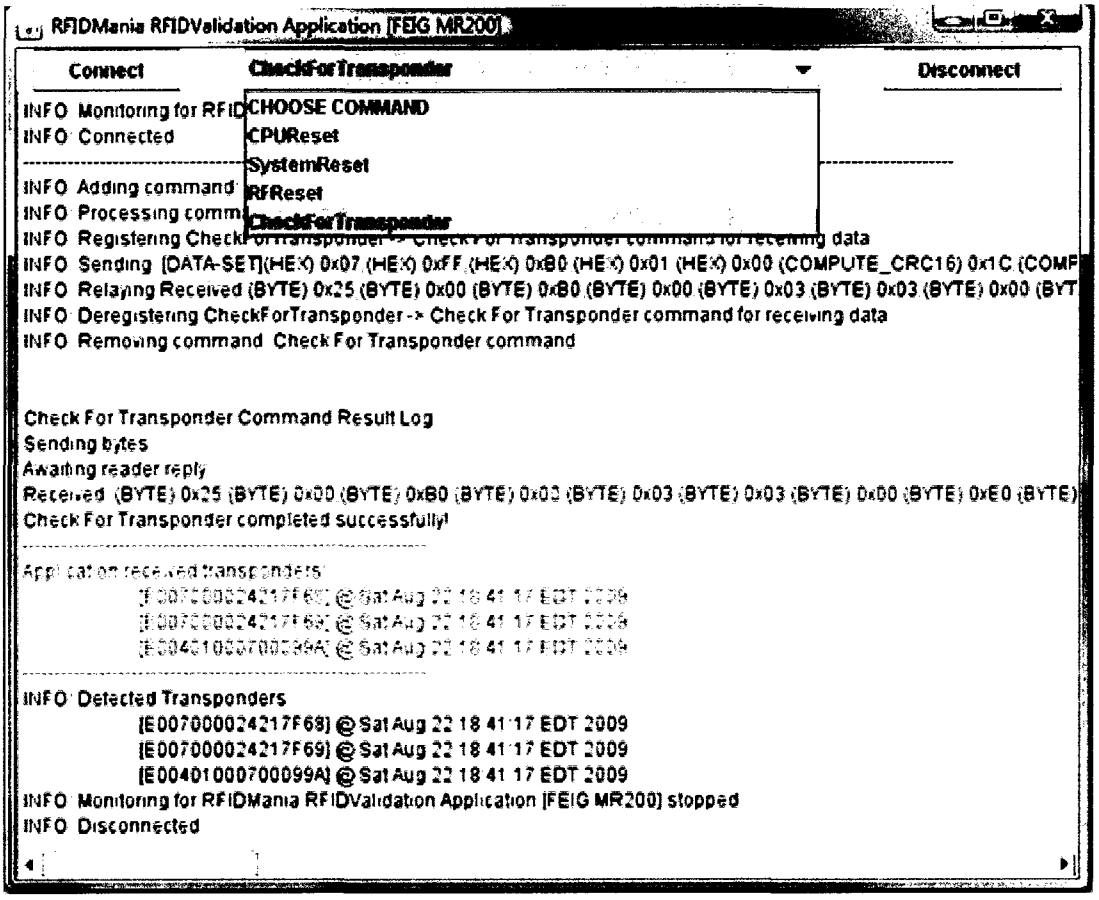

Figure 3.7: RFIDValidation GUI with drop-down list of commands to validate with

Figure 3.8 shows a class diagram of Listing 3.1 and highlights the main classes and interfaces in use between the *FeigMR200Validation* and *RFIDValidation* classes. This code snippet for Listing 3.1 is provided and explained in full in Listing 4.2 in Chapter 4.

The *Utilities* package provides a set of shared classes that are used throughout the middleware. Examples of such classes include the *RFIDByte* class, which essentially is the class responsible for performing data conversion from and to the reader. Other classes of this package include the *CRC16* and *CRC8* checksum classes. Figure 3.9 provides a UML

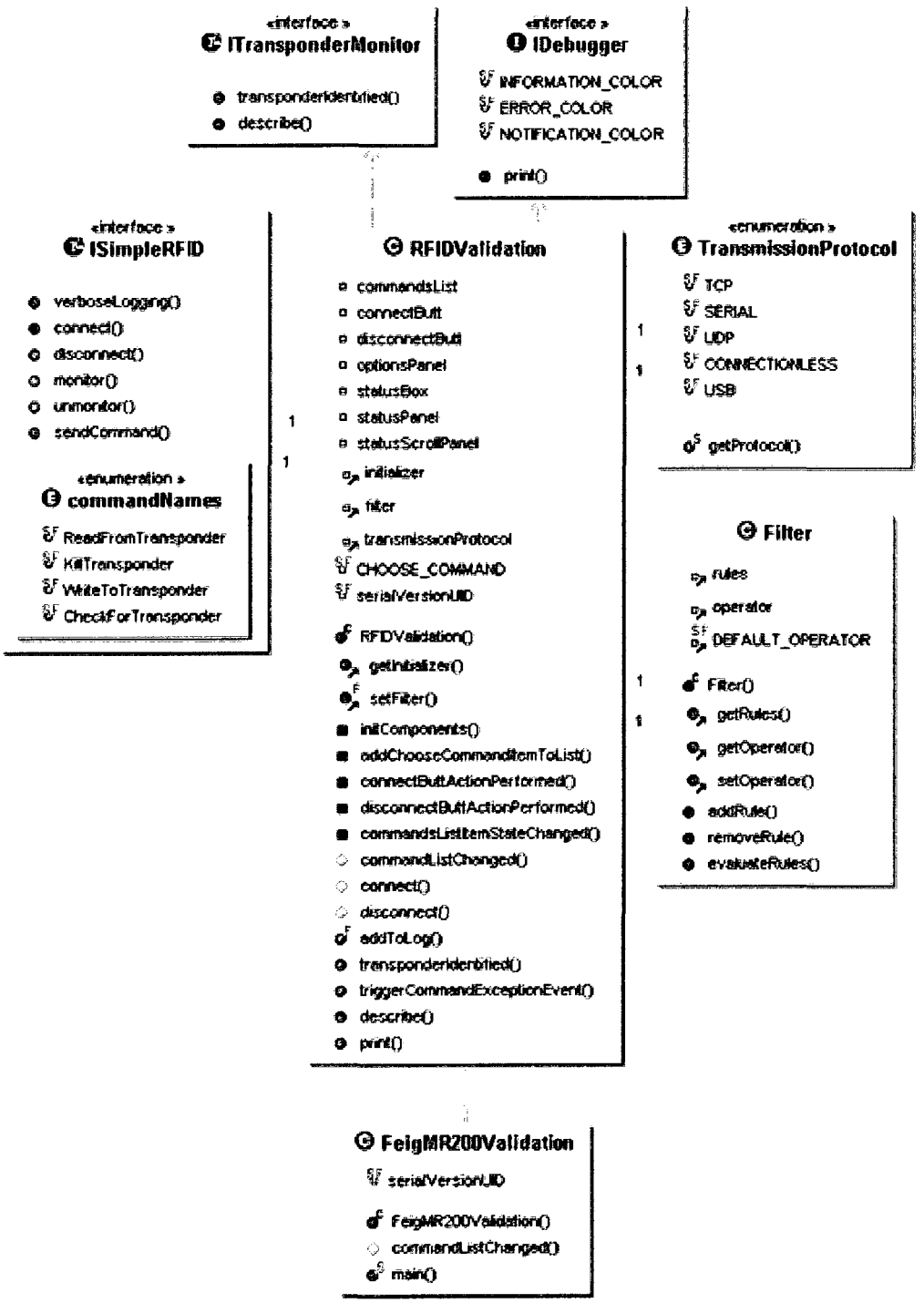

Figure 3.8: UML class diagram of Listing 3.1

representation of classes of the *Utilities* package

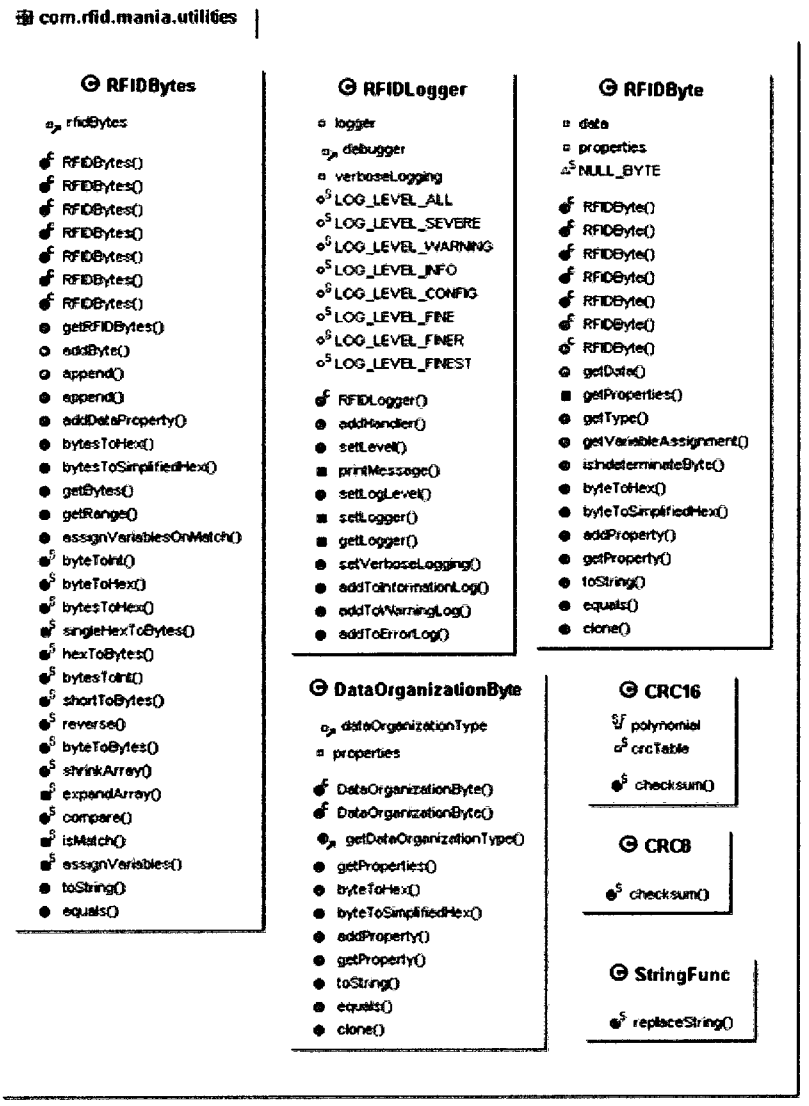

Figure 3.9: *Utilities* package classes

3.2 Protocol Configuration File

A Protocol Configuration File (PCF) is an XML-based configuration file that is used to describe all known air interface protocols that illustrate the rules that govern how tags and readers communicate. Different readers support different tag protocols and most readers do not support all the different protocols available. Each air transfer protocol is given a unique byte code, which tags must carry as part of their memory. The protocol code is an important piece that the reader looks for when it detects a radio frequency in its vicinity. Through the protocol code, the reader can decide to ignore a tag if it does not support working with its protocol.

The PCF is used by RFIDMania to extract the full protocol name of a tag based on its protocol code. In other words, the PCF helps to map protocol codes to protocol names, making it easier for the application to filter out tags of a certain protocol - for instance. A protocol object is provided as part of the Transponder object that RFIDMania provides to the application when it detects a transponder. Listing 3.2 illustrates how the I-Code1 [42], Tag-it [47] and ISO15693 [32] protocols are described in a PCF.

Listing 3.2: Example air interface protocols in the PCF

```
<Protocol>
    <Code>0x00</Code>
    <Name>I-Code1</Name>
</Protocol>
<Protocol>
    <Code>0x01</Code>
    <Name>Tag-it</Name>
</Protocol>
<Protocol>
```

```
10      <Code>0x03</Code>
11      <Name>ISO15693</Name>
12  </Protocol>
```

Appendix B provides a PCF listing all known RFID air interface protocols while Appendix A provides the schema for it.

3.3 Radio-Frequency Configuration File

The Radio-Frequency Configuration File (RFCF) is an XML-based configuration file that is used to describe all known radio frequencies that RFID tags/transponders use. A unique byte code is assigned to each radio frequency and is globally recognized. The radio frequency byte codes can be translated to describe the radio frequency in words. This is where the RFCF comes in handy as RFIDMania uses the RFCF to provide a more comprehensive description of the detected transponders/tags to the application. Applications can also make use of the RFCF to filter tags of a certain radio frequency. Listing 3.3 illustrates how the Low frequency and Ultra-high frequency are described in the RFCF.

Listing 3.3: Example radio frequencies in the RFCF

```
1  <RadioFrequency>
2      <Code>0x1</Code>
3      <Name>LF</Name>
4      <Description>Low frequency</Description>
5  </RadioFrequency>
6  <RadioFrequency>
7      <Code>0x2</Code>
8      <Name>UHF</Name>
9      <Description>Ultra-high frequency</Description>
```

```
10 </RadioFrequency>
```

Appendix D provides a RFCF listing all known RFID radio frequencies while Appendix C provides the schema for it.

3.4 Manufacturer Configuration File

The Manufacturer Configuration File (MCF) is an XML-based configuration file that is used to describe all the major manufacturers of RFID tags/transponders. Tags generally have a byte reserved to indicate their manufacturer code. The manufacturer code of a detected tag is mapped to one in the MCF to get the fully spelled out name of the manufacturer. In other words, the MCF file maps manufacturer codes to manufacturer names. Instead of having the application figure out the name corresponding to the manufacturer code, RFIDMania creates a Manufacturer object and attaches it to a Transponder object, which the application can then use to obtain the manufacturer name or manufacturer code. The application can also make use of the MCF file to filter out tags from a certain manufacturer without necessarily having to work with manufacturer codes. Listing 3.4 illustrates how the Philips, Infineon and Texas Instruments manufacturers are described in the MCF.

It is important to note that the manufacturer code is not always unique as there doesn't exist any standardization in this area. In other words, it is possible for two companies to have the same manufacturer code. Therefore, the developer should be aware of this when creating their MCF that there may be more than one manufacturer block with the same code.

Listing 3.4: Example manufacturers in the MCF

```
<Manufacturer>
   <Code>0x04</Code>
   <Name>Philips Semiconductors</Name>
</Manufacturer>
<Manufacturer>
   <Code>0x05</Code>
   <Name>Infineon</Name>
</Manufacturer>
<Manufacturer>
   <Code>0x07</Code>
   <Name>Texas Instruments</Name>
</Manufacturer>
```

Appendix F provides a MCF listing all known RFID manufacturers while Appendix E provides the schema for it.

3.5 Transponder Configuration File

The Transponder Configuration File (TCF) is an XML-based configuration file that spells out the different types of transponders, which are available in the market today. The aim here is to keep a very small database of frequently used transponders/tags and some details concerning each. A combination of the PCF, MCF and RCF files is used by RFIDMania to cross-reference detected tag protocol codes and manufacturer codes with the list of ones that are available in the TCF file to provide the application with the complete picture of the tag that was detected in a Transponder object form. This is very useful to the application as it can just reference the Transponder object to get its properties, such as the model name/number, protocol type, manufacturer name, internal memory size and whether or not

it is locked for reading or writing. Furthermore, the TCF file can also be useful to readers that do not provide error handling of locked tags or read-only tags so the application can check the properties of the detected tag to determine if it can write to it before issuing the *WriteToTransponder* command.

Figure 3.10 provides a UML Class Diagram of the Transponder, Memory, Protocol and Manufacturer classes. Listing 3.5 provides a definition of a read-only Fujitsu MB89R116/118 tag with the ISO15693 protocol.

Listing 3.5: Fujitsu MB89R116/118 tag definition in the TCF

```
<Transponder>
   <Name>MB89R116</Name>
   <Name>MB89R118</Name>
   <Protocol>ISO15693</Protocol>
   <Manufacturer>Fujitsu Limited</Manufacturer>
   <Memory>
      <Size>256</Size>
      <BytesPerBlock>16</BytesPerBlock>
      <User>
         <Read>true</Read>
         <Write>false</Write>
      </User>
</Memory>
</Transponder>
```

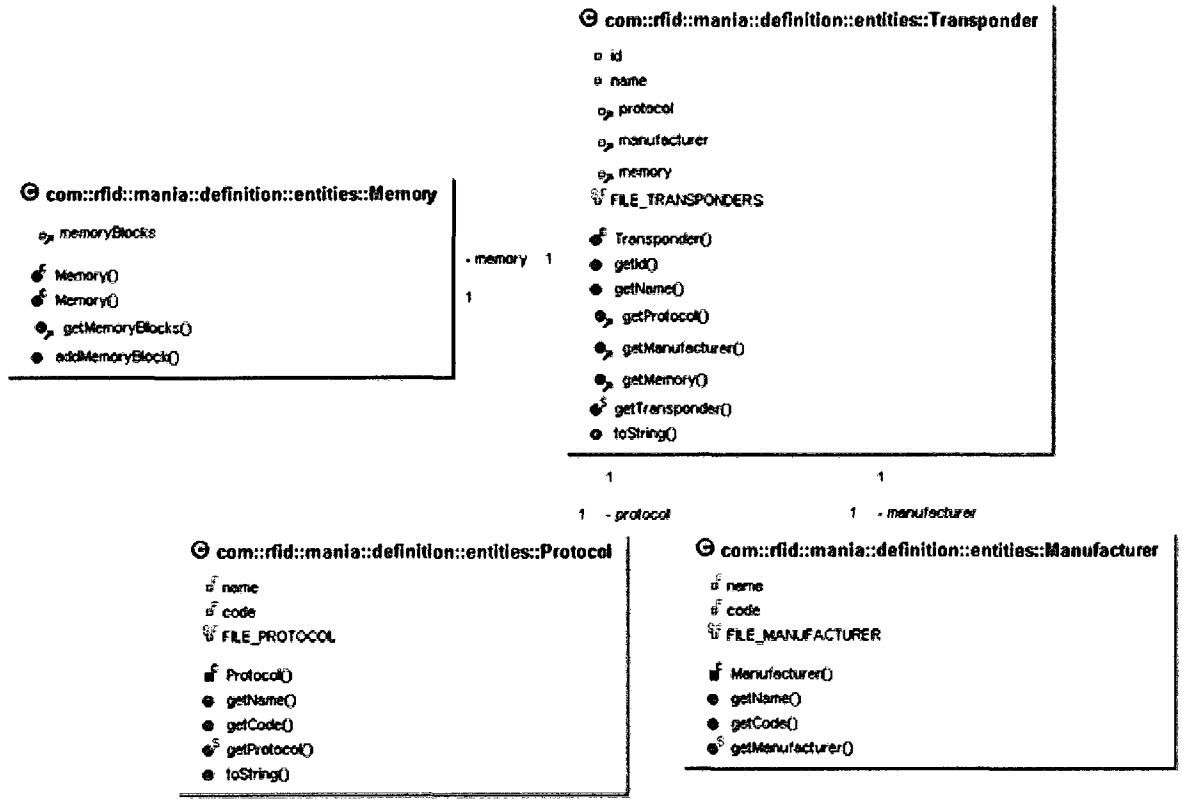

Figure 3.10: UML Class Diagram of the Transponder class and its dependencies

Appendix H provides a TCF listing a number of RFID tags/transponders while Appendix G provides the schema for it.

3.6 Reader Configuration Files

Reader Configuration Files (RCFs) play a big role in RFIDMania allowing reader-specific details to be kept in a separate XML configuration file, based on the schema described in Appendix I, which the middleware then uses to communicate with the reader. Those details include everything from specifying the default data transfer protocol to use (*USB*, *Serial*, *TCP* or *UDP*) with the reader to specifying all commands that the reader understands. The

details of each command are also kept in the same RCF, including what parameters the software needs to provide in order to call-up the command, what values are to be sent to the reader and in what format, any checksum calculations that need to be done as part of the command data before it is sent out and any return data that the reader may send back as a response to the command.

An RCF can also specify acceptable reader responses for each command it defines, which RFIDMania then uses to trigger a successful command invocation event. Similarly, the RCF can be used to define any known error codes or messages that the reader may send back if the command fails, which are used to trigger a failure command invocation event. An RCF can be thought of as a device-driver for each reader, except it contains no actual code and is entirely configurable and easy to modify without affecting the application.

RFIDMania specifies a set of commonly used command names that should be defined in all RCFs regardless of the type of reader. The set of command names is available through an enumeration and can also be used by the application to call-up a command. This ensures that all RCFs have at least those commands defined and provides some standardization across RCFs and the code that developers write to call-up commonly used commands. The enumeration contains the following command names: *CheckForTransponder*, *ReadFromTransponder*, *WriteToTransponder*, *KillTransponder*. Figure 3.11 provides a UML representation of this enumeration.

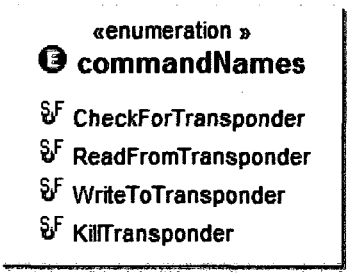

Figure 3.11: Enumeration of commonly used reader command names

An RCF includes a *Features* section that allows a boolean value to be set for each feature that the reader supports or does not support. Examples of features include the ability to read tags, write to tags or *kill* tags. The list of features can differ from one configuration file to the next depending on the number of features that a reader supports or does not support. Features are defined locally on a per-reader basis and therefore it is possible for two readers to have a completely different set of features. Listing 3.6 illustrates the *Features* section for a reader that is configurable, allows resetting, can read data from transponders in its vicinity but can not write-to or *kill* any transponder.

Listing 3.6: Example *Features* section in an RCF

```
1 <Features>
2    <ReaderConfigurations>true</ReaderConfigurations>
3    <ReaderReset>true</ReaderReset>
4    <TransponderKill>false</TransponderKill>
5    <TransponderReading>true</TransponderReading>
6    <TransponderWriting>false</TransponderWriting>
7 </Features>
```

Lines 10-17 of the RCF schema in Appendix I correspond to the example in the above listing.

An RCF is also responsible for defining the default data transfer protocol (*USB*, *Serial*, *TCP* and *UDP*), which is to be used when communicating with the reader in special section called *Settings*. The application can override the default data transfer protocol specified in the RCF. This is useful for readers that support multiple data transfer protocols where if the application detects that the reader is unable to communicate over one protocol, it can simply choose another as long as it is defined in the RCF. The settings for each data transfer protocol that the reader supports is also defined in the *Settings* section. For example, the IP address, port number and timeout are specified for the *TCP/IP* transfer protocol and the port number, baud rate, data-bits, stop-bits, parity and timeout are specified for the *Serial* data transfer protocol. Listing 3.7 illustrates the *Settings* section for a reader that supports the *Serial* and *USB* data transfer protocols, with the *Serial* data transfer protocol being its default.

