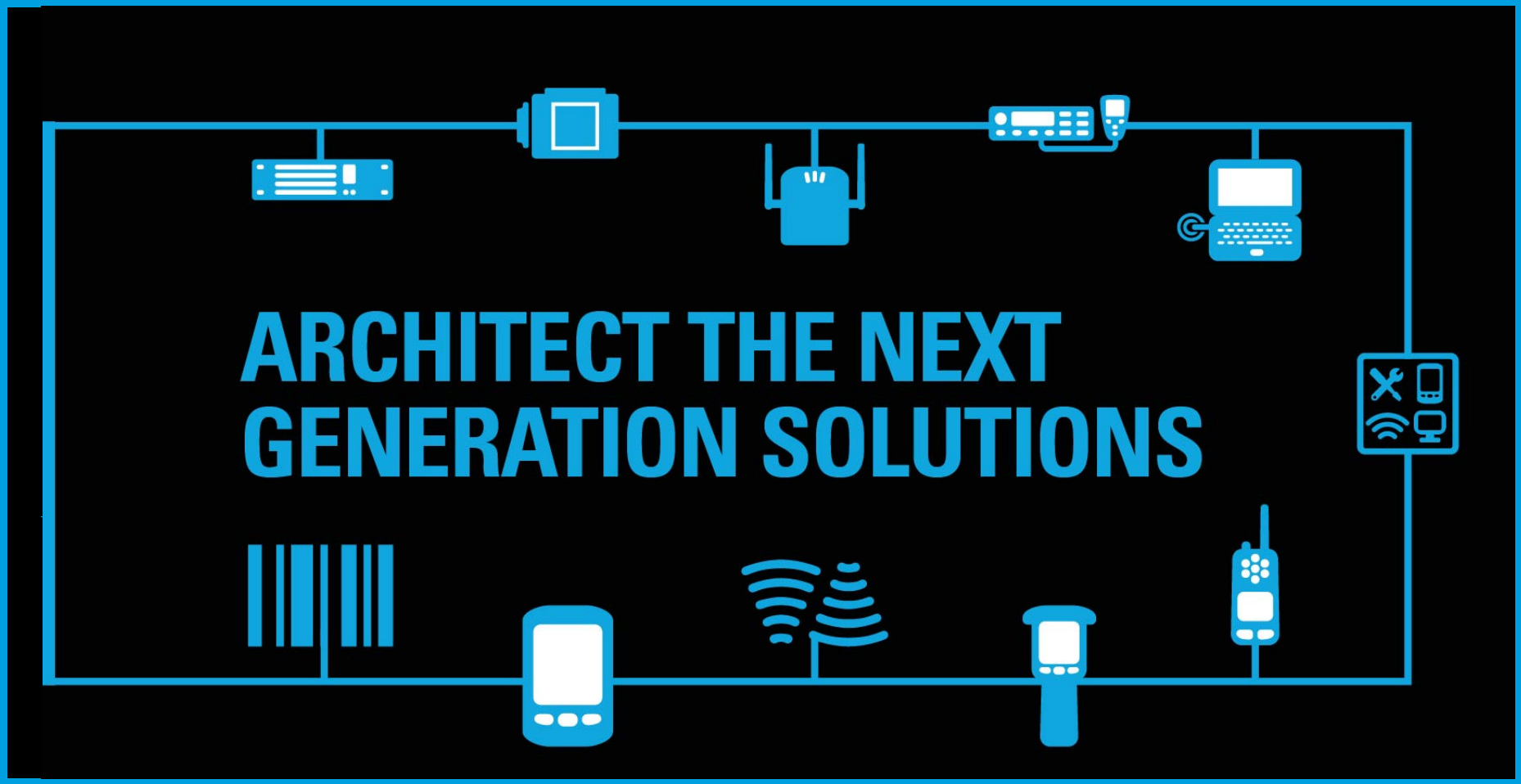


MOTOROLA SOLUTIONS

APPFORUM Americas 2012

Working with Motorola RFID

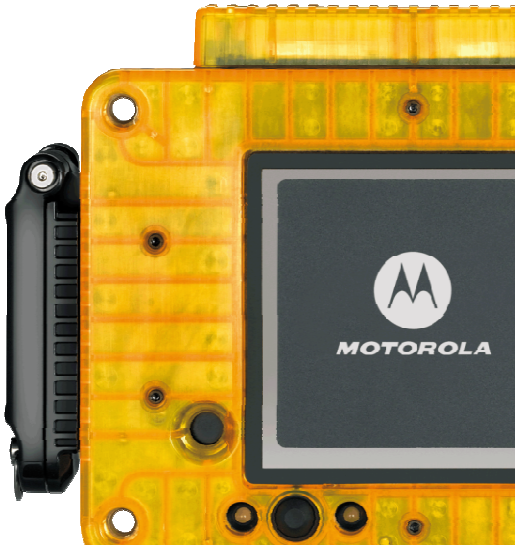
Andy Doorty I Alliance Manager

Adithya Krishnamurthy I Solution Manager

APPFORUM Americas 2012

MOTOROLA SOLUTIONS

- RFID Product Overview
- RFID Architecture
- RFID API3 for .NET
- Information Location
- Best Practice

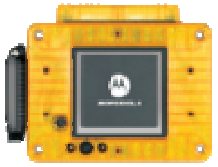


RFID Adoption

RFID Product Suite

Industrial

- Warehousing & Distribution
- Supply Chain
- Manufacturing
- Energy

Fixed	Fixed	Handheld	Mobile
			
FX9500	XR480	MC9090-Z	RD5000

Business

- Retail Inventory Management
- Asset Management
- File tracking
- Healthcare

Fixed	Handheld	Hands-Free
		
FX7400	MC3190-Z	DS9808-R

FX9500

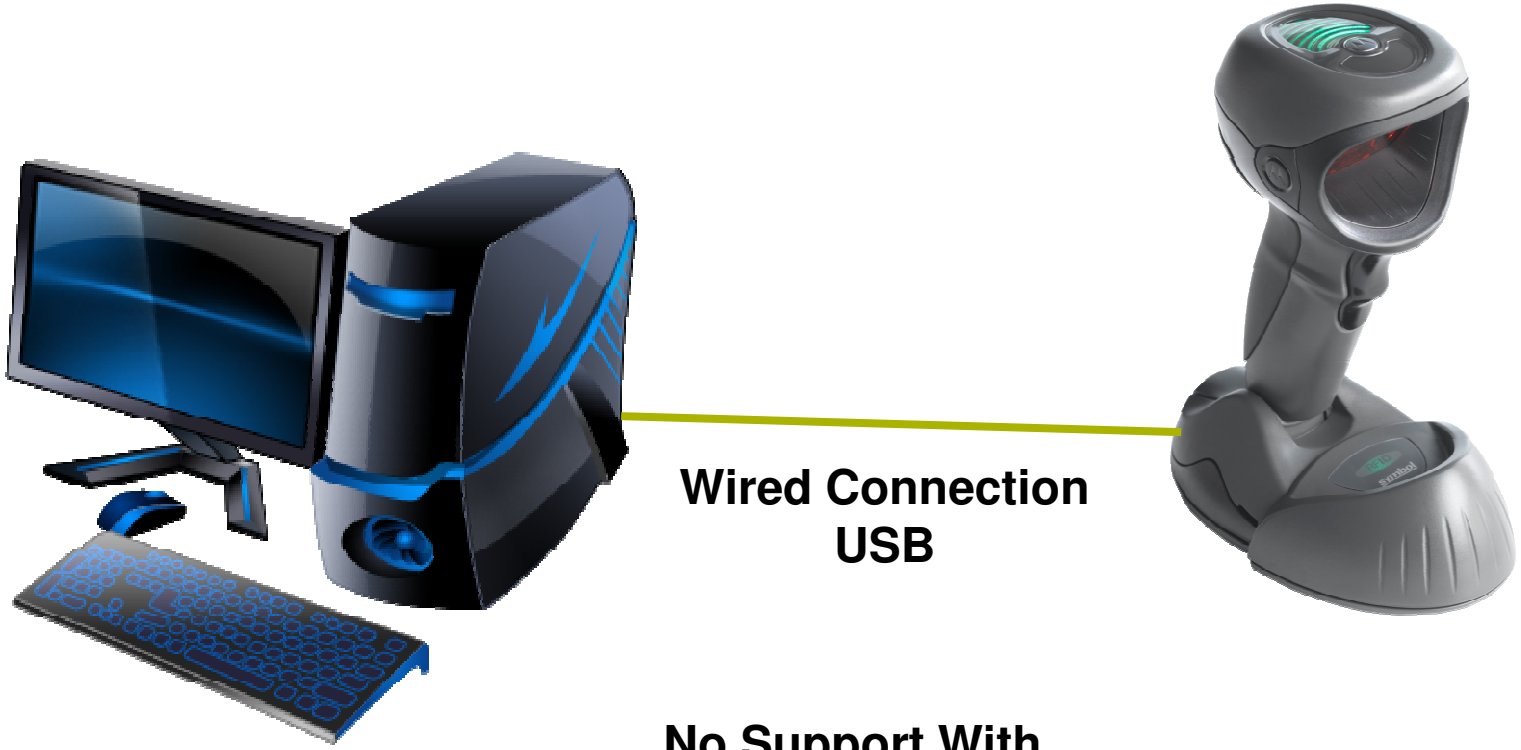
- **INCREASED SENSITIVITY:**
  - **GREATER ACCURACY** for challenging materials like metals and liquids
  - **LONGER READ RANGES** for large DC's and yard management applications
  - **HIGHER THROUGHPUT** rates for reading more, and densely packed goods

