Sargas On-Reader Application Development

(Version 2)

This document provides instructions for downloading and running pre-compiled applications that run on the Sargas reader, as well as instructions for creating and compiling your own applications for the Sargas reader.

Contents

Government Limited Rights Notice: All documentation and manuals were developed at private expense and no part of it was developed using Government funds.

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose the technical data contained herein are restricted by paragraph (b)(3) of the Rights in Technical Data — Noncommercial Items clause (DFARS 252.227-7013(b)(3)), as amended from time-to-time. Any reproduction of technical data or portions thereof marked with this legend must also reproduce the markings. Any person, other than the
U.S. Government, who has been provided access to such data must promptly notify ThingMagic.

ThingMagic, A Division of Trimble 1 Merrill St.
Woburn, MA 01801

Revision 2
August 2016

Overview

The Sargas reader contains a full-featured Debian Linux system, which supports all the features of a conventional Linux desktop or server, including

- Standard Debian package manager
    - Software may be installed directly from web sites (debian.org) using apt-get
    - Custom apps are managed and deployed via ".deb" packages
- Onboard, pre-installed gcc compiler
    - No cross-compiler necessary. Out of the box, you can code and build directly on the device itself.
- Broad language support
    - Node.js, Python, Ruby, Perl, gcc, and g++ are pre-installed. Other languages are easily available through the package manager.

These features make it straightforward to develop on-reader applications without any additional tools. This document tells users how to build an on-reader application for the Sargas reader and also walks through an example.

Note that the examples for accessing the Sargas reader over the network use its hostname, (example: "sargas-123456") assuming that there is a DNS server in the network that can resolve the hostname into an IP address. If this does not work, add ".local" to the host name to try to resolve the name via MDNS. If this does not work, substitute the IP address of the Sargas reader in place of the hostname.

Introduction for M6 Programmers

There are two primary differences between Sargas on-reader applications and those written for the Mercury 6 or Astra-EX reader:

1. Creating on-reader programs for the M6/Astra-EX reader requires a cross-compiler that runs on a Linux workstation. The compiler tools for Sargas are present on the Sargas reader itself, and app development can be done on the Sargas platform itself.
2. Applications for the M6/Astra-EX readers had to be written entirely in "C". For Sargas, as long as your program can call a compiled "C" program, you can write your application in a variety of languages. Node.js, Python, Ruby, Perl, gcc, and g++ are pre-installed on the Sargas platform.
3. Sargas provides access to the internal web server pages so the user can add pages that are specific to controlling or reporting from their on-reader application. The M6/Astra-EX web page was bundles as part of the operational code and could not be altered by the user.

Applications written for the M6 and Astra-EX can be re-compiled and run on the Sargas as long as they do not rely on features that Sargas does not support, such as:

- The M6 and Astra-EX support DC-based antenna detection and do not need to have antenna explicitly activated if the antennas have a DC resistance of less than 10 kOhms. The Sargas reader does not support this method of antenna detection, so antennas must be explicitly activated (although their presence can be manually tested by the user's application using RF return loss measurements).
- Sargas and Astra-EX have 2 antenna ports, the M6 has 4
- Sargas has 2 GPI ports and 2 GPO ports. The M6 and Astra-EX have 4 GPI ports and 4 GPO ports.

Downloading Pre-Compiled Applications

Log In

To gain enough access to install arbitrary files, log in to the Sargas as

Username: debian

Password: rootsecure.

From Windows, use an SSH client such as PuTTY (or "screen" for MaxOS or LINUX workstations, for example: "'screen /dev/tty.usbmodem1413 115200")

From a Unix-like OS (e.g., Linux, Mac OS)

```
ssh debian@sargas-name
```

where *sargas-name* is the same name you use to access the Sargas Web UI (be sure to omit http://).

Files Necessary for Creating Own Applications

The first Sargas reader release did not contain all the files necessary to create on-reader applications. Here are the instructions for adding the necessary utilities for on-reader programming. *This is in addition to upgrading to firmware 5.1.2 from 5.1.1 if necessary, as detailed in the firmware release notes.*

Download from ThingMagic Web Site and Install

Download the 4 "*.deb" files from the Thingmagic Support website (http://www.thingmagic.com/manuals-firmware) :

- xsltproc_1.1.26-14.1+deb7u1_armhf.deb
- libtinfo-dev_5.9-10_armhf.deb

- libreadline6-dev_6.2+dfsg-0.1_armhf.deb
- libreadline-dev_6.2+dfsg-0.1_armhf.deb

If you're installing from the command line, download them all and install them all at once (sudo dpkg -i *.deb). The dpkg tool will figure out the correct order to satisfy dependencies.

If you have to install them one at a time (e.g., via the firmware update page on the reader's own web server), the correct order is: libtinfo-dev, libreadline6-dev, libreadline-dev. xsltproc isn't dependent on anything else, so it can go anywhere in the order.)

Download the files from the Debian Web Site

"apt-get" is an alternative, easier way to install them, but it requires that the reader be connected to the Internet, so it can reach the debian.org servers. If the reader has Internet access, run

```
sudo apt-get update
sudo apt-get install libreadline-dev xsltproc
```

If any confirmation prompts appear, answer "yes".