Listing 3.7: Example *Settings* section in an RCF

```
<!-- Settings -->
<Settings default='Serial'>
   <Serial>
      <COMPort>1</COMPort>
      <BaudRate>38400</BaudRate>
      <Timeout>6000</Timeout>
      <DataBits>8</DataBits>
      <StopBits>1</StopBits>
      <Parity>2</Parity>
   </Serial>
   <Usb>
      <ManufacturerCode>0x06C2</ManufacturerCode>
      <ProductCode>0x0031</ProductCode>
      <SerialNumber>63635</SerialNumber>
      <Configuration>0x01</Configuration>
```

```
16        <Interface>0x00</Interface>
17        <AlternativeInterface>0x00</AlternativeInterface>
18        <ResetDeviceOnOpen>false</ResetDeviceOnOpen>
19        <Timeout>3000</Timeout>
20        <CheckForDataArrivalEvery>200</CheckForDataArrivalEvery>
21        <ReceiveTransferMode>interrupt</ReceiveTransferMode>
22        <SendTransferMode>interrupt</SendTransferMode>
23        <InPipeAddress>0x81</InPipeAddress>
24        <OutPipeAddress>0x00</OutPipeAddress>
25        <ReconnectOnTimeout>true</ReconnectOnTimeout>
26    </Usb>
27 </Settings>
```

Lines 21-79 of the RCF schema in Appendix I correspond to the example in the above listing.

3.7 Reader Command Definitions

An RCF can contain an unlimited number of reader command definitions . Each command structure must adhere to specific rules that RFIDMania expects in order to parse out the command parts properly. Failure to follow these rules will result in a user-friendly error message when RFIDMania compares the structure of the RCF to an XSD. A command definition is expected to have four main parts: *Info*, *Settings*, *Commands* and *Errors*. Figure 3.12 provides a schema representation of the main sections in an RCF.

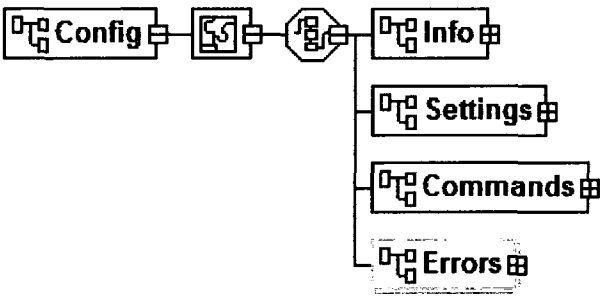

Figure 3.12: Diagram of Reader Configuration File Schema

3.7.1 Startup Commands

Startup commands are pointers to commands in the RCF that are called automatically by RFIDMania when the *connect()* command is issued by the application. This could be used to configure the reader or set it up initially before the program can start sending it commands. By defining Startup Commands at the RCF level, the application does not need to know what commands it needs to send to the reader to configure it and can focus on business requirements as opposed to reader setup requirements. A good example of a Startup Command is one that involves resetting the antenna of the reader or to turn on or off advanced features such as tag collision handling. Listing 3.8 shows an example of the Startup Command section in an RCF.

Listing 3.8: Example *Startup Commands* that reset the reader's CPU and antenna and turns on collision detection on startup

```
1 <!-- Startup commands -->
2 <Startup>
3    <Call-Command>ResetCPU</Call-Command>
4    <Call-Command>ResetAntenna</Call-Command>
5    <Call-Command>TurnOnCollisionDetection</Call-Command>
6 </Startup>
```

Lines 83-90 of the RCF schema in Appendix I correspond to the example in the above listing.

3.7.2 Scheduled Commands

Scheduled Commands are pointers to commands in the RCF that are called automatically by RFIDMania on an interval basis. The interval defines the frequency of executing the Scheduled Commands in milliseconds. Scheduled Commands are especially important for commands that need to repeat for the program to get the latest data from the reader. For instance, some readers will send data out about a new tag that has entered their vicinity only if they receive a request to do so. By allowing RFIDMania to repeatedly send commands to the reader, the application can focus more on business requirements. For instance, the application can send a request *CheckForTransponder* just once and get updated with an event by RFIDMania whenever a new tag is detected by the reader. Listing 3.9 shows an example of the Scheduled Command section in an RCF.

Listing 3.9: Example *Scheduled Commands* that check for new tags every two seconds

```
1 <!-- Scheduled commands -->
2 <Scheduled interval='2000'>
3    <Call-Command>CheckForTransponder</Call-Command>
4 </Scheduled>
```

Lines 99-106 of the RCF schema in Appendix I correspond to the example in the above listing.

3.7.3 Goodbye Commands

Goodbye Commands are similar to Startup and Scheduled Commands in that they are also point to commands defined in the RCF but they are only called when the application disconnects from RFIDMania by calling the *disconnect()* method. Goodbye commands can be used to perform cleanup commands on a reader when the application is done using it. For instance, the application can easily ask the reader to turn off its CPU and antenna to reduce power consumption when not in use. This allows the application to focus on business requirements and not have to worry about writing code to perform cleanup tasks that are specific to the reader they are working with. Listing 3.10 shows an example of the Goodbye Command section in an RCF.

Listing 3.10: Example *Goodbye Commands* that perform turning off the reader's CPU and antenna on exit

```
1 <!-- Goodbye commands -->
2 <Goodbye>
3    <Call-Command>TurnOffCPU</Call-Command>
4    <Call-Command>TurnOffAntenna</Call-Command>
5 </Goodbye>
```

Lines 91-98 of the RCF schema in Appendix I correspond to the example in the above listing.

3.7.4 Command Definitions

Perhaps the most critical section in an RCF file is the *Commands* block because that is where all reader commands are described. The Commands block allows commands to define what data needs to be sent out to the reader to issue the command, what data is to be expected back from the reader as a response to the command and what errors can occur while performing the command. A typical command has five main parts: the *Description* block where a textual explanation of the command is provided for display purposes, the *OnSuccessLog* block where a textual message can be defined in the log whenever the command completes successfully, the *Prerequisites* block where the command requires a set of features - as defined in the *Features* block discussed previously - to be supported by the reader before the command can be issued, the *Send* block where all data that needs to be sent out to the reader to issue the command are defined and the Receive block where all data that the application expects back from the reader are defined.

There are other settings that can also be optionally specified per command, such as if logging should be available for the command or not, if the transponder-detected-event should be triggered when the command completes successfully, if the command should block and not allow other commands to run in parallel with it until it completes or times out and the ability to specify another command that should be called automatically by RFIDMania when the command completes successfully - this is used to chain commands. Figure 3.13 provides a schema representation of the Command block in an RCF. Listing 3.11 shows command named *CheckForTransponder* that has logging enabled, should block until it completes or times-out, triggers a transponder-detected-event and calls up the *TurnOnLed* command if it completes successfully.

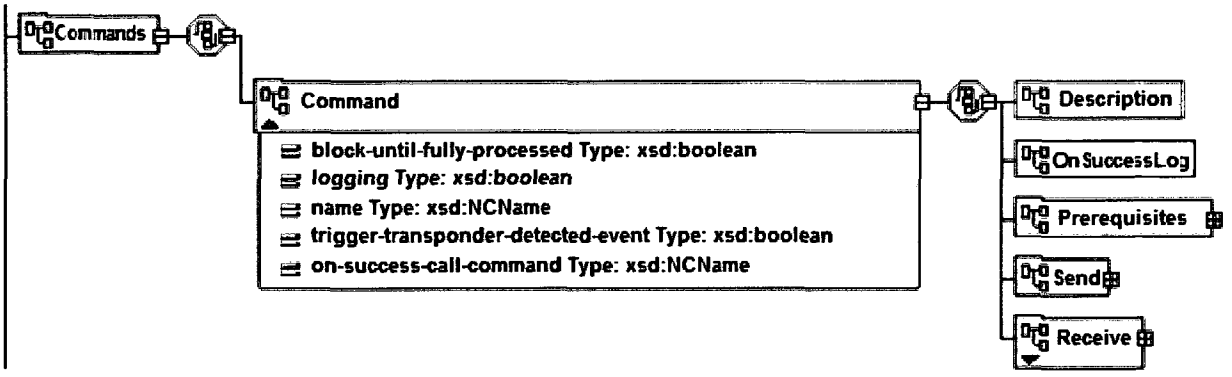

Figure 3.13: Schema diagram of the main parts of the *Command* block

Listing 3.11: Command options

```
1 <Command name='CheckForTransponder' trigger-transponder-detected-event='true' ↩
      logging='true' on-success-call-command='TurnOnLed'>
2     ..
3     <!-- see lines 78-87 in Appendix K for an example Command block body -->
4     ..
5 </Command>
```

Lines 107-226 of the RCF schema in Appendix I correspond to the example in the above listing.

The structure of each command can differ from one command to the next and therefore RFIDMania allows the user to have complete control over structuring of the command. For instance, some commands can be structured such that they send a few bytes to the reader but do not necessarily need a reply back from the reader; a good example of such commands are ones that pertain to resetting the reader's CPU or configuring the reader where the reader does not send any data out. Other command structure differences include

the ability to allow data of any size to come back and not necessarily wait for a specific data size as a valid response from the reader for the command issued. This is necessary where the data that is coming back from the reader depends on other factors, such as the number of tags available in the reader's vicinity and therefore the size of data coming back may not always be the same.

In addition to allowing the user to define the structure of the send and receive data blocks of each command, the user can also specify the data type of each byte it expects to send or receive. For instance, the user can configure a hexadecimal set of bytes to be sent out to the reader to invoke a message or they may wish to define the same set of bytes in ASCII format. Depending on the data type specified, RFIDMania will perform the appropriate byte conversion accordingly before data is sent out and report any errors that may occur during the conversion process. The same applies for data that the command expects to receive, where the type of data can also be specified. In addition to defining data types, the user can use wild characters to specify variability within data coming back. This is useful if a reader sends a different set of bytes each time the same command is issued and the application is only interested in the last byte, for example, and therefore the first set of bytes can be masked. A good example of this is when the application requests the reader to provide the date and time of the last seen tag but is only interested in processing and storing the date portion of the data coming back. Masking out the time portion of the data will achieve this functionality. Listing 3.12 provides an example of a *Receive* block that ignores the first 4 bytes, ensures that the 5th byte equals nine (for the current year) and stores the remaining six bytes in a variable called *$detected_time*.

Listing 3.12: Example *Receive* block with the *Any* data type

```
1  <Data type='any' length='4' />
2  <Data type='hex' value='0x09' />
3  <Data type='to_variable' assign-to='$detected_time' length='6'/>
```

Lines 173-180 of the RCF schema in Appendix I correspond to the example in the above listing.

User-defined variables can also be used by the user to specify input data in the *Receive* block or to store data coming back in the *Send* block. User defined variables are useful for linked commands where the output of one command can be used as input for another. Variables are assigned to data blocks by specifying the *type* of the data block as either a *from-variable* or as a *to-variable* along with the name of the variable from which to read data from or to store data to. The application needs to pass a *VariableList* object when it calls a command so RFIDMania can substitute the necessary input commands with the ones passed in. The *VariableList* object is made up of *Variable* object that contains a name and value pair. RFIDMania uses the name of the variable when it makes the necessary substitutions and therefore all passed in variable names must match exactly what the command expects. Variable names do not have a specific naming convention but must all begin with the dollar-sign ($) character. There are system reserved variable names that are used by RFIDMania to interpret received data as tag data. For instance, all variable names that begin with *$trans_* are used by RFIDMania to makeup a tag object that is passed to the application in the form of an application event when a tag is detected. That is, when RFIDMania sees a data block assigned to a variable by the name $trans_id, it will automatically assign it to the tag object's ID field and when it sees a data type assigned to a

$trans_manufacturer variable name, it will use it to set the tag object's manufacturer field and so on. Listing 3.13 provides an example of two data blocks: the first gets substituted with the value of a variable called $CFG-ADR or the value zero if the variable value is missing and the value of the second data block of length one is assigned to a variable called $data_length.

Listing 3.13: Example of a from-variable and a to-variable data block

```
1  <Data type='from_variable' value="$CFG-ADR" default='0x00' />
2  <Data type='to_variable' assign-to='$data_length' length='1' />
```

Lines 173-180 of the RCF schema in Appendix I correspond to the example in the above listing.

Data blocks can also take on the form of computed values at runtime. For instance, a data block can be assigned the type *COMPUTE_CRC16* to denote that RFIDMania should calculate a CRC16 checksum of all data previous to the position of the byte and insert it into the set of bytes being sent out to the reader to issue a command. Furthermore, a data block can take on the type *COMPUTE_CRC8* for a CRC8 calculation. Similarly, the *CHECK_CRC16* or *CHECK_CRC8* can be used to specify that a calculation must be done on the data that came back and its value must match the specified data block value to determine if the command was run successfully or not. This allows a complete command validation to occur at the RCF level and therefore the application does not need to worry about having to calculate a checksum of data that has come back to determine its validity. Listing 3.14 provides an example of two data blocks: the first computes a CRC16 checksum of all data blocks proceeding it and inserts the value in its position and the second

performs a CRC16 checksum calculation on data received up to that point and compares the value of the calculation with the value of the data bytes received at that position to determine if a command was successful or not.

Listing 3.14: Example of a compute and verify CRC16 checksum data blocks

```
1  <Data type='compute_crc16' />
2  <Data type='check_crc16' length='2' />
```

Lines 173-180 of the RCF schema in Appendix I correspond to the example in the above listing.

A *Receive* block can have a number of optional parameters such as the number of seconds that RFIDMania should wait after sending the bytes defined in the *Send* block before it receives any data from the reader. Other optional parameters include the ability to check for errors based on the data received by assigning an error block name to the receive block. Furthermore, there is the option to not expect any data to come back after issuing the command request - this is important for one-way commands where the reader does not return anything. Lastly, there is an option to allow the user to specify a time-out interval in milliseconds, which RFIDMania should wait before it considers the command to have failed if it does not receive the data it is expecting for the issued command. Figure 3.14 provides a schema representation of the *Receive* block and its optional parameters. Listing 3.15 shows an example of a *Receive* block of a CheckForTransponder command that expects data to come back, pausing for fifty milliseconds to receive data after the command is issued, times out after two seconds if no data came back and traps for all possible errors defined in the CheckForTransponderErrors *Error* block.

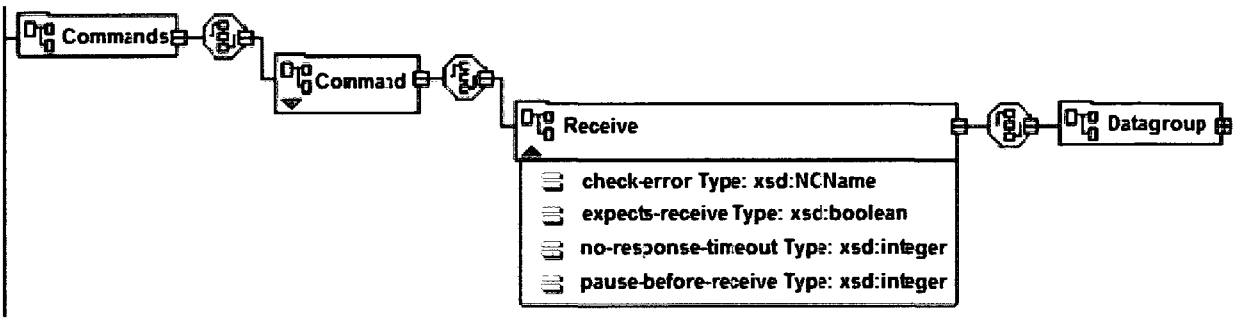

Figure 3.14: Schema diagram of the *Receive* block

Listing 3.15: Example *Receive* block with optional paramaters

```
1 <Receive check-error='CheckForTransponderErrors' expects-receive='true' ↩
      pause-before-receive='50' no-response-timeout='2000'>
2     ..
3     <!-- see lines 67-72 in Appendix K for an example Receive block body -->
4     ..
5 </Receive>
```

Lines 164-215 of the RCF schema in Appendix I correspond to the example in the above listing.

In the *Receive* block, the user can specify what data it expects back from the reader after a successful command run. The expected data needs to be structured into *Datagroups*, which allow the user to specify multiple possible sets of data to be treated as valid data back from the reader. This is necessary when the reader could provide two or more different outcomes to the same command, all of which are valid. Datagroups contain datasets, which group sets of data together as a single large set, which is then used by RFIDMania to compare

expected data to received data to determine if is the command was run successfully or not.

Dataloops are another instruction blocks that inform RFIDMania that all data defined into their body must be repeated. The number of times to repeat the data inside a Dataloop instruction is specified as a constant or can be obtained from a variable. This is especially useful when a reader returns a set of data blocks with information regarding all the tags within its vicinity and specifies the number of tags in one of the bytes. The number of tags byte can then be stored in a variable and used to loop on the remaining data blocks using the Dataloops instruction to capture information for tags. Datasets take on a different form when found inside a Dataloop instruction in that they are treated as choice blocks. For instance, to create a loop that looks to see if the current tag at position i is using the ISO 15693 protocol then parse it one way versus a tag with a different protocol, we would simply create a Dataloop instruction and embed a Dataset block inside of it for each protocol type that the reader supports. Figure 3.15 provides a schema representation of the Dataloop instruction in an RCF. Listing 3.16 shows an example of a Dataloop instruction that loops as many times as the number of detected tags.

Listing 3.16: Example *Dataloop* instruction

```
<Receive>
   <Datagroup>
      <Dataset>
         <Data type='to_variable' assign-to='$number_of_transponders' length='1'/>
      </Dataset>
      <Dataloop type='from_variable' value="$number_of_transponders">
         <!-- ISO 15693 -->
         <Dataset>
            <Data type='to_variable' assign-to='$trans_protcl' length='1' value='0x03'/>
            <Data type='to_variable' assign-to='$trans_DSFID' length='1'/>
            <Data type='to_variable' assign-to='$trans_id' length='8'/>
         </Dataset>
         <!-- ISO 14443A -->
         <Dataset>
            <Data type='to_variable' assign-to='$trans_protcl' length='1' value='0x04'/>
            <Data type='to_variable' assign-to='$trans_TR_INFO' length='1'/>
            <Data type='to_variable' assign-to='$trans_OPT_INFO' length='1'/>
            <Data type='to_variable' assign-to='$trans_id' length='7'/>
         </Dataset>
      </Dataloop>
   </Datagroup>
</Receive>
```

Lines 164-215 of the RCF schema in Appendix I correspond to the example in the above listing.

The *Send* block defined all data that should be sent to the reader in order to issue a command. Similar to the *Receive* block, the *Send* block is made up of *Data* blocks enclosed in any number of *Datasets* within a *Datagroup*. Unlike the *Receive* block, no *Dataloops*

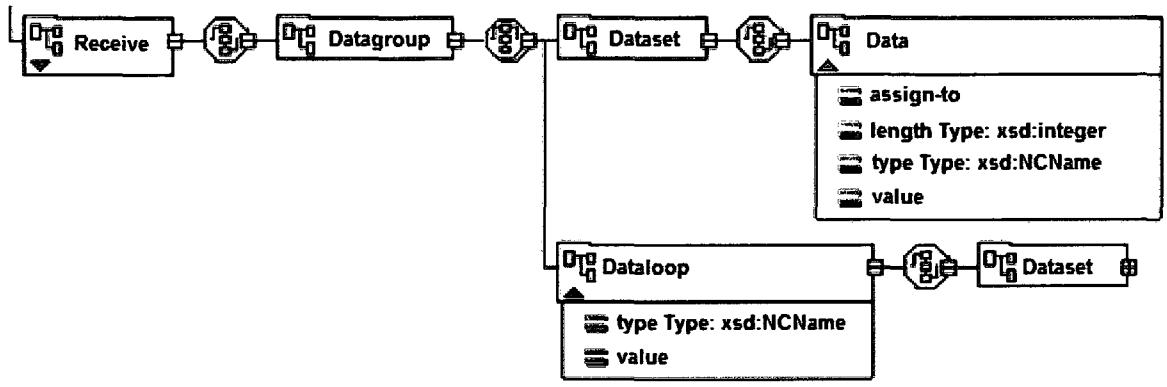

Figure 3.15: Schema diagram of the *Dataloop* instruction

can be defined but the user can make use of *Variables* to make commands more generic. In addition to being able to specify the data that should be sent out to the reader to issue a command, the *Send* block can also specify a *Control Message*, which is a set of bytes that are sent to readers that are connected through a USB interface to setup the device. Figure 3.16 provides a schema representation of the *Send* block in an RCF. Listing 3.17 shows an example of a *Send* block that sends a *Control Message* followed by three hex bytes (0x05, 0xFF and 0x690) and finally a *CRC16* checksum byte value of all proceeding bytes.

Listing 3.17: Example *Send* block with a *Control Message*

```
<Send>
   <ControlMessage BMRequestType='0x21' BRequest='0x09' WValue='0x0200' WIndex='0x00' />
   <Datagroup>
      <Dataset>
         <Data type='hex' value='0x05,0xFF,0x69' />
         <Data type='compute_crc16' />
      </Dataset>
   </Datagroup>
</Send>
```

Lines 129-163 of the RCF schema in Appendix I correspond to the example in the above listing.

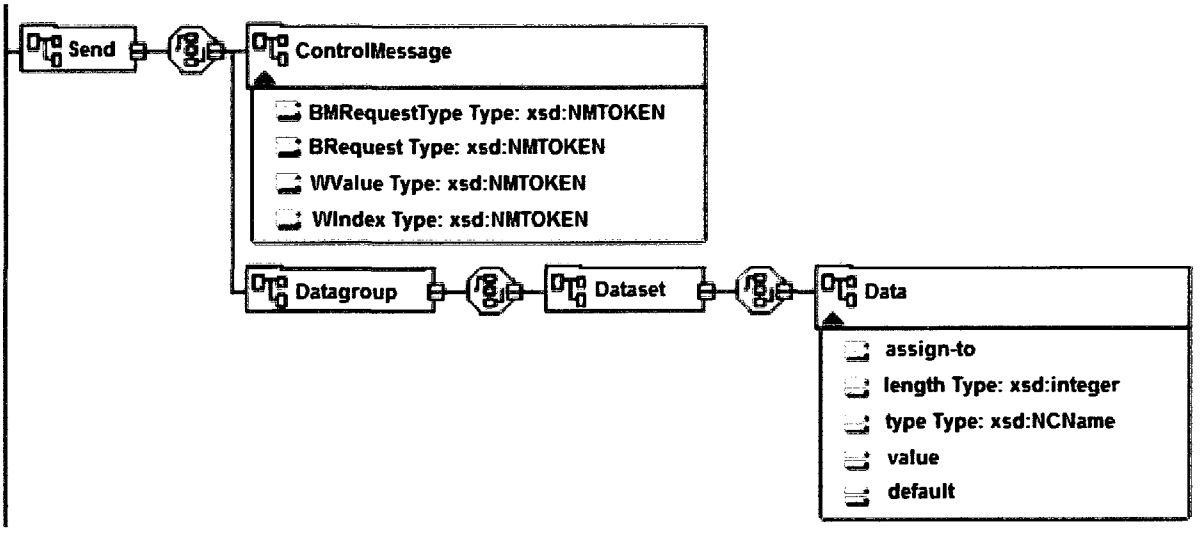

Figure 3.16: Schema diagram of the *Send* block

Finally, the RCF *Errors* section allows the user to define different types of *Error* blocks and group them all with a single name that may then be reference by a command to link it to a set of possible errors that may occur after the command is issued. For example, consider a *CheckForTransponder* command where the reader is requested to provide a list of all tags within its vicinity. The reader may respond with an error when no tags are found or when its antenna is turned off. This means that there are two known errors that the reader may respond back to the *CheckForTransponder* request. The user can then write up two separate error blocks - one for each error - and group them both within an *Errors* block and assign it the name *CheckForTransponderErrors*, which may then be referenced by the

CheckForTransponder command with the *check-error* optional parameter.