- **NEATER, MORE MANAGEABLE FOOTPRINT**
  - All cabling and input/output ports are on one side for a smaller installation footprint and better manageability

Host Based Applications

Non RFID MC

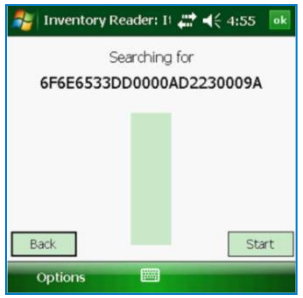
Other Host Based Applications



Wired Connection
USB

No Support With
RFID3 API or LLRP
Interface

Embedded Applications



Mobile Application

Headless Application

LLRP
Low Level Reader Protocol

What is LLRP?

- It stands for Low Level Reader Protocol, defined by the EPCGlobal organization
- It is the open standard protocol that defines communication interface between readers and applications.
- It is a message-oriented protocol
  - Messages from the Client to the Reader include:
    Getting and setting configuration of Readers, reader's capabilities discovery, and managing the inventory and access operations
  - Messages from the Reader to the Client include:
    Reporting of Reader status, inventory and access results, and various events
- It is the standard RFID interface to Motorola mobile as well as Motorola fixed readers

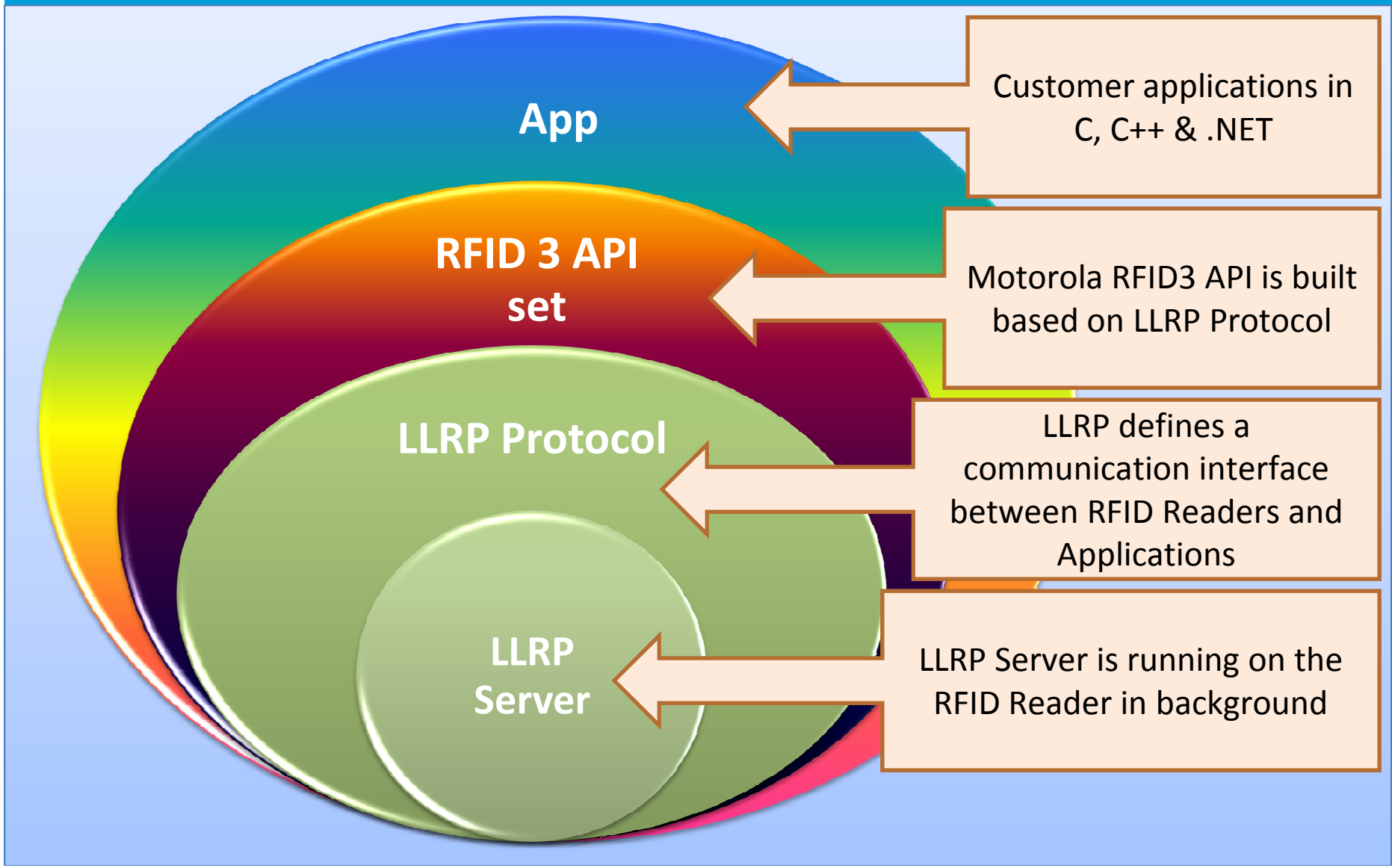
LLRP Protocol

For Reading Enjoyment

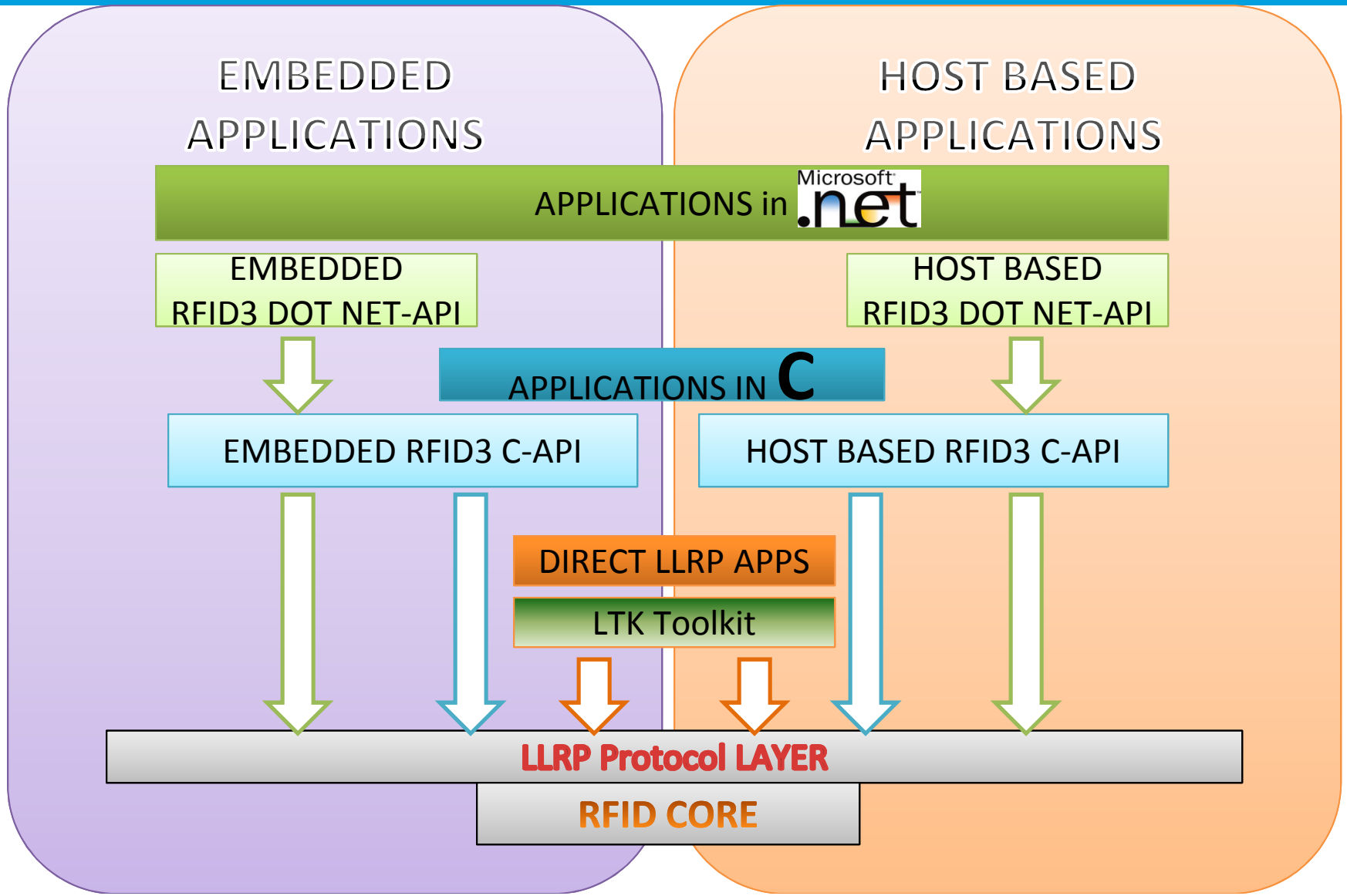
LLRP Specification can be downloaded from EPCGlobal

http://www.gs1.org/gsmp/kc/epcglobal/llrp

Architecture of Motorola RFID System



Architecture of Motorola RFID System



Functionality of Motorola RFID Reader

- Reading Tags
- Writing Tags
- Locating Tags
- Direction of Tags

LLRP Specific Functionality

2 Major Operations

- Reader Operations – Antenna Inventory
- Access Operations – Read, Write Tag's Memory

RFID3 API Interface

- Provide API set for Motorola's LLRP based Fixed & Hand-held readers
  - Fixed Reader: XR Series (LLRP Enabled), FX7400, FX9500
  - Hand-Held Reader: MC3090Z, MC3190Z, MC9090Z
- Support multiple readers access and is thread-safe
- Designed for fast application development to support both simple and advanced Reader Operations.
- Provide a Full-fledged user-friendly interface for
  - Generic reader functionality (over LLRP) , and
  - Reader Manageability (over Http or custom LLRP extensions)

Comparison between using RFID3 API and LTK toolkit

SAME FUNCTIONALITY

Using RFID3 API

Start Inventory

```
// ------------------------------
reader.Actions.Inventory.Perform();
// ------------------------------
```

Stop Inventory

```
// ------------------------------
reader.Actions.Inventory.Stop();
// ------------------------------
```

Using LTK Toolkit

Start Inventory

1. Build RoSpec package
2. Send the following LLRP messages to LLRP Server
   - ADD_ROSPEC
   - ENABLE_ROSPEC
   - START_ROSPEC

Stop Inventory

1. STOP_ROSPEC
2. DISABLE_ROSPEC
3. DELETE_ROSPEC

LESS COMPLEXITY

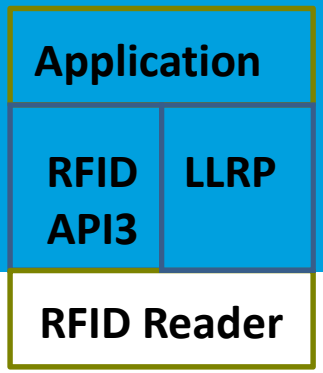
