Developer Guide for Read-a-Card

Read-a-Card software from v3.3.0

Revised April 2019 v1.02

If you need help to set up or use Read-a-Card, beyond what is contained in this Developer Guide and the User Guide, then please contact our support team.

Before you get in touch it would be helpful if you could check which version of Read-a-Card you are using. You will find this on the **About tab**.

Website: **https://readacard.com/support**

Email: **support@read-a-card.com**

Telephone UK and Europe: +44 (0) 1428 685861

Telephone North America and Latin America: +1 888-262-9642 or +1 565-262-9642

If you have any feedback on setting up or using Read-a-Card software or this documentation, then please contact our support team. The product is constantly being reviewed and improved and we value feedback about your experience.

Dot Origin Ltd
Unit 7, Coopers Place Business Park, Combe Lane, Wormley
Godalming GU8 5SZ United Kingdom
+44 (0) 1428 685861

Contents

1 Introduction for Developers

Read-a-Card has various features aimed at software developers, which hide the complexities of the contactless card and reader communication layers so that unique RFID card IDs and other data can be read easily, without prior knowledge of the technologies involved.

You can choose different approaches depending on your needs and experience. Read-a-Card will give you compatibility with the widest range of readers and cards possible. This flexibility should make it easy to integrate smart cards into your solution.

1.1 Controlled by `.ini` file

All settings available in the user interface, including the lock feature, can be enabled by directly editing the `Read-a-Card.ini` configuration text file. In this way your own software can take control of how Read-a-Card operates.

1.2 Web server functionality

A web site or application in the browser can receive the information that Read-a-Card has read from a contactless card or tag, using the local web server functionality that runs on the client.

1.3 Messaging API

Read-a-Card includes a Windows messaging based API (application programming interface), allowing other software applications to be developed that make use of Read-a-Card's functionality. Applications that register to receive Read-a-Card's notification messages will receive the card ID, card type, reader ID (where available) and timestamp, whenever a new card is presented.

1.4 Intended use

Read-a-Card is designed to be configured by an integrator or administrator using its tab-based user interface, or by software changing the configuration files and settings directly. Read-a-Card is then typically run in the background by an end-user, with limited user privileges, to avoid changes to Read-a-Card configuration parameters in normal use. Configuration changes can also be prevented by using a password.

This Developer guide will show you the facilities in Read-a-Card which you can use to integrate card reading seamlessly into third-party applications.

It is assumed you are already familiar with the initial Read-a-Card set up, described in the separate **User guide**.

2 Using the Read-a-Card web server

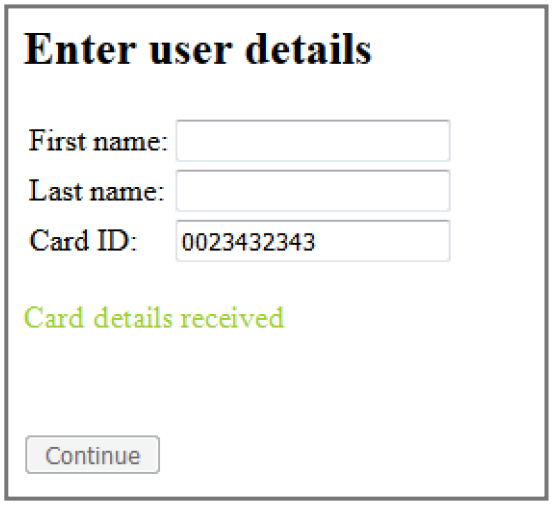
A web site or application in the browser can receive the information that Read-a-Card has read from a contactless card or tag, using the local web server functionality that runs on the client. This is enabled inside Read-a-Card on the **Web Server tab**.

The Read-a-Card built-in web server will then allow your web page, with the addition of some Javascript, to read card information from the Read-a-Card user's web browser. The page can then send this information back to your server application. This allows end-users to submit online forms that are automatically populated with the relevant information from a presented card.

For an example of how to add this ability to your web site, see the **Web page example** in the Appendix. (This example code is available as a file called `Card.htm` installed with the Read-a-Card application, usually in `C:/Program Files (x86)/Read-a-Card`.).

