



**RFID
APPLICATION
PROGRAMMING
INTERFACE**

for

DEVELOPERS

DEVELOPER'S GUIDE

RFID Application Programming Interface for Developers Developer's Guide

V2.2

April 11, 2008

© 2008 Sirit Inc., All Rights Reserved. "Sirit", the Sirit Design, "RFID by Sirit", the RFID by Sirit Design and "vision beyond sight" are all trademarks of Sirit Inc. All other trademarks are the property of their respective owners. Specifications are subject to change without notice.

Disclaimer and Limitation of Liability

The content of this manual is for information use only and is subject to change without notice. Sirit assumes no responsibility or liability for any errors or inaccuracies that may appear in this publication. No part of this manual may be reproduced in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of Sirit.

Sirit products are not designed, intended, authorized or warranted to be suitable for life support applications or any other life critical applications which could involve potential risk of death, personal injury, property damage, or environmental damage.

About Sirit

Sirit Inc. designs, develops, manufactures and sells Radio Frequency Identification (RFID) technology. Targeted at a diverse set of markets RFID technology has become a core technology for applications including: electronic toll collection, access control, cashless payment systems, product identification, and supply chain management systems including logistics, warehousing and manufacturing, and asset management.

Head Office - Canada

372 Bay Street, Suite 1100
Toronto, Ontario, M5H 2W9 Canada
Tel: 416.367.1897
Fax: 416.367.1435
Toll Free: 1.800.498.8760
Email: mail@sirit.com

Sirit, Inc. - US

1321 Valwood Parkway, Suite 620
Carrollton, Texas 75006 United States
Tel: 972.243.7208
Fax: 972.243.8034
Toll Free: 1.866.338.9586

Web: www.sirit.com

Preface

How to Use this Manual

This manual is intended as an overview of RAPID. For specific details on individual commands or functions, you should refer to the online RAPID help file (**<application_name>.chm**). This file is located with the same directory as the RAPID DLL. The help file is only installed if you select the option at installation.

Intended audience

This guide is intended for application and RFID system developers who desire a simple, easy-to-use, type safe interface for controlling and managing Sirit RFID readers.

RAPID supports both Java and .NET programming environments and makes extensive use of object-oriented properties. As a result, you should have an understanding of the important concepts of the object-oriented paradigm. In addition to being proficient with these programming environments, you must also understand the following:

- Windows®-based software installation and operation
- Device communication parameters including Ethernet and serial communications
- RFID reader configuration including antenna placement and RF
- Basic digital input/output control

What's in this guide

The RAPID Developer's Guide introduces developers to the process of developing programs to interface with the reader. A brief overview of the architecture is provided along with more detailed views of the Mapping, Data Management, and Discovery Layers.

RAPID Overview – This chapter provides a brief overview of RAPID.

Architecture – This chapter provides an overview of the RAPID architecture.

Mapping Layer – This chapter briefly describes the Mapping Layer.

Data Management Layer – This chapter briefly describes the Data Management Layer.

Discovery Layer – This chapter briefly describes the Discovery Layer.

Conventions used in this manual

The following conventions are used in this manual:

Bold courier font indicates code entered by the user

(values) within parentheses indicate parameters

(values) in italics indicate user defined variables.

<n> indicates a variable number used in a function that can apply to several different devices such as antennas or I/O ports.

NOTES

Important information and other tips are presented in light blue boxes to the left of the applicable section.



WARNING: *Warnings advise the reader that a hazardous condition can be created by a particular action that can cause bodily injury or extreme damage to equipment*



Caution: *Cautions advise the reader that a condition can be created by a particular action that can cause equipment damage or result in equipment operation that violates regulatory requirements.*

Table of Contents

Chapter 1 – RAPID Overview	1
Overview.....	1
System/Software Requirements	1
Installation	1
Chapter 2 – Architecture	2
Architecture Overview	2
Mapping Layer.....	3
Data Management Layer	3
Discovery Layer	3
Chapter 3 – Mapping Layer.....	4
Overview.....	4
BaseCommand class	Error! Bookmark not defined.
Chapter 4 – Data Management Layer.....	5
Overview.....	5
DataManager Class.....	5
Public Instance Constructors	5
Public Instance Properties.....	5
Public Instance Methods	6
ConfigurationManager Class	7
Public Instance Constructors	7
Public Instance Properties.....	7
Public Instance Methods	7
Chapter 5 – Discovery Layer	8
Chapter 6 – Sample Code Segment	10

This page intentionally left blank.

RAPID Overview

Overview

RAPID (RFID Application Programming Interface for Developers) is an API that allows system developers to extend the functionality of Sirit fixed position readers by providing server-side access to the reader's configuration and control functions.