Error blocks are similar to both *Receive* and *Send* blocks in that they define the *Data* blocks of the error bytes that the reader may respond back with. Error blocks provide the capability for RFIDMania to translate a set of error bytes to a textual error message that can be provided to the application. By doing so, the application can easily trap for the error message and display it for the user without having to decode the error bytes. Finally, error blocks have an optional parameter that when enabled triggers the notification of the application that issued the command that failed with the error. Not setting this parameter will trigger an application error event that may be useful if the application does not need to be notified when some commands fail. Figure 3.17 provides a schema representation of the *Error* block in an RCF. Listing 3.18 shows an example of an *Error* block that defines two possible errors, which may occur when checking for a new tag.

Listing 3.18: Example *Error* block

```
1  <!-- Check for transponder errors -->
2  <Errors name='CheckForTransponderErrors' trigger-exception-event='true'>
3     <Error>
4        <Description>No transponder found!</Description>
5        <Datagroup>
6           <Dataset>
7              <Data type='hex' value='0x01,0x08,0x00,0x00,0x00,0x00,0x00' />
8           </Dataset>
9        </Datagroup>
10    </Error>
11    <Error>
12       <Description>Antenna Off</Description>
13       <Datagroup>
```

```
14          <Dataset>
15             <Data type='hex' value='0x01,0x01,0x02,0x03,0x04,0x05,0x06' />
16          </Dataset>
17       </Datagroup>
18    </Error>
19 </Errors>
```

Lines 227-260 of the RCF schema in Appendix I correspond to the example in the above listing.

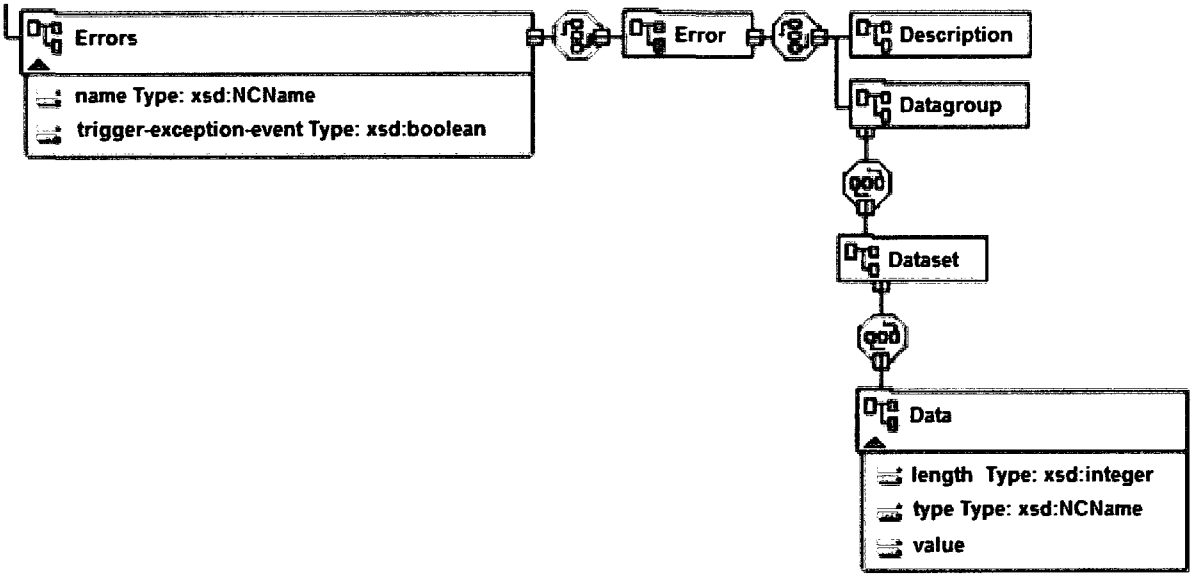

Figure 3.17: Schema diagram of the *Error* block

A complete RCF schema (XSD) is provided in Appendix I. This schema is also used by RFIDMania to validate an RCF before it can be used to interact with a reader. If a RCF fails the validation, the user is presented with an appropriate error message and the RCF needs to be fixed accordingly before it can be used by the middleware.

3.8 RFIDMania API

In order for an application to communicate with the RFIDMania it must make use of the *ISimpleRFID* interface, which declares six important methods - the API - which are implemented by the *Initializer* class of the *Implementation* package. The *verboseLogging()* method allows the application to control the verbosity of the events being logged by the RFIDMania middleware, the *connect()* method is used to establish a connection to a reader using settings defined in the RCF, the *disconnect()* method is used to disconnect an already established connection with a reader, the *monitor()* method is used to inform the middleware that the application is interested in receiving events of tags as they are detected, the *unmonitor()* method, which un-subscribes the application from receiving future tag detection notifications and finally the *sendCommand()* method, which allows the application to issue a command to a reader through the middleware. All of these methods, except for the *disconnect()* method, take in parameters and most throw exceptions using the *RFIDException* class, which the application needs to handle in the event that a method does not complete successfully. Listing 3.19 depicts the RFIDMania API.

Listing 3.19: RFIDMania API

```
1  //global enumeration to declare commonly used command names
2  static enum commandNames
3  {
4     CheckForTransponder,
5     ReadFromTransponder,
6     WriteToTransponder,
7     KillTransponder;
8  }
9
10 /**
```

```
11  * function to enable or disable verbose logging
12  * @param logging
13  */
14 public void verboseLogging(boolean logging);
15
16 /**
17  * connect to the RFID device using the given transmission protocol
18  * @param transmissionProtocol
19  * @throws RFIDException
20  */
21 public void connect(TransmissionProtocol transmissionProtocol) throws RFIDException;
22
23 /**
24  * disconnect from RFID device
25  * @throws RFIDException
26  */
27 public void disconnect() throws RFIDException;
28
29 /**
30  * inform the given transponder monitor object when transponders are detected unless ↩
        the transponder is filtered according to the filter rules
31  * @param transponderMonitor
32  * @param filter
33  */
34 public void monitor(ITransponderMonitor transponderMonitor, Filter filter) throws ↩
      RFIDException;
35
36 /**
37  * stop informing the given transponder monitor object of the detection of transponders
38  * @param transponderMonitor
39  */
40 public void unmonitor(ITransponderMonitor transponderMonitor) throws RFIDException;
41
42 /**
43  * function to send the given command name, block and return the result when done
44  * @param name
```

```
45  * @param variableList
46  * @return CommandResult
47  * @throws RFIDException
48  */
49 public CommandResult sendCommand(String name, VariableList variableList) throws ↩
        RFIDException;
```

3.9 Design Decisions

One of the major decisions taken when designing RFIDMania was to use XML files to store all textual data - such as RFIDMania specific error message - and all data that may differ from one user environment to the next. XML files are easy to manipulate, can be validated against a schema easily and do not force the user to purchase a special reader to edit them. The decision to store all details of each reader in its own separate file meant that the user can have total control over specifying what commands their readers support and what transmission protocol to use with each reader without having to generalize and create a global standard that may work for some readers but not others. With this setup, users can easily extend their RCF to include more commands or to edit exiting details without having to recompile their code as the changes get picked up automatically by the system. XML files are able to provide exactly what is necessary to make it feasible for RFIDMania to scale easily without any change to the core of the middleware and without having to write any additional code.

Other design decisions include using the Java programming language to write RFIDMania, thereby making the middleware very portable to numerous Operating Systems and taking advantage of the object oriented syntax that the Java programming language offers to create

an easy to edit and extend middleware for today's large community of Java developers.

3.10 Summary

In this chapter, a standardization in the form of a specification and schema was introduced for describing RFID protocols/radio-frequencies/manufacturers/tags and readers in configuration files. A middleware reference implementation - called RFIMania - was also introduced to allow software applications to communicate with heterogeneous RFID readers using a common interface through a set of API calls which are also defined in this chapter. The next chapter will validate the approach introduced in this chapter against two different RFID readers.

4 Results

This chapter introduces the results obtained by writing two validation classes that interact with the RFIDMania middleware. This chapter also showcases how the goals and objective, introduced in Sections 1.3 and 1.4, were successfully met.

4.1 Validation

Two RFID readers were chosen to validate the RFIDMania middleware. The first reader is a FEIG ELECTRONIC model OBID i-scan HF MR200 [20]. This reader is a multi-tag reader and has an anti-collision function built-in that allows it to identify transponders of different manufacturers and ISO-standards. The reader is also able to read and write to transponders of various makes as long as they conform to one of the reader's supported protocols, such as the Tag-it and ISO15693 protocols. This reader falls under the *Fixed RFID Readers* category and provides communication access through its Serial and TCP interfaces. The second reader is a PhidgetRFID model 1023 [41] and falls under the category of *Development Kit Readers*. This reader comes in as part of an RFID kit that includes a number of tags of different shapes and sizes. The reader is not capable of writing to any tag but can read data from all the tags that conform to the *EM4102* [18] protocol. Tags must be brought in within 3 inches to the reader to be detected and if more than one tag is present in the reader's vicinity at once it will report none due to its lack of support for collision detection and multi-tag support. The reader has an on-board led, which can be manipulated (on or off) by sending a specific command to the reader. Support for Serial or

TCP interfaces is not available but the reader works on a USB interface. The differences among these readers provide a good validation for the RFIDMania middleware.

The first step in validating the RFIDMania middleware involved creating a Reader Configuration File for each of the readers to demonstrate and evaluate the expressive power of the corresponding specification(s). The FEIG reader comes with its own technical manual [19] that spells out the different commands it supports and all possible exceptions and errors that may occur from having issued those commands. The RCF for the FEIG reader was based solely on this manual. On the other hand, the PhidgetRFID reader had no technical manual available that spelled out the details of each command it supported but came bundled with its own software that controls it. In order to understand the data flow for each command and what data is to be expected to issue the command and what data comes back, USB monitoring software - called USB Monitor Pro [48] - was used to monitor the traffic in and out of the USB port that the reader was connected on. After performing analysis of the data flow, it became apparent how the commands were constructed and the reader's own RCF was created based on that. Appendix J and K provide an RCF for the FEIG and PhidgetRFID readers respectively. Figure 4.1 shows pictures of the FEIG MR200 antenna and the PhidgetRFID 1023 reader.

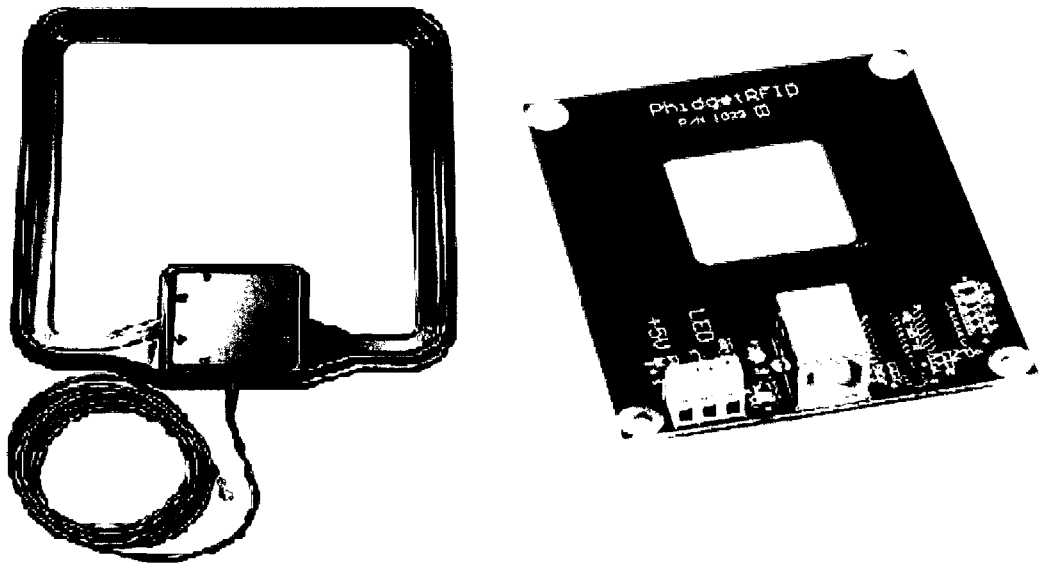

Figure 4.1: FEIG MR200 antenna and PhidgetRFID 1023 reader

The Reader Configuration Files defined in Appendix J and K show that the RCF schema specification, outlined in chapter 3, was sophisticated enough to allow for the expression of reader-commands of varying total byte size, structure and data types. For instance, the FEIG RCF outline in Appendix J made use of all possible data organization blocks such as *DataSets* and *DataLoops* with a variety of data types such as *hex* and *compute_crc16*. For the FEIG reader, a handful of commands were chosen from its manual to be implemented in the RCF; namely *Radio Frequency Reset (RFReset)* command, *CPU Reset (CPUReset)* command, *System Reset (SystemReset)* command and finally the *Check For Transponder (CheckForTransponder)* command. Similarly, for the PhidgetRFID reader, the *Check For Transponder (CheckForTransponder)* command was implemented along with the *Turn On Led (TurnOnLed)* command, which is triggered automatically when the *CheckForTransponder* command succeeds.

The next step involved writing a validation class for each of the readers to perform commands as defined in the RCFs. Both validation classes issue the standard *CheckForTransponder* command while the FEIG validation class issues a few more commands, such as the *CPU Reset* and *SystemReset* commands, which are not supported by the PhidgetRFID reader. The validation classes were written to take advantage of the *RFIDValidation* class that RFIDMania provides to make validation easy and provides the user with an interactive GUI, as shown in Section 3.1 - Figure 3.7 with logging capabilities. Listings 4.1 and 4.2 provide the code for the FEIG and PhidgetRFID validation classes respectively.

Listing 4.1: FEIG validation code using *RFIDValidation* class

```
1  //FEIG MR 200 validation class
2  public class FeigMR200Validation extends RFIDValidation {
3
4     public FeigMR200Validation()
5     {
6        //instantiate RFIDValidation using the 'FiegMR200.xml' Reader Configuration File
7        //allow validation of all commands defined in the RCF
8        //uses the default data transfer protocol defined in the RCF
9        super("FEIG MR200", "FiegMR200.xml", true, null);
10    }
11
12    //start validation
13    public static void main(String args[])
14    { new FeigMR200Validation(); }
15 }
```

The FEIG validation class extends the *RFIDValidation* class - described in Section 3.1 - on line 2 and initializes it on line 9 with 'FEIG MR200' as the validation application GUI title and the path of where the FEIG MR200. The *FIDValidation* automatically retrieves a list

of all commands defined in the 'FEIG MR200' RCF and adds them to its GUI drop-down list (as depicted in Figure 3.7 in Chapter 3). When a user selects a command from the GUI drop-down list, the *RFIDValidation* class takes care of sending the selected command to the reader and outputs the outcome of the invocation process in its log. Error messages are provided in red and information messages are provided in blue in the log.

Listing 4.2: PhidgetRFID validation code using *RFIDValidation* class

```
1  //PhidgetRFID 1023 validation class
2  public class PhidgetRFID1023Validation extends RFIDValidation{
3
4     public PhidgetRFID1023Validation()
5     {
6        //instantiate RFIDValidation using the 'PhidgetRFID1023.xml' Reader Configuration ↩
             File
7        //allow validation of all commands defined in the RCF
8        //uses the default data transfer protocol defined in the RCF
9        super("Phidget 1023", "PhidgetRFID1023.xml", true, null);
10
11       //setup filter
12       setupFilter();
13    }
14
15    //class methods
16    private void setupFilter()
17    {
18       Filter filter = new Filter();
19       filter.addRule(new Rule("1B001428BA00","Exactly: 1B001428BA00", ↩
             TransponderProperty.ID.getVariableName(), Operator.EQUALS, "1B001428BA00"));
20       filter.addRule(new Rule("1600790*","Start with: 1600790", ↩
             TransponderProperty.ID.getVariableName(), Operator.STARTS_WITH, "1600790"));
21       filter.addRule(new Rule("*BC88DC00","End with: BC88DC00", ↩
             TransponderProperty.ID.getVariableName(), Operator.ENDS_WITH, "BC88DC00"));
22
```

```
23          //set filter to this filter
24          setFilter(filter);
25      }
26
27      //start validation
28      public static void main(String args[])
29      { new PhidgetRFID1023Validation(); }
30  }
```

Similar to the FEIG validation class, the Phidget validation class also extends the *RFIDValidation* class - described in Section 3.1 - on line 2 and initializes it on line 9 with 'Phidget 1023' as the validation application GUI title and the path of where the PhidgetRFID 1023 RCF is stored. The class then sets up filters on line 12 that should be used by the middleware to filter out data before it arrives at the validation class, thereby eliminating data that the validation class is not interested in. The filter for this validation class includes three rules that are described on lines 19, 20 and 21 respectively. The first rule, on line 19, tells RFIDMania that it is not interested in receiving any application events for tags whose id matches '1B001428BA00'. The second rule, on line 20, tells RFIDMania that it is not interested in receiving any application events for tags whose id starts with '1600790'. The third and last rule, on line 21, tells RFIDMania that it is not interested in receiving any application events for tags whose id ends with 'BC88DC00'.

For the purpose of showing all events generated in the middleware by the validation application, the *verbose logging* option is implicitly enabled in the *RFIDValidation* class. The PhidgetRFID validation class defines a number of tags to filter out based on different criteria as shown in the *setupFilter()* method in Listing 4.2 lines 16-25. It is important to note

that the same validation classes could have easily been written to interact directly with the RFIDMania middleware API, described in Section 3.8, and not make use of the *RFIDValidation* class. Listings 4.3 and 4.4 show the same validation classes re-written without making use of the *RFIDValidation* class, which is how a typical application would be coded to interact directly with RFIDMania using its API.

For the purpose of displaying results in an easy-to-read format - e.g., color-coded logs differentiating errors from success messages - the first set of validation classes are used to display all output in Appendix M. The FEIG validation class depicted in Listing 4.3 differs from the FEIG validation class in Listing 4.1 in that the developer needs to implement the *ITransponderMonitor* interface - line 2 - and therefore must implement the *transponderIdentified()* - lines 85-87, *triggerCommandExceptionEvent()* - lines 90-92 - and the *describe()* - lines 95-97 - methods. In addition, the developer would need to write code to send commands to their reader (as shown on lines 21, 30, 39 and 48) using the *sendCommand()* API call described in Section 3.8. In addition, the developer would need to write code to subscribe and un-subscribe its application list from receiving notifications when a command fails or when a tag is detected by using the *monitor()* and *unmonitor()* API call as shown on lines 15 and 64 respectively. Lastly, the developer would also be responsible for writing code to connect to their RFID reader using the *connect()* command as shown on line 18 and *disconnect()* from their reader as shown on line 67. By extending the *RFIDValidation* class as in Listings 4.1 and 4.2 , the developer benefits from the shortcut that the *RFIDValidation* class provides in allowing quick means to test out reader commands - through an interactive GUI - against one or more readers simply by providing the path to a RCF.

Listing 4.3: FEIG validation class

```
1  //FEIG MR 200 validation class
2  public class FeigMR200Validation implements ITransponderMonitor, IDebugger {
3
4     //method to run validations
5     public void runValidations()
6     {
7        ISimpleRFID initializer = new Initializer("FiegMR200.xml", this);
8
9        try {
10
11          //enable verbose logging
12          initializer.verboseLogging(true);
13
14          //add monitor
15          initializer.monitor(this, new Filter());
16
17          //connect using default data transfer protocol
18          initializer.connect(null);
19
20          //send CPU reset command
21          CommandResult CPUResetCommand = initializer.sendCommand("CPUReset", null);
22
23          print("\n\nCPU Reset Command Result Log:\n" + CPUResetCommand.getLog(), ↩
                FeigMR200Validation.INFORMATION_COLOR);
24
25          //if failure
26          if (!CPUResetCommand.isSuccessful())
27             print("\n\nCPU Reset Command Result Error:\n" + ↩
                   CPUResetCommand.getErrDescription(), FeigMR200Validation.ERROR_COLOR);
28
29          //send system reset command
30          CommandResult systemResetCommand = initializer.sendCommand("SystemReset", null);
31
32          print("\n\nSystem Reset Command Result Log:\n" + systemResetCommand.getLog(), ↩
                FeigMR200Validation.INFORMATION_COLOR);
```

```
33
34          //if failure
35          if (!systemResetCommand.isSuccessful())
36             print("\n\nSystem Reset Command Result Error:\n" + ↩
                   systemResetCommand.getErrDescription(), FeigMR200Validation.ERROR_COLOR);
37
38          //send RF Reset command
39          CommandResult rfReset = initializer.sendCommand("RFReset", null);
40
41          print("\n\nRF Reset Command Result Log:\n" + rfReset.getLog(), ↩
                FeigMR200Validation.INFORMATION_COLOR);
42
43          //if failure
44          if (!rfReset.isSuccessful())
45             print("\n\nRF Reset Command Result Error:\n" + rfReset.getErrDescription(), ↩
                   FeigMR200Validation.ERROR_COLOR);
46
47          //send Check For Transponder command
48          CommandResult checkForTagCommand = ↩
                initializer.sendCommand(ISimpleRFID.commandNames.
49                CheckForTransponder.toString(), null);
50
51          print("\n\nCheck For Transponder Command Result Log:\n" + ↩
                checkForTagCommand.getLog(), FeigMR200Validation.INFORMATION_COLOR);
52
53          //if failure
54          if (!checkForTagCommand.isSuccessful())
55             print("\n\nCheck For Transponder Command Result Error:\n" + ↩
                   checkForTagCommand.getErrDescription(), FeigMR200Validation.ERROR_COLOR);
56
57       } catch (RFIDException e) {
58          print("Command Error: " + e.toString(), FeigMR200Validation.ERROR_COLOR);
59       }
60       finally {
61          try{
62
```

```
63              //remove monitor
64              initializer.unmonitor(this);
65
66              //disconnect
67              initializer.disconnect();
68          }
69          catch (RFIDException e) {
70              print("Error: " + e.toString(), FeigMR200Validation.ERROR_COLOR);
71          }
72      }
73  }
74
75  //start validation
76  public static void main(String args[])
77  {
78      FeigMR200Validation feigMR200Validation = new FeigMR200Validation();
79
80      //run validations
81      feigMR200Validation.runValidations();
82  }
83
84  //method invoked by RFIDMania when one or more transponders/tags are detected
85  public void transponderIdentified(ITransponderList transponderList) {
86      System.out.println("Application received transponder(s): " + ↩
            transponderList.toString());
87  }
88
89  //method invoked by RFIDMania when a command fails
90  public void triggerCommandExceptionEvent(String commandName, String errorMsg) {
91      System.out.println("Application received an error for command " +  commandName + ↩
            ": " + errorMsg);
92  }
93
94  //method to describe validation class
95  public String describe() {
96      return ("FEIG MR200 Validations Class");
```

```
97      }
98
99      //method to print a message to console
100     public void print(String msg, Color color) {
101
102        //if it's an error message
103        if (color.equals(PhidgetRFID1023Validation.ERROR_COLOR))
104           System.out.println("ERROR: " + msg);
105        else if (color.equals(IDebugger.INFORMATION_COLOR))  //if it's an information ↩
               message
106           System.out.println("INFO: " + msg);
107        else if (color.equals(IDebugger.NOTIFICATION_COLOR )) //if it's a notification ↩
               message
108           System.out.println("NOTIF: " + msg);
109     }
110 }
```

Unlike the first FEIG validation class, this FEIG validation class does not extend the *RFID-Validation* class. Instead, it implements the *ITransponderMonitor* class - as depicted in Section 3.1, Figure 3.5 - to receive notifications from RFIDMania when one or more tags are detected or when a command fails. Furthermore, it also implements the *IDebugger* class - as depicted in Section 3.1, Figure 3.5 - which forces it to declare the *print()* to print notifications and error messages to the console, as shown on line 100-109. The *run-Validations()* method is called from the *main()* method to start the validation process. On line 7, the validation application instantiates a copy of the *Initializer* class which implements the *ISimpleRFID* interface - as depicted in Section 3.1, Figure 3.5 and specifies where the FEIG MR200 RCF is located. Verbosity logging is then set on line 12. Line 15 makes an API with the *monitor()* method to setup this validation class to start receiving

notification events when a tag is detected or when a command fails. On Line 18 the validation class requests RFIDMania to establish a connection with the FEIG MR200 RFID reader using the default data transfer protocol defined in the FEIG MR200 RCF. Line 21 asks RFIDMania to send the CPURest command to the FEIG MR200 reader. The outcome of processing the command is then printed to the console on line 23. Lastly, a check is made to test if the *CPUReset* command had failed and if it did then the command's error message is printed on the screen on line 27. The same is done for the *SystemReset*, *RFReset* and *CheckForTransponder* commands on lines 30-36, 39-45, 48-55 respectively. All command exceptions are caught on line 57 and the appropriate error messages are printed to the console on line 58. Upon processing the previously mentioned commands, a call is made to halt receiving notifications from RFIDMania using the *unmonitor()* API call on line 64 and to disconnect RFIDMania from the FEIG reader using the *disconnect()* API call on line 67. The *transponderIdenfitied()* and *triggerCommandExceptionEvent()* methods are implemented by the validation application on lines 85 and 90 to display to the console any detected tags or command error messages on lines 86 and 91 respectively.