2.1 Web server example

When an end-user presents a card to a reader, with Read-a-Card set up in this way, a web page will be populated with data from the presented card.



Steps to set this up in Read-a-Card:

1. **Card Type tab** - change the default from ☑ All Cards, if you only want Read-a-Card to react to certain card types.
2. **Reader Type tab** - change the default from ◉ All supported PC/SC Contactless Card Readers only if you want to restrict the readers to use.
3. **Action tab** - optionally, select ☑ Play sound to give confirmation a card has been read.
4. **Format tab** - control how data should be read from the card.
5. **Web Server tab** - choose ☑ Enable Read-a-Card web server on this computer and optionally alter the port number from its default and set the domain name of the requesting page, or type `*` to allow any page.

External to Read-a-Card you will need to set up your web page to make use of this card information. The **Web page example** is useful as a starting point.

Versions of the Read-a-Card web server may also be available for Linux and Mac OSX. Please contact **support@read-a-card.com** for more information.

3 Using the Read-a-Card Messaging API

If you are a Windows application developer introducing contactless card technology, you can communicate directly with Read-a-Card through our Windows Messaging API. This is the cleanest way to add Read-a-Card functionality to your software, by receiving Windows messages from Read-a-Card that contain the information you need. You can also send messages back to Read-a-Card to change the actions that it takes, depending on the status of your system.

3.1 Messaging API

To use the Read-a-Card messaging API, your application must register with Read-a-Card by sending a custom Windows message to the Read-a-Card main window. That message provides Read-a-Card with your application's window handle.

Since your application will be assigned a different window handle each time it runs, your application will need to send the register message to Read-a-Card whenever it or Read-a-Card is restarted.

When registered, Read-a-Card will notify your application whenever a card is presented. To do this, Read-a-Card sends a `WM_COPYDATA` message to your application, using the window handle that was provided in the registration step. The `WM_COPYDATA` message is a standard method of transferring data in shared memory between two applications under Windows. The data transferred in this message includes the card ID, card type, reader ID (where available) and timestamp, as 4 unicode character strings.

Ideally, when your application closes or no longer requires the Read-a-Card notification messages, it should send an unregister message to Read-a-Card to stop the notification messages being sent.

3.2 Implementing messaging

This messaging scheme may be implemented using any Windows application development environment that supports the Win32 API. Examples are provided, in Visual C++ using MFC and in VB.NET, for download from **https://readacard.com/support**. The steps involved are described here in more detail to allow other development environments to be used:

To register with Read-a-Card:

1. Find the Read-a-Card window handle. This can be achieved using the `win32api.FindWindow` function to look for a window with the title Read-a-Card.
2. Create a custom message ID using the `win32api.RegisterWindowMessage` function with `READACARD` as the argument.
3. Use `win32api.SendMessage` to send a message to the Read-a-Card window. The message

ID should be the one created in step 2 and the `LPARAM` should be your application's window handle. The `WPARAM` is an 8 bit action mask that tells Read-a-Card what actions it should take itself when a card is presented.

The following is the bitwise representation of the Action Mask:

```
X X X X   X X X X
7 6 5 4   3 2 1 0
```

Bit	Use	Enable	Disable
7	Not Used	N/A	N/A
6	Log Action	0	1
5	URL/HTTP Action	0	1
4	Command Action	0	1
3	Sound Action	0	1
2	Clipboard Action	0	1
1	Keyboard Action	0	1
0	Not used	N/A	N/A

For example, a `WPARAM` of integer value 1 will register with Read-a-Card and leave all functions on, while a value of 3 will register with Read-a-Card and also turn off the keyboard action.

When registered, your application's window handler will receive `WM_COPYDATA` messages from Read-a-Card whenever a card is presented.

The `lpData` member of the received `COPYDATASTRUCT` object can be interpreted as a pointer to a Read-a-Card structure, comprising 4 null-terminated unicode character arrays each having a maximum length of 256 bytes.