Generic LLRP Reader Interface

- Talks with LLRP based Motorola Readers via LLRP interface
- Allows applications to perform:
  - Connection Management
  - Knowing Reader Capabilities
  - Configuring the Reader
  - Managing Events
  - Managing Tags
  - Setting Filters and Triggers
  - Inventory & Access Operations
  - .....

Reader Management Interface

- Talks over the custom interface as supported by Reader
  - Via xml over http (for XR Series, FX Series)
  - Via custom LLRP extensions (for MC3090Z, MC3190Z, MC9090Z)
- Allows application to perform
  - Login/Logout
  - Get Reader Information
  - Change Antenna Mode (only for XR Series)
  - Enable/Disable antenna port (only for XR Series, FX Series)
  - Update Software/Firmware
  - Import/Export Profiles (only for XR Series, FX Series)
  - Get System Info (only for XR Series, FX Series)
  - ....

Application Choices



APPLICATION

HOST PC

Loopback

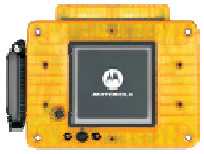
RFID MC

Non RFID MC

READERS

FIXED

HANDHELD



MOBILE

A Simple "Inventory" Code Snippet

```
// Establish connection to the RFID Reader
RFIDReader reader = new RFIDReader("157.235.88.153", 0, 0);
reader.Connect();
```

```
// Specify that TagData should  be reported along with the ReadNotify Event
// Handler
reader.Events.AttachTagDataWithReadEvent = true;
// registering for read tag event notification
reader.Events.ReadNotify += new Events.ReadNotifyHandler(Events_ReadNotify);
```

```
// perform simple inventory
reader.Actions.Inventory.Perform();
// Read for 5 seconds
Thread.Sleep(5000);
// stop the inventory
reader.Actions.Inventory.Stop();

// Disconnects reader and performs clean-up
reader.Disconnect();
```

Programming Tip

Question:

I am using the Motorola Handheld RFID reader to inventory tags and there are over hundreds tags in the FOV. I observe that the tag read rate is quite slow and my application responds slowly as well. Sometime the system even crashes. What could possibly cause this issue?

Answer:

When writing embedded applications on the handheld readers, try NOT to implement computation heavy and complex logic, like frequently update database, play sound, communicate with server. This could have the undesirable effect of crippling the core functionality of the RFID.

Programming Tip

Question:

Does RFID3 API allow user to control the inventory operation via trigger button on Hand-held reader? – Start Inventory when trigger is pulled, and stop inventory when trigger is released

Answer:

Yes. A special trigger type is added in the RFID3 API, which is Handheld trigger. The sample code in the next slide shows how to use handheld trigger to control the inventory operation

Programming Tip

Question:

Can RFID3 API support reading large-capacity tag, Eg. EPC ID > 12 byts, MB size > 64 bytes?