ThingMagic doesn't have to serve anything for these particular packages, since they are part of the standard debian.org libraries.

Compiling Your Own Applications

Build [C] Mercury API

- Log in as the development user (has ability to escalate privileges so apps can be installed to auto-start at boot time.)
  - Username: debian
  - Password: rootsecure
- A Mercury API source code package is preinstalled in the debian user's home directory (/home/debian). Unpack and build.

```
tar xvf sargas-mercuryapi*.tar.gz        # substitute the API version for "*"

cd mercuryapi*/c/src/api                 # substitute the API version for "*"

time make
```

```
# NOTE: "time" is optional and provides details on know how long builds took.  The
first build of Mercury API typically takes 6.5 minutes.  Subsequent builds are faster -- it
doesn't have to rebuild files whose source code hasn't changed.
```

- Run a sample program (which was built along with the API)

```
./read tmr://localhost --ant 1
# This command assumes you have an antenna connected on port 1.
# For different antenna configurations, substitute one of the following commands:
# ./read tmr://localhost --ant 2
# ./read tmr://localhost --ant 1,2
```

- You should see a list of tag reads

```
EPC:AD0C1000136545958F0000CE ant:1 count:1 Time:2016-06-
24T14:51:04.969969+0000
EPC:AD0C100013667195900000EB ant:1 count:1 Time:2016-06-
24T14:51:04.989989+0000
...
```

Modify Code

Try making a small change to the "read" codelet. Let's add the RSSI (Return Signal Strength Indicator) field to the output to get an idea of how strongly tags are responding.

First, take a backup of the original file. This will help in rolling back in case of unforeseen trouble

```
cp -p ../samples/read.c ../samples/read.c.ORIG
```

Here's the change to make. Find the line that prints out the tag read. It's near the end of the file.

```
  printf("EPC:%s ant:%d count:%d Time:%s\n", epcStr, trd.antenna, trd.readCount, timeStr);
```

Add a printout of the RSSI (changes in bold).

```
  printf("EPC:%s rssi:%d ant:%d count:%d Time:%s\n", epcStr, trd.rssi, trd.antenna,
trd.readCount, timeStr);
```

Edit Directly on Sargas

If you are comfortable with text-mode editors, you can edit directly on the Sargas -- vim and nano are pre-installed. (If you're a hardcore command-line devotee, sed is pre-installed, too.)

```
vi ../samples/read.c
OR
nano ../samples/read.c
```

If your Sargas has a connection to the Internet, you can also use the Debian package manager to download other editors such as emacs.

Edit Remotely on Host, Sync to Sargas

Most people will want to edit code on their own machine, where you already have your favorite editor and source control tools.

For Windows, use WinSCP or a similar program to access the Sargas filesystem via the SSH protocol. For WinSCP,

- Create a new session
  - File protocol: SFTP
  - Host name: sargas-*name*
  - Port number: 22
  - User name: debian
  - Click Save, then OK
- Click Login
  - Click Continue when presented with the Authentication Banner
  - Enter the password (rootsecure). Enable Remember password for this session if you want to save some time later. Click OK.
- The right-hand pane (or only pane, if you're in single-pane mode) should be in /home/debian.
  - Click down into mercuryapi-*/c/src/samples/.
  - Scroll down to read.c
    - To use WinSCP's own internal editor, double-click read.c
    - To use your own editor, right-click read.c, select Edit With, and pick your own editor program.

- Whichever editor you use, when you save, WinSCP detects the change and automatically copies the file back to Sargas. This may take a minute if the session login has timed out while you were editing. Then you'll have to wait for the session to be reestablished.

For a Unix-like OS (e.g., Linux, Mac OS) run rsync on your own machine. From a terminal shell,

- Sync the code from Sargas to your machine

```
rsync -aP debian@sargas-name:'mercuryapi*/c/src/samples/' samples/
```

- Edit the code (samples/read.c) with your own favorite editor.
- Sync the code back to Sargas

```
rsync -aP samples/ debian@sargas-name:'mercuryapi*/c/src/samples/'
```

NOTE: To speed up SSH-based access (which includes both SFTP and rsync) look into the following features:

- SSH Keys: Remove the need to enter passwords manually
- SSH ControlMaster (a.k.a. SSH Multiplexing): Share an already-established SSH connection, to avoid the lengthy login delay.

Rebuild Modified Code

- Back in /home/debian/mercuryapi*/c/src/api (You should still be there if your terminal is still open from building the Mercury API.)

```
time make
# NOTE: "time" is optional, but I find it nice to know how long builds took.  If read.c is
the only thing you changed, it should only take about 5 seconds.
```

- Run the read codelet again.

```
./read tmr://localhost --ant 1

# This command assumes you have an antenna connected on port 1.
# For different antenna configurations, substitute one of the following commands:
# ./read tmr://localhost --ant 2
# ./read tmr://localhost --ant 1,2
```

- You should see a list of tag reads with RSSI information.

```
EPC:AD0C100013669D908F0000F0 rssi:-28 ant:1 count:1 Time:2016-06-
27T16:04:08.861861+0000
EPC:AD0C10001366999590000F1 rssi:-41 ant:1 count:1 Time:2016-06-
```

```
27T16:04:08.865865+0000
...
```

How to install Java In Sargas Reader

In the Sargas reader, the processor is ARM71 32 bit . So we need to download the respective package from official java download website(oracle).