Listing 4.4: PhidgetRFID validation class

```
1  //PhidgetRFID 1023 validation class
2  public class PhidgetRFID1023Validation implements ITransponderMonitor, IDebugger {
3
4     //method to run validations
5     public void runValidations()
6     {
7        ISimpleRFID initializer = new Initializer("PhidgetRFID1023.xml", this);
8
9        try {
```

```
10
11          //enable verbose logging
12          initializer.verboseLogging(true);
13
14          //setup filter
15          Filter filter = new Filter();
16      filter.addRule(new Rule("1B001428BA00","Exactly: 1B001428BA00", ↩
            TransponderProperty.ID.getVariableName(), Operator.EQUALS, "1B001428BA00"));
17      filter.addRule(new Rule("1600790*","Start with: 1600790", ↩
            TransponderProperty.ID.getVariableName(), Operator.STARTS_WITH, "1600790"));
18      filter.addRule(new Rule("*BC88DC00","End with: BC88DC00", ↩
            TransponderProperty.ID.getVariableName(), Operator.ENDS_WITH, "BC88DC00"));
19
20          //add monitor
21          initializer.monitor(this, filter);
22
23          //connect using default data transfer protocol
24          initializer.connect(null);
25
26          //call check for transponder command
27          CommandResult checkForTransponder = ↩
                initializer.sendCommand(ISimpleRFID.commandNames.
28                CheckForTransponder.toString(), null);
29
30          print("\n\nCheckForTransponder Command Result Log:\n" + ↩
                checkForTransponder.getLog(), PhidgetRFID1023Validation.INFORMATION_COLOR);
31
32          //if failure
33          if (!checkForTransponder.isSuccessful())
34             print("\n\nCheckForTransponder Command Result Error:\n" + ↩
                   checkForTransponder.getErrDescription(), ↩
                   PhidgetRFID1023Validation.ERROR_COLOR);
35
36      } catch (RFIDException e) {
37          print("Command Error: " + e.toString(), PhidgetRFID1023Validation.ERROR_COLOR);
38      }
```

```
39      finally {
40          try{
41              //remove monitor
42              initializer.unmonitor(this);
43
44              //disconnect
45              initializer.disconnect();
46          }
47          catch (RFIDException e) {
48              print("Error: " + e.toString(), PhidgetRFID1023Validation.ERROR_COLOR);
49          }
50      }
51  }
52
53  //start validation
54  public static void main(String args[])
55  {
56      PhidgetRFID1023Validation phidgetRFID1023Validation = new ↩
            PhidgetRFID1023Validation();
57
58      //run validations
59      phidgetRFID1023Validation.runValidations();
60  }
61
62  //method invoked by RFIDMania when one or more transponders/tags are detetced
63  public void transponderIdentified(ITransponderList transponderList) {
64      System.out.println("Application received transponder(s): " + ↩
            transponderList.toString());
65  }
66
67  //method invoked by RFIDMania when a command fails
68  public void triggerCommandExceptionEvent(String commandName, String errorMsg) {
69      System.out.println("Application received an error for command " +  commandName + ↩
            ": " + errorMsg);
70  }
71
```

```
72      //method to describe the validation class
73      public String describe() {
74          return ("PhidgetRFID 1023 Validation Class");
75      }
76
77      //method to print a message to console
78      public void print(String msg, Color color) {
79
80          //if it's an error message
81          if (color.equals(PhidgetRFID1023Validation.ERROR_COLOR))
82              System.out.println("ERROR: " + msg);
83          else if (color.equals(IDebugger.INFORMATION_COLOR))  //if it's an information ↩
                message
84              System.out.println("INFO: " + msg);
85          else if (color.equals(IDebugger.NOTIFICATION_COLOR )) //if it's a notification ↩
                message
86              System.out.println("NOTIF: " + msg);
87      }
88  }
```

To validate the detection capabilities of the middleware, a number of tags were brought in close to the readers. Some of those tags were filtered out by the middleware as specified by the PhidgetRIFD validation application, as described in Listing 4.4 - lines 15-18. The validation classes shown above confirm that the RFIDMania middleware was successful in encapsulating all data transfer protocol details and reader command details. The application was responsible for issuing the command by name and fetching a *CommandResult* object to obtain results for the command invoked. The *CommandResult* object was able to provide the application with an indicator to determine whether the command was issued successfully or not. Furthermore, the *CommandResult* object provided means for the application to determine if there were any error messages and to examine the log produced

by issuing each command. The logging property of the *CommandResult* object was used by the validation applications to log errors to the console for further analysis.

The validation applications received all detected tags and errors in the form of an application event and displayed the tag information on the console. The default data transfer protocol specified in the FEIG RCF was changed from *Serial* to *TCP* and the same exact results were reproduced without any code changes or the need to recompile any of the validation applications. Furthermore, the validation applications were run side-by-side to confirm that the middleware had support for multiple concurrent reader and did not run into any conflicts dismantling the data flowing in and out of the readers. Finally, the *EM4102* and *Tag-it* tags that were brought in close to the readers were randomly picked and the results showed that the middleware was able to handle as many tags detected at once as each reader had support for. Listings 18.1 and 18.2 in Appendix M show the output of running the *CPUReset* - lines 4-17, *SystemRest* - lines 19-32, *RFReset* - lines 34-47, *CheckForTransponder* - lines 49-229 - commands on the FEIG validation application and the *CheckForTransponder* - lines 4-209 - command on the PhidgetRFID validation application respectively.

Finally, to confirm that the specification schema is able to suit other types of readers, a different class of RFID readers by Intermic - called the IF61 Enterprise Reader [31] - was examined and a Reader Configuration File was created for two of its commands: the *Check For Transponder (CheckForTransponder)* and *Boot Reset (BootReset)* commands. The Intermec IF61 reader is a "powerful combination of reader and network appliance for running

RFID applications" [31]. It has support for filtering out unwanted tags, storing and manipulating tag information and can be queried using a custom communication protocol called the Basic Reader Interface (BRI) [30]. Appendix L shows the RCF for the Intermic IF61 reader for the two commands mentioned above. The RCF was built solely on the details provided by the manual [30] and no actual validations were performed on the reader because it was not physically available.

4.2 Summary

The results in Appendix M successfully validated that the RFIDMania middleware was able to provide the application with a generic interface and a set of API calls, described in Section 3.8, to communicate with two complete different RFID readers. The code remained free of RFID reader-specific details, such as how to parse the data coming down from the reader or how to decode the data properly and was completely focused on meeting the requirements of the validation application. Additionally, all communication details with both readers were hidden by the RFIDMania middleware from the application, which would allow the same application to communicate with the same reader through different data transfer protocols (USB, Serial, TCP or UDP) without having to change or recompile any code. The results in Appendix M also show that the details of each command that the application issued against both readers were kept in the Reader Configuration Files and not intertwined with the application code itself, thus keeping a clear separation between code that serves to address business requirements and code that manages the details of each command.

5 Conclusion

This thesis is set out to provide a middleware and schema specification for describing RFID protocols, radio-frequencies, manufacturers, tags and readers. The RFIDMania middleware is built to interpret user generated configuration files based on the specifications and schemas outlined in Appendices A, C, E, G and I. The middleware, specifications and schemas work together to encapsulate all communication details with an RFID reader from the application layer, provide data processing capabilities and present the application with application events instead of raw data and allow developers to describe their RFID readers in XML-based configuration files.

The Reader Configuration File specification is created based on researching the different reader communication protocols in the market. The expressive structure of the Reader Configuration File specification and schema make it possible to describe readers of different makes and hardware components alike and allow complex reader protocols to be expressed in an easy to read format. The Reader Configuration File allows for complete description of a reader's features, what the reader supports in terms of commands, how the commands are formed in terms of expected input and output bytes, what errors should be handled and for what commands, what data transfer protocols are supported by the reader and what protocol to use by default if the application does not specify one.

The RFIDMania middleware reference implementation, described in Section 3.1, is built to provide consistency across heterogeneous readers in that it provides a single set of API

calls that applications can use to access their readers. The API, described in Section 3.8, is extensible in terms of being able to describe commands that heterogeneous readers can understand and provide command name standardization for the most commonly supported commands by today's readers, such as the ability to read from a tag, to write to it, to disable it and finally to get an inventory list of tags in a reader's vicinity. In addition, the RFIDMania middleware make use of RCFs, described in Section 3.6, to determine how it should communicate with readers and what commands a reader is able to understand and in what data format. It also makes use of RCFs to render data coming back from readers to determine its validity and to check for appropriate errors and inform the application accordingly. Unlike existing middleware solutions, applications do not need to provide a programmed solution for their readers in order to gain access to them with RFIDMania. This alone offers an advantage over existing solutions in terms of consistency and extensibility as well as allowing anyone with basic XML skills to write reader drivers in an RCF format.

The Reader Configuration File specification and schema, described in section 3.6 and Appendix I, offer an advantage over existing solutions in that the user does not need to write code to express the details of their readers as it can all be defined in an XML format. A RCF acts much like a device-driver to the RFIDMania middleware where new readers can be configured and made available instantly to the application by simply specifying a new RCF path in an application's code. An application benefits a great deal from the RFIDMania middleware as it provides an encapsulates layer from all details of data translation and transmission. Lastly, an application is able to work with detected tags or transponders as objects and therefore does not need to know the details of how to manipulate or parse the raw tag data to capture its different parts, such as the manufacturer name and tag ID.

In conclusion, the *Results* Chapter show that the schema, described in the Appendix I, is generic enough to allow the details of various readers to be implemented in Reader Configuration Files. Moreover, the RFIDMania middleware is able to perform all data translations, data encoding and decoding - based on the details specified in RCFs - and notify the application in the form of an application event when a command error or a tag is detected. By doing so, the application code keep complete focus on meeting business requirements and business logic and is not intertwined with RFID reader-specific command details. Lastly, all communication details is taken care of by the RFIDMania middleware and the application is not required to implement any data transfer protocols to send and receive data from readers.

5.1 Review of Goals and Contributions

In this thesis, we have shown that the RFIDMania middleware is able to encapsulate all communication details with an RFID reader from the application layer, allow users to define all commands understood by their RFID readers in a Reader Configuration Files, provide users with a set of APIs that they can use to communicate with their RFID readers, provide implementation of basic data transfer protocols such as *Serial*, *USB*, *TCP* and *UDP*, provide data processing capabilities and finally present the application with application events instead of raw data. We believe that these abilities demonstrate that we have reached our goal and have met our contributions successfully.

5.2 Future Work

Given more time, the Bluetooth and GPRS data transfer protocols would have been implemented. It is important to note that none of the readers used to validate the middleware with had support for the UDP protocol and therefore it remains untested. While RFIDMania offers great portability due to the nature of the Java language in which it is written, the USB package [46] that it makes use of is Windows based only and would have to be replaced with a similar package for the USB component in RFIDMania to work with other operating systems. Compatibility with EPCglobal's EPC Network standards is not currently available and would give RFIDMania a greater boost over existing solutions if implemented. It is important to note that reader commands that write to tags can also be specified in RCFs and are fully supported by RFIDMania. However, due to time constraints such commands have not been validated and are left for future work.

Because XML-based configuration files - such as the Reader Configuration File - can get very lengthy, it makes sense to create a GUI-based application that allows users to create their RCF easily without having to type everything up. This means that RCFs can automatically be generated by software given that the software knows about the user's business requirements and the details of the reader. This enhancement is not currently available and is left for future work.

References

[1] a1-rfid.com. *RFID Readers and the different types for all purposes.* 7 July 2009 <http://www.a1-rfid.com/rfid-readers.htm>

[2] APSX LLC. *RW-110 RFID Reader.* 11 June 2009 <http://www.apsx.com/RW110.aspx?gclid=CNyTgOmnoZwCFQxM5QodwzaNfg>

[3] Ashwin. "RFID - The Future." *Article Alley* (Aug. 2007) 21 June 2009 <http://www.articlealley.com/article_198294_11.html>.

[4] Aspire. *AspireRFID.* 12 Dec. 2005 <http://wiki.aspire.ow2.org/xwiki/bin/view/Main>

[5] Atwell, Allen. "Capturing, Managing and Using RFID Information." *RFID and Telecommunication Services Workshop* (May 2004) 6 May 2009 <http://portal.etsi.org/docbox/ERM/open/RFIDWorkshop/RFID_12AllenAtwell_Oracle.ppt>.

[6] Barcoding Inc. *IF5 Fixed "Smart" Reader.* 8 Aug. 2009 <http://www.barcoding.com/rfid/choosing_rfid_reader.shtml#if5>

[7] ——. *Intermec F4IF4 Fixed Serial Reader.* 8 Aug. 2009 <http://www.barcoding.com/rfid/choosing_rfid_reader.shtml#if4>

[8] ——. *Intermec PM4i.* 4 Aug. 2009 <http://www.barcoding.com/common/intermec/intermec_PM4i.shtml>

[9] ——. *RFID Reader Options by Application.* 18 July 2009 <http://www.barcoding.com/rfid/rfid_reader_options_by_application.shtml>

[10] ——. *Symbol DC600 RFID Portal.* 18 July 2009 <http://www.barcoding.com/common/symbol/symbol_DC600.shtml>

[11] ——. *Symbol MC 9090-G RFID Reader.* 1 Aug. 2009 <http://www.barcoding.com/common/symbol/symbol_mc9000G_RFID.shtml>

[12] ——. *Symbol RD5000 Mobile RFID Reader.* 1 Aug. 2009 <http://www.barcoding.com/common/symbol/symbol_RD5000.shtml>

[13] ——. *Symbol XR440 RFID Reader.* 8 Aug. 2009 <http://www.barcoding.com/common/symbol/symbol_XR440.shtml>

[14] B.S. Prabhu, Xiaoyong Su and Harish Ramamurthy. "WinRFID A Middleware for the enablement of Radio Frequency Identification (RFID) based Applications." PhD thesis. University of California, 2005.

[15] Burkett, Bobby. "RFID Software Basics." *Dynasys Technologies* (2004) 16 Sept. 2009 <http://rfidusa.com/superstore/pdf/RFID_Software_Basics.pdf>.

[16] Burnell, John. "What Is RFID Middleware and Where Is It Needed?" (Aug. 2006) 2 Aug. 2009 <http://www.rfidupdate.com/articles/index.php?id=1176>.

[17] Cervantes, Louie F., et al. "A Hybrid Middleware for RFID-based Parking Management System Using Group Communication in Overlay Networks." *IPC '07: Proceedings of the The 2007 International Conference on Intelligent Pervasive Computing*, 26 Feb. 2009. Washington, DC, USA: IEEE Computer Society, 2007. 521–526.

[18] EM Microelectronic-Marin SA. "EM4102." *Read Only Contactless Identification Device* (Feb. 2005): 1–7 13 16 2009 <http://www.emmicroelectronic.com/webfiles/Product/RFID/DS/EM4102_DS.pdf>.

[19] FEIG Electronics. *FEIG ID CPR-Family Manual* (29 Mar. 2006). 8 Aug. 2009 <http://www.ing.unibs.it/~sardini/Lucidi_SILSIS/Modulo20RFID/H20800-10e-ID-B.pdf>

[20] ——. *ID ISC.MR200-A /-E.* 8 Aug. 2009 <http://www.feig.de/index.php?option=com_content&task=view&id=101&Itemid=126>

[21] Fosstrak.org. *Fosstrak HAL - FEIG Configuration* (20 Oct. 2008). 16 July 2009 <http://www.fosstrak.org/hal/docs/user-feig.html>

[22] GAO. "Understanding RFID - a thorough understanding of RFID technology and standards is crucial to fully realizing its benefits." *RFID Design* (Sept. 2007) 15 Sept. 2009 <http://rfdesign.com/mag/709RFDEf2.pdf>.

[23] GS1 Australia, Impetus 2006 Conference, 19 June 2009. Melbourne, Australia 2006.

[24] GS1 EPCglobal. "Specification for RFID Air Interface." *EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID Protocol for Communications at 60 MHz 960 MHz* (2005): 1–94 17 May 2009 <http://www.epcglobalinc.org/standards/uhfc1g2/uhfc1g2_1_0_9-standard-20050126.pdf>.

[25] GS1.org. *EPCGlobal* (10 June 2009). 5 May 2009 <http://www.epcglobalinc.org>

[26] ——. *International Organization for Standardization.* 15 Sept. 2009 <http://www.iso.org>

[27] Gupta, Alka and Mayank Srivastava. "Developing Auto-ID Solutions using Sun Java System RFID Software." *Sun Developer Network (SDN)* (Oct. 2004) 9 May 2009 <http://java.sun.com/developer/technicalArticles/Ecommerce/rfid/sjsrfid/RFID.html>.

[28] Hahn, Rebecca and Daylan Burlison. "Oracle Delivers Enhanced RFID Technology in Oracle^{R} Fusion Middleware." *Oracle Press Release* (May 2006) 24 May 2009 <http://www.oracle.com/corporate/press/2006_may/sensor-edge-server-10g-release3.html>.

[29] Inc, SkyRFID. *RFID UHF Fixed Readers — RFID UHF Handheld Readers.* 5 June 2009 <http://www.skyrfid.com/Readers_UHF_860960.php>

[30] Intermec Technologies Corp. *Basic Reader Interface (BRI) - Programmer's Reference Manual* (July 2009). 16 July 2009 <http://epsfiles.intermec.com/eps_files/eps_man/937-000.pdf>

[31] ——. *IF61 Enterprise Reader.* 3 Aug. 2009 <http://www.intermec.com/products/rfidif61a>

[32] ISO.org. "ISO/IEC 15693." *Identification cards - Contactless integrated circuit cards - Vicinity cards - Part 2: Air interface and initialization* (Dec. 2006): 1–14 1 May 2009 <http://wwsw.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39695>.

[33] J. Al-Jaroodi, Junaid Aziz and N. Mohamed, The 33rd Annual IEEE International Computer Software and Applications Conference, 12 May 2009. Seattle, Washington, USA: IEEE Computer Society Press, 2009.

[34] Kabir, Ashad, et al., eds., Dynamics in Logistics: First International Conference, LDIC 2007, Bremen, Germany, August 2007, Proceedings, 8 June 2009. Busan, Korea: Springer Berlin Heidelberg, 2008.

[35] Mak, Andy, Anthony Lam, and Daiming Qu. "CUHK RFID Middleware - System Design Document." *The Chinese University of Hong Kong* (Aug. 2007) 15 Sept. 2009 <http://mobitec.ie.cuhk.edu.hk/rfid/middleware/doc/Middleware_SDD_v1.0.pdf>.

[36] Microsoft Corp. *Microsoft BizTalk RFID Server*. 5 May 2009 <http://www.microsoft.com/biztalk/en/us/rfid.aspx>

[37] MobiTec - Mobile Technologies Centre. *CUHK RFID Middleware 1.0*. 5 June 2009 <http://mobitec.ie.cuhk.edu.hk/rfid/middleware/project.htm>

[38] Motorola Inc. *RD5000 Mobile RFID Reader*. 7 Feb. 2009 <http://www.multisystems.com/new/pdf_lib/Datasheet_RD5000_EN.pdf>

[39] Nurminen, Timo. "The End of RFID Middleware?" *RFID Journal* (Jan. 2006) 18 May 2009 <http://www.rfidjournal.com/article/view/2035>.

[40] Oracle Inc. *Oracle Sensor Edge Services: Technical Presentation*. 22 June 2009 <http://www.oracle.com/technology/products/sensor_edge_server/collateral/Oracle_SES_1013_Technical_Presentation.pdf>

[41] Phidget Inc. *1023 - PhidgetRFID* (8 Aug. 2009). 9 Aug. 2009 <http://www.phidgets.com/products.php?category=14&product_id=1023>

[42] Philips Semiconductors. "I-CODE1 Label IC." *Data sheet* (May 2000): 1–20 3 Aug. 2009 <http://www.nxp.com/acrobat_download/datasheets/SL1ICS3001_2.pdf>.

[43] Power 7 Technology. *Internet RFID Security Mouse.* 15 Apr. 2009 <http://power7.en.alibaba.com/product/51031269-50181782/RFID_Internet_Security_Mouse.html>

[44] Preradovic, Stevan and Nemai C. "Modern RFID Readers." *Microwave Journal* (Sept. 2007) 7 Jan. 2009 <http://www.mwjournal.com/article.asp?HH_ID=AR_4830>.

