Windows® is a trademark of Microsoft Corporation.

CryptoRF® is a trademark of Atmel Corporation.

This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit (http://www.openssl.org).

This product includes software developed by Lee Thomason and Yves Berquin and distributed asTinyXML (http://www.sourceforge.net/projects/tinyxml).

This product includes software developed by Todd Fast, Timothy Wall and Wayne Meissner and distributed as Java Native Access (JNA) (https://github.com/twall/jna).

© Copyrights 2011 ProximaRF Technology Corp. All rights reserved

No part of this manual may be copied or distributed, transcribed, stored in a retrieval system, or translated into any human or computer language, in any form or by any means, electronic, mechanical, magnetic, manual, or otherwise, without the express written permission of ProximaRF Technology Corp.

Authors: Jeroen Spreeuwenberg and Alexandre Naverniouk

ProximaRF Technology Corp.
1740 H. Dell Range Blvd., Unit #406
Cheyenne, WY 82009
U.S.A
Contents

Intro .................................................................................................................................................. 6
Universal RFID API structure ........................................................................................................ 6
RFID Core ......................................................................................................................................... 8
RFID Core functions ........................................................................................................................ 9
  s_create ......................................................................................................................................... 9
  s_close .......................................................................................................................................... 9
  s_getch .......................................................................................................................................... 10
  s_putch .......................................................................................................................................... 10
  s_write .......................................................................................................................................... 11
  s_write_string ............................................................................................................................... 11
  s_getlength ................................................................................................................................... 11
  s_seekp .......................................................................................................................................... 12
  s_tellp .......................................................................................................................................... 12
  s_get_string.................................................................................................................................. 13
RFID .................................................................................................................................................. 13
RFID .NET .......................................................................................................................................... 14
RFID .NET functions ........................................................................................................................ 14
  Call ............................................................................................................................................... 14
  GetError ....................................................................................................................................... 14
  GetParameter .................................................................................................................................. 15
  EncodeHex .................................................................................................................................... 15
  DecodeHex ...................................................................................................................................... 15
RFID .NET Helper classes ............................................................................................................... 16
RFID Java .......................................................................................................................................... 18
RFID Java functions ........................................................................................................................ 18
  Call ............................................................................................................................................... 18
  GetError ....................................................................................................................................... 18
  GetParameter .................................................................................................................................. 19
  EncodeHex .................................................................................................................................... 19
<table>
<thead>
<tr>
<th>Request Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DecodeHex</td>
<td>19</td>
</tr>
<tr>
<td>RFID XML Requests</td>
<td>20</td>
</tr>
<tr>
<td>Structure of Universal RFID API requests and responses</td>
<td>20</td>
</tr>
<tr>
<td>General Requests</td>
<td>21</td>
</tr>
<tr>
<td>Initialize request</td>
<td>21</td>
</tr>
<tr>
<td>GetAPIInfo requests</td>
<td>22</td>
</tr>
<tr>
<td>Sleep request</td>
<td>23</td>
</tr>
<tr>
<td>WakeUp request</td>
<td>23</td>
</tr>
<tr>
<td>Close request</td>
<td>24</td>
</tr>
<tr>
<td>ISO15693 Requests</td>
<td>25</td>
</tr>
<tr>
<td>GetTagInfo request</td>
<td>25</td>
</tr>
<tr>
<td>ReadTag request</td>
<td>26</td>
</tr>
<tr>
<td>WriteTag request</td>
<td>27</td>
</tr>
<tr>
<td>CryptoRF® Requests</td>
<td>28</td>
</tr>
<tr>
<td>GetTagInfo request</td>
<td>28</td>
</tr>
<tr>
<td>ReadTag request</td>
<td>29</td>
</tr>
<tr>
<td>WriteTag request</td>
<td>30</td>
</tr>
<tr>
<td>MLX90129 Requests</td>
<td>32</td>
</tr>
<tr>
<td>GetTagInfo request</td>
<td>32</td>
</tr>
<tr>
<td>ReadRegisterFile request</td>
<td>33</td>
</tr>
<tr>
<td>WriteRegisterFile request</td>
<td>34</td>
</tr>
<tr>
<td>UpdateRegisterFile request</td>
<td>34</td>
</tr>
<tr>
<td>ReadInternalMemory request</td>
<td>35</td>
</tr>
<tr>
<td>WriteInternalMemory request</td>
<td>36</td>
</tr>
<tr>
<td>ReadExternalMemory request</td>
<td>37</td>
</tr>
<tr>
<td>WriteExternalMemory request</td>
<td>38</td>
</tr>
<tr>
<td>ReadInternalDevice request</td>
<td>39</td>
</tr>
<tr>
<td>WriteInternalDevice request</td>
<td>40</td>
</tr>
<tr>
<td>RFID .NET XML Examples</td>
<td>41</td>
</tr>
<tr>
<td>Read data from an ISO15693 compliant RFID tag</td>
<td>41</td>
</tr>
</tbody>
</table>
Write data to a CryptoRF® compliant RFID tag..............................................................................44
Read sensor data from an MLX90129 based RFID sensor tag..........................................................46
RFID .NET Helper Examples.............................................................................................................49
Read data from an ISO15693 compliant RFID tag ........................................................................49
Write data to a CryptoRF® compliant RFID tag.............................................................................50
Read sensor data from an MLX90129 based RFID sensor tag..........................................................51
Demo project ..........................................................................................................................................53
Appendix: Error codes..........................................................................................................................56
**Intro**

RFID stands for Radio Frequency IDentification. RFID tags are small integrated circuits connected to an antenna, which can respond to an interrogating RF signal with simple identifying information, or with more complex signals depending on the size of the integrated circuit.