Currently, Sirit fixed position readers support a command and control protocol that provides an interface to the reader's connectivity layer. RAPID provides the application developer with a high level interface to the reader's data layer.

System/Software Requirements

- .NET Framework, Version 2.0 or higher
- Java Runtime, Version 1.3.1 and higher

Installation

RAPID is provided in two versions: .NET and Java. The .NET version is automatically installed when you install the Reader Startup Tool (RST). If you wish to install the Java version or install any of the associated help files, you must select the RADID SDK installation option during RST installation.

Refer to your reader's User's Guide for more information on installing RST.

Architecture

Architecture Overview

The RAPID architecture is divided into three layers to componentize the software and accommodate different interfacing requirements. These layers are as follows:

- Mapping Layer
- Data Management Layer
- Discovery Layer

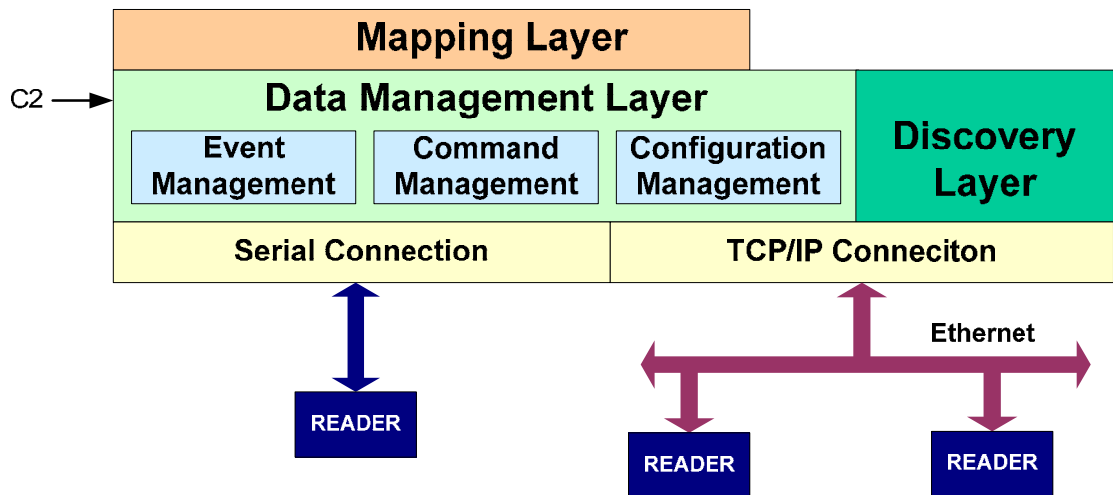


Figure 1 RAPID Architecture

Mapping Layer

The Mapping Layer provides a type safe interface for sending commands to the reader. This layer provides an abstraction layer on top of the reader's protocol commands.

Data Management Layer

This layer handles communication with the reader and presents the reader interface from a data perspective. Use this layer to instantiate the reader and extract data from the reader.

As shown in Figure 1, the Data Management layer has three major components:

- Event Management – Registers and receives event notifications from the reader.
- Command Management – Provides support for sending C2 commands to the reader and parsing responses from the reader.
- Configuration Management – Provides easy to use functions to load and retrieve configurations to and from the reader.

Discovery Layer

This layer returns all the readers found on the TCP/IP network.

Mapping Layer

Overview

The Mapping layer provides a type safe interface to the commands that can be sent to the reader. This layer gives an abstraction layer on top of reader protocol commands. The following section lists the high level classes available on the reader.

Class	Description
EventInfo	Base class for all events.
BaseManager	Base class for <namespace>Manager classes. Contains variables and methods common across <namespace>Manager .
ReaderException	This exception class is raised whenever the reader returns a response other than OK .

All other classes are based on namespaces. Each class has functions to manage each namespace.

Class	Description
AntennasManager	Antenna related features
ComManager	Communication related features
DioManager	Digital IO related features
InfoManager	Information related features
ReaderManager	Reader functions
ModemManager	Modem related features
SetupManager	Setup related features
TagManager	Tag Related features
UserManager	User related features
VersionManager	Version related features

For details on each of class and functions please refer to the compiled help file. The name of the file will match the name of the DLL file for the .NET version **<dll_file.chm>**. For the Java version, the help file is a **.zip** file in **<install_directory>/java**. The file name matches the Java jar file name **<java_jar_file.zip>**.

Data Management Layer

Overview

The Data Management Layer handles communication for the reader and provides an entry point into the reader for most applications. This layer contains the DataManager Class (which handles Event Management and Command Management) and the ConfigurationManager Class.