1. Download latest version of java from oracle website.
   URL: link to download
2. Then copy downloaded file into SARGAS.
3. Untar the file .
   cmd :tar xvf *<location to the file>*
   example :

   ```
   tar xvf jdk-8u101-linux-arm32-vfp-hflt.tar.gz
   ```
4. Then add java installed location /etc/profile file like below
   example :

   ```
   export JAVA_HOME=/home/debian/jdk1.8.0_101
   export PATH=$PATH:$JAVA_HOME/bin
   ```
   Then save the file and reboot the reader
5. Then check java installed or not with below commands
   cmd:

   ```
   which java, java -version, java , javac
   ```

How to run an On Reader Java Application

1. When Mercury API source code SDK is not available on the reader

a. Download the latest mercuryapi available from the ThingMagic website using the below link:

http://www.thingmagic.com/index.php/manuals-firmware#Mercury_API

b. Extract the API SDK .

c. Copy the “ltkjava-1.0.0.6.jar” , “mercuryapi.jar” from the /SDK/Java/ to a directory(xx) on the reader via SCP.

d. Copy the required sample application from /SDK/Java/samples_nb/src/samples (ex: read.java) to the same directory(xx) on the reader via SCP.

e. Log in as the development user (has ability to escalate privileges so apps can be installed to auto-start at boot time.)

- Username: debian
- Password: rootsecure

f. In the terminal window, change the directory to xx . Edit the sample java application (example: read.java) and comment out the line “PACKAGES=samples;”. Save the file.

g. Compile the application using the following command.

- javac -cp .:ltkjava-1.0.0.6.jar:mercuryapi.jar <sampleapplication.java>

(Example: for “read.java” sample application, send below command to compile)

- javac -cp .:ltkjava-1.0.0.6.jar:mercuryapi.jar read.java

h. Run the application using following command.

- java -cp .:ltkjava-1.0.0.6.jar:mercuryapi.jar <sampleapplication> <comport> <--ant 1,2>

(Example: for “read.java” sample application, send below command to run)

- java -cp .:ltkjava-1.0.0.6.jar:mercuryapi.jar read tmr://localhost --ant 1

This command assumes you have an antenna connected on port 1.

For different antenna configurations, substitute one of the following commands:

- java -cp .:ltkjava-1.0.0.6.jar:mercuryapi.jar read tmr://localhost --ant 2
- java -cp .:ltkjava-1.0.0.6.jar:mercuryapi.jar read tmr://localhost --ant 1,2

2. When Mercury API source code SDK is available on the reader

a. Log in as the development user (has ability to escalate privileges so apps can be installed to auto-start at boot time.)

- Username: debian
- Password: rootsecure

b. A Mercury API source code package is preinstalled in the debian user's home directory (/home/debian). Unpack using the below command

```
tar xvf sargas-mercuryapi*.tar.gz                    # substitute the API version for "*"
```

c. Change the directory to mercuryapi*/java # substitute the API version for "*"

d. Check for the availability of the jars "ltkjava-1.0.0.6.jar" , "mercuryapi.jar" and the sample codelet (example: read.java) under the directory "mercuryapi*/java".

e. Now follow the steps from "f to h" from the section, "1. When Mercury API source code SDK is not available on the reader" for compiling and running on reader java application**.**

Uploading Compiled Program for Distribution

There are two recommended methods for uploading programs once they are created, so they can be distributed to other readers: SSH and creating a temporary web server on a reader.

SSH

SSH (the Secure SHell protocol) includes file transfer capabilities.
From a Unix-like system, there are a couple of programs that are likely to be installed already. The first option is scp (Secure Copy):

```
scp debian@sargas-123456:mercuryapi-1.29.0.63/c/src/api/read
```

A more sophisticated option is rsync (Remote Sync). For this simple use case, it won't make much of a difference, but it also has the ability to do incremental copies of large files or entire directories. Using hashing techniques on both sides, it can avoid transferring parts of files that have not changed.

```
rsync -aP debian@sargas-123456:mercuryapi-1.29.0.63/c/src/api/read
```

Temporary web server

Python comes with a simple web server library that includes its own demo program.

From a Sargas command line,

```
cd mercuryapi-1.29.0.63/c/src/api
python -m SimpleHTTPServer
```

and wait for the message

Serving HTTP on 0.0.0.0 port 8000 ...

Now from the host machine, you should be able to open the following URL in a web browser.

http://sargas-123456:8000

Simply click on the file you want to open or download it.

NOTE: You will need to reapply the “executable” permission to the file after copying it to the target system, since web server download does not preserve this property. After copying the read program to the target system, log in to that target system and run

```
chmod +x  read
```

in order to be able to run it again.

To stop the web server, hold down **Ctrl-C** until you see the Sargas command line prompt again. (Just pressing once doesn't always break completely out of the program.)