RFID couplers (also called RFID interrogators, RFID readers or RFID scanners) are electronic devices which can read/write information from/to RFID tags. Universal RFID API (Application Program Interface) is a set of functions which helps programmers to interface with our RFID couplers. The Universal RFID API is system and language agnostic. It is built on a simple, clear, XML based protocol.

This document describes the common structure of the Universal RFID API, its low level core API and the available XML requests.

**Universal RFID API structure**

The Universal RFID API has two levels. First, the so called core API. The core API is written in C++ and it is responsible for the major functionality of the API.

Above this core level there are different platform specific modules, which help to access the core functions from different platforms.
Provided is a Microsoft .NET module. The Microsoft .NET platform supports a wide variety of computer languages, including C#, Programmers can easily access the API from any of the languages supported by Microsoft .NET. In addition to this module we also provide protocol specific (ISO15693, CryptoRF®, MLX90129 and Sensor) Helper classes, designed to speed development even further.

Also provided is a Java module. It provides Java developers with the same functions as the Microsoft .NET module.

The core API can also be accessed directly. It is implemented as a DLL and can be called from any program that can load the DLL into memory and call its functions. RFID_Core.dll can be called directly from C/C++ programs. RFID_Core.h can be found in the Universal RFID API’s C++ folder.
RFID Core

The RFID Core API provides two groups of functions. The first group allows the creation and use of byte streams. The core API also provides the main API call function 'RFID'. This function takes a stream handler as the only parameter and returns a stream handler which represents the response stream.

The RFID Core API is implemented as a dll called RFID_Core.dll. RFID_Core.dll must always be present in the same folder as the executable of the application using the core API.

Calling the core API directly is done as follows:

1. Create new stream
2. Write XML request into the stream
3. Call RFID function, which returns the response stream handle
4. Read the response from the stream
5. Close both streams (request and response)

The C++ sample code below shows these steps:

```cpp
int request, response;
char *xml;

// Create the request stream
request = s_create();

// Create the XML request
xml = "<RFID>
    <command>Initialize</command>
    <device>PRFD</device>
    <port>COM1</port>
</RFID>";

// Write the XML request into the stream
s_write( request, xml, strlen( xml ) );

// Call RFID function and get the response stream handle
response = RFID( request );
```
// Read the XML response
xml = s_get_string( request );

// Parse the response
...

// Close both the request and response streams
s_close( request );
s_close( response );

RFID Core functions

s_create
Function s_create() creates a stream which can be used for RFID request.

Syntax:

int s_create();

Defined:
RFID_Core.h

Returns:
Handle of the stream

Note:
The number of open streams are limited, so do not forget to close and dispose all the streams when finished.

s_close
Function s_close() closes the stream and releases all the memory used.

Syntax:

void s_close( int handler );
Defined:
RFID_Core.h

Parameters:
handler – handle of the open stream

s_getch
Function \textit{s\_getch()} reads one byte from the stream.

\textbf{Syntax:}

\begin{verbatim}
int s_getch( int handler );
\end{verbatim}

Defined:
RFID_Core.h

Parameters:
handler – handle of the opened stream

Returns:
Byte from the stream or -1 if there is no more bytes to r

s_putch
Function \textit{s\_putch()} writes a byte into the stream.

\textbf{Syntax:}

\begin{verbatim}
void s_putch( int handler, char ch );
\end{verbatim}

Defined:
RFID_Core.h

Parameters:
handler – handle of the opened stream
ch – byte to write
s_write
Function **s_write()** writes a bytes array into the stream.

**Syntax:**

```c
void s_write( int handler, char const *data, unsigned int len );
```

**Defined:**
RFID_Core.h

**Parameters:**
- **handler** – handle of the opened stream
- **data** – pointer to the bytes
- **len** – number of bytes to write

s_write_string
Function **s_write_string()** writes an ASCII 0-terminated string into the stream.

**Syntax:**

```c
void s_write_string( int handler, char const * str );
```

**Defined:**
RFID_Core.h

**Parameters:**
- **handler** – handle of the opened stream
- **str** – ASCII 0-terminated string

s_getlength
Function **s_getlength()** returns the number of bytes in the stream.

**Syntax:**

```c
unsigned int s_getlength( int handler );
```
Defined:
RFID_Core.h

Parameters:
handler – handle of the opened stream

Returns:
Total number of bytes in the stream

`s_seekp`
Function `s_seekp()` moves pointer of the stream to specified location in the stream.

Syntax:

```c
unsigned int s_seekp( int handler, unsigned int pos );
```

Defined:
RFID_Core.h

Parameters:
handler – handle of the opened stream
pos – position from the beginning of the stream

Returns:
New position in the stream.

`s_tellp`
Function `s_tellp()` returns current position in the stream.

Syntax:

```c
unsigned int s_tellp( int handler );
```

Defined:
RFID_Core.h

Parameters:
handler – handle of the opened stream

Returns:
Current position in the stream.
s_get_string
Function s_get_string() returns pointer to the internal stream buffer.

Syntax:

```
char *s_get_string( int handler );
```

Defined:
RFID_Core.h

Parameters:
handler – handle of the opened stream

Returns:
Pointer to the internal buffer of the stream. If it is needed the function first adds 0 byte into the end of the stream.

Comments:
Note to be cautious and not to overrun the buffer. You can get the length of the stream by calling s_getlength().

RFID
Function RFID() is the main function of the core API. It processes the XML request and returns the result of the operation. All the XML RFID requests are described later in this document.

Syntax:

```
int RFID( int request_handler );
```

Defined:
RFID_Core.h

Parameters:
handler – handle of the stream with the XML request.