[45] Roberti, Mark. "Auto-ID Lab Releases Accada RFID Prototyping Platform." *RFID Journal* (June 2007) 12 July 2009 <http://www.rfidjournal.com/article/articleview/3405>.

[46] Sourceforge.net. *Java libusb/libusb-win32 wrapper.* 5 July 2009 <http://sourceforge.net/projects/libusbjava>

[47] Texas Instruments. "Tag-it." *Transponder Protocol Reference Manual* (Mar. 2000): 1–55 2 Aug. 2009 <http://www.ti.com/rfid/docs/manuals/refmanuals/tag-it_transponder_protocol.pdf>.

[48] USB Monitor Pro. *USB Analyzer, USB Capture* (21 Aug. 2009). 6 Feb. 2009 <http://www.usb-monitor.com>

[49] Wikipedia. *EPCglobal.* 1 Mar. 2008 <http://en.wikipedia.org/wiki/EPCglobal>

Appendix A

Listing 6.1: XSD Schema of Protocol Configuration File (PCF)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
 <xsd:element name='Protocols'>
  <xsd:complexType>
   <xsd:sequence maxOccurs='unbounded'>
  <xsd:element name="Protocol" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:element name="Code" type="xsd:hexBinary" minOccurs="1" maxOccurs="1"/>
     <xsd:element name="Name" type="xsd:string" maxOccurs="1" minOccurs="1"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Appendix B

Listing 7.1: Protocol Configuration File

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- RFID Protocols -->
3  <Protocols>
4
5    <Protocol>
6      <Code>0x00</Code>
7      <Name>I-Code1</Name>
8    </Protocol>
9
10   <Protocol>
11     <Code>0x01</Code>
12     <Name>Tag-it</Name>
13   </Protocol>
14
15   <Protocol>
16     <Code>0x03</Code>
17     <Name>ISO15693</Name>
18   </Protocol>
19
20   <Protocol>
21     <Code>0x04</Code>
22     <Name>ISO14443A</Name>
23   </Protocol>
24
25   <Protocol>
26     <Code>0x05</Code>
27     <Name>ISO14443B</Name>
28   </Protocol>
29
```

```
<Protocol>
  <Code>0x06</Code>
  <Name>ICodeEPC</Name>
</Protocol>

<Protocol>
  <Code>0x07</Code>
  <Name>ICodeUID</Name>
</Protocol>

<Protocol>
  <Code>0x08</Code>
  <Name>InnovisionJewel</Name>
</Protocol>

<Protocol>
  <Code>0x80</Code>
  <Name>ISO18000_6A</Name>
</Protocol>

<Protocol>
  <Code>0x81</Code>
  <Name>ISO18000_6B</Name>
</Protocol>

<Protocol>
  <Code>0x83</Code>
  <Name>EM4222</Name>
</Protocol>

<Protocol>
  <Code>0x84</Code>
  <Name>EPCclass1Gen2</Name>
</Protocol>

<Protocol>
```

```
66      <Code>0x88</Code>
67      <Name>EPCclass0Gen1</Name>
68    </Protocol>
69
70    <Protocol>
71      <Code>0x89</Code>
72      <Name>EPCclass1Gen1</Name>
73    </Protocol>
74
75    <Protocol>
76      <Code>0xff</Code>
77      <Name>OTHER</Name>
78    </Protocol>
79
80  </Protocols>
```

Appendix C

Listing 8.1: XSD Schema of Radio Frequency Configuration File (PFCF)

```
1  <?xml version="1.0"?>
2  <xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
3   <xsd:element name='RadioFrequencies'>
4    <xsd:complexType>
5     <xsd:sequence maxOccurs='unbounded'>
6    <xsd:element name="RadioFrequency" minOccurs="0" maxOccurs="unbounded">
7     <xsd:complexType>
8      <xsd:sequence>
9       <xsd:element name="Code" type="xsd:hexBinary" maxOccurs="1" minOccurs="1"/>
10      <xsd:element name="Name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
11      <xsd:element name="Description" type="xsd:string" minOccurs="1" maxOccurs="1"/>
12     </xsd:sequence>
13    </xsd:complexType>
14   </xsd:element>
15    </xsd:sequence>
16   </xsd:complexType>
17  </xsd:element>
18 </xsd:schema>
```

Appendix D

Listing 9.1: Radio Frequency Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- RFID Radio Frequencies -->
<RadioFrequencies>

  <RadioFrequency>
    <Code>0x0</Code>
    <Name>HF</Name>
    <Description>High frequency</Description>
  </RadioFrequency>

  <RadioFrequency>
    <Code>0x1</Code>
    <Name>LF</Name>
    <Description>Low frequency</Description>
  </RadioFrequency>

  <RadioFrequency>
    <Code>0x2</Code>
    <Name>UHF</Name>
    <Description>Ultra-high frequency</Description>
  </RadioFrequency>

  <RadioFrequency>
    <Code>0x3</Code>
    <Name>UNKNOWN</Name>
    <Description>Unknown frequency</Description>
  </RadioFrequency>

</RadioFrequencies>
```

Appendix E

Listing 10.1: XSD Schema of Manufacturer Configuration File (MCF)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
 <xsd:element name='Manufacturers'>
  <xsd:complexType>
   <xsd:sequence maxOccurs='unbounded'>
  <xsd:element name="Manufacturer" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
    <xsd:sequence>
  <xsd:element name="Code" type="xsd:hexBinary" minOccurs="1" maxOccurs="1"/>
     <xsd:element name="Name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
   </xsd:complexType>
  </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

Appendix F

Listing 11.1: Manufacturer Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Transponder Manufacturers -->
<Manufacturers>

  <Manufacturer>
    <Code>0x01</Code>
    <Name>Motorola</Name>
  </Manufacturer>

  <Manufacturer>
    <Code>0x02</Code>
    <Name>ST Microelectronics</Name>
  </Manufacturer>

  <Manufacturer>
    <Code>0x03</Code>
    <Name>Hitachi</Name>
  </Manufacturer>

  <Manufacturer>
    <Code>0x04</Code>
    <Name>Philips Semiconductors</Name>
  </Manufacturer>

  <Manufacturer>
    <Code>0x05</Code>
    <Name>Infineon</Name>
  </Manufacturer>

```

```
<Manufacturer>
  <Code>0x07</Code>
  <Name>Texas Instruments</Name>
</Manufacturer>

<Manufacturer>
  <Code>0x06</Code>
  <Name>Cylinc</Name>
</Manufacturer>

<Manufacturer>
  <Code>0x08</Code>
  <Name>Fujitsu Limited</Name>
</Manufacturer>

<Manufacturer>
  <Code>0x09</Code>
  <Name>Matsushita Electric Industrial</Name>
</Manufacturer>

<Manufacturer>
  <Code>0x0A</Code>
  <Name>NEC</Name>
</Manufacturer>

<Manufacturer>
  <Code>0x0B</Code>
  <Name>Oki Electric</Name>
</Manufacturer>

<Manufacturer>
  <Code>0x0C</Code>
  <Name>Toshiba</Name>
</Manufacturer>

<Manufacturer>
```

```
    <Code>0x0D</Code>
    <Name>Mitsubishi Electric</Name>
  </Manufacturer>

  <Manufacturer>
    <Code>0x0E</Code>
    <Name>Samsung Electronics</Name>
  </Manufacturer>

  <Manufacturer>
    <Code>0x0F</Code>
    <Name>Hyundai Electronics</Name>
  </Manufacturer>

  <Manufacturer>
    <Code>0x10</Code>
    <Name>LG Semiconductors</Name>
  </Manufacturer>

  <Manufacturer>
    <Code>0x16</Code>
    <Name>EMarin Microelectronic</Name>
  </Manufacturer>

  <Manufacturer>
    <Code>0x17</Code>
    <Name>KSW</Name>
  </Manufacturer>

  <Manufacturer>
    <Code>0xff</Code>
    <Name>Innovision Jewel</Name>
  </Manufacturer>

  <Manufacturer>
    <Code>0xff</Code>
```

```
102      <Name>Unknown</Name>
103    </Manufacturer>
104
105 </Manufacturers>
```

Appendix G

Listing 12.1: XSD Schema of Transponder Configuration File (TCF)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd='http://www.w3.org/2001/XMLSchema'>
 <xsd:element name='Transponders'>
  <xsd:complexType>
   <xsd:sequence maxOccurs='unbounded'>
  <xsd:element name="Transponder" minOccurs="0" maxOccurs="unbounded">
   <xsd:complexType>
    <xsd:sequence>
     <xsd:sequence maxOccurs="unbounded">
      <xsd:element name="Name" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     </xsd:sequence>
     <xsd:element name="Protocol" type="xsd:string" minOccurs="1" maxOccurs="1"/>
     <xsd:element name="Manufacturer" type="xsd:string" minOccurs="1" maxOccurs="1"/>
   <xsd:element name="Memory" minOccurs="0" maxOccurs="1">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="Size" type="xsd:integer" minOccurs="1" maxOccurs="1"/>
       <xsd:element name="BytesPerBlock" type="xsd:integer" maxOccurs="1" minOccurs="1"/>
    <xsd:element name="User" minOccurs="1" maxOccurs="1">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="Read" type="xsd:boolean" minOccurs="1" maxOccurs="1"/>
       <xsd:element name="Write" type="xsd:boolean" minOccurs="1" maxOccurs="1"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
```

```
30      </xsd:sequence>
31     </xsd:complexType>
32    </xsd:element>
33     </xsd:sequence>
34    </xsd:complexType>
35   </xsd:element>
36 </xsd:schema>
```

Appendix H

Listing 13.1: Transponder Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- RFID Transponders -->
<Transponders>

  <Transponder>
    <Name>MB89R116</Name>
    <Name>MB89R118</Name>
    <Protocol>ISO15693</Protocol>
    <Manufacturer>Fujitsu Limited</Manufacturer>
    <Memory>
      <Size>256</Size>
      <BytesPerBlock>16</BytesPerBlock>
      <User>
        <Read>true</Read>
        <Write>true</Write>
      </User>
    </Memory>
  </Transponder>

  <Transponder>
    <Name>SRF55V10P</Name>
    <Protocol>ISO15693</Protocol>
    <Manufacturer>Infineon</Manufacturer>
    <Memory>
      <Size>128</Size>
      <BytesPerBlock>8</BytesPerBlock>
      <User>
        <Read>true</Read>
        <Write>true</Write>
```

```
      </User>
    </Memory>
  </Transponder>

  <Transponder>
    <Name>SRF55V02P</Name>
    <Protocol>ISO15693</Protocol>
    <Manufacturer>Infineon</Manufacturer>
    <Memory>
      <Size>1</Size>
      <BytesPerBlock>8</BytesPerBlock>
      <User>
        <Read>true</Read>
        <Write>true</Write>
      </User>
    </Memory>
  </Transponder>

  <Transponder>
    <Name>ICodeSLI</Name>
    <Protocol>ISO15693</Protocol>
    <Manufacturer>Philips Semiconductors</Manufacturer>
    <Memory>
      <Size>32</Size>
      <BytesPerBlock>4</BytesPerBlock>
      <User>
        <Read>true</Read>
        <Write>true</Write>
      </User>
    </Memory>
  </Transponder>

  <Transponder>
    <Name>LRI512</Name>
    <Protocol>ISO15693</Protocol>
    <Manufacturer>ST Microelectronics</Manufacturer>
```

```
    <Memory>
      <Size>16</Size>
      <BytesPerBlock>4</BytesPerBlock>
      <User>
        <Read>true</Read>
        <Write>true</Write>
      </User>
    </Memory>
  </Transponder>

  <Transponder>
    <Name>Tag-it</Name>
    <Protocol>Tag-it</Protocol>
    <Manufacturer>Texas Instruments</Manufacturer>
    <Memory>
      <Size>64</Size>
      <BytesPerBlock>4</BytesPerBlock>
      <User>
        <Read>true</Read>
        <Write>true</Write>
      </User>
    </Memory>
  </Transponder>

  <Transponder>
    <Name>SLE55R04</Name>
    <Protocol>ISO14443</Protocol>
    <Manufacturer>Infineon</Manufacturer>
    <Memory>
      <Size>82</Size>
      <BytesPerBlock>10</BytesPerBlock>
      <User>
        <Read>true</Read>
        <Write>true</Write>
      </User>
    </Memory>
```

```
102   </Transponder>
103
104   <Transponder>
105     <Name>SLE55R08</Name>
106     <Protocol>ISO14443</Protocol>
107     <Manufacturer>Infineon</Manufacturer>
108     <Memory>
109       <Size>133</Size>
110       <BytesPerBlock>10</BytesPerBlock>
111       <User>
112         <Read>true</Read>
113         <Write>true</Write>
114       </User>
115     </Memory>
116   </Transponder>
117
118   <Transponder>
119     <Name>SLE55R16</Name>
120     <Protocol>ISO14443</Protocol>
121     <Manufacturer>Infineon</Manufacturer>
122     <Memory>
123       <Size>261</Size>
124       <BytesPerBlock>10</BytesPerBlock>
125       <User>
126         <Read>true</Read>
127         <Write>true</Write>
128       </User>
129     </Memory>
130   </Transponder>
131
132   <Transponder>
133     <Name>MF1ICS50</Name>
134     <Protocol>ISO14443</Protocol>
135     <Manufacturer>Philips Semiconductors</Manufacturer>
136     <Memory>
137       <Size>47</Size>
```

```
      <BytesPerBlock>16</BytesPerBlock>
      <User>
        <Read>true</Read>
        <Write>true</Write>
      </User>
    </Memory>
  </Transponder>

  <Transponder>
    <Name>MF1ICS70</Name>
    <Protocol>ISO14443</Protocol>
    <Manufacturer>Philips Semiconductors</Manufacturer>
    <Memory>
      <Size>256</Size>
      <BytesPerBlock>16</BytesPerBlock>
      <User>
        <Read>true</Read>
        <Write>true</Write>
      </User>
    </Memory>
  </Transponder>

  <Transponder>
    <Name>MifareUltraLight</Name>
    <Protocol>ISO14443</Protocol>
    <Manufacturer>Philips Semiconductors</Manufacturer>
    <Memory>
      <Size>16</Size>
      <BytesPerBlock>4</BytesPerBlock>
      <User>
        <Read>true</Read>
        <Write>true</Write>
      </User>
    </Memory>
  </Transponder>

```

```
174   <Transponder>
175     <Name>IRT5001W</Name>
176     <Name>IRT5001E</Name>
177     <Protocol>InnovisionJewel</Protocol>
178     <Manufacturer>Innovision Jewel</Manufacturer>
179     <Memory>
180       <Size>120</Size>
181       <BytesPerBlock>1</BytesPerBlock>
182       <User>
183         <Read>true</Read>
184         <Write>true</Write>
185       </User>
186     </Memory>
187   </Transponder>
188
189   <!-- Unknown Transporter -->
190   <Transponder>
191     <Name>Unknown</Name>
192     <Name>Unknown</Name>
193     <Protocol>Unknown</Protocol>
194     <Manufacturer>Unknown</Manufacturer>
195     <Memory>
196       <Size>0</Size>
197       <BytesPerBlock>0</BytesPerBlock>
198       <User>
199         <Read>false</Read>
200         <Write>false</Write>
201       </User>
202     </Memory>
203   </Transponder>
204
205 </Transponders>
```

Appendix I

Listing 14.1: XSD Schema of Reader Configuration File (RCF)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="Config">
  <xsd:complexType mixed="false">
   <xsd:sequence>
    <xsd:element name="Info" minOccurs="1" maxOccurs="1">
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name="Name" type="xsd:string"/>
       <xsd:element name="Features" minOccurs="1" maxOccurs="1">
        <xsd:complexType>
         <xsd:sequence>
          <xsd:any maxOccurs="unbounded" minOccurs="0" processContents="skip">
          </xsd:any>
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name="Settings" minOccurs="1" maxOccurs="1">
     <xsd:complexType>
      <xsd:sequence maxOccurs="1">
       <xsd:element name="Serial" minOccurs="0" maxOccurs="1">
        <xsd:complexType>
         <xsd:sequence>
          <xsd:element name="COMPort" type="xsd:integer" minOccurs="1" maxOccurs="1"/>
          <xsd:element name="BaudRate" type="xsd:integer" minOccurs="1" maxOccurs="1"/> <!-- bits/sec -->
          <xsd:element name="Timeout" type="xsd:integer" minOccurs="1" maxOccurs="1"/> <!-- millisecond -->
          <xsd:element name="DataBits" type="xsd:integer" minOccurs="1" maxOccurs="1"/> <!-- bits -->
          <xsd:element name="StopBits" type="xsd:integer" minOccurs="1" maxOccurs="1"/> <!-- bits -->
          <xsd:element name="Parity" type="xsd:integer" minOccurs="1" maxOccurs="1"/> <!-- bit -->
         </xsd:sequence>
        </xsd:complexType>
       </xsd:element>
       <xsd:element name="Usb" minOccurs="0" maxOccurs="1">
        <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="ManufacturerCode" type="xsd:NMTOKEN" minOccurs="1" maxOccurs="1"/>
            <xsd:element name="ProductCode" type="xsd:NMTOKEN" minOccurs="1" maxOccurs="1"/>
```

```
41            <xsd:element name="SerialNumber" type="xsd:integer" minOccurs="1" maxOccurs="1"/>
42            <xsd:element name="Configuration" type="xsd:NMTOKEN" minOccurs="1" maxOccurs="1"/>
43            <xsd:element name="Interface" type="xsd:NMTOKEN" minOccurs="1" maxOccurs="1"/>
44            <xsd:element name="AlternativeInterface" type="xsd:NMTOKEN" minOccurs="1" maxOccurs="1"/>
45            <xsd:element name="ResetDeviceOnOpen" type="xsd:boolean" minOccurs="1" maxOccurs="1"/>
46            <xsd:element name="Timeout" type="xsd:integer" minOccurs="1" maxOccurs="1"/> <!-- millisecond -->
47            <xsd:element name="CheckForDataArrivalEvery" type="xsd:integer" minOccurs="1" maxOccurs="1"/> <!-- ↩
                  millisecond -->
48            <xsd:element name="ReceiveTransferMode" type="xsd:NCName" minOccurs="1" maxOccurs="1"/>
49            <xsd:element name="SendTransferMode" type="xsd:NCName" minOccurs="1" maxOccurs="1"/>
50            <xsd:element name="InPipeAddress" type="xsd:NMTOKEN" minOccurs="1" maxOccurs="1"/>
51            <xsd:element name="OutPipeAddress" type="xsd:NMTOKEN" minOccurs="1" maxOccurs="1"/>
52            <xsd:element name="ReconnectOnTimeout" type="xsd:boolean" minOccurs="1" maxOccurs="1"/> <!-- millisecond -->
53          </xsd:sequence>
54         </xsd:complexType>
55        </xsd:element>
56        <xsd:element name="TCP" minOccurs="0" maxOccurs="1">
57         <xsd:complexType>
58          <xsd:sequence>
59            <xsd:element name="IPAddress" type="xsd:NMTOKEN" minOccurs="1" maxOccurs="1"/>
60            <xsd:element name="PortNumber" type="xsd:integer" minOccurs="1" maxOccurs="1"/>
61            <xsd:element name="Timeout" type="xsd:integer" minOccurs="1" maxOccurs="1"/> <!-- millisecond -->
62            <xsd:element name="CheckForDataArrivalEvery" type="xsd:integer" minOccurs="1" maxOccurs="1"/> <!-- ↩
                  millisecond -->
63          </xsd:sequence>
64         </xsd:complexType>
65        </xsd:element>
66        <xsd:element name="UDP" minOccurs="0" maxOccurs="1">
67         <xsd:complexType>
68          <xsd:sequence>
69            <xsd:element name="IPAddress" type="xsd:NMTOKEN" minOccurs="1" maxOccurs="1"/>
70            <xsd:element name="PortNumber" type="xsd:integer" minOccurs="1" maxOccurs="1"/>
71            <xsd:element name="Timeout" type="xsd:integer" minOccurs="1" maxOccurs="1"/> <!-- millisecond -->
72            <xsd:element name="CheckForDataArrivalEvery" type="xsd:integer" minOccurs="1" maxOccurs="1"/> <!-- ↩
                  millisecond -->
73          </xsd:sequence>
74         </xsd:complexType>
75        </xsd:element>
76       </xsd:sequence>
77       <xsd:attribute name="default" use="required" type="xsd:NCName"/>
78      </xsd:complexType>
79     </xsd:element>
80     <xsd:element name="Commands" minOccurs="1" maxOccurs="1">
81      <xsd:complexType>
82       <xsd:sequence>
83        <xsd:element name="Startup" minOccurs="0" maxOccurs="1">
84         <xsd:complexType>
85          <xsd:sequence>
```

```
86            <xsd:element name="Call-Command" type="xsd:NCName" minOccurs="0" maxOccurs="unbounded"/>
87           </xsd:sequence>
88           <xsd:attribute name="interval" use="optional" type="xsd:integer"/>
89          </xsd:complexType>
90         </xsd:element>
91         <xsd:element name="Goodbye" minOccurs="0" maxOccurs="1">
92          <xsd:complexType>
93           <xsd:sequence>
94            <xsd:element name="Call-Command" type="xsd:NCName" minOccurs="0" maxOccurs="unbounded"/>
95           </xsd:sequence>
96           <xsd:attribute name="interval" use="optional" type="xsd:integer"/>
97          </xsd:complexType>
98         </xsd:element>
99         <xsd:element name="Scheduled" minOccurs="0" maxOccurs="1">
100         <xsd:complexType>
101          <xsd:sequence>
102           <xsd:element name="Call-Command" type="xsd:NCName" minOccurs="0" maxOccurs="unbounded"/>
103          </xsd:sequence>
104          <xsd:attribute name="interval" use="optional" type="xsd:integer"/>
105         </xsd:complexType>
106        </xsd:element>
107        <xsd:element name="Command" maxOccurs="unbounded">
108         <xsd:complexType>
109          <xsd:sequence>
110           <xsd:element name="Description" type="xsd:string" minOccurs="1" maxOccurs="1"/>
111           <xsd:element name="OnSuccessLog" type="xsd:string" minOccurs="1" maxOccurs="1"/>
112           <xsd:element name="Prerequisites" minOccurs="0" maxOccurs="1">
113            <xsd:complexType>
114             <xsd:sequence>
115              <xsd:element name="Features" minOccurs="1" maxOccurs="1">
116               <xsd:complexType>
117                <xsd:sequence>
118                 <xsd:element minOccurs="1" name="Feature" maxOccurs="unbounded">
119                  <xsd:complexType>
120                   <xsd:attribute name="name" type="xsd:NCName" use="required"/>
121                  </xsd:complexType>
122                 </xsd:element>
123                </xsd:sequence>
124               </xsd:complexType>
125              </xsd:element>
126             </xsd:sequence>
127            </xsd:complexType>
128           </xsd:element>
129           <xsd:element name="Send" minOccurs="0">
130            <xsd:complexType>
131             <xsd:sequence>
132              <xsd:element name="ControlMessage" minOccurs="0">
133                  <xsd:complexType>
```

```
                    <xsd:attribute name="BMRequestType" type="xsd:NMTOKEN" use="required"/>
                <xsd:attribute name="BRequest" type="xsd:NMTOKEN" use="required"/>
                <xsd:attribute name="WValue" type="xsd:NMTOKEN" use="required"/>
                <xsd:attribute name="WIndex" type="xsd:NMTOKEN" use="required" />
                  </xsd:complexType>
               </xsd:element>
               <xsd:element name="Datagroup" minOccurs="1" maxOccurs="1">
                <xsd:complexType>
                 <xsd:sequence minOccurs="1" maxOccurs="1">
                  <xsd:element name="Dataset" minOccurs="0" maxOccurs="unbounded">
                   <xsd:complexType>
                    <xsd:sequence>
                     <xsd:element name="Data" minOccurs="1" maxOccurs="unbounded">
                      <xsd:complexType>
                       <xsd:attribute name="assign-to"/>
                       <xsd:attribute name="length" type="xsd:integer"/>
                       <xsd:attribute name="type" use="required" type="xsd:NCName"/>
                       <xsd:attribute name="value"/>
                       <xsd:attribute name="default"/>
                      </xsd:complexType>
                     </xsd:element>
                    </xsd:sequence>
                   </xsd:complexType>
                  </xsd:element>
                 </xsd:sequence>
                </xsd:complexType>
               </xsd:element>
              </xsd:sequence>
             </xsd:complexType>
            </xsd:element>
            <xsd:element name="Receive" minOccurs="0" maxOccurs="1">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="Datagroup" maxOccurs="unbounded" minOccurs="1">
                            <xsd:complexType>
                                <xsd:choice maxOccurs="unbounded">
                                    <xsd:element name="Dataset" minOccurs="0" maxOccurs="unbounded">
                                        <xsd:complexType>
                                            <xsd:sequence>
                                                <xsd:element name="Data" maxOccurs="unbounded" minOccurs="1">
                                                    <xsd:complexType>
                                                        <xsd:attribute name="assign-to"/>
                                                        <xsd:attribute name="length" type="xsd:integer"/>
                                                        <xsd:attribute name="type" use="required" type="xsd:NCName"/>
                                                        <xsd:attribute name="value"/>
                                                    </xsd:complexType>
                                                </xsd:element>
                                            </xsd:sequence>
```

```
                                    </xsd:complexType>
                                </xsd:element>
                                <xsd:element name="Dataloop" minOccurs="0" maxOccurs="unbounded">
                                    <xsd:complexType>
                                        <xsd:sequence>
                                            <xsd:element name="Dataset" minOccurs="1" maxOccurs="unbounded">
                                                <xsd:complexType>
                                                    <xsd:sequence>
                                                        <xsd:element name="Data" minOccurs="1" maxOccurs="unbounded">
                                                            <xsd:complexType>
                                                                <xsd:attribute name="assign-to"/>
                                                                <xsd:attribute name="length" type="xsd:integer"/>
                                                                <xsd:attribute name="type" use="required" ↩
                                                                    type="xsd:NCName"/>
                                                                <xsd:attribute name="value"/>
                                                            </xsd:complexType>
                                                        </xsd:element>
                                                    </xsd:sequence>
                                                </xsd:complexType>
                                            </xsd:element>
                                        </xsd:sequence>
                                        <xsd:attribute name="type" use="required" type="xsd:NCName"/>
                                        <xsd:attribute name="value" use="required"/>
                                    </xsd:complexType>
                                </xsd:element>
                            </xsd:choice>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
                <xsd:attribute name="check-error" use="required" type="xsd:NCName"/>
                <xsd:attribute name="expects-receive" use="required" type="xsd:boolean"/>
                <xsd:attribute name="nc-response-timeout" use="required" type="xsd:integer"/>
                <xsd:attribute name="pause-before-receive" use="required" type="xsd:integer"/>
            </xsd:complexType>
        </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="block-until-fully-processed" type="xsd:boolean" use="optional"/>
        <xsd:attribute name="logging" use="optional" type="xsd:boolean"/>
        <xsd:attribute name="name" use="required" type="xsd:NCName"/>
        <xsd:attribute name="trigger-transponder-detected-event" type="xsd:boolean" use="optional"/>
        <xsd:attribute name="on-success-call-command" type="xsd:NCName" use="optional"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
<xsd:element name="Errors" minOccurs="0" maxOccurs="unbounded">
    <xsd:complexType>
```

```
229             <xsd:sequence>
230                 <xsd:element name="Error" minOccurs="1" maxOccurs="unbounded">
231                     <xsd:complexType>
232                         <xsd:sequence>
233                             <xsd:element name="Description" type="xsd:string" minOccurs="1" maxOccurs="1"/>
234                             <xsd:element name="Datagroup" minOccurs="1" maxOccurs="1">
235                                 <xsd:complexType>
236                                     <xsd:sequence minOccurs="1" maxOccurs="1">
237                                         <xsd:element name="Dataset" minOccurs="0" maxOccurs="unbounded">
238                                             <xsd:complexType>
239                                                 <xsd:sequence>
240                                                     <xsd:element name="Data" minOccurs="1" maxOccurs="unbounded">
241                                                         <xsd:complexType>
242                                                             <xsd:attribute name="length" type="xsd:integer"/>
243                                                             <xsd:attribute name="type" use="required" ↩
                                                                    type="xsd:NCName"/>
244                                                             <xsd:attribute name="value"/>
245                                                         </xsd:complexType>
246                                                     </xsd:element>
247                                                 </xsd:sequence>
248                                             </xsd:complexType>
249                                         </xsd:element>
250                                     </xsd:sequence>
251                                 </xsd:complexType>
252                             </xsd:element>
253                         </xsd:sequence>
254                     </xsd:complexType>
255                 </xsd:element>
256             </xsd:sequence>
257             <xsd:attribute name="name" use="required" type="xsd:NCName"/>
258             <xsd:attribute name="trigger-exception-event" use="required" type="xsd:boolean"/>
259         </xsd:complexType>
260     </xsd:element>
261    </xsd:sequence>
262   </xsd:complexType>
263  </xsd:element>
264 </xsd:schema>
```