DataManager Class

The DataManager is a logical connection to the reader and provides functions to communicate with the reader. You can form the connection with any of the base protocols implemented for the readers like serial or sockets. The DataManager helps form the connection to a specific reader via any of the connection protocol implementations.

Public Instance Constructors

The following constructor builds a connection with the reader.

```
public DataManager(ConnectionTypes, string name,
int rate)
```

where:

type – Connection defined in `DataManager.ConnectionTypes`

name - Port name for serial connection type or IP Address for socket connection type

rate – Baud rate for serial connection type or command timeout value for socket connection type

The following constructor creates the Data Manager with a Hash table of parameters. The parameters are based on what type of connection needs to be formed. For example, a TCP based connection would pass the "ipaddress" and type="socket".

```
public DataManager(Hashtable)
```

Public Instance Properties

Property	Description
Live	Live property can be used to check if connection

Public Instance Methods

Method	Description
OpenConnection()	Opens the connection to the reader. Throws a <code>Driver.ConnectionException</code> if an error has occurred while trying to open the connection.
Close()	Closes the connection to the reader. Throws a <code>Driver.ConnectionException</code> if an error has occurred while trying to close the connection.
Send()	Sends a raw C2 command to the reader. Throws a <code>Driver.ConnectionException</code> if an error has occurred while trying to send the command to the reader.
Get()	Returns the value of a configuration variable. Throws a <code>Driver.ConnectionException</code> if an error has occurred while trying to get the configuration variable from the reader. Throws a <code>Mapping.ReaderException</code> if the reader returns an error.
Set()	Sets the value of a configuration variable. Throws a <code>Driver.ConnectionException</code> if an error has occurred while trying to set the configuration variable on the reader. Throws a <code>Mapping.ReaderException</code> if the reader returns an error.
Exec()	Executes a C2 function on the reader. There are two forms of this method, one takes a string as the parameters for the function and the other takes a <code>NameValueCollection</code> as the parameters. Throws a <code>Driver.ConnectionException</code> if an error has occurred while trying to execute the function on the reader. Throws a <code>Mapping.ReaderException</code> if the reader returns an error.
GetEventChannel()	Creates an event channel and returns the id for the event channel.
RegisterEvent()	Registers to receive certain event notifications on an event channel. Throws a <code>Driver.ConnectionException</code> if an error has occurred while trying to register for events.
UnregisterEvent()	Stop receiving certain event notifications on an event channel. Throws a <code>Driver.ConnectionException</code> if an error has occurred while trying to stop event notifications.
UpgradeFirmware()	Upgrades the reader's firmware. Throws a <code>Driver.ConnectionException</code> if an error has occurred while trying to upgrade the reader's firmware.
UploadApp()	Uploads either a Java or python application file to the reader. Throws a <code>Driver.ConnectionException</code> if an error has occurred while trying to upload the application file.

ConfigurationManager Class

The ConfigurationManager provides a snapshot of all the configuration variables on a reader. This class has methods that allow modification of these configuration variables. The configuration may be saved to and read from an XML file. It may also be read from and loaded onto a reader.

Public Instance Constructors

The following constructor builds a construction with the reader

```
public ConfigurationManager(DataManager)
```

Public Instance Properties

Property	Description
ConfigurationProperties	Returns the configuration variables in a NameValueCollection.

Public Instance Methods

Method	Description
GetProperty()	Returns the value of a configuration variable from the local configuration.
SetProperty()	Sets the value of a configuration variable in the local configuration.
GetConfigurationFromReader()	Loads the local configuration with a snapshot of the current configuration on the reader.
LoadConfiguration()	Loads the local configuration from an XML file and then loads it on the reader. Throws a ConnectionException if an error occurs while trying to load the configuration.
SaveConfigurationToFile()	Saves the local configuration to an XML file. Throws a ConnectionException if an error occurs while trying to save the configuration.

Discovery Layer

The Discovery Layer is used to identify all readers on the network. Once reader information is received you can instantiate the Data Manager.

To return a list of all readers on the network, instantiate the **ConnectionDiscovery** class and then call its **findConnections** function. You can register for the **ConnectionFoundEvent** and get events whenever a reader is found. You can also use **ConnectionInfoArgs** to get the latest list of readers.

ConnectionDiscovery also allows you to reconfigure the network settings of the reader over UDP multicasting. This allows you to configure a reader even if you cannot directly connect to it.

Public Instance Constructor

The following is the default constructor for **ConnectionDiscovery**. The default constructor uses the default multicast address **239.192.1.101** to listen for readers on the LAN. The other constructor allows the user to use a different multicast address.