Returns:
Handle of the stream with the XML response.
RFID .NET

As mentioned in the introduction a .NET platform module is provided with the Universal RFID API. This .NET platform module comes in the form of a .NET assembly specially created to help access the Universal RFID API from the Microsoft .NET environment.

The assembly is distributed as RFID_NET.dll and can be found in the Universal RFID API’s .NET folder.

To use the module you must add a reference to the RFID_NET.dll into your .NET project. You can also add the module’s source files directly into your C# .NET project.

To access the module’s functionality in your source files you have to specify usage of the class package ProximaRF.RFID

For example in C#:

```csharp
using ProximaRF.RFID;
```

RFID NET functions

Call

Call is the main function of RFID_NET. The method takes an XmlDocument containing the XML request in and returns an XmlDocument with the XML response.

```csharp
public static XmlDocument Call( XmlDocument req );
```

In addition to the Call function, RFID_NET also has some auxiliary functions which help process the XML result

getError

getError function returns the error code’s value from the XML response

```csharp
public static int GetError( XmlDocument res )
```
**GetParameter**
GetParameter function returns the text value of the specified parameter in the XML response.

```csharp
public static string GetParameter(XmlDocument res, string name);
```

**EncodeHex**
EncodeHex function encodes a byte array into a hexadecimal string.

```csharp
public static string EncodeHex(byte[] data);
```

**DecodeHex**
DecodeHex function decodes a byte array from a hexadecimal string.

```csharp
public static byte[] DecodeHex(string hex);
```
RFID NET Helper classes
In addition to the above functions RFID_NET also contains 4 Helper classes:

- ISO15693Helper
- CryptoRFHelper
- MLX90129Helper
- SensorHelper
  - SensorTagHelper
  - DataLoggerTagHelper
  - SensTagHelper

These helper classes are available from the RFID_NET module. You can also add the helper classes’ source files directly into your C# .NET project. Each helper class allows you to create an object in your source code that is specific to an RFID protocol and/or Proxima RF (sensor) product.

Each helper class provides you with functions that can be called directly from your source code. It hides the Universal RFID API’s XML response and request structure from your code allowing for an even faster implementation. The available functions are explained in great detail using commentary in the available source code. Information is also made available while coding through the Microsoft Visual Studio IntelliSense interface.

Below is an example of how to use the SensorTag Helper class to read the current temperature from a compatible SensorTag passive RFID temperature tag:

```csharp
using ProximaRF.RFID;

private void button1_Click(object sender, EventArgs e)
{
    // Declare member variables
    SensorTagHelper SensorTagh = new SensorTagHelper();

    // Open communication with the Proxima RF Desktop RFID reader
    SensorTagh.Initialize("PRFD", "COM1");
}
```
// Read sensor 0 from a SensorTag to get the temperature
Single sSensor0 = this.SensorTagh.ReadSensor(0);

// Close communication with the Proxima RF Desktop RFID reader
SensorTagh.Close();

// Convert the temperature to Celsius and Fahrenheit
Single sTempC = sSensor0;
Single sTempF = (sTempC * 9 / 5) + 32;

// Show the temperature
MessageBox.Show("sTempC.ToString("#0.0") + " °C\r\n" + sTempF.ToString("#0.0") + " °F", "TEMPERATURE");
}
RFID Java
As mentioned in the introduction a Java platform module is provided with the Universal RFID API. This Java platform module comes in the form of a Java library specially created to help access the Universal RFID API from the Java runtime environment.

The assembly is distributed as RFID_Java.jar and can be found in the Universal RFID API’s Java folder.

To use the module you must include the RFID_Java.jar library into your Java project. You can also add the module’s source files directly into your Java project.

To access the module’s functionality in your source files you have to import the class package ProximaRF.RFID.

```java
import ProximaRF.RFID;
```

RFID Java functions

Call
Call is the main function of RFID_Java. The method takes a String containing the XML request in and returns a String with the XML response.

```java
public static String Call( String req );
```

In addition to the Call function, RFID_Java also has some auxiliary functions which help process the XML result

getError
getError function returns the error code’s value from the XML response

```java
public static int GetError( String res )
```
**GetParameter**

GetParameter function returns the text value of the specified parameter in the XML response.

```csharp
public static string GetParameter( String res, String name );
```

**EncodeHex**

EncodeHex function encodes a byte array into a hexadecimal string.

```csharp
public static String EncodeHex( byte [] data )
```

**DecodeHex**

DecodeHex function decodes a byte array from a hexadecimal string.

```csharp
public static byte [] DecodeHex( String hex )
```
RFID XML Requests
Universal RFID API’s interface is built on an XML (eXtensible Markup Language) based protocol. All requests and responses to/from the API are simple text XML packages. Using XML ensures elimination of platform and language specific aspects.

Structure of Universal RFID API requests and responses
All the XML requests have the same general structure. The XML root element should always be 'RFID'. The body of the XML request can contain multiple different parameters. It should always contain the 'command' parameter.

```xml
<RFID>
    <command>({Command Name})
    {additional parameters}
</RFID>
```

The XML response also has the 'RFID' root element. In addition every response has an 'error-code' parameter (see Appendix). If the 'error-code' parameter is not 0, an 'error-text' parameter is also provided. It contains the matching textual error message.

```xml
<RFID>
    <error-code>{0 if no errors, or some numeric error code}
</error-code>
    <error-text>{text commentary to the error}
</error-text>
    {additional data tags}
</RFID>
```
General Requests
Below is a list of general, non-protocol or device specific Universal RFID API requests and their responses. Each example shows the requests XML structure including the mandatory and optional parameters. Mandatory parameters are indicated with a *.