Appendix J

Listing 15.1: FEIG MR 200 Reader Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<Config>
  <!-- Info & Features -->
  <Info>
    <Name>FEIG MR 200</Name>
    <Features>
      <ReaderConfigurations>true</ReaderConfigurations>
      <ReaderReset>true</ReaderReset>
      <TransponderKill>true</TransponderKill>
      <TransponderReading>true</TransponderReading>
      <TransponderWriteID>true</TransponderWriteID>
      <TransponderWriting>true</TransponderWriting>
    </Features>
  </Info>

  <!-- Settings -->
  <Settings default='Serial'>
    <Serial>
      <COMPort>1</COMPort>
      <BaudRate>38400</BaudRate>
      <Timeout>6000</Timeout>
      <DataBits>8</DataBits>
      <StopBits>1</StopBits>
      <Parity>2</Parity>
    </Serial>
    <TCP>
      <IPAddress>192.168.2.12</IPAddress>
      <PortNumber>6666</PortNumber>
      <Timeout>6000</Timeout>
```

```
30      <CheckForDataArrivalEvery>50</CheckForDataArrivalEvery>
31    </TCP>
32  </Settings>
33
34  <!-- list of supported commands and their definitions -->
35  <Commands>
36
37    <!-- Startup commands -->
38    <Startup />
39
40    <!-- Exit commands -->
41    <Goodbye />
42
43    <!-- Scheduled commands -->
44    <Scheduled interval='2000'>
45      <Call-Command>CheckForTransponder</Call-Command>
46    </Scheduled>
47
48    <!-- RF Reset command, page 68 -->
49    <Command name='RFReset' logging='true'>
50      <Description>RF Reset command</Description>
51      <OnSuccessLog>RF Reset completed successfully!</OnSuccessLog>
52      <Prerequisites>
53        <Features>
54          <Feature name='ReaderReset' />
55        </Features>
56      </Prerequisites>
57      <Send>
58        <Datagroup>
59          <Dataset>
60            <Data type='hex' value='0x05,0xFF,0x69' />
61            <Data type='compute_crc16' />
62          </Dataset>
63        </Datagroup>
64      </Send>
```

```
65      <Receive check-error='RFResetErrors' expects-receive='true' ↩
            pause-before-receive='50' no-response-timeout='2000'>
66        <Datagroup>
67          <Dataset>
68            <Data type='hex' value='0x06,0x00,0x69,0x00' />
69            <Data type='check_crc16' length='2' />
70          </Dataset>
71        </Datagroup>
72      </Receive>
73    </Command>
74
75    <!-- CPU Reset command, page 61 -->
76    <Command name='CPUReset' logging='true'>
77      <Description>CPU Reset command</Description>
78      <OnSuccessLog>CPU Reset completed successfully!</OnSuccessLog>
79      <Prerequisites>
80        <Features>
81          <Feature name='ReaderReset' />
82        </Features>
83      </Prerequisites>
84      <Send>
85        <Datagroup>
86          <Dataset>
87            <Data type='hex' value='0x05,0xFF,0x63' />
88            <Data type='compute_crc16' />
89          </Dataset>
90        </Datagroup>
91      </Send>
92      <Receive check-error='CPUResetErrors' expects-receive='true' ↩
            pause-before-receive='50' no-response-timeout='2000'>
93        <Datagroup>
94          <Dataset>
95            <Data type='hex' value='0x06,0x00,0x63,0x00' />
96            <Data type='check_crc16' length='2' />
97          </Dataset>
98        </Datagroup>
```

```
99      </Receive>
100   </Command>
101
102   <!-- System Reset command -->
103   <Command name='SystemReset' logging='true'>
104     <Description>System Reset command</Description>
105     <OnSuccessLog>System Reset completed successfully!</OnSuccessLog>
106     <Prerequisites>
107       <Features>
108         <Feature name='ReaderReset' />
109       </Features>
110     </Prerequisites>
111     <Send>
112       <Datagroup>
113         <Dataset>
114           <Data type='hex' value='0x06,0xFF,0x64,0x00' />
115           <Data type='compute_crc16' />
116         </Dataset>
117       </Datagroup>
118     </Send>
119     <Receive check-error='SystemResetErrors' expects-receive='true' ↩
          pause-before-receive='50' no-response-timeout='2000'>
120       <Datagroup>
121         <Dataset>
122           <Data type='hex' value='0x06,0x00,0x64,0x00' />
123           <Data type='check_crc16' length='2' />
124         </Dataset>
125       </Datagroup>
126     </Receive>
127   </Command>
128
129   <!-- Check For Transponder command, page ?? -->
130   <Command name='CheckForTransponder' trigger-transponder-detected-event='true' ↩
        logging='true'>
131     <Description>Check For Transponder command</Description>
132     <OnSuccessLog>Check For Transponder completed successfully!</OnSuccessLog>
```

```
133     <Prerequisites>
134       <Features>
135         <Feature name='TransponderReading' />
136       </Features>
137     </Prerequisites>
138     <Send>
139       <Datagroup>
140         <Dataset>
141           <Data type='hex' value='0x07,0xff,0xb0,0x01,0x00' />
142           <Data type='compute_crc16' />
143         </Dataset>
144       </Datagroup>
145     </Send>
146     <Receive check-error='CheckForTransponderErrors' expects-receive='true' ↩
          pause-before-receive='50' no-response-timeout='10000'>
147       <Datagroup>
148         <Dataset>
149           <Data type='to_variable' assign-to='$data_length' length='1' />
150           <Data type='hex' value='0x00,0xb0,0x00' />
151           <Data type='to_variable' assign-to='$number_of_transponders' length='1' />
152         </Dataset>
153         <Dataloop type='from_variable' value="$number_of_transponders">
154           <!-- ISO 15693 -->
155           <Dataset>
156             <Data type='to_variable' assign-to='$trans_protocol' length='1' ↩
                  value='0x03' />
157             <Data type='to_variable' assign-to='$trans_DSFID' length='1' />
158             <Data type='to_variable' assign-to='$trans_id' length='8' />
159           </Dataset>
160           <!-- ISO 14443A -->
161           <Dataset>
162             <Data type='to_variable' assign-to='$trans_protocol' length='1' ↩
                  value='0x04'/>
163             <Data type='to_variable' assign-to='$trans_TR_INFO' length='1' />
164             <Data type='to_variable' assign-to='$trans_OPT_INFO' length='1' />
165             <Data type='to_variable' assign-to='$trans_id' length='7' />
```

```
166             </Dataset>
167             <!-- ISO 14443B -->
168             <Dataset>
169               <Data type='to_variable' assign-to='$trans_protocol' length='1' ↩
                      value='0x05'/>
170               <Data type='to_variable' assign-to='$trans_PROTO_INFO' length='1' />
171               <Data type='to_variable' assign-to='$trans_APP_DATA' length='4' />
172               <Data type='to_variable' assign-to='$trans_id' length='4' />
173             </Dataset>
174             <!-- I-Code EPC -->
175             <Dataset>
176               <Data type='to_variable' assign-to='$trans_protocol' length='1' ↩
                      value='0x06'/>
177               <Data type='to_variable' assign-to='$trans_id' length='8' />
178             </Dataset>
179             <!-- Jewel -->
180             <Dataset>
181               <Data type='to_variable' assign-to='$trans_protocol' length='1' ↩
                      value='0x08'/>
182               <Data type='any' length='2' />
183               <Data type='to_variable' assign-to='$trans_id' length='6' />
184             </Dataset>
185           </Dataloop>
186           <Dataset>
187             <Data type='check_crc16' length='2' />
188           </Dataset>
189         </Datagroup>
190       </Receive>
191     </Command>
192
193   </Commands>
194
195   <!-- Set RF reset errors, page 138 in manual -->
196   <Errors name='RFResetErrors' trigger-exception-event='true'>
197     <Error>
198       <Description>Unable to reset RF</Description>
```

```
199       <Datagroup>
200         <Dataset>
201           <Data type='hex' length="6" />
202         </Dataset>
203       </Datagroup>
204     </Error>
205   </Errors>
206
207   <!-- Set CPU reset errors, page 138 in manual -->
208   <Errors name='CPUResetErrors' trigger-exception-event='true'>
209     <Error>
210       <Description>Unable to reset CPU</Description>
211       <Datagroup>
212         <Dataset>
213           <Data type='hex' length="6" />
214         </Dataset>
215       </Datagroup>
216     </Error>
217   </Errors>
218
219   <!-- Set system reset errors, page 138 in manual -->
220   <Errors name='SystemResetErrors' trigger-exception-event='true'>
221     <Error>
222       <Description>Unable to reset system</Description>
223       <Datagroup>
224         <Dataset>
225           <Data type='hex' length="6" />
226         </Dataset>
227       </Datagroup>
228     </Error>
229   </Errors>
230
231   <!-- Set check for transponder errors -->
232   <Errors name='CheckForTransponderErrors' trigger-exception-event='true'>
233     <Error>
234       <Description>No Transponder in Reader Field</Description>
```

```
235       <Datagroup>
236         <Dataset>
237           <Data type='hex' value="0x06,0x00,0xB0,0x01" />
238           <Data type='check_crc16' length='2' />
239         </Dataset>
240       </Datagroup>
241     </Error>
242   </Errors>
243
244 </Config>
```

Appendix K

Listing 16.1: Phidget RFID 1023 Reader Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<Config>
  <!-- Info & Features -->
  <Info>
    <Name>Phidget RFID 1023</Name>
    <Features>
      <ReaderConfigurations>true</ReaderConfigurations>
      <TransponderKill>false</TransponderKill>
      <TransponderReading>true</TransponderReading>
      <TransponderWriteID>false</TransponderWriteID>
      <TransponderWriting>false</TransponderWriting>
    </Features>
  </Info>

  <!-- Settings -->
  <Settings default='Usb'>
    <Usb>
      <ManufacturerCode>0x06C2</ManufacturerCode>
      <ProductCode>0x0031</ProductCode>
      <SerialNumber>63635</SerialNumber>
      <Configuration>0x01</Configuration>
      <Interface>0x00</Interface>
      <AlternativeInterface>0x00</AlternativeInterface>
      <ResetDeviceOnOpen>false</ResetDeviceOnOpen>
      <Timeout>3000</Timeout>
      <CheckForDataArrivalEvery>200</CheckForDataArrivalEvery>
      <ReceiveTransferMode>interrupt</ReceiveTransferMode>
      <SendTransferMode>interrupt</SendTransferMode>
      <InPipeAddress>0x81</InPipeAddress>
```

```
30      <OutPipeAddress>0x00</OutPipeAddress>
31      <ReconnectOnTimeout>true</ReconnectOnTimeout>
32    </Usb>
33  </Settings>
34
35  <!-- list of supported commands and their definitions -->
36  <Commands>
37
38    <!-- Startup commands -->
39    <Startup />
40
41    <!-- Exit commands -->
42    <Goodbye />
43
44    <!-- Scheduled commands -->
45    <Scheduled interval='2000'>
46      <Call-Command>CheckForTransponder</Call-Command>
47    </Scheduled>
48
49    <!-- Check for transponder command -->
50    <Command name='CheckForTransponder' trigger-transponder-detected-event='true' ↩
          on-success-call-command='TurnOnLed'>
51      <Description>Check for transponder command</Description>
52      <OnSuccessLog>Checking for transponder completed successfully!</OnSuccessLog>
53      <Prerequisites>
54        <Features>
55          <Feature name='TransponderReading' />
56        </Features>
57      </Prerequisites>
58      <Send>
59        <ControlMessage BMRequestType='0x21' BRequest='0x09' WValue='0x0200' ↩
              WIndex='0x00' />
60        <Datagroup>
61          <Dataset>
62            <Data type='hex' value='0x08,0x00,0x00,0x00' />
63          </Dataset>
```

```
64          </Datagroup>
65        </Send>
66        <Receive check-error='CheckForTransponderErrors' expects-receive='true' ↩
              pause-before-receive='50' no-response-timeout='2000'>
67          <Datagroup>
68            <Dataset>
69              <Data type='hex' value='0x00' />
70              <Data type='to_variable' assign-to='$trans_id' length='6' />
71            </Dataset>
72          </Datagroup>
73        </Receive>
74      </Command>
75
76      <!-- Turn on led command -->
77      <Command name='TurnOnLed'>
78        <Description>Turn on led command</Description>
79        <OnSuccessLog>Led turned on successfully!</OnSuccessLog>
80        <Send>
81          <ControlMessage BMRequestType='0x21' BRequest='0x09' WValue='0x0200' ↩
                WIndex='0x00' />
82          <Datagroup>
83            <Dataset>
84              <Data type='hex' value='0x0C,0x00,0x00,0x00' />
85            </Dataset>
86          </Datagroup>
87        </Send>
88      </Command>
89
90      <!-- Turn off led command -->
91      <Command name='TurnOffLed'>
92        <Description>Turn off led command</Description>
93        <OnSuccessLog>Led turned off successfully!</OnSuccessLog>
94        <Send>
95          <ControlMessage BMRequestType='0x21' BRequest='0x09' WValue='0x0200' ↩
                WIndex='0x00' />
96          <Datagroup>
```

```
97            <Dataset>
98              <Data type='hex' value='0x08,0x00,0x00,0x00' />
99            </Dataset>
100         </Datagroup>
101       </Send>
102     </Command>
103   </Commands>
104
105   <!-- Check for transponder errors -->
106   <Errors name='CheckForTransponderErrors' trigger-exception-event='true'>
107     <Error>
108       <Description>No transponder found!</Description>
109       <Datagroup>
110         <Dataset>
111           <Data type='hex' value='0x01,0x08,0x00,0x00,0x00,0x00,0x00' />
112         </Dataset>
113       </Datagroup>
114     </Error>
115     <Error>
116       <Description>Unknown error</Description>
117       <Datagroup>
118         <Dataset>
119           <Data type='hex' value='0x01,0x00,0x00,0x00,0x00,0x00,0x00' />
120         </Dataset>
121       </Datagroup>
122     </Error>
123   </Errors>
124
125 </Config>
```

Appendix L

Listing 17.1: Intermec IF61 Reader Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<Config>
  <!-- Info & Features -->
  <Info>
    <Name>Intermec IF61</Name>
    <Features>
      <ReaderConfigurations>true</ReaderConfigurations>
      <ReaderReset>true</ReaderReset>
      <TransponderKill>true</TransponderKill>
      <TransponderReading>true</TransponderReading>
      <TransponderWriteID>true</TransponderWriteID>
      <TransponderWriting>true</TransponderWriting>
    </Features>
  </Info>

  <!-- Settings -->
  <Settings default='TCP'>
    <TCP>
      <IPAddress>192.168.2.11</IPAddress>
      <PortNumber>5555</PortNumber>
      <Timeout>6000</Timeout>
      <CheckForDataArrivalEvery>50</CheckForDataArrivalEvery>
    </TCP>
  </Settings>

  <!-- list of supported commands and their definitions -->
  <Commands>

    <!-- Startup commands -->
```

```
30    <Startup />
31
32    <!-- Exit commands -->
33    <Goodbye />
34
35    <!-- Scheduled commands -->
36    <Scheduled interval='2000'>
37      <!-- <Call-Command>CheckForTransponder</Call-Command> -->
38    </Scheduled>
39
40    <!-- Boot Reset command, page 66 -->
41    <Command name='bootReset' logging='true'>
42      <Description>Boot Reset</Description>
43      <OnSuccessLog>RF Reset completed successfully!</OnSuccessLog>
44      <Prerequisites>
45        <Features>
46          <Feature name='ReaderReset' />
47        </Features>
48      </Prerequisites>
49      <Send>
50        <Datagroup>
51          <Dataset>
52            <Data type='string' value='RESET' />
53            <Data type='hex' value='0x0D,0x0A' />
54          </Dataset>
55        </Datagroup>
56      </Send>
57      <Receive expects-receive='true' pause-before-receive='50' ↩
            no-response-timeout='2000'>
58        <Datagroup>
59          <Dataset>
60            <Data type='string' value='OK>' />
61            <Data type='hex' value='0x0D,0x0A' />
62          </Dataset>
63        </Datagroup>
64      </Receive>
```

```
65    </Command>
66
67    <!-- Check For Transponder command, page 60 -->
68    <Command name='CheckForTransponder' trigger-transponder-detected-event='true' ↩
          logging='true'>
69      <Description>Check For Transponder command</Description>
70      <OnSuccessLog>Check For Transponder completed successfully!</OnSuccessLog>
71      <Prerequisites>
72        <Features>
73          <Feature name='TransponderReading' />
74        </Features>
75      </Prerequisites>
76      <Send>
77        <Datagroup>
78          <Dataset>
79            <Data type='string' value='READ "TagType:"TAGTYPE "TagID:"TAGID' />
80            <Data type='hex' value='0x0D,0x0A' />
81          </Dataset>
82        </Datagroup>
83      </Send>
84      <Receive check-error='CheckForTransponderErrors' expects-receive='true' ↩
          pause-before-receive='50' no-response-timeout='10000'>
85        <Datagroup>
86          <Dataloop type='COUNTER' value="TagType:">
87            <!-- MIXED -->
88            <Dataset>
89              <Data type='string' value='TagType:MIXED TagType:' />
90              <Data type='to_variable' assign-to='$trans_id' length='16' />
91            </Dataset>
92            <!-- ISO6BG1 -->
93            <Dataset>
94              <Data type='string' value='TagType:ISO6BG1 TagType:' />
95              <Data type='to_variable' assign-to='$trans_id' length='16' />
96            </Dataset>
97            <!-- ISO6BG2 -->
98            <Dataset>
```