Answer:

Yes. In order to read large-capacity tag, application should change the default API tag storage settings, which are:

Maximum size of EPC Data in Bytes: 12 bytes (96 bits)

Maximum size of tag's Memory Bank: 64 bytes(512 bits)

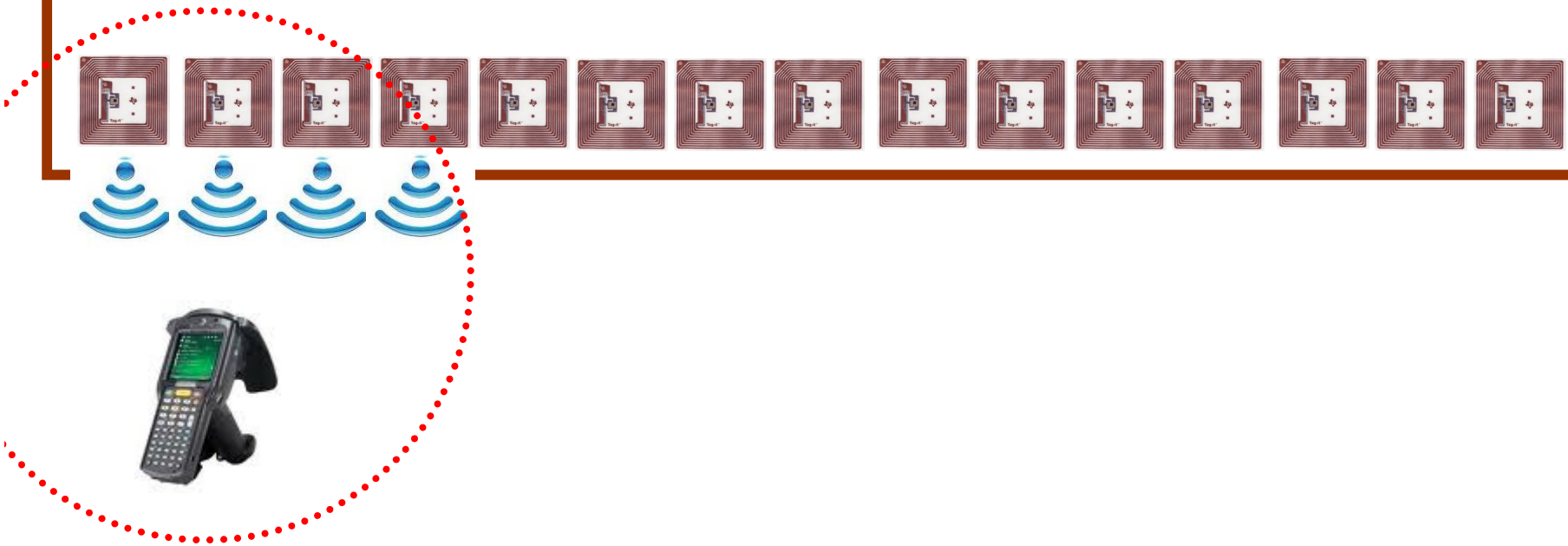
Change TagStorageSettings

```
TagStorageSettings tagStorageSettings = new TagStorageSettings();
tagStorageSettings.MaxTagIDLength = 30;          // New maximum EPC ID: 30 bytes
tagStorageSettings. MaxSizeMemoryBank = 128;    // New maximum memory bank size: 128 bytes

rfid3Reader.Config.SetTagStorageSettings(tagStorageSettings);
```