Initialize request
The Initialize request should be send to the API before any other requests. It opens communication with the RFID coupler and configures it according to the specified parameters.

Request:

```
<RFID>
    <command>Initialize</command>*
    <device>{AP1/AP2/AV-X/PRFD/PRFM}</device>*
    <protocol>{ISO15693/CryptoRF (default is ISO15693)}</protocol>
    <subcarrier>{single/dual (default is dual)}</subcarrier>
    <modulation>{10%/100% (default is 10%)}<modulation>
    <port>{COM1/COM2/COM3/...}</port>*
    <baud>{9200/57600/... (default is 57600)}<baud>
    <parity>{0/1 (default is 0)}</parity>
    <stop-bits>{0/1/2 (default is 0)}</stop-bits>
    <flow-control>{none/hardware (default is none)}</flow-control>
    <mode>{text/binary (default is text)}</mode>
    <field>{auto/manual (default is auto, manual required for MLX90129)}</field>
    <connector>{scannerport/xmod (default is scannerport)}</connector>
    <manufacturer-code>{hexadecimal manufacturer byte to be added into MLX90129 commands (default is 1F)}</manufacturer-code>
    <memory-addressing-bits>{8/16/24 (default is 8)}</memory-addressing-bits>
</RFID>
```
Response:

```xml
<RIFD>
    <version>{API version}</version>
    <error-code>0</error-code>
</RIFD>
```

Notes:
Windows CE/Mobile requires port names to end with ':' (COM3:).

Values of the stop-bits parameter have the following meaning:
- 0 – 1 stop bit
- 1 – 1.5 stop bits
- 2 – 2 stop bits

'connector' parameter only applies when using the AV-X device.

'manufacturer-code' and 'memory-addressing-bits' parameters only apply when using a MLX90129 based (SensorTag) tag.

For more information on 'protocol', 'subcarrier' and 'modulation' refer to the ISO15693 protocol specification available from ISO/IEC and/or the CryptoRF datasheet available from Atmel.

For more information on 'manufacturer-code' and 'memory-addressing-bits' refer to the MLX90129 datasheet available from Melexis.

GetAPIInfo requests
The GetAPIInfo request returns the version number of the currently used API.

Request:

```xml
<RIFD>
    <command>GetAPIInfo</command>*
</RIFD>
```
Response:

```xml
<RFID>
  <version>{API version}</version>
  <error-code>0</error-code>
</RFID>
```

**Sleep request**
The Sleep request will put the RFID coupler into sleep mode and turn off the RF field if the field parameter was set to manual in the Initialize request.

**Request:**

```xml
<RFID>
  <command>Sleep</command>*
</RFID>
```

**Response:**

```xml
<RFID>
  <error-code>0</error-code>
</RFID>
```

**WakeUp request**
The WakeUp request will put the RFID coupler into operating mode and turn on the RF field if the field parameter was set to manual in the Initialize request.

**Request:**

```xml
<RFID>
  <command>WakeUp</command>*
</RFID>
```
Response:

```xml
<RFID>
  <$error-code>0</$error-code>
</RFID>
```

**Close request**
The Close request will close the connection with the RFID coupler.

**Request:**

```xml
<RFID>
  <$command>Close</$command>*
</RFID>
```

**Response:**

```xml
<RFID>
  <$error-code>0</$error-code>
</RFID>
```
ISO15693 Requests
Below is a list of ISO15693 specific Universal RFID API requests and their responses. Each example shows the requests XML structure including the mandatory and optional parameters. Mandatory parameters are indicated with a *.

For more information about the ISO15693 protocol please refer to the ISO15693 specification available from ISO/IEC.

GetTagInfo request
The GetTagInfo request will make the RFID coupler communicate with a compliant RFID tag and will return that tag’s information.

Request:

```
<RFID>
  <command>GetTagInfo</command>*
</RFID>
```

Response:

```
<RFID>
  <tag-type>{Type of the Tag}</tag-type>
  <tag-id>{Unique Tag ID (UID)}</tag-id>
  <error-code>0</error-code>
</RFID>
```
ReadTag request
The ReadTag request will read information, as specified, from a compliant RFID tag’s memory. The amount of available memory depends on the type of the Tag.

Request:

```xml
<RFID>
    <command>ReadTag</command>*
    <tag-type>{Tag type}</tag-type>
    <tag-id>{Tag ID}</tag-id>
    <read-from>{Number (address) of the first byte to read}</read-from>*
    <total-bytes>{Total number (length) of bytes to read}</total-bytes>*
</RFID>
```

Response:

```xml
<RFID>
    <error-code>0</error-code>
    <data>{Data in hexadecimal format}</data>
</RFID>
```

Note:
When the 'tag-type' and 'tag-id' parameters are not present the RFID coupler will communicate with the RFID tag in the unaddressed mode.
WriteTag request

The WriteTag request writes data, as specified to a compliant RFID tag’s memory. The amount of available memory depends on the type of the Tag.

Request:

```xml
<RFID>
  <command>WriteTag</command>*
  <tag-type>{Tag type}</tag-type>
  <tag-id>{Tag ID}</tag-id>
  <write-from>{Number (address) of the first byte to write}</write-from>*
  <total-bytes>{Total number (length) of bytes to write}</total-bytes>*
  <data>{Data in hexadecimal form}</data>*
</RFID>
```

Response:

```xml
<RFID>
  <error-code>0</error-code>
</RFID>
```

Note:

When the 'tag-type' and 'tag-id' parameters are not present the RFID coupler will communicate with the RFID tag in the unaddressed mode.
CryptoRF® Requests
Below is a list of CryptoRF® specific Universal RFID API requests and their responses. Each example shows the requests XML structure including the mandatory and optional parameters. Mandatory parameters are indicated with a *.