ConnectionDiscovery Constructor

Public Instance Properties

Property	Description
BytesReceived	The number bytes received while listening for readers.
BytesSent	The number of bytes sent as Update packets
ReaderList	Returns a list of readers found so far. Only valid if there is no ConnectionFound specified. This property returns a ConnectionInfoArgsList class which contains a list of ConnectionInfoArgs. This class allows easy stepping through the discovered readers.

Public Instance Delegates

Property	Description
ConnectionFound	Called whenever a reader is found on the network.

Public Instance Methods

Method	Description
findConnections	Starts the discovery process. There are two forms of this method, one uses the default multicast address of "239.192.1.100" to initiate discovery. The other allows for a different multicast address to be used.
Stop	Stops the discovery process.
update	Updates the information for the reader over UDP. There are two forms of this method, one uses the default multicast address of 239.192.1.100 to update a reader. The other allows for a different multicast address to be used.

Public Instance Events

Constructor	Description
ConnectionFoundEvent	ConnectionFoundEvent is thrown whenever a reader is found on the network.

Sample Code Segment

The following code sample is an example of using the Data Management Layer.

```
static void Main(string [] parameters)
{

    DataManager dataManager = null;

    try
    {

        // Create the DataManager and open the connection
        dataManager = new DataManager(DataManager.ConnectionTypes.Socket,
        parameters[0], 0);
        dataManager.OpenConnection();

        // Get Configuration variables - using DataManager
        string zone = dataManager.Get("info.zone");

        // Get Configuration variables - using typesafe class
        ComManager comManager = new ComManager(dataManager);
        string ipAddress = comManager.NetworkIpAddress;
        string subnetMask = comManager.NetworkSubnetMask
        comManager = null;

        // Set Configuration variable - using DataManager
        dataManager.Set("info.zone", "New Zone");
```

```
// Set Configuration variable - using typesafe class
InfoManager infoManager = new InfoManager(dataManager);
infoManager.Location = "New Location";
InfoManager = null;

// Execute a function to the reader - tag.read_id()
// using DataManager
    Hashtable hashResponse = dataManager.Exec("tag.read_id", null);
    if(hashResponse != null)
        Console.WriteLine("Tag ID: " + hashResponse["value"]);

// Execute a function to the reader - tag.read_id()
// using typesafe class
TagManager tagManager = new TagManager(dataManager);
string response = tagManager.ReadId("");
if(response != null)
    Console.WriteLine("Tag ID: " + response);
tagManager = null;

// Create an event channel and register a callback function
string id = dataManager.GetEventChannel(new EventFound(EventReceivedHandler));
// Use the channel id to register for event
if(id != null)
    dataManager.RegisterEvent(id, "event.tag.report");

// Close Connection
dataManager.Close();
dataManager = null;
}
catch(Exception exc)
{
    Console.WriteLine("Error: " + exc.Message);
}
}
```

```
// EventFound delegate is used to notify interested entities whenever an
// event is triggered on the reader
public void EventReceivedHandler(object sender, EventInfo eventInfo)
{
    Console.WriteLine("Event Information: " + eventInfo.Data);

    switch(eventInfo.Type)
    {
        case DIO_OUT_1: // event.dio.out.1
            // Handle the event
            break;
            ...
    }
}
```

Using Discovery Layer

```
static void Main(string [] parameters)
{
    DiscoveryConnection discovery = null;

    try
    {
        // Start the discovery process
        discovery = new ConnectionDiscovery();
        discovery.ConnectionFoundEvent += new ConnectionFound(ConnectionFound);
        discovery.FindConnections();

        // Stop the discovery process
        discovery.Stop();
        discovery = null;
    }
    catch(Exception exc)
    {
        Console.WriteLine("Error: " + exc.Message);
    }
}
```

```
}

// Handles when a reader has been discovered.
public void ConnectionFound(object sender, ConnectionInfoArgs e)
{
    try
    {
        Console.WriteLine("Mac Address: " + e.MacAddress);
        Console.WriteLine("IP Address: " + e.IpAddress);
        Console.WriteLine("Location: " + e.Location);
    }
    catch(Exception exc)
    {
        Console.WriteLine("Error: " + exc.Message);
    }
}
```

This page intentionally left blank.



SIRIT - CANADA
372 Bay Street, Suite 1100
Toronto, ON M5H 2W9 Canada
Tel: 416.367.1897
Fax: 416.367.1435

SIRIT - USA
1321 Valwood Parkway, Suite 620
Carrollton, Texas 75006 USA
Tel: 972.243.7208
Fax: 972.243.8034

For more information
call toll free:
1.800.498.8760 (CA)
1.866.338.9586 (US)