```
99              <Data type='string' value='TagType:ISO6BG2 TagType:' />
100             <Data type='to_variable' assign-to='$trans_id' length='16' />
101           </Dataset>
102           <!-- ICODE119 -->
103           <Dataset>
104             <Data type='string' value='TagType:ICODE119 TagType:' />
105             <Data type='to_variable' assign-to='$trans_id' length='16' />
106           </Dataset>
107           <!-- UCODE119 -->
108           <Dataset>
109             <Data type='string' value='TagType:UCODE119 TagType:' />
110             <Data type='to_variable' assign-to='$trans_id' length='16' />
111           </Dataset>
112           <!-- EPCC1G1 -->
113           <Dataset>
114             <Data type='string' value='TagType:EPCC1G1 TagType:' />
115             <Data type='to_variable' assign-to='$trans_id' length='16' />
116           </Dataset>
117           <!-- EPCC1G2 -->
118           <Dataset>
119             <Data type='string' value='TagType:EPCC1G2 TagType:' />
120             <Data type='to_variable' assign-to='$trans_id' length='16' />
121           </Dataset>
122         </Dataloop>
123         <Dataset>
124           <Data type='string' value='OK>' />
125           <Data type='hex' value='0x0D,0x0A' />
126         </Dataset>
127       </Datagroup>
128     </Receive>
129   </Command>
130
131 </Commands>
132
133 <!-- Set check for transponder errors, page 61 -->
134 <Errors name='CheckForTransponderErrors' trigger-exception-event='true'>
```

```
135     <Error>
136       <Description>A general BRI error has occurred</Description>
137       <Datagroup>
138         <Dataset>
139           <Data type='string' value='ERR' />
140           <Data type='hex' value='0x0D,0x0A' />
141           <Data type='string' value='OK>' />
142           <Data type='hex' value='0x0D,0x0A' />
143         </Dataset>
144       </Datagroup>
145     </Error>
146     <Error>
147       <Description>The data read from the tag was invalid</Description>
148       <Datagroup>
149         <Dataset>
150           <Data type='string' value='RDERR' />
151           <Data type='hex' value='0x0D,0x0A' />
152           <Data type='string' value='OK>' />
153           <Data type='hex' value='0x0D,0x0A' />
154         </Dataset>
155       </Datagroup>
156     </Error>
157     <Error>
158       <Description>The address used in the read command was not available on the ↩
              tag</Description>
159       <Datagroup>
160         <Dataset>
161           <Data type='string' value='MEMOVRN' />
162           <Data type='hex' value='0x0D,0x0A' />
163           <Data type='string' value='OK>' />
164           <Data type='hex' value='0x0D,0x0A' />
165         </Dataset>
166       </Datagroup>
167     </Error>
168     <Error>
```

```
169       <Description>The amount of data requested from the tag will not fit into the ↩
              output buffer in the BRI service</Description>
170       <Datagroup>
171         <Dataset>
172           <Data type='string' value='DISPLAYERR' />
173           <Data type='hex' value='0x0D,0x0A' />
174           <Data type='string' value='OK>' />
175           <Data type='hex' value='0x0D,0x0A' />
176         </Dataset>
177       </Datagroup>
178     </Error>
179     <Error>
180       <Description>No Transponder in Reader Field</Description>
181       <Datagroup>
182         <Dataset>
183           <Data type='string' value='NOTAG' />
184           <Data type='hex' value='0x0D,0x0A' />
185           <Data type='string' value='OK>' />
186           <Data type='hex' value='0x0D,0x0A' />
187         </Dataset>
188       </Datagroup>
189     </Error>
190   </Errors>
191
192 </Config>
```

Appendix M

Listing 18.1: FEIG validation application output

```
1  INFO: Monitoring for RFIDMania RFIDValidation Application [FEIG MR200] started
2  INFO: Connected
3  ------------------------------------------------------------------------------------------
4  INFO: Adding command: CPU Reset command
5  INFO: Processing command: CPU Reset command
6  INFO: Registering CPUReset -> CPU Reset command for receiving data
7  INFO: Sending: [DATA-SET](HEX) 0x05,(HEX) 0xFF,(HEX) 0x63,(COMPUTE_CRC16) ↩
       0xD3,(COMPUTE_CRC16) 0xAE[/DATA-SET]
8  INFO: Relaying Received (BYTE) 0x06,(BYTE) 0x00,(BYTE) 0x63,(BYTE) 0x00,(BYTE) ↩
       0x86,(BYTE) 0x07 to: [CPUReset -> CPU Reset command]
9  INFO: Deregistering CPUReset -> CPU Reset command for receiving data
10 INFO: Removing command: CPU Reset command
11
12
13 CPU Reset Command Result Log:
14 Sending bytes...
15 Awaiting reader reply
16 Received: (BYTE) 0x06,(BYTE) 0x00,(BYTE) 0x63,(BYTE) 0x00,(BYTE) 0x86,(BYTE) 0x07 @ Wed ↩
       Aug 12 23:24:58 EDT 2009
17 CPU Reset completed successfully!
18 ------------------------------------------------------------------------------------------
19 INFO: Adding command: System Reset command
20 INFO: Processing command: System Reset command
21 INFO: Registering SystemReset -> System Reset command for receiving data
22 INFO: Sending: [DATA-SET](HEX) 0x06,(HEX) 0xFF,(HEX) 0x64,(HEX) 0x00,(COMPUTE_CRC16) ↩
       0x7D,(COMPUTE_CRC16) 0x8C[/DATA-SET]
23 INFO: Relaying Received (BYTE) 0x06,(BYTE) 0x00,(BYTE) 0x64,(BYTE) 0x00,(BYTE) ↩
       0x8E,(BYTE) 0x4A to: [SystemReset -> System Reset command]
24 INFO: Deregistering SystemReset -> System Reset command for receiving data
```

```
25 INFO: Removing command: System Reset command
26
27
28 System Reset Command Result Log:
29 Sending bytes...
30 Awaiting reader reply
31 Received: (BYTE) 0x06,(BYTE) 0x00,(BYTE) 0x64,(BYTE) 0x00,(BYTE) 0x8E,(BYTE) 0x4A @ Wed ↩
       Aug 12 23:25:01 EDT 2009
32 System Reset completed successfully!
33 ------------------------------------------------------------------------------------------
34 INFO: Adding command: RF Reset command
35 INFO: Processing command: RF Reset command
36 INFO: Registering RFReset -> RF Reset command for receiving data
37 INFO: Sending: [DATA-SET](HEX) 0x05,(HEX) 0xFF,(HEX) 0x69,(COMPUTE_CRC16) ↩
       0x89,(COMPUTE_CRC16) 0x01[/DATA-SET]
38 INFO: Relaying Received (BYTE) 0x06,(BYTE) 0x00,(BYTE) 0x69,(BYTE) 0x00,(BYTE) ↩
       0xF6,(BYTE) 0xFA to: [RFReset -> RF Reset command]
39 INFO: Deregistering RFReset -> RF Reset command for receiving data
40 INFO: Removing command: RF Reset command
41
42
43 RF Reset Command Result Log:
44 Sending bytes...
45 Awaiting reader reply
46 Received: (BYTE) 0x06,(BYTE) 0x00,(BYTE) 0x69,(BYTE) 0x00,(BYTE) 0xF6,(BYTE) 0xFA @ Wed ↩
       Aug 12 23:25:04 EDT 2009
47 RF Reset completed successfully!
48 ------------------------------------------------------------------------------------------
49 INFO: Adding command: Check For Transponder command
50 INFO: Processing command: Check For Transponder command
51 INFO: Registering CheckForTransponder -> Check For Transponder command for receiving data
52 INFO: Sending: [DATA-SET](HEX) 0x07,(HEX) 0xFF,(HEX) 0xB0,(HEX) 0x01,(HEX) ↩
       0x00,(COMPUTE_CRC16) 0x1C,(COMPUTE_CRC16) 0x56[/DATA-SET]
```

```
53 INFO: Relaying Received (BYTE) 0x1B,(BYTE) 0x00,(BYTE) 0xB0,(BYTE) 0x00,(BYTE) ↩
       0x02,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
       0x24,(BYTE) 0x21,(BYTE) 0x7F,(BYTE) 0x68,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) ↩
       0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x24,(BYTE) 0x21,(BYTE) 0x7F,(BYTE) 0x69,(BYTE) ↩
       0x59,(BYTE) 0x3F to: [CheckForTransponder -> Check For Transponder command]
54 INFO: Deregistering CheckForTransponder -> Check For Transponder command for receiving ↩
       data
55 INFO: Removing command: Check For Transponder command
56
57
58 Check For Transponder Command Result Log:
59 Sending bytes...
60 Awaiting reader reply
61 Received: (BYTE) 0x1B,(BYTE) 0x00,(BYTE) 0xB0,(BYTE) 0x00,(BYTE) 0x02,(BYTE) ↩
       0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x24,(BYTE) ↩
       0x21,(BYTE) 0x7F,(BYTE) 0x68,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) ↩
       0x00,(BYTE) 0x00,(BYTE) 0x24,(BYTE) 0x21,(BYTE) 0x7F,(BYTE) 0x69,(BYTE) 0x59,(BYTE) ↩
       0x3F @ Wed Aug 12 23:26:00 EDT 2009
62 Check For Transponder completed successfully!
63 ----------------------------------------------------------------
64 Application received transponders:
65    [E007000024217F68] @ Wed Aug 12 23:26:02 EDT 2009
66    [E007000024217F69] @ Wed Aug 12 23:26:02 EDT 2009
67 ----------------------------------------------------------------
68 INFO: Detected Transponders
69    [E007000024217F68] @ Wed Aug 12 23:26:02 EDT 2009
70    [E007000024217F69] @ Wed Aug 12 23:26:02 EDT 2009
71 ------------------------------------------------------------------------------------
72 INFO: Adding command: Check For Transponder command
73 INFO: Processing command: Check For Transponder command
74 INFO: Registering CheckForTransponder -> Check For Transponder command for receiving data
75 INFO: Sending: [DATA-SET](HEX) 0x07,(HEX) 0xFF,(HEX) 0xB0,(HEX) 0x01,(HEX) ↩
       0x00,(COMPUTE_CRC16) 0x1C,(COMPUTE_CRC16) 0x56[/DATA-SET]
```

```
76 INFO: Relaying Received (BYTE) 0x39,(BYTE) 0x00,(BYTE) 0xB0,(BYTE) 0x00,(BYTE) ↩
       0x05,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
       0x24,(BYTE) 0x21,(BYTE) 0x7F,(BYTE) 0x68,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) ↩
       0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x24,(BYTE) 0x21,(BYTE) 0x7F,(BYTE) 0x69,(BYTE) ↩
       0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x18,(BYTE) ↩
       0xEF,(BYTE) 0x3C,(BYTE) 0x6B,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) ↩
       0x00,(BYTE) 0x00,(BYTE) 0x18,(BYTE) 0xEF,(BYTE) 0x3C,(BYTE) 0x6C,(BYTE) 0x03,(BYTE) ↩
       0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x11,(BYTE) 0xFE,(BYTE) ↩
       0x66,(BYTE) 0xC8,(BYTE) 0x34,(BYTE) 0x30 to: [CheckForTransponder -> Check For ↩
       Transponder command]
77 INFO: Deregistering CheckForTransponder -> Check For Transponder command for receiving ↩
       data
78 INFO: Removing command: Check For Transponder command
79
80
81 Check For Transponder Command Result Log:
82 Sending bytes...
83 Awaiting reader reply
84 Received: (BYTE) 0x39,(BYTE) 0x00,(BYTE) 0xB0,(BYTE) 0x00,(BYTE) 0x05,(BYTE) ↩
       0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x24,(BYTE) ↩
       0x21,(BYTE) 0x7F,(BYTE) 0x68,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) ↩
       0x00,(BYTE) 0x00,(BYTE) 0x24,(BYTE) 0x21,(BYTE) 0x7F,(BYTE) 0x69,(BYTE) 0x03,(BYTE) ↩
       0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x18,(BYTE) 0xEF,(BYTE) ↩
       0x3C,(BYTE) 0x6B,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) ↩
       0x00,(BYTE) 0x18,(BYTE) 0xEF,(BYTE) 0x3C,(BYTE) 0x6C,(BYTE) 0x03,(BYTE) 0x00,(BYTE) ↩
       0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x11,(BYTE) 0xFE,(BYTE) 0x66,(BYTE) ↩
       0xC8,(BYTE) 0x34,(BYTE) 0x30 @ Wed Aug 12 23:26:27 EDT 2009
85 Check For Transponder completed successfully!
86 ------------------------------------------------------------
87 Application received transponders:
88    [E007000024217F68] @ Wed Aug 12 23:26:29 EDT 2009
89    [E007000024217F69] @ Wed Aug 12 23:26:29 EDT 2009
90    [E007000018EF3C6B] @ Wed Aug 12 23:26:29 EDT 2009
91    [E007000018EF3C6C] @ Wed Aug 12 23:26:29 EDT 2009
92    [E007000011FE66C8] @ Wed Aug 12 23:26:30 EDT 2009
93 ------------------------------------------------------------
```

```
INFO: Detected Transponders
   [E007000024217F68] @ Wed Aug 12 23:26:29 EDT 2009
   [E007000024217F69] @ Wed Aug 12 23:26:29 EDT 2009
   [E007000018EF3C6B] @ Wed Aug 12 23:26:29 EDT 2009
   [E007000018EF3C6C] @ Wed Aug 12 23:26:29 EDT 2009
   [E007000011FE66C8] @ Wed Aug 12 23:26:30 EDT 2009
--------------------------------------------------------------------------------
INFO: Adding command: Check For Transponder command
INFO: Processing command: Check For Transponder command
INFO: Registering CheckForTransponder -> Check For Transponder command for receiving data
INFO: Sending: [DATA-SET](HEX) 0x07,(HEX) 0xFF,(HEX) 0xB0,(HEX) 0x01,(HEX) ↩
    0x00,(COMPUTE_CRC16) 0x1C,(COMPUTE_CRC16) 0x56[/DATA-SET]
INFO: Relaying Received (BYTE) 0x39,(BYTE) 0x00,(BYTE) 0xB0,(BYTE) 0x00,(BYTE) ↩
    0x05,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
    0x24,(BYTE) 0x21,(BYTE) 0x7F,(BYTE) 0x68,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) ↩
    0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x24,(BYTE) 0x21,(BYTE) 0x7F,(BYTE) 0x69,(BYTE) ↩
    0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x18,(BYTE) ↩
    0xEF,(BYTE) 0x3C,(BYTE) 0x6B,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) ↩
    0x00,(BYTE) 0x00,(BYTE) 0x18,(BYTE) 0xEF,(BYTE) 0x3C,(BYTE) 0x6C,(BYTE) 0x03,(BYTE) ↩
    0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x11,(BYTE) 0xFE,(BYTE) ↩
    0x66,(BYTE) 0xC8,(BYTE) 0x34,(BYTE) 0x30 to: [CheckForTransponder -> Check For ↩
    Transponder command]
INFO: Deregistering CheckForTransponder -> Check For Transponder command for receiving ↩
    data
INFO: Removing command: Check For Transponder command


Check For Transponder Command Result Log:
Sending bytes...
Awaiting reader reply
```

```
113 Received: (BYTE) 0x39,(BYTE) 0x00,(BYTE) 0xB0,(BYTE) 0x00,(BYTE) 0x05,(BYTE) ↩
    0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x24,(BYTE) ↩
    0x21,(BYTE) 0x7F,(BYTE) 0x68,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) ↩
    0x00,(BYTE) 0x00,(BYTE) 0x24,(BYTE) 0x21,(BYTE) 0x7F,(BYTE) 0x69,(BYTE) 0x03,(BYTE) ↩
    0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x18,(BYTE) 0xEF,(BYTE) ↩
    0x3C,(BYTE) 0x6B,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) ↩
    0x00,(BYTE) 0x18,(BYTE) 0xEF,(BYTE) 0x3C,(BYTE) 0x6C,(BYTE) 0x03,(BYTE) 0x00,(BYTE) ↩
    0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x11,(BYTE) 0xFE,(BYTE) 0x66,(BYTE) ↩
    0xC8,(BYTE) 0x34,(BYTE) 0x30 @ Wed Aug 12 23:26:43 EDT 2009
114 Check For Transponder completed successfully!
115 ----------------------------------------------------------------
116 Application received transponders:
117    [E007000024217F68] @ Wed Aug 12 23:26:45 EDT 2009
118    [E007000024217F69] @ Wed Aug 12 23:26:45 EDT 2009
119    [E007000018EF3C6B] @ Wed Aug 12 23:26:45 EDT 2009
120    [E007000018EF3C6C] @ Wed Aug 12 23:26:45 EDT 2009
121    [E007000011FE66C8] @ Wed Aug 12 23:26:45 EDT 2009
122 ----------------------------------------------------------------
123 INFO: Detected Transponders
124    [E007000024217F68] @ Wed Aug 12 23:26:45 EDT 2009
125    [E007000024217F69] @ Wed Aug 12 23:26:45 EDT 2009
126    [E007000018EF3C6B] @ Wed Aug 12 23:26:45 EDT 2009
127    [E007000018EF3C6C] @ Wed Aug 12 23:26:45 EDT 2009
128    [E007000011FE66C8] @ Wed Aug 12 23:26:45 EDT 2009
129 ----------------------------------------------------------------------------------------
130 INFO: Adding command: Check For Transponder command
131 INFO: Processing command: Check For Transponder command
132 INFO: Registering CheckForTransponder -> Check For Transponder command for receiving data
133 INFO: Sending: [DATA-SET](HEX) 0x07,(HEX) 0xFF,(HEX) 0xB0,(HEX) 0x01,(HEX) ↩
    0x00,(COMPUTE_CRC16) 0x1C,(COMPUTE_CRC16) 0x56[/DATA-SET]
```

```
134 INFO: Relaying Received (BYTE) 0x43,(BYTE) 0x00,(BYTE) 0xB0,(BYTE) 0x00,(BYTE) ↩
    0x06,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
    0x11,(BYTE) 0xFE,(BYTE) 0x66,(BYTE) 0xC8,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) ↩
    0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x24,(BYTE) 0x21,(BYTE) 0x7F,(BYTE) 0x69,(BYTE) ↩
    0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x04,(BYTE) 0x01,(BYTE) 0x00,(BYTE) 0x07,(BYTE) ↩
    0x00,(BYTE) 0x09,(BYTE) 0x9A,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) ↩
    0x00,(BYTE) 0x00,(BYTE) 0x18,(BYTE) 0xEF,(BYTE) 0x3C,(BYTE) 0x6B,(BYTE) 0x03,(BYTE) ↩
    0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x18,(BYTE) 0xEF,(BYTE) ↩
    0x3C,(BYTE) 0x6C,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) ↩
    0x00,(BYTE) 0x24,(BYTE) 0x21,(BYTE) 0x7F,(BYTE) 0x68,(BYTE) 0x4F,(BYTE) 0xA8 to: ↩
    [CheckForTransponder -> Check For Transponder command]
135 INFO: Deregistering CheckForTransponder -> Check For Transponder command for receiving ↩
    data
136 INFO: Removing command: Check For Transponder command
137
138
139 Check For Transponder Command Result Log:
140 Sending bytes...
141 Awaiting reader reply
142 Received: (BYTE) 0x43,(BYTE) 0x00,(BYTE) 0xB0,(BYTE) 0x00,(BYTE) 0x06,(BYTE) ↩
    0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x11,(BYTE) ↩
    0xFE,(BYTE) 0x66,(BYTE) 0xC8,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) ↩
    0x00,(BYTE) 0x00,(BYTE) 0x24,(BYTE) 0x21,(BYTE) 0x7F,(BYTE) 0x69,(BYTE) 0x03,(BYTE) ↩
    0x00,(BYTE) 0xE0,(BYTE) 0x04,(BYTE) 0x01,(BYTE) 0x00,(BYTE) 0x07,(BYTE) 0x00,(BYTE) ↩
    0x09,(BYTE) 0x9A,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) ↩
    0x00,(BYTE) 0x18,(BYTE) 0xEF,(BYTE) 0x3C,(BYTE) 0x6B,(BYTE) 0x03,(BYTE) 0x00,(BYTE) ↩
    0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x18,(BYTE) 0xEF,(BYTE) 0x3C,(BYTE) ↩
    0x6C,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
    0x24,(BYTE) 0x21,(BYTE) 0x7F,(BYTE) 0x68,(BYTE) 0x4F,(BYTE) 0xA8 @ Wed Aug 12 ↩
    23:27:07 EDT 2009
143 Check For Transponder completed successfully!
144 ----------------------------------------------------------------
145 Application received transponders:
146    [E007000011FE66C8] @ Wed Aug 12 23:27:08 EDT 2009
147    [E0070000024217F69] @ Wed Aug 12 23:27:08 EDT 2009
148    [E00401000700099A] @ Wed Aug 12 23:27:09 EDT 2009
```

```
149    [E007000018EF3C6B] @ Wed Aug 12 23:27:09 EDT 2009
150    [E007000018EF3C6C] @ Wed Aug 12 23:27:09 EDT 2009
151    [E007000024217F68] @ Wed Aug 12 23:27:09 EDT 2009
152 ------------------------------------------------------------
153 INFO: Detected Transponders
154    [E007000011FE66C8] @ Wed Aug 12 23:27:08 EDT 2009
155    [E007000024217F69] @ Wed Aug 12 23:27:08 EDT 2009
156    [E00401000700099A] @ Wed Aug 12 23:27:09 EDT 2009
157    [E007000018EF3C6B] @ Wed Aug 12 23:27:09 EDT 2009
158    [E007000018EF3C6C] @ Wed Aug 12 23:27:09 EDT 2009
159    [E007000024217F68] @ Wed Aug 12 23:27:09 EDT 2009
160 ------------------------------------------------------------------------------------------
161 INFO: Adding command: Check For Transponder command
162 INFO: Processing command: Check For Transponder command
163 INFO: Registering CheckForTransponder -> Check For Transponder command for receiving data
164 INFO: Sending: [DATA-SET](HEX) 0x07,(HEX) 0xFF,(HEX) 0xB0,(HEX) 0x01,(HEX) ↩
        0x00,(COMPUTE_CRC16) 0x1C,(COMPUTE_CRC16) 0x56[/DATA-SET]
165 INFO: Relaying Received (BYTE) 0x11,(BYTE) 0x00,(BYTE) 0xB0,(BYTE) 0x00,(BYTE) ↩
        0x01,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x04,(BYTE) 0x01,(BYTE) 0x00,(BYTE) ↩
        0x07,(BYTE) 0x00,(BYTE) 0x09,(BYTE) 0x9A,(BYTE) 0xC2,(BYTE) 0xE9 to: ↩
        [CheckForTransponder -> Check For Transponder command]
166 INFO: Deregistering CheckForTransponder -> Check For Transponder command for receiving ↩
        data
167 INFO: Removing command: Check For Transponder command
168
169
170 Check For Transponder Command Result Log:
171 Sending bytes...
172 Awaiting reader reply
173 Received: (BYTE) 0x11,(BYTE) 0x00,(BYTE) 0xB0,(BYTE) 0x00,(BYTE) 0x01,(BYTE) ↩
        0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x04,(BYTE) 0x01,(BYTE) 0x00,(BYTE) 0x07,(BYTE) ↩
        0x00,(BYTE) 0x09,(BYTE) 0x9A,(BYTE) 0xC2,(BYTE) 0xE9 @ Wed Aug 12 23:27:30 EDT 2009
174 Check For Transponder completed successfully!
175 ------------------------------------------------------------
176 Application received transponders:
177    [E00401000700099A] @ Wed Aug 12 23:27:31 EDT 2009
```