For more information about CryptoRF® please refer to the CryptoRF® datasheet available from Atmel.

GetTagInfo request
The GetTagInfo request will make the RFID coupler communicate with a compliant RFID tag and will return that tag’s information.

Request:

```
<RFID>
    <command>GetTagInfo</command>*
</RFID>
```

Response:

```
<RFID>
    <tag-type>{Type of the Tag}</tag-type>
    <tag-id>{Unique Tag ID (UID)}</tag-id>
    <error-code>0</error-code>
</RFID>
```
ReadTag request
The ReadTag request will read information, as specified, from a compliant RFID tag’s memory. The amount of available memory depends on the type of the Tag.

Request:

```xml
<RFID>
  <command>ReadTag</command> *
  <tag-type>{Tag type}</tag-type>
  <tag-id>{Tag ID}</tag-id>
  <read-from>{Number (address) of the first byte to read}</read-from> *
  <total-bytes>{Total number (length) of bytes to read}</total-bytes> *
  <user-zone>{User zone to read from}</user-zone> *
  <password>{Read password associated with specified user zone}</password>
  <password-index>{Index associated with specified password}</password-index>
  <authentication-key>{Authentication key associated with specified user zone}</authentication-key>
  <authentication-key-index>{Index associated with specified authentication key}</authentication-key-index>
  <encryption-key>{Encryption key associated with specified user zone}</encryption-key>
</RFID>
```

Response:

```xml
<RFID>
  <error-code>0</error-code>
  <data>{Data in hexadecimal format}</data>
</RFID>
```

Note:
When the 'tag-type' and 'tag-id' parameters are not present the RFID coupler will communicate with the RFID tag in the unaddressed mode.

For more information on 'user-zone', 'password', 'password-index', 'authentication-key',
'authentication-key-index' and 'encryption-key' parameters please refer to the CryptoRF® datasheet available from Atmel.

WriteTag request
The WriteTag request writes data, as specified to a compliant RFID tag’s memory. The amount of available memory depends on the type of the Tag.

Request:

```
<RFID>
  <command>WriteTag</command>*
  <tag-type>{Tag type}</tag-type>
  <tag-id>{Tag ID}</tag-id>
  <write-from>{Number (address) of the first byte to write}</write-from>*
  <total-bytes>{Total number (length) of bytes to write}</total-bytes>*
  <data>{Data in hexadecimal form}</data>*
  <user-zone>{User zone to read from}</user-zone>*
  <password>{Write password associated with specified user zone}</password>
  <password-index>{Index associated with specified password}</password-index>
  <authentication-key>{Authentication key associated with specified user zone}</authentication-key>
  <authentication-key-index>{Index associated with specified authentication key}</authentication-key-index>
  <encryption-key>{Encryption key associated with specified user zone}</encryption-key>
</RFID>
```

Response:

```
<RFID>
  <error-code>0</error-code>
</RFID>
```

Note:
When the 'tag-type' and 'tag-id' parameters are not present the RFID coupler will communicate with the RFID tag in the unaddressed mode.
For more information on 'user-zone', 'password', 'password-index', 'authentication-key', 'authentication-key-index' and 'encryption-key' parameters please refer to the CryptoRF® datasheet available from Atmel.
MLX90129 Requests
Below is a list of MLX90129 specific Universal RFID API requests and their responses. Each example shows the requests XML structure including the mandatory and optional parameters. Mandatory parameters are indicated with a *.

For more information about the MLX90129 please refer to the MLX90129 datasheet available from Melexis.

GetTagInfo request
The GetTagInfo request will make the RFID coupler communicate with a compliant RFID tag and will return that tag’s information.

Request:

```
<RIFD>
    <command>GetTagInfo</command>*
</RIFD>
```

Response:

```
<RIFD>
    <tag-type>{Type of the Tag}</tag-type>
    <tag-id>{Unique Tag ID (UID)}</tag-id>
    <error-code>0</error-code>
</RIFD>
```
**ReadRegisterFile request**
The ReadRegisterFile request will read information, as specified, from a MLX90129 based tag’s register file.

**Request:**

```
<RFID>
    <command>ReadRegisterFile</command>*
    <tag-type>{Tag type}</tag-type>
    <tag-id>{Tag ID}</tag-id>
    <read-from>{Number (address) of the first byte to read}</read-from>*
    <total-bytes>{Total number (length) of bytes to read}</total-bytes>*
</RFID>
```

**Response:**

```
<RFID>
    <error-code>0</error-code>
    <data>{Data in hexadecimal format}</data>
</RFID>
```

**Note:**
When the 'tag-type' and 'tag-id' parameters are not present the RFID coupler will communicate with the RFID tag in the unaddressed mode.
WriteRegisterFile request
The WriteRegisterFile request writes data, as specified to a MLX90129 based tag’s register file.

Request:

```xml
<RFID>
    <command>WriteRegisterFile</command>*
    <tag-type>{Tag type}</tag-type>
    <tag-id>{Tag ID}</tag-id>
    <write-from>{Number (address) of the first byte to write}</write-from>*
    <total-bytes>{Total number (length) of bytes to write}</total-bytes>*
    <data>{Data in hexadecimal form}</data>*
</RFID>
```

Response:

```xml
<RFID>
    <error-code>0</error-code>
</RFID>
```

Note:
When the 'tag-type' and 'tag-id' parameters are not present the RFID coupler will communicate with the RFID tag in the unaddressed mode.

UpdateRegisterFile request
The UpdateRegisterFile request will update the register file of a MLX90129 based tag.