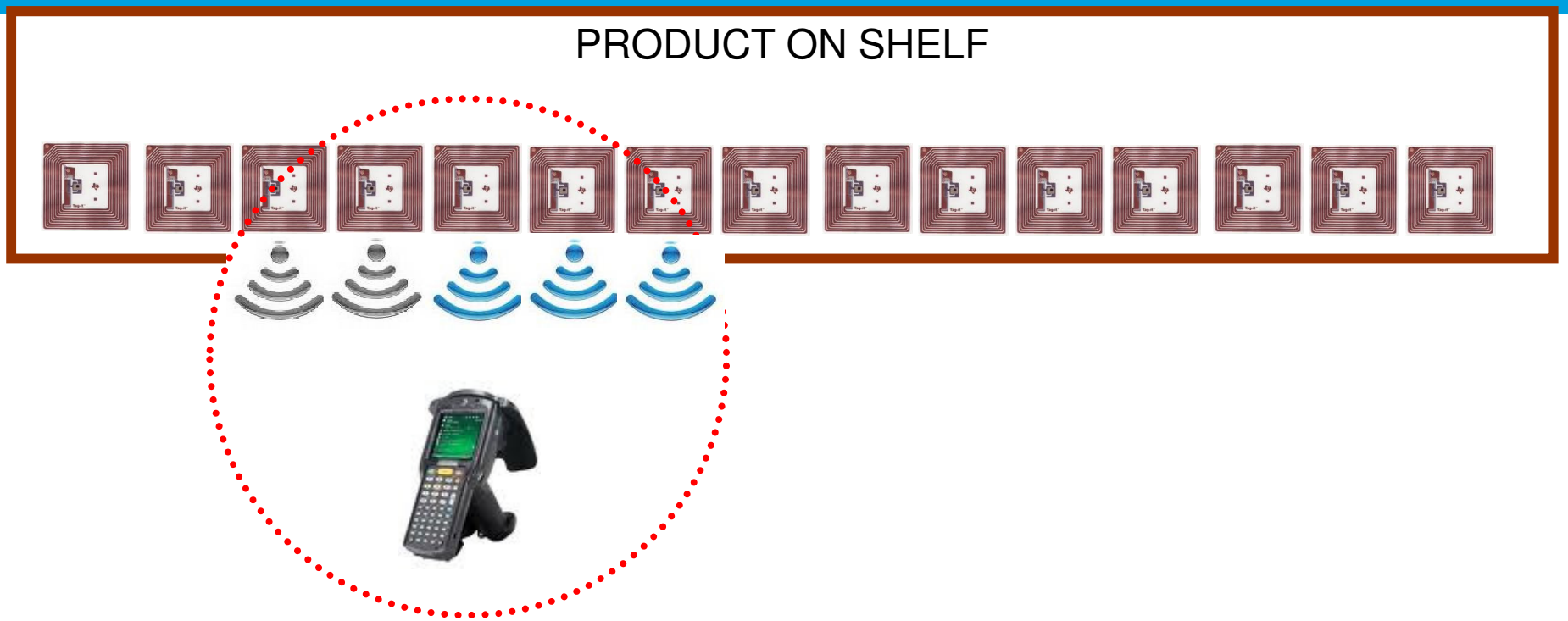
Inventory with no Singulation

PRODUCT ON SHELF

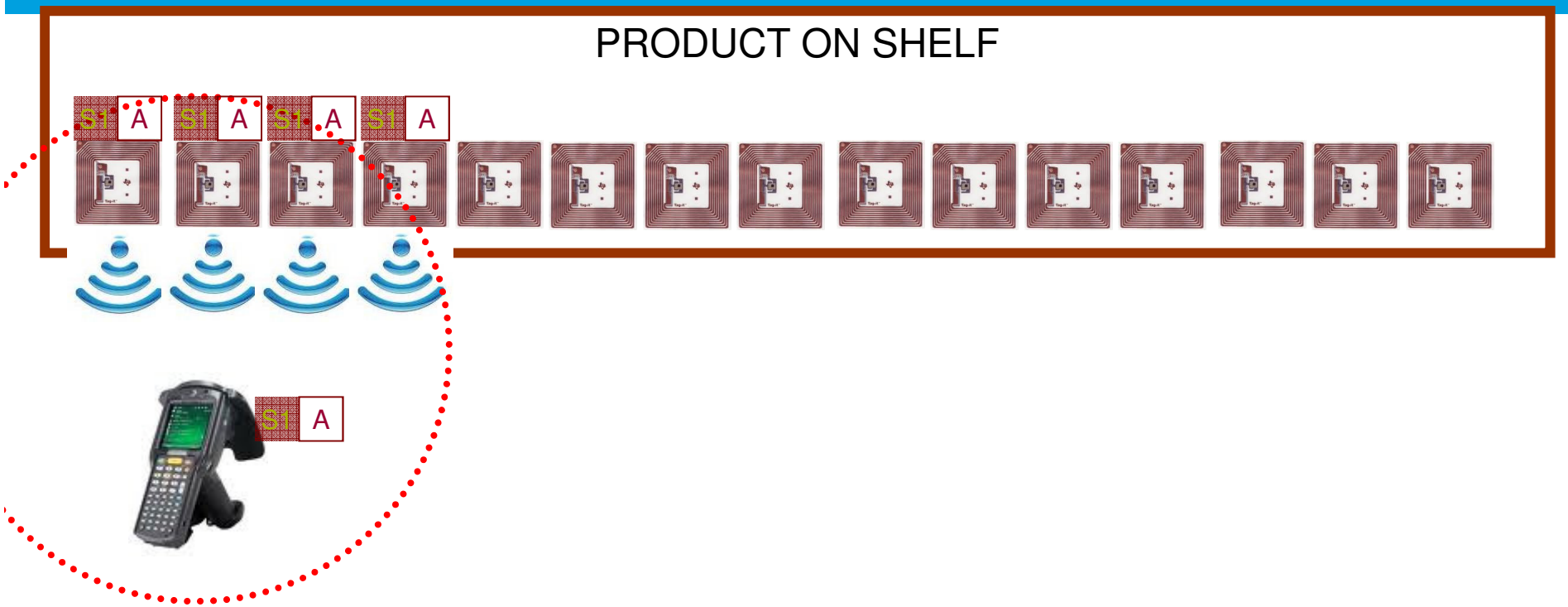


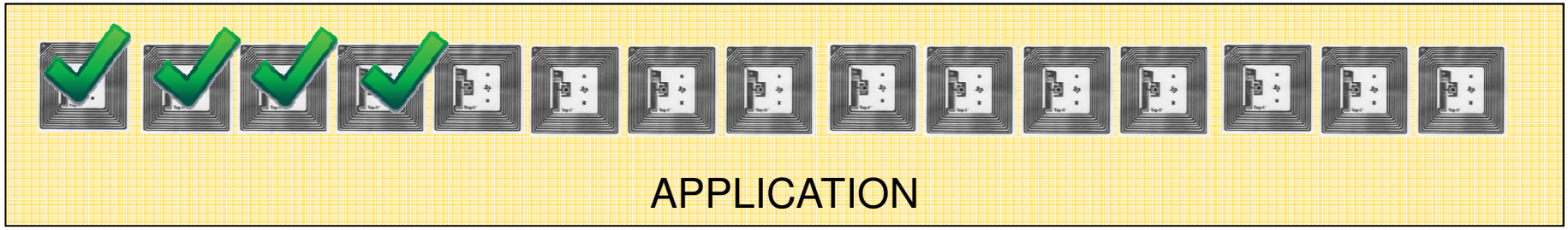
APPLICATION

Inventory with no Singulation

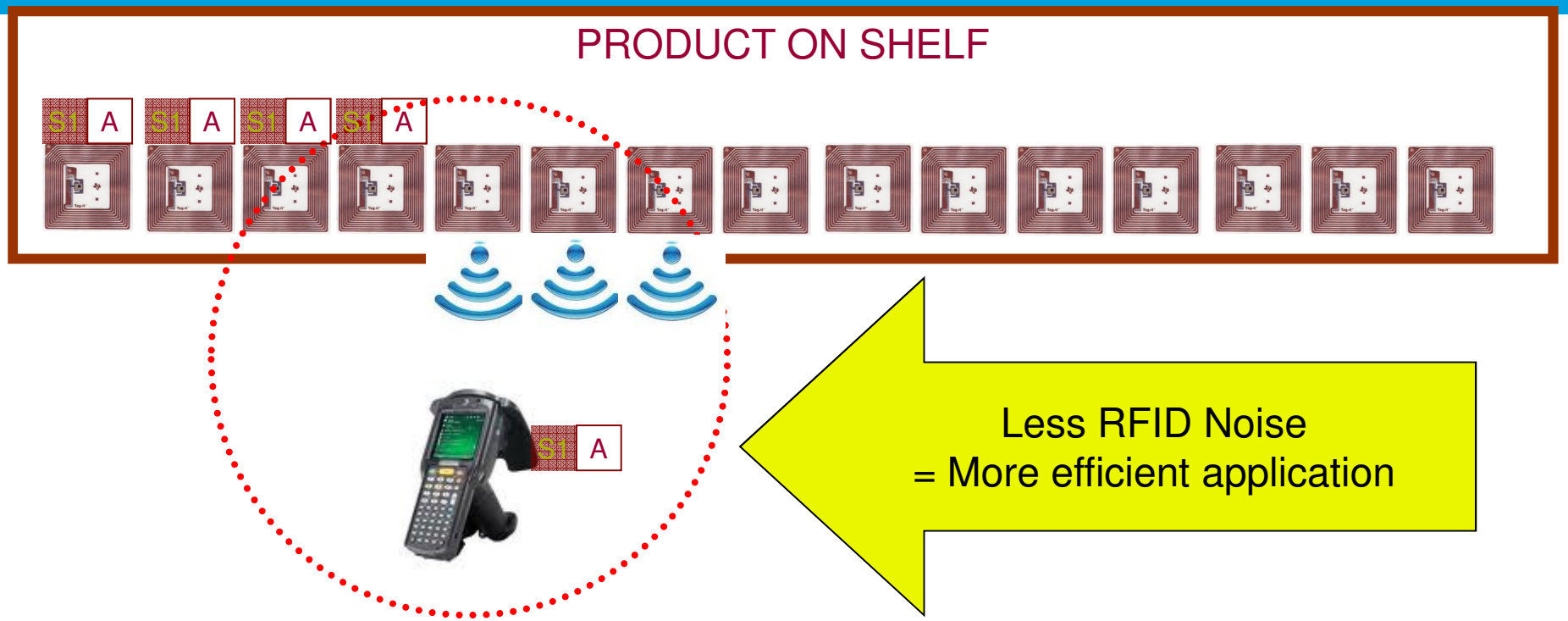


Inventory with Singulation – S1





Inventory with Singulation S1



Sessions

S0	S1	S2	S3
A/B	A/B	A/B	A/B

Tags provide 4 sessions (denoted S0, S1, S2, and S3) and maintain an independent inventoried flag for each session

Singulation

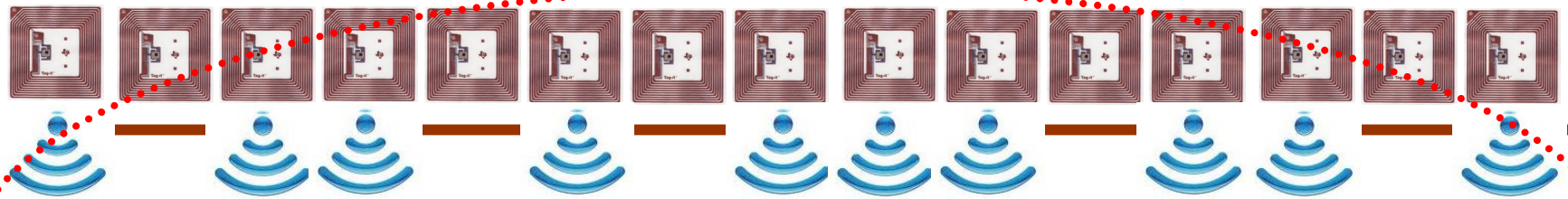
- State Aware – More complex – application handles
- State Unaware – Reader Handles

The following are the steps to use pre-filters:

1. Add pre-filters
   - Check **ReaderCapabilites.MaxNumPreFilters**
   - Use **StateUnawareAction** for tag selection control
2. Set appropriate Singulation controls
   - Session S0 – no select persistency
   - Session S1-S3 – Tag stays ‘selected’ for configurable time
3. Perform Inventory or Access operation