The following shows the C++ structure for this data:

```
typedef struct {
      WCHAR cardID[256];
      WCHAR readerSerial[256];
      WCHAR cardType[256];
      WCHAR timeStamp[256];
} READ_A_CARD_DATA;
```

Depending on the development environment of your application, these character arrays may require conversion to be displayed as string types. See the `VB.NET` sample code for an example of this type of conversion.

3.3 Support using the messaging API

We include both the source code and executable versions of two example applications that demonstrate the use of Read-a-Card via the **Messaging API** for download from **https://readacard.com/support**. These have been built using standard Microsoft compilers, and should provide enough information to guide you through creating your own integrated solution.

If you have any queries about these examples, or the API in general, please contact us by email using the address **support@read-a-card.com** although we may not always be able to help with queries about other languages or development environments.

A Web page example

This is an example web page to show what is required to communicate with the Read-a-Card web server. You must enable the Read-a-Card web server from within the Read-a-Card software in order for this to work. This example code is available as a file called `Card.htm` installed with the Read-a-Card application, usually in `C:/Program Files (x86)/Read-a-Card`.

The Javascript code contained in this example page first opens an asynchronous connection with the web server. It then polls the Read-a-Card web server located at the predefined URL and port (once a second) with a GET message. If there is no change in status, the response will be NO CARD otherwise the response message will be in the format: OK,<cardID>,<reader_serial>,<card_type>,<time_stamp>

This message can then be split to extract the required data elements.

In this example page, when a card is detected the card ID is inserted into the appropriate field in an HTML form, and the continue button is enabled.

```
<HEAD>
<script language=Javascript>

var http = CreateHTTP();
var url = "http://localhost:21059/pollcard";
```

```
function CreateHTTP()
{
      var obj;

      if (window.XDomainRequest) {
            // IE8
          obj = new window.XDomainRequest();
      }
      else {
           // Everything else
      obj = new XMLHttpRequest();
      }
      return obj;
}
```

```
function Poll()
{
 /*Avoid browser caching problems by putting a random number on
   each request*/
      var randURL = url + "/" + Math.random();

      if (http) {
            if (window.XDomainRequest) {
                  // IE8
                  http.onload = EnterResult;
                  http.open("GET", randURL, true);
            }
            else {
                  http.open('GET', randURL, true);
                  http.onreadystatechange = StateChangeHandler;
            }
      }

      http.send();
      setTimeout("Poll()", 1000);
}
```

```
function StateChangeHandler(evtXHR)
{
      if (http.readyState == 4) {
          if (http.status == 200) {
              EnterResult();
          }
      }
}
```

```
function EnterResult()
{
 /*document.getElementByID ("debug").innerHTML +=
   "<BR>" + http.responseText;*/
      var values = http.responseText.split(",");
          if (values && values[0] == "OK") {
                  // values[1] is the card ID
                  if (values[1]) {
                  /* Enter the value into the field, change the
                      text and enable the OK button*/
                     document.userform.cardID.value = values[1];
                     Hide("promptforcard");
                     Show("promptcardok");
                     document.userform.okbutton.disabled = false;
                  }
          }
}
```

```
function Hide(object) {
        // sub-function used to hide objects
        document.getElementById(object).style.display='none';
}
```

```
function Show(object) {
        // sub-function used to display objects
        document.getElementById(object).style.display='inline';
}

</script>
```

```
</HEAD>
<BODY onload="Poll();">
<H2>Enter user details</H2>
<FORM name=userform>
```

```
<TABLE>
<TR>
<TD>First name:</TD>
<TD><INPUT type=TEXT name=firstName value=></TD>
</TR>
<TR>
<TD>Last name:</TD>
<TD><INPUT type=TEXT name=lastName value=></TD>
</TR>
<TR>
<TD>Card ID:</TD>
<TD><INPUT type=TEXT name=cardID value=></TD>
</TR>
</TABLE>
```

```
<DIV id=promptforcard style="display:inline;">
<p>Present card to continue
</DIV>
<DIV id=promptcardok style="display:none;">
<p><font color=339933>Card details received</font>
</DIV>
<BR><BR>
<INPUT type=SUBMIT name=okbutton value=Continue disabled=true>
```

```
</FORM>
<DIV id=debug>
</DIV></BODY>
```