Request:

```xml
<RFID>
    <command>UpdateRegisterFile</command>*
</RFID>
```
Response:

```xml
<Response>
  <RFID>
    <error-code>0</error-code>
  </RFID>
</Response>
```

**ReadInternalMemory request**

The ReadInternalMemory request will read information, as specified, from a MLX90129 based tag’s internal memory (EEPROM).

**Request:**

```xml
<RFID>
  <command>ReadInternalMemory</command>*
  <tag-type>{Tag type}</tag-type>
  <tag-id>{Tag ID}</tag-id>
  <read-from>{Number (address) of the first byte to read}</read-from>*
  <total-bytes>{Total number (length) of bytes to read}</total-bytes>*
</RFID>
```

Response:

```xml
<Response>
  <RFID>
    <error-code>0</error-code>
    <data>{Data in hexadecimal format}</data>
  </RFID>
</Response>
```

**Note:**

When the 'tag-type' and 'tag-id' parameters are not present the RFID coupler will communicate with the RFID tag in the unaddressed mode.
WriteInternalMemory request
The WriteInternalMemory request writes data, as specified to a MLX90129 based tag’s internal memory (EEPROM).

Request:

```
<RFID>
  <command>WriteInternalMemory</command>*
  <tag-type>{Tag type}</tag-type>
  <tag-id>{Tag ID}</tag-id>
  <write-from>{Number (address) of the first byte to write}</write-from>*
  <total-bytes>{Total number (length) of bytes to write}</total-bytes>*
  <data>{Data in hexadecimal form}</data>*
</RFID>
```

Response:

```
<RFID>
  <error-code>0</error-code>
</RFID>
```

Note:
When the 'tag-type' and 'tag-id' parameters are not present the RFID coupler will communicate with the RFID tag in the unaddressed mode.
**ReadExternalMemory request**

The ReadExternalMemory request will read information, as specified, from a MLX90129 based tag’s external memory. The amount of available memory depends on the external memory used.

**Request:**

```
<RFID>
  <command>ReadExternalMemory</command>*
  <tag-type>{Tag type}</tag-type>
  <tag-id>{Tag ID}</tag-id>
  <read-from>{Number (address) of the first byte to read}</read-from>*
  <total-bytes>{Total number (length) of bytes to read}</total-bytes>*
</RFID>
```

**Response:**

```
<RFID>
  <error-code>0</error-code>
  <data>{Data in hexadecimal format}</data>
</RFID>
```

**Note:**

When the 'tag-type' and 'tag-id' parameters are not present the RFID coupler will communicate with the RFID tag in the unaddressed mode.
**WriteExternalMemory request**

The WriteExternalMemory request writes data, as specified to a MLX90129 based tag’s external memory. The amount of available memory depends on the external memory used.

**Request:**

```
<RFID>
    <command>WriteExternalMemory</command>*
    <tag-type>{Tag type}</tag-type>
    <tag-id>{Tag ID}</tag-id>
    <write-from>{Number (address) of the first byte to write}</write-from>*
    <total-bytes>{Total number (length) of bytes to write}</total-bytes>*
    <data>{Data in hexadecimal form}</data>*
</RFID>
```

**Response:**

```
<RFID>
    <error-code>0</error-code>
</RFID>
```

**Note:**

When the 'tag-type' and 'tag-id' parameters are not present the RFID coupler will communicate with the RFID tag in the unaddressed mode.
**ReadInternalDevice request**

The ReadInternalDevice request will read information, as specified, from a MLX90129 based tag’s internal device.

**Request:**

```
<RFID>
    <command>ReadInternalDevice</command>*
    <tag-type>{Tag type}</tag-type>
    <tag-id>{Tag ID}</tag-id>
    <read-from>{Number (address) of the first byte to read}</read-from>*
    <total-bytes>{Total number (length) of bytes to read}</total-bytes>*
</RFID>
```

**Response:**

```
<RFID>
    <error-code>0</error-code>
    <data>{Data in hexadecimal format}</data>
</RFID>
```

**Note:**

When the 'tag-type' and 'tag-id' parameters are not present the RFID coupler will communicate with the RFID tag in the unaddressed mode.
WriteInternalDevice request
The WriteInternalDevice request writes data, as specified to a MLX90129 based tag’s internal device.

Request:

```xml
<RFID>
  <command>WriteInternalDevice</command>*
  <tag-type>{Tag type}</tag-type>
  <tag-id>{Tag ID}</tag-id>
  <write-from>{Number (address) of the first byte to write}</write-from>*
  <total-bytes>{Total number (length) of bytes to write}</total-bytes>*
  <data>{Data in hexadecimal form}</data>*
</RFID>
```

Response:

```xml
<RFID>
  <error-code>0</error-code>
</RFID>
```

Note:
When the 'tag-type' and 'tag-id' parameters are not present the RFID coupler will communicate with the RFID tag in the unaddressed mode.
RFID .NET XML Examples

This part of the document contains 3 samples on how to use the Universal RFID API. The samples are in C#. Note that you must include RFID.cs or add a reference to RFID_NET.dll to your solution in order to use these samples. Both files can be found in the .NET folder of the Universal RFID API.