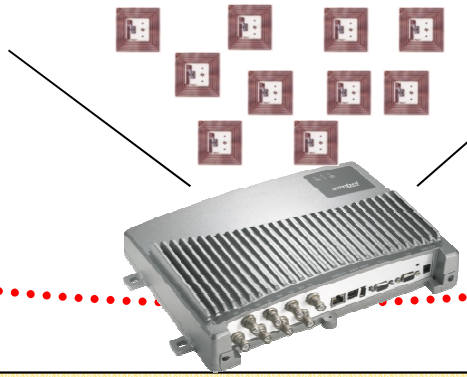
Pre-Filtering

PRODUCT



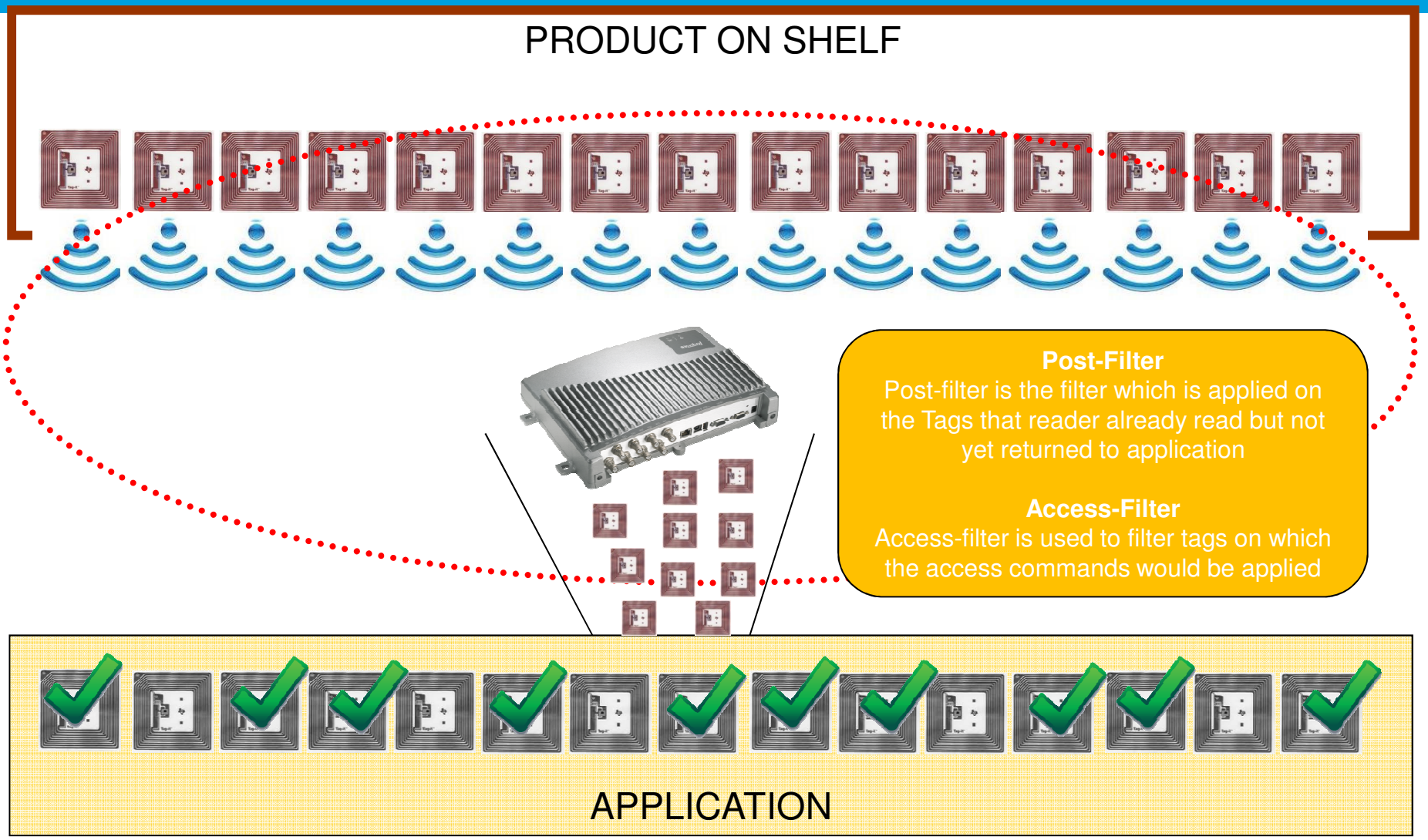
Pre-Filter:
Pre-filter are same as the Select command of C1G2 specification.

Once applied, pre-filters are applied prior to Inventory and Access operations



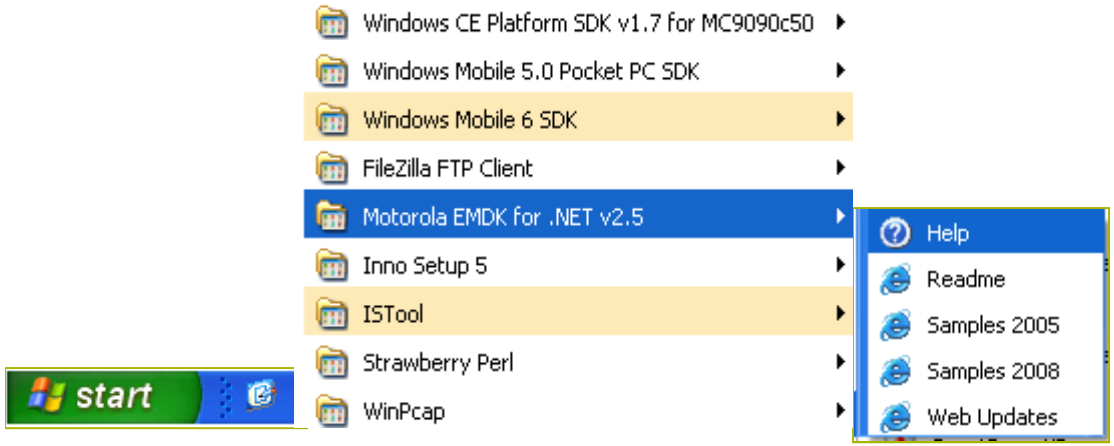
APPLICATION

Post And Access Filters



RFID3 API Interface Download

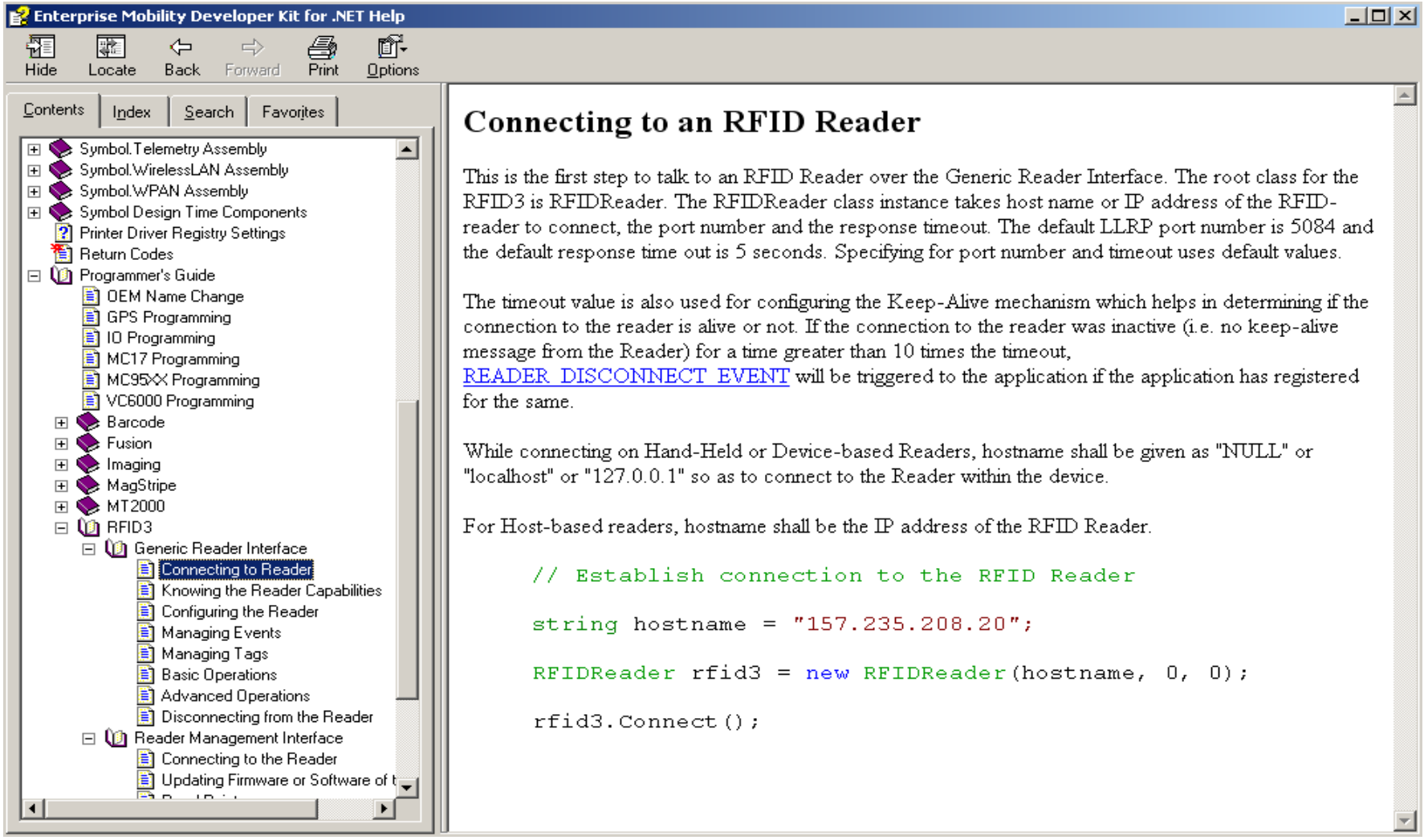
The EMDKs can be downloaded from Symbol support website:
http://support.symbol.com/