```
178 ----------------------------------------------------------------
179 INFO: Detected Transponders
180    [E00401000700099A] @ Wed Aug 12 23:27:31 EDT 2009
181 ----------------------------------------------------------------------------------------
182 INFO: Adding command: Check For Transponder command
183 INFO: Processing command: Check For Transponder command
184 INFO: Registering CheckForTransponder -> Check For Transponder command for receiving data
185 INFO: Sending: [DATA-SET](HEX) 0x07,(HEX) 0xFF,(HEX) 0xB0,(HEX) 0x01,(HEX) ↩
        0x00,(COMPUTE_CRC16) 0x1C,(COMPUTE_CRC16) 0x56[/DATA-SET]
186 INFO: Relaying Received (BYTE) 0x06,(BYTE) 0x00,(BYTE) 0xB0,(BYTE) 0x01,(BYTE) ↩
        0x5C,(BYTE) 0x63 to: [CheckForTransponder -> Check For Transponder command]
187 INFO: Deregistering CheckForTransponder -> Check For Transponder command for receiving ↩
        data
188 INFO: Removing command: Check For Transponder command
189 Application received an error for command CheckForTransponder: No Transponder in Reader ↩
        Field
190
191
192 Check For Transponder Command Result Log:
193 Sending bytes...
194 Awaiting reader reply
195 Received: (BYTE) 0x06,(BYTE) 0x00,(BYTE) 0xB0,(BYTE) 0x01,(BYTE) 0x5C,(BYTE) 0x63 @ Wed ↩
        Aug 12 23:27:48 EDT 2009
196 Received data ignored because it didn't match expected data
197 Actual receive bytes do not match expected receive bytes
```

```
198 Printing expected receive bytes: [DATA-SET](TO_VARIABLE) 0x??,(HEX) 0x00,(HEX) ↩
    0xB0,(HEX) 0x00,(TO_VARIABLE) 0x??[/DATA-SET][DATA-LOOP][DATA-SET](TO_VARIABLE) ↩
    0x03,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) ↩
    0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) ↩
    0x??,(TO_VARIABLE) 0x??[/DATA-SET][DATA-SET](TO_VARIABLE) 0x04,(TO_VARIABLE) ↩
    0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) ↩
    0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) ↩
    0x??[/DATA-SET][DATA-SET](TO_VARIABLE) 0x05,(TO_VARIABLE) 0x??,(TO_VARIABLE) ↩
    0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) ↩
    0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) ↩
    0x??[/DATA-SET][DATA-SET](TO_VARIABLE) 0x06,(TO_VARIABLE) 0x??,(TO_VARIABLE) ↩
    0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) ↩
    0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??[/DATA-SET][DATA-SET](TO_VARIABLE) ↩
    0x08,(ANY) 0x??,(ANY) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) ↩
    0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) ↩
    0x??[/DATA-SET][/DATA-LOOP][DATA-SET](CHECK_CRC16) 0x??,(CHECK_CRC16) 0x??[/DATA-SET]
199 Printing last actual receive bytes: (BYTE) 0x06,(BYTE) 0x00,(BYTE) 0xB0,(BYTE) ↩
    0x01,(BYTE) 0x5C,(BYTE) 0x63
200 No Transponder in Reader Field
201
202
203 Check For Transponder Command Result Error:
204 No Transponder in Reader Field
205 ------------------------------------------------------------------------------------
206 INFO: Adding command: Check For Transponder command
207 INFO: Processing command: Check For Transponder command
208 INFO: Registering CheckForTransponder -> Check For Transponder command for receiving data
209 INFO: Sending: [DATA-SET](HEX) 0x07,(HEX) 0xFF,(HEX) 0xB0,(HEX) 0x01,(HEX) ↩
    0x00,(COMPUTE_CRC16) 0x1C,(COMPUTE_CRC16) 0x56[/DATA-SET]
210 INFO: Relaying Received (BYTE) 0x25,(BYTE) 0x00,(BYTE) 0xB0,(BYTE) 0x00,(BYTE) ↩
    0x03,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
    0x24,(BYTE) 0x21,(BYTE) 0x7F,(BYTE) 0x68,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) ↩
    0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x24,(BYTE) 0x21,(BYTE) 0x7F,(BYTE) 0x69,(BYTE) ↩
    0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x04,(BYTE) 0x01,(BYTE) 0x00,(BYTE) 0x07,(BYTE) ↩
    0x00,(BYTE) 0x09,(BYTE) 0x9A,(BYTE) 0x40,(BYTE) 0xB2 to: [CheckForTransponder -> ↩
    Check For Transponder command]
```

```
INFO: Deregistering CheckForTransponder -> Check For Transponder command for receiving ↩
    data
INFO: Removing command: Check For Transponder command


Check For Transponder Command Result Log:
Sending bytes...
Awaiting reader reply
Received: (BYTE) 0x25,(BYTE) 0x00,(BYTE) 0xB0,(BYTE) 0x00,(BYTE) 0x03,(BYTE) ↩
    0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x24,(BYTE) ↩
    0x21,(BYTE) 0x7F,(BYTE) 0x68,(BYTE) 0x03,(BYTE) 0x00,(BYTE) 0xE0,(BYTE) 0x07,(BYTE) ↩
    0x00,(BYTE) 0x00,(BYTE) 0x24,(BYTE) 0x21,(BYTE) 0x7F,(BYTE) 0x69,(BYTE) 0x03,(BYTE) ↩
    0x00,(BYTE) 0xE0,(BYTE) 0x04,(BYTE) 0x01,(BYTE) 0x00,(BYTE) 0x07,(BYTE) 0x00,(BYTE) ↩
    0x09,(BYTE) 0x9A,(BYTE) 0x40,(BYTE) 0xB2 @ Wed Aug 12 23:28:14 EDT 2009
Check For Transponder completed successfully!
------------------------------------------------------------
Application received transponders:
   [E007000024217F68] @ Wed Aug 12 23:28:16 EDT 2009
   [E007000024217F69] @ Wed Aug 12 23:28:16 EDT 2009
   [E00401000700099A] @ Wed Aug 12 23:28:16 EDT 2009
------------------------------------------------------------
INFO: Detected Transponders
   [E007000024217F68] @ Wed Aug 12 23:28:16 EDT 2009
   [E007000024217F69] @ Wed Aug 12 23:28:16 EDT 2009
   [E00401000700099A] @ Wed Aug 12 23:28:16 EDT 2009
```

Listing 18.2: PhidgetRFID validation application output

```
INFO: Monitoring for RFIDMania RFIDValidation Application [Phidget 1023] started
INFO: Connected
------------------------------------------------------------------------------------------
INFO: Adding command: Check for transponder command
INFO: Processing command: Check for transponder command
INFO: Registering CheckForTransponder -> Check for transponder command for receiving data
```

```
7  INFO: Relaying Received (BYTE) 0x00,(BYTE) 0x1B,(BYTE) 0x00,(BYTE) 0x14,(BYTE) ↩
       0x28,(BYTE) 0xBA,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
       command]
8  INFO: Relaying Received (BYTE) 0x00,(BYTE) 0x1B,(BYTE) 0x00,(BYTE) 0x14,(BYTE) ↩
       0x28,(BYTE) 0xBA,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
       command]
9  INFO: Deregistering CheckForTransponder -> Check for transponder command for receiving ↩
       data
10 INFO: Removing command: Check for transponder command
11 INFO: Transponder [1B001428BA00] detected but filtered out!
12 INFO: Detected Transponders
13    [1B001428BA00] @ Thu Aug 13 04:53:57 EDT 2009
14 INFO: Adding chain command: Turn on led command
15 INFO: Processing command: Turn on led command
16 INFO: Removing command: Turn on led command
17
18
19 CheckForTransponder Command Result Log:
20 Sending control message [DATA-SET](HEX) 0x08,(HEX) 0x00,(HEX) 0x00,(HEX) ↩
       0x00[/DATA-SET] ...
21 Control message sent...
22 Awaiting reader reply
23 Received: (BYTE) 0x00,(BYTE) 0x1B,(BYTE) 0x00,(BYTE) 0x14,(BYTE) 0x28,(BYTE) ↩
       0xBA,(BYTE) 0x00 @ Thu Aug 13 04:53:56 EDT 2009
24 Checking for transponder completed successfully!
25 ------------------------------------------------------------------------------------
26 INFO: Adding command: Check for transponder command
27 INFO: Processing command: Check for transponder command
28 INFO: Registering CheckForTransponder -> Check for transponder command for receiving data
29 INFO: Relaying Received (BYTE) 0x00,(BYTE) 0x1E,(BYTE) 0x00,(BYTE) 0x9A,(BYTE) ↩
       0x64,(BYTE) 0xA2,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
       command]
30 INFO: Deregistering CheckForTransponder -> Check for transponder command for receiving ↩
       data
31 INFO: Removing command: Check for transponder command
32 ------------------------------------------------------------
```

```
Application received transponders:
   [1E009A64A200] @ Thu Aug 13 04:54:25 EDT 2009
------------------------------------------------------------
INFO: Detected Transponders
   [1E009A64A200] @ Thu Aug 13 04:54:25 EDT 2009
INFO: Adding chain command: Turn on led command
INFO: Processing command: Turn on led command
INFO: Removing command: Turn on led command


CheckForTransponder Command Result Log:
Sending control message [DATA-SET](HEX) 0x08,(HEX) 0x00,(HEX) 0x00,(HEX) ↩
    0x00[/DATA-SET] ...
Control message sent...
Awaiting reader reply
Received: (BYTE) 0x00,(BYTE) 0x1E,(BYTE) 0x00,(BYTE) 0x9A,(BYTE) 0x64,(BYTE) ↩
    0xA2,(BYTE) 0x00 @ Thu Aug 13 04:54:25 EDT 2009
Checking for transponder completed successfully!
------------------------------------------------------------------------------
INFO: Adding command: Check for transponder command
INFO: Processing command: Check for transponder command
INFO: Registering CheckForTransponder -> Check for transponder command for receiving data
INFO: Relaying Received (BYTE) 0x00,(BYTE) 0x1C,(BYTE) 0x00,(BYTE) 0xEF,(BYTE) ↩
    0xB6,(BYTE) 0x29,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
    command]
INFO: Deregistering CheckForTransponder -> Check for transponder command for receiving ↩
    data
INFO: Removing command: Check for transponder command
------------------------------------------------------------
Application received transponders:
   [1C00EFB62900] @ Thu Aug 13 04:54:40 EDT 2009
------------------------------------------------------------
INFO: Detected Transponders
   [1C00EFB62900] @ Thu Aug 13 04:54:40 EDT 2009
INFO: Adding chain command: Turn on led command
INFO: Processing command: Turn on led command
```

```
64 INFO: Removing command: Turn on led command
65
66
67 CheckForTransponder Command Result Log:
68 Sending control message [DATA-SET](HEX) 0x08,(HEX) 0x00,(HEX) 0x00,(HEX) ↩
       0x00[/DATA-SET] ...
69 Control message sent...
70 Awaiting reader reply
71 Received: (BYTE) 0x00,(BYTE) 0x1C,(BYTE) 0x00,(BYTE) 0xEF,(BYTE) 0xB6,(BYTE) ↩
       0x29,(BYTE) 0x00 @ Thu Aug 13 04:54:40 EDT 2009
72 Checking for transponder completed successfully!
73 ----------------------------------------------------------------------------------------
74 INFO: Adding command: Check for transponder command
75 INFO: Processing command: Check for transponder command
76 INFO: Registering CheckForTransponder -> Check for transponder command for receiving data
77 INFO: Relaying Received (BYTE) 0x01,(BYTE) 0x0C,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
       0x00,(BYTE) 0x00,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
       command]
78 INFO: Relaying Received (BYTE) 0x01,(BYTE) 0x08,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
       0x00,(BYTE) 0x00,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
       command]
79 INFO: Relaying Received (BYTE) 0x00,(BYTE) 0x1A,(BYTE) 0x00,(BYTE) 0x8C,(BYTE) ↩
       0xE7,(BYTE) 0xA0,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
       command]
80 INFO: Deregistering CheckForTransponder -> Check for transponder command for receiving ↩
       data
81 INFO: Removing command: Check for transponder command
82 ------------------------------------------------------------
83 Application received transponders:
84    [1A008CE7A000] @ Thu Aug 13 04:54:55 EDT 2009
85 ------------------------------------------------------------
86 INFO: Detected Transponders
87    [1A008CE7A000] @ Thu Aug 13 04:54:55 EDT 2009
88 INFO: Adding chain command: Turn on led command
89 INFO: Processing command: Turn on led command
90 INFO: Removing command: Turn on led command
```

```
91
92
93 CheckForTransponder Command Result Log:
94 Sending control message [DATA-SET](HEX) 0x08,(HEX) 0x00,(HEX) 0x00,(HEX) ↩
       0x00[/DATA-SET] ...
95 Control message sent...
96 Awaiting reader reply
97 Received: (BYTE) 0x01,(BYTE) 0x08,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
       0x00,(BYTE) 0x00 @ Thu Aug 13 04:54:54 EDT 2009
98 Received data ignored because it didn't match expected data
99 Received: (BYTE) 0x00,(BYTE) 0x1A,(BYTE) 0x00,(BYTE) 0x8C,(BYTE) 0xE7,(BYTE) ↩
       0xA0,(BYTE) 0x00 @ Thu Aug 13 04:54:55 EDT 2009
100 Checking for transponder completed successfully!
101 ------------------------------------------------------------------------------------------
102 INFO: Adding command: Check for transponder command
103 INFO: Processing command: Check for transponder command
104 INFO: Registering CheckForTransponder -> Check for transponder command for receiving data
105 INFO: Relaying Received (BYTE) 0x01,(BYTE) 0x0C,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
       0x00,(BYTE) 0x00,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
       command]
106 INFO: Relaying Received (BYTE) 0x00,(BYTE) 0x17,(BYTE) 0x00,(BYTE) 0x4A,(BYTE) ↩
       0xBC,(BYTE) 0xA0,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
       command]
107 INFO: Deregistering CheckForTransponder -> Check for transponder command for receiving ↩
       data
108 INFO: Removing command: Check for transponder command
109 ------------------------------------------------------------
110 Application received transponders:
111    [17004ABCA000] @ Thu Aug 13 04:55:14 EDT 2009
112 ------------------------------------------------------------
113 INFO: Detected Transponders
114    [17004ABCA000] @ Thu Aug 13 04:55:14 EDT 2009
115 INFO: Adding chain command: Turn on led command
116 INFO: Processing command: Turn on led command
117 INFO: Removing command: Turn on led command
118
```

```
119
120 CheckForTransponder Command Result Log:
121 Sending control message [DATA-SET](HEX) 0x08,(HEX) 0x00,(HEX) 0x00,(HEX) ↩
    0x00[/DATA-SET] ...
122 Control message sent...
123 Received: (BYTE) 0x01,(BYTE) 0x0C,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
    0x00,(BYTE) 0x00 @ Thu Aug 13 04:55:13 EDT 2009
124 Awaiting reader reply
125 Received data ignored because it didn't match expected data
126 Received: (BYTE) 0x00,(BYTE) 0x17,(BYTE) 0x00,(BYTE) 0x4A,(BYTE) 0xBC,(BYTE) ↩
    0xA0,(BYTE) 0x00 @ Thu Aug 13 04:55:14 EDT 2009
127 Checking for transponder completed successfully!
128 ----------------------------------------------------------------------------------------
129 INFO: Adding command: Check for transponder command
130 INFO: Processing command: Check for transponder command
131 INFO: Registering CheckForTransponder -> Check for transponder command for receiving data
132 INFO: Relaying Received (BYTE) 0x00,(BYTE) 0x1C,(BYTE) 0x00,(BYTE) 0xD6,(BYTE) ↩
    0x63,(BYTE) 0xB9,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
    command]
133 INFO: Deregistering CheckForTransponder -> Check for transponder command for receiving ↩
    data
134 INFO: Removing command: Check for transponder command
135 ------------------------------------------------------------
136 Application received transponders:
137    [1C00D663B900] @ Thu Aug 13 04:55:28 EDT 2009
138 ------------------------------------------------------------
139 INFO: Detected Transponders
140    [1C00D663B900] @ Thu Aug 13 04:55:28 EDT 2009
141 INFO: Adding chain command: Turn on led command
142 INFO: Processing command: Turn on led command
143 INFO: Removing command: Turn on led command
144
145
146 CheckForTransponder Command Result Log:
147 Sending control message [DATA-SET](HEX) 0x08,(HEX) 0x00,(HEX) 0x00,(HEX) ↩
    0x00[/DATA-SET] ...
```

```
148 Control message sent...
149 Awaiting reader reply
150 Received: (BYTE) 0x00,(BYTE) 0x1C,(BYTE) 0x00,(BYTE) 0xD6,(BYTE) 0x63,(BYTE) ↩
        0xB9,(BYTE) 0x00 @ Thu Aug 13 04:55:28 EDT 2009
151 Checking for transponder completed successfully!
152 ----------------------------------------------------------------------------
153 INFO: Adding command: Check for transponder command
154 INFO: Processing command: Check for transponder command
155 INFO: Registering CheckForTransponder -> Check for transponder command for receiving data
156 INFO: Relaying Received (BYTE) 0x00,(BYTE) 0x15,(BYTE) 0x00,(BYTE) 0xBC,(BYTE) ↩
        0x88,(BYTE) 0xDC,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
        command]
157 INFO: Relaying Received (BYTE) 0x00,(BYTE) 0x15,(BYTE) 0x00,(BYTE) 0xBC,(BYTE) ↩
        0x88,(BYTE) 0xDC,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
        command]
158 INFO: Deregistering CheckForTransponder -> Check for transponder command for receiving ↩
        data
159 INFO: Removing command: Check for transponder command
160 INFO: Transponder [1500BC88DC00] detected but filtered out!
161 INFO: Detected Transponders
162    [1500BC88DC00] @ Thu Aug 13 04:55:43 EDT 2009
163 INFO: Adding chain command: Turn on led command
164 INFO: Processing command: Turn on led command
165 INFO: Removing command: Turn on led command
166
167
168 CheckForTransponder Command Result Log:
169 Sending control message [DATA-SET](HEX) 0x08,(HEX) 0x00,(HEX) 0x00,(HEX) ↩
        0x00[/DATA-SET] ...
170 Control message sent...
171 Awaiting reader reply
172 Received: (BYTE) 0x00,(BYTE) 0x15,(BYTE) 0x00,(BYTE) 0xBC,(BYTE) 0x88,(BYTE) ↩
        0xDC,(BYTE) 0x00 @ Thu Aug 13 04:55:43 EDT 2009
173 Checking for transponder completed successfully!
174 ----------------------------------------------------------------------------
175 INFO: Adding command: Check for transponder command
```

```
176 INFO: Processing command: Check for transponder command
177 INFO: Registering CheckForTransponder -> Check for transponder command for receiving data
178 INFO: Relaying Received (BYTE) 0x01,(BYTE) 0x08,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
        0x00,(BYTE) 0x00,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
        command]
179 INFO: Relaying Received (BYTE) 0x01,(BYTE) 0x08,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
        0x00,(BYTE) 0x00,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
        command]
180 INFO: Relaying Received (BYTE) 0x01,(BYTE) 0x08,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
        0x00,(BYTE) 0x00,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
        command]
181 INFO: Relaying Received (BYTE) 0x01,(BYTE) 0x08,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
        0x00,(BYTE) 0x00,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
        command]
182 INFO: Relaying Received (BYTE) 0x01,(BYTE) 0x08,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
        0x00,(BYTE) 0x00,(BYTE) 0x00 to: [CheckForTransponder -> Check for transponder ↩
        command]
183 INFO: Deregistering CheckForTransponder -> Check for transponder command for receiving ↩
        data
184 INFO: Removing command: Check for transponder command
185 Application received an error for command CheckForTransponder: No transponder found!
186
187
188 CheckForTransponder Command Result Log:
189 Sending control message [DATA-SET](HEX) 0x08,(HEX) 0x00,(HEX) 0x00,(HEX) ↩
        0x00[/DATA-SET] ...
190 Control message sent...
191 Awaiting reader reply
192 Received: (BYTE) 0x01,(BYTE) 0x08,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
        0x00,(BYTE) 0x00 @ Thu Aug 13 04:55:54 EDT 2009
193 Received data ignored because it didn't match expected data
194 Received: (BYTE) 0x01,(BYTE) 0x08,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
        0x00,(BYTE) 0x00 @ Thu Aug 13 04:55:54 EDT 2009
195 Received data ignored because it didn't match expected data
196 Received: (BYTE) 0x01,(BYTE) 0x08,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
        0x00,(BYTE) 0x00 @ Thu Aug 13 04:55:55 EDT 2009
```

```
197 Received data ignored because it didn't match expected data
198 Received: (BYTE) 0x01,(BYTE) 0x08,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
        0x00,(BYTE) 0x00 @ Thu Aug 13 04:55:55 EDT 2009
199 Received data ignored because it didn't match expected data
200 Received: (BYTE) 0x01,(BYTE) 0x08,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x00,(BYTE) ↩
        0x00,(BYTE) 0x00 @ Thu Aug 13 04:55:55 EDT 2009
201 Received data ignored because it didn't match expected data
202 Actual receive bytes do not match expected receive bytes
203 Printing expected receive bytes: [DATA-SET](HEX) 0x00,(TO_VARIABLE) 0x??,(TO_VARIABLE) ↩
        0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) 0x??,(TO_VARIABLE) ↩
        0x??[/DATA-SET]
204 Printing last actual receive bytes: (BYTE) 0x01,(BYTE) 0x08,(BYTE) 0x00,(BYTE) ↩
        0x00,(BYTE) 0x00,(BYTE) 0x00,(BYTE) 0x00
205 No transponder found!
206
207
208 CheckForTransponder Command Result Error:
209 No transponder found!
210 INFO: Monitoring for RFIDMania RFIDValidation Application [Phidget 1023] stopped
211 INFO: Disconnected
```