Read data from an ISO15693 compliant RFID tag

```csharp
using ProximaRF.RFID;

private void button1_Click(object sender, EventArgs e)
{
    // Declare member variables
    XmlDocument request, response;

    // Open communication with and initiliaze the Proxima RF reader
    // Prepare the XML Initialze request
    request = new XmlDocument();
    request.LoadXml("<RFID>
        <command>Initialize</command>
        <device>PRFD</device>
        <port>COM1</port>
        <protocol>ISO15693</protocol>
    </RFID>);

    // Send the Initilaize request to the API and parse the XML repsonse
    response = RFID.Call(request);
    if (RFID.GetError(response) != 0)
        throw new Exception(RFID.GetParameter(response, "error-text"));

    // Scan for an ISO15693 compliant tag and get its information (UID)
    // Prepare the XML GetTagInfo request
    request = new XmlDocument();
    request.LoadXml("<RFID>
        <command>GetTagInfo</command>
    </RFID>);
```

// Send the GetTagInfo request to the API and parse the XML response
response = RFID.Call(request);

if (RFID.GetError(response) != 0)
    throw new Exception(RFID.GetParameter(response, "error-text"));

// Get the tag info from the response
string tagtype = RFID.GetParameter(response, "tag-type");
string tagid = RFID.GetParameter(response, "tag-id");

// Read 10 bytes of data from an ISO15693 compliant tag starting at address 0
// Prepare the XML ReadTag request
request = new XmlDocument();
request.LoadXml("<RFID>
    <command>ReadTag</command>
    <tag-type>"+tagtype+"</tag-type>
    <tag-id>"+tagid+"</tag-id>
    <read-from>0</read-from>
    <total-bytes>10</total-bytes>
</RFID>");

// Send the ReadTag request to the API and parse the XML response
response = RFID.Call(request);
if (RFID.GetError(response) != 0)
    throw new Exception(RFID.GetParameter(response, "error-text"));

// Get the requested data from the response
byte[] data = RFID.DecodeHex(RFID.GetParameter(response, "data"));

// Parse the data (put your own parsing code here)
MessageBox.Show("Tag ID:" + tagid + "\nData: " + RFID.EncodeHex(data));

// Close communication with the Proxima RF reader
// Prepare the XML Close request
request = new XmlDocument();
request.LoadXml("<RFID>
    <command>Close</command>
  </RFID>");

// Send the Close request to the API and parse the XML response
response = RFID.Call(request);

if (RFID.GetError(response) != 0)
    throw new Exception(RFID.GetParameter(response, "error-text"));
}
Write data to a CryptoRF® compliant RFID tag

```csharp
using ProximaRF.RFID;

private void button1_Click(object sender, EventArgs e)
{
    // Declare member variables
    XmlDocument request, response;

    // Open communication with and initialize the Proxima RF reader
    // Prepare the XML Initialize request
    request = new XmlDocument();
    request.LoadXml("<RFID>
        <command>Initialize</command>
        <device>PRFD</device>
        <port>COM1</port>
        <protocol>CryptoRF</protocol>
    </RFID>");

    // Send the Initialize request to the API and parse the XML response
    response = RFID.Call(request);
    if (RFID.GetError(response) != 0)
        throw new Exception(RFID.GetParameter(response, "error-text"));

    // Write 10 bytes of data to an CryptoRF® compliant tag starting at address 0
    // Prepare the XML WriteTag request
    request = new XmlDocument();
    request.LoadXml("<RFID>
        <command>WriteTag</command>
        <write-from>0</write-from>
        <total-bytes>10</total-bytes>
        <user-zone>0</user-zone>
        <data>0123456789ABCDEF0123</data>
    </RFID>");

    // Send the WriteTag request to the API and parse the XML response
    response = RFID.Call(request);
```
if (RFID.GetError(response) != 0)
    throw new Exception(RFID.GetParameter(response, "error-text"));

    // Close communication with the Proxima RF reader
    // Prepare the XML Close request
    request = new XmlDocument();
    request.LoadXml("<RFID>
        <command>Close</command>
    </RFID>");

    // Send the Close request to the API and parse the XML response
    response = RFID.Call(request);
    if (RFID.GetError(response) != 0)
        throw new Exception(RFID.GetParameter(response, "error-text"));
}
Read sensor data from an MLX90129 based RFID sensor tag

using ProximaRF.RFID;

private void button1_Click(object sender, EventArgs e)
{
    // Declare member variables
    XmlDocument request, response;

    // Open communication with and initiliaze the Proxima RF reader
    // Prepare the XML Initialze request
    request = new XmlDocument();
    request.LoadXml("<RFID>
        <command>Initialize</command>
        <device>PRFD</device>
        <port>COM1</port>
        <protocol>ISO15693</protocol>
        <field>manual</field>
        <manufacturer-code>1F</manufacturer-code>
        <memory-addressing-bits>8</memory-addressing-bits>
    </RFID>");

    // Send the Initilaize request to the API and parse the XML repsonse
    response = RFID.Call(request);
    if (RFID.GetError(response) != 0)
        throw new Exception(RFID.GetParameter(response, "error-text"));

    // Read internal device address 0x06 from an MLX90129 based tag (sensor 0)
    // Prepare the XML ReadInternalDevice request
    request = new XmlDocument();
    request.LoadXml("<RFID>
        <command>ReadInternalDevice</command>
        <read-from>12</read-from>
        <total-bytes>2</total-bytes>
    </RFID>");
// Send the ReadInternalDevice request to the API and ignore the XML response
RFID.Call(request);

// Wait for sensor conversion to complete
// Please refer to chapter 2.5 of the "Sensor Tag Application Note MLX90129"
// available from Melexis to determine the sensor conversion time (sct)
int sct = 1000;
System.Threading.Thread.Sleep(sct);

// Read internal device address 0x02 from an MLX90129 based tag (sensor buffer)
// Prepare the XML ReadInternalDevice request
request = new XmlDocument();
request.LoadXml("<RFID>
    <command>ReadInternalDevice</command>
    <read-from>4</read-from>
    <total-bytes>2</total-bytes>
</RFID>");