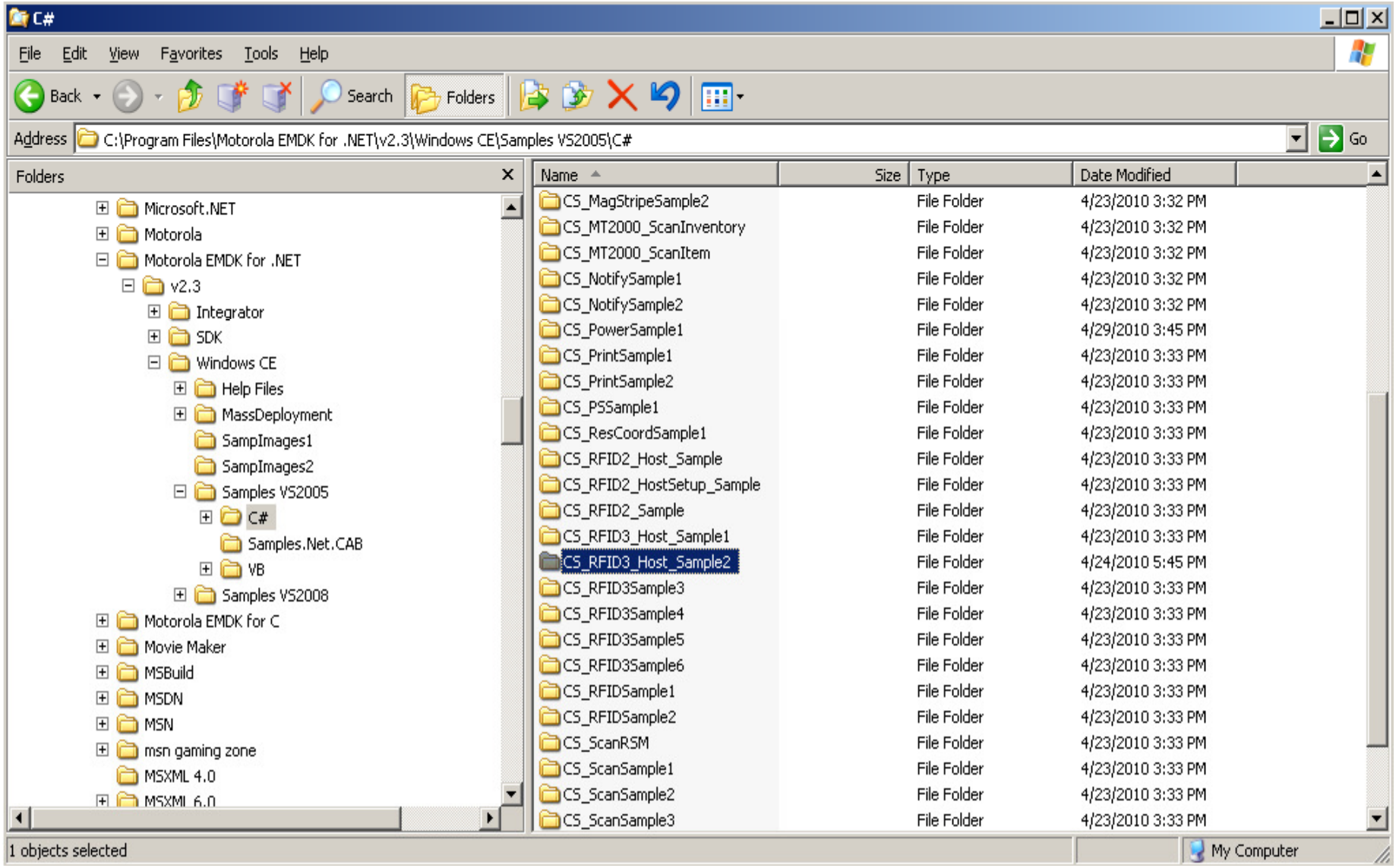
RFID3 API Programmer Guide & RFID3 Sample Applications

API3 for .NET Reference Guide

Start -> Programs -> Motorola EMDK for .NET V2.3 -> Help



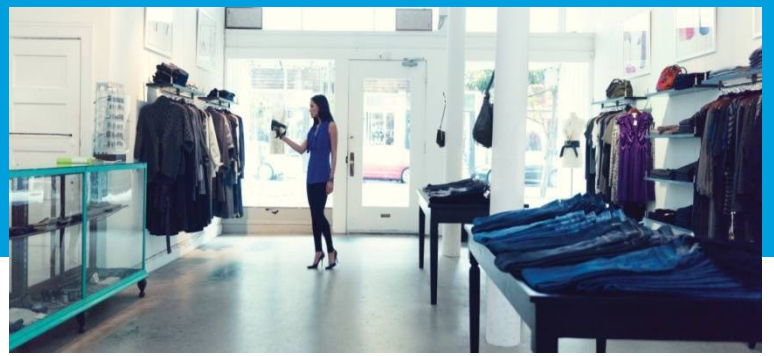
API3 for .NET Sample Application Guide



Samples in VS2005 and VS2008

EMDK samples list			
Samples in .Net	**Samples in C**	**Type of application**	**Target devices**
CS_RFID3_Host_Sample1	BasicRFIDHost1	Host based	FX series, XR(with LLRP)
CS_RFID3_Host_Sample2	RFIDHostSample1	Host based	FX series, XR(with LLRP)
CS_RFID3_Sample3	BasicRFID2	Embedded	FX740x, XR(with LLRP)
CS_RFID3_Sample4	RFIDSample4	Embedded	FX740x, XR(with LLRP)
CS_RFID3_Sample5	BasicRFID1	Embedded	MC3x90-Z and MC9x90-Z
CS_RFID3_Sample6	RFIDSample3	Embedded	MC3x90-Z and MC9x90-Z

MOTOROLA | RFID



On The Floor

QUESTIONS?

THANK YOU

In The Field

In The Warehouse

More Details on RFID3 API

FAQs On Using RFID3 API

Question:

How to efficiently retrieve inventory results from RFID3 API ?

Answer:

The RFID3 API allows users to get notified when tag read events occur. But more importantly, the tag data can be retrieved in two ways:

1. Completely relies on tag events to get tag data
   In this way, associated tag data will be attached to the tag event handler
   provide by application when the tag event occurs
2. Retrieve tag data directly from API tag buffer when the tag event occurs
   To do this, application should explicitly turn off 'ATTACH TAG DATA TO EVENT'

FOR SYSTEMS WHICH SEE HIGH TAG POPULATIONS, WE RECOMMEND USING METHOD #2.

Inventory operations – Method #2

Main reader object = RFIDReader

```
// --Reader declarations -----
RFIDReader reader;
// ------------------------------
```

Initialize and Connect to RFID reader

```
// --Reader initializations -----
reader = new RFIDReader("157.235.88.253", 0, 0);
reader.Connect();
// ------------------------------
```

Disable 'AttachTagDataWithReadEvent'

```
// ------------------------------
reader.Events.AttachTagDataWithReadEvent = false;
// ------------------------------
```

Register for read notifications

```
// ------------------------------
reader.Events.ReadNotify +=
new Events.ReadNotifyHandler(Events_ReadNotify);
// ------------------------------
```

Start polling

```
// ------------------------------
reader.Actions.Inventory.Perform();
// ------------------------------
```

Callback function

```
// ------------------------------
void Events_ReadNotify(object sender,
                       Events.ReadEventArgs e)
{
   TagData[ ] tagData = reader.Actions.GetReadTags(1000);

    if ((tagData != null) && (tagData.Length > 0))
    {
        foreach(TagData t in tagData)
        {
            UpdateListBox(t.TagID+" : "+
                          t.AntennaID.ToString());
        }
    }
}
// ------------------------------
```

Stop polling

```
// ------------------------------
reader.Actions.Inventory.Stop();
// ------------------------------
```

Disconnect from RFID reader

```
// --Reader initializations -----
reader.Disconnect();
// ------------------------------
```

Question:

What is the difference between the synchronous and asynchronous access operation supported by RFID3 API?