// Send the ReadInternalDevice request to the API and parse the XML response
response = RFID.Call(request);
if (RFID.GetError(response) != 0)
    throw new Exception(RFID.GetParameter(response, "error-text"));

// Get the requested data form the response
byte[] ADC = RFID.DecodeHex(RFID.GetParameter(response, "data"));

// Parse the sensor's ADC value (put your own parsing code here)
MessageBox.Show(RFID.EncodeHex(data));

// Close communication with the Proxima RF reader
// Prepare the XML Close request
request = new XmlDocument();
request.LoadXml("<RFID>
    <command>Close</command>
</RFID>");
// Send the Close request to the API and parse the XML response
response = RFID.Call(request);
if (RFID.GetError(response) != 0)
    throw new Exception(RFID.GetParameter(response, "error-text"));
}
RFID .NET Helper Examples

This part of the document contains 3 samples on how to use the Universal RFID API’s .NET Helper classes. The samples are in C#. Note that you must include RFID.cs, ISO15693Helper.cs, CryptoRFHelper.cs, MLX90129Helper.cs and SensorHelper.cs or add a reference to RFID_NET.dll to your solution in order to use these samples. The files can be found in the .NET folder of the Universal RFID API.

Read data from an ISO15693 compliant RFID tag

```csharp
using ProximaRF.RFID;

private void button1_Click(object sender, EventArgs e)
{
    // Declare member variables
    ISO15693Helper ISO15693h = new ISO15693Helper();
    string tagtype, tagid;

    // Initialize the coupler
    ISO15693h.Initialize("AV-X", "COM3:");

    // Get the tag info
    ISO15693h.GetTagInfo(out tagtype, out tagid);

    Read 10 bytes of data from an ISO15693 compliant tag starting at address 0
    byte[] data = ISO15693h.Read(tagtype, tagid, 0, 10);

    // Parse the data (put your own parsing code here)
    MessageBox.Show("Tag ID:" + tagid + "\nData: " + RFID.EncodeHex(data));

    // Close the coupler
    ISO15693h.Close();
}  
```
Write data to a CryptoRF® compliant RFID tag

using ProximaRF.RFID;

private void button1_Click(object sender, EventArgs e)
{
    // Declare member variables
    CryptoRFHelper CryptoRFh = new CryptoRFHelper();
    string tagtype, tagid;

    // Initialize the coupler
    CryptoRFh.Initialize("AV-X", "COM3:");

    // Get the tag info
    CryptoRFh.GetTagInfo(out tagtype, out tagid);

    // Write 10 bytes of data to an CryptoRF® compliant tag starting at address 0
    byte[] data = new byte[10];
    CryptoRFh.Write(tagtype, tagid, data, 0, 100, 0);

    // Close the coupler
    CryptoRFh.Close();
}
Read sensor data from an MLX90129 based RFID sensor tag

```csharp
using ProximaRF.RFID;

private void button1_Click(object sender, EventArgs e)
{
    // Declare member variables
    SensorTagHelper SensorTagh = new SensorTagHelper();
    string tagtype, tagid;
    Single tempC, tempF;

    // Initialize the coupler
    SensorTagh.Initialize("AV-X", "COM3:");

    // Get the tag info
    SensorTagh.GetTagInfo(out tagtype, out tagid);

    // Check the field strength (only accept sensor readings if RF field is strong enough)
    if (SensorTagh.CheckFieldStrength(tagtype, tagid, 2))
    {
        // Read the validated temperature in celcius
        tempC = SensorTagh.ReadSensor(0, tagtype, tagid);
        tempF = (tempC * 9 / 5) + 32;

        // Parse the temperature (put your own parsing code here)
        MessageBox.Show(tempC.ToString("#0.0") + "°C\n" + tempF.ToString("#0.0") + "°F");
    }
}
```
// Close the coupler
SensorTagh.Close();
}
Demo project
Provided with the Universal RFID API is a .NET demo project written in C#. The program dialog has two text fields, one for an XML request and one for an XML response.
You can select one of the predefined XML requests from the 'Commands' menu.

Now click on the Call button, and the program will call the Universal RFID API using the selected XML request. After the call has been completed you will see the API’s XML response in the bottom text field.
The source code for the program is provided. You can find it in the Universal RFID API’s 'demo' folder. You can investigate the code, modify it and test any command with any parameters you want.
### Appendix: Error codes

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFIDE_SYSTEM_ERROR</td>
<td>-1</td>
<td>System error. For more information please see 'error-text' field.</td>
</tr>
<tr>
<td>RFIDE_XML_PARSER_INITIALIZATION</td>
<td>100</td>
<td>Error of initialization of the XML parser.</td>
</tr>
<tr>
<td>RFIDE_REQUEST_PARSING_ERROR</td>
<td>101</td>
<td>Bad XML request</td>
</tr>
<tr>
<td>RFIDE_BAD_PARAMETERS</td>
<td>102</td>
<td>Bad parameters for the command</td>
</tr>
<tr>
<td>RFIDE_HARDWARE_PROBLEM</td>
<td>103</td>
<td>Hardware problems</td>
</tr>
<tr>
<td>RFIDE_LICENSE_PROBLEM</td>
<td>104</td>
<td>Wrong license</td>
</tr>
<tr>
<td>RFIDE_PROTOCOL_ERROR</td>
<td>105</td>
<td>Incorrect sequence of commands</td>
</tr>
<tr>
<td>RFIDE_NOT_ENOUGH_MEMORY</td>
<td>106</td>
<td>Not enough memory to process the command</td>
</tr>
<tr>
<td>RFIDE_SECURITY_ERROR</td>
<td>107</td>
<td>Security error</td>
</tr>
</tbody>
</table>