Answer:

1. Synchronous access operations, like ReadWait(), WriteWaite(), is performed on a
specific tag. And application will be blocked until the function call returns. It is a safe
and easy way to access a tag, but it does not offer the best performance on speed.
(high reliability, easy to program but low performance on speed)

2.Asynchronous access operation, like ReadEvent(), WriteEvent(), is performed on all tags that are in the field of view of reader's antenna(s) (Broadcasting command). The asynchronous function call issues the access operation then immediately returns, therefore, application should rely on AccessStopEvent to be notified when the access operation completes. It increases the programming complexity but get high performance on speed.

Note: Application can issue asynchronous access operation in conjunction with the

Access operations

Synchronous Access operation

```
// Write user memory bank data
string tagId = "1234ABCD00000000000025B1";

TagAccess.WriteAccessParams writeParams = new TagAccess.WriteAccessParams();

byte[] writeData = new byte[4] {0x11, 0x22, 0x33, 0x44};
writeParams.AccessPassword = 0;
writeParams.WriteDataLength = writeData.Length;
writeParams.MemoryBank = MEMORY_BANK.MEMORY_BANK_USER;
writeParams.ByteOffset = 0;
writeParams.WriteData = writeData;

// antenna Info is null – performs on all antenna
// Application waits until function returns
rfid3.Actions.TagAccess.WriteWait(tagId, writeAccessParams,  null);
```

Access operations

Asynchronous Access operation

```
// Create Event to signify access operation complete
AutoResetEvent AccessComplete =
                        new AutoResetEvent(false);

rfid3.Events.NotifyAccessStopEvent = true;
rfid3.Events.StatusNotify +=
    new Events.StatusNotifyHandler(Events_StatusNotify);
……..
……
// Status Notification event handler
public void Events_StatusNotify(object sender,
                                    Events.StatusEventArgs e)
{
   switch (e.StatusEventData.StatusEventType)
   {
     ……….
     case Events.STATUS_EVENT_TYPE.ACCESS_STOP_EVENT:
          AccessComplete.Set();
          break;
     …….
   }
}
….
….
```

Asynchronous Access operation (Continue)

```
// Write user memory bank data
TagAccess.WriteAccessParams writeParams =
                new TagAccess.WriteAccessParams();
byte[] writeData = new byte[4] { 0xff, 0xff, 0xff, 0xff  };
writeParams.AccessPassword = 0;
writeParams.MemoryBank =
                MEMORY_BANK.MEMORY_BANK_USER;
writeParams.ByteOffset = 0;
writeParams.WriteDataLength = writeData.Length;
writeParams.WriteData = writeData;

AccessComplete.Reset();

// Asynchronous write operation
rfid3.Actions.TagAccess.WriteEvent(writeParams, null, null);

// wait for access operation to complete
AccessComplete.WaitOne();
```

Question:

Can RFID3 API support reading large-capacity tag, Eg. EPC ID > 12 byts, MB size > 64 bytes?

Answer:

Yes. In order to read large-capacity tag, application should change the default API tag storage settings, which are:

Maximum size of EPC Data in Bytes: 12 bytes (96 bits)

Maximum size of tag's Memory Bank: 64 bytes(512 bits)

Change TagStorageSettings

```
TagStorageSettings tagStorageSettings = new TagStorageSettings();
tagStorageSettings.MaxTagIDLength = 30;          // New maximum EPC ID: 30 bytes
tagStorageSettings. MaxSizeMemoryBank = 128;    // New maximum memory bank size: 128 bytes

rfid3Reader.Config.SetTagStorageSettings(tagStorageSettings);
```

Question:

I don't want the same tag getting reported multiple times. Does RFID3 API provide the mechanism to report tag during the inventory only when this is the new tag first seen by reader or this tag's visibility is changed?

Answer:

Yes. RFID3 API implements the autonomous mode. In this mode, applications can subscribe for only new tag event in which case only new tags will get reported, or subscribe the visibility changed event so that tag gets reported only when it is read by different antenna.

Enable autonomous mode and only receive new tags (unique tags)

```
TriggerInfo triggerInfo = new TriggerInfo();

triggerInfo.EnableTagEventReport = true;  // Enable Autonomous mode

// only register new tag event
triggerInfo.TagEventReportInfo.ReportNewTagEvent = TAG_EVENT_REPORT_TRIGGER. IMMEDIATE;
triggerInfo.TagEventReportInfo.ReportTagInvisibleEvent = TAG_EVENT_REPORT_TRIGGER.NEVER;
triggerInfo.TagEventReportInfo.ReportTagBackToVisibilityEvent = TAG_EVENT_REPORT_TRIGGER.NEVER;

rfid3.Actions.Inventory.Perform(null, triggerInfo, null);

// TAG_EVENT reported as part of  TAG_DATA indicates the visibility state of  the Tag.
```

Question:

Does RFID3 API allow user to control the inventory operation via trigger button on Hand-held reader? – Start Inventory when trigger is pulled, and stop inventory when trigger is released

Answer:

Yes. A special trigger type is added in the RFID3 API, which is Handheld trigger. The sample code in the next slide shows how to use handheld trigger to control the inventory operation

Handheld Trigger Type

```
// --TriggerInfo declarations -----
TriggerInfo triggerInfo;
// -------------------------------

// -- Setting up start and stop trigger to be handheld triggers----------
triggerInfo = new TriggerInfo();
triggerInfo.TagReportTrigger = 1; // This will cause tags to be reported to application as soon as they are available

triggerInfo.StartTrigger.Type = START_TRIGGER_TYPE.START_TRIGGER_TYPE_HANDHELD;
triggerInfo.StartTrigger.Handheld.HandheldEvent = HANDHELD_TRIGGER_EVENT_TYPE.HANDHELD_TRIGGER_PRESSED;

triggerInfo.StopTrigger.Type = STOP_TRIGGER_TYPE.STOP_TRIGGER_TYPE_HANDHELD_WITH_TIMEOUT;
triggerInfo.StopTrigger.Handheld.HandheldEvent = HANDHELD_TRIGGER_EVENT_TYPE.HANDHELD_TRIGGER_RELEASED;
triggerInfo.StopTrigger.Handheld.Timeout = 0;   // no timeout setting
// -------------------------------------
```

Start polling

```
// -------------------------------
reader.Actions.Inventory.Perform(null, triggerInfo, null); // the TriggerInfo object is passed in when RFID reading is performed
// -------------------------------
```

Stop polling

```
// -------------------------------
reader.Actions.Inventory.Stop();
// -------------------------------
```

Question:

I already write a non-zero value AccessPassword to a tag. How come I still can access that tag without providing password?

Answer:

Writing a non-zero value AccessPassword to a tag does not mean the tag is protected from the subsequent Read/Write operation. The next thing user needs to do is to perform the lock operation on a specific memory bank or individual passwords of that tag by calling LockWait() and LockEvent() from RFID3 API.

Question:

I already write a non-zero value AccessPassword to a tag and perform the lock operation on it. How come I still can read data from tag's EPC, TID, and User memory bank without providing the access password?

Answer:

C1G2 tag protocol specifies that only tag's Kill & Access password can be protected from both read and write operation; Reading tag's EPC, TID and User memory bank does not require providing the access password even though these memory banks have already been locked. Some special C1G2 tags are able to support the read protection via the extended read command. Eg, NXP tag

Masks and Associated Action Fields

	Kill pwd		Access pwd		EPC memory		TID memory		User memory	
	19	18	17	16	15	14	13	12	11	10
Mask	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write	skip/ write
	9	8	7	6	5	4	3	2	1	0
Action	pwd read/ write	perma lock	pwd read/ write	perma lock	pwd write	perma lock	pwd write	perma lock	pwd write	perma lock

Figure 6.24 – *Lock* payload and usage

Question:

I am using the Motorola Handheld RFID reader to inventory tags and there are over hundreds tags in the FOV. I observe that the tag read rate is quite slow and my application responds slowly as well. Sometime the system even crashes. What could possibly cause this issue?

Answer:

When writing embedded applications on the handheld readers, try NOT to implement computation heavy and complex logic, like frequently update database, play sound, communicate with server. This could have the undesirable effect of crippling the core functionality of the RFID.