**Getting Started with DINC Package**

Copyright © 2013-2014 Mobric Inc. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language, in any form by any means, without the written permission of Mobric Inc. Mobric Inc. reserves the right to modify or discontinue any product or piece of literature at anytime without prior notice.

**History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Editor** |
| 2014/3/14 | 1.0 | Public announcement version. | Stan Shih |
| 2014/08/19 | 1.1 | Support external USB adaptor case. | Stan Shih |
| 2015/07/08 | 1.2 | Log for HAL raw data support | Stan Shih |
| 2015/08/14 | 1.3 | Auto server lookup. | Stan Shih |
| 2016/01/07 | 1.4 | Chapter of command line and trouble shooting | Stan Shih |

**Introduction**

**Purpose**

The purpose of this document is to help readers to have a basic concept of **DINC** (Device Intelligence Client) and guide them to integrate a **DINC** package into their products or devices. Moreover, this document also provides a practice to create a new data collector and a new profile for their product.

**Scope**

This document describes the system overview of **DINC** and explains the components of **DINC** in order to lead readers to integrate **DINC** into product step by step; understanding and recognizing all modules of **DINC** package; the lifecycle and resource consumption; the guideline of porting this package into their products; the additional deployment of this package; a tutorial for creating a new data collector and along with the de-serializing components on **Service Insight**.

**Audiences**

The audiences of this document are expected to have intermediate C programming skill and basic telecom knowledge.

**Definitions and abbreviations**

|  |  |
| --- | --- |
| **Service Insight** | Mobric Device Intelligence Server. |
| **DINC** | Mobric Device Intelligence Network Client. |
| **DC** | Data Collector Module of DINC |
| **HAL** | Hardware Abstraction Layer |
| **Toolchain** | A set of tools including compiler, libraries, headers and etc. |

**Notations**

The variable **$module** used in this document stands for one of the following data collection modules:

|  |  |
| --- | --- |
| **dev** | General device information. |
| **Wifi\_sta** | Wi-Fi (station mode). |
| **wifi\_ap** | Wi-Fi (access point mode). |
| **wmx** | WiMAX. |
| **2g** | 2G networks. |
| **3g** | 3G networks. |
| **lte** | LTE networks. |
| **voip** | VoIP. |
| **ethernet** | Ethernet. |
| **extern\_src** | Outside network source such as USB wireless adaptor |

**System Overview**

**Software Architecture**



**Runtime Diagram**



**Modules**

* **DINC Device Intelligence Network Client**

1. **DINC is a communication hub between server and DC,** it handles authentication, server look up and packet delivery.
2. **DINC** should be started on system up.
3. **DINC** is a persistent service.

* **DC Data Collector**

1. When **DC** is called with **--start\_dc**, **DINC** forks a child process and execute **DC** main routine.
2. There are three types of **DCs**:
   * 1. device
     2. network interface
     3. service
3. Packets sent from **DC** and forwarded by **DINC** to server.

* **HAL Hardware Abstraction Layer**

1. **HAL** is a set of machine dependent data access functions. All **DC** collected data are retrieved from **HAL** layer.
2. Profiles are machine dependant codes.
3. Profile is selected on compile time.

**Module Life Cycle**

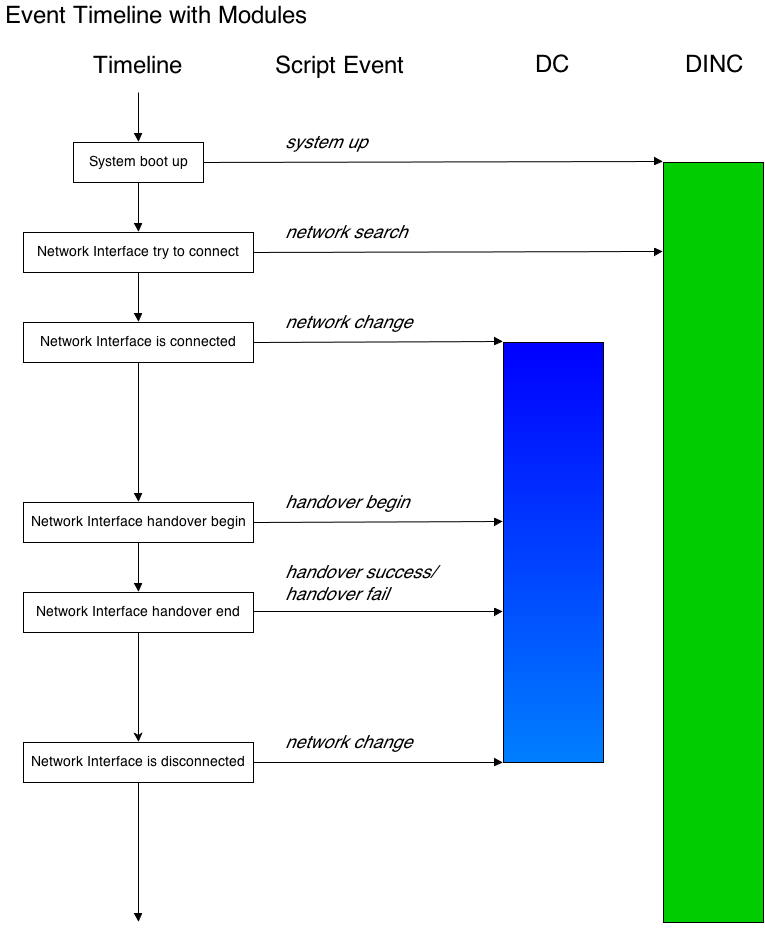
* **DINC**

**DINC** should be activated when system boots up and should not be terminated even server does not exists.

* **DC**

Every **DC** module is an independent process:

1. **Network** **DC** only activated when the related network interface is connected and terminated if the network is disconnected. For example, LTE DC only **up** when LTE network is connected and **down** when LTE network is disconnected. A shell script (**dinc\_util.sh**) is provided to help you to notify **DINC** of the network status change.
2. **Service** **DC** only activated when the related service is started and terminated if the network is disconnected or service stopped. For example, VoIP DC only **up** when VoIP service is started and **down** when VoIP service is stopped or network is disconnected.
3. **Device** **DC** only activated when system is up and then terminated after its own task is finished.



**System and Resource Requirements**

First of all to avoid memory leakage, there is no dynamic memory allocation in **DINC**.

**Recommended Hardware Specification:**

1. Processor Speed: 200 MHz
2. Flash Memory Size: at least 200 KB free space.

|  |  |  |  |
| --- | --- | --- | --- |
| **Module** | **Binary Code Size (flash memory)** | **Memory Usage (runtime code size)** | **CPU Usage** |
| **DINC** | **169.6 KB** | **2712 KB**: Which Include processes:   1. **din** 2. **dc 2g** 3. **dc 3g** 4. **dc ethernet** 5. **dc lte** 6. **dc voip** 7. **dc wifi\_ap** 8. **dc wifi\_sta** 9. **dc wmx** | < **1 %** |
| **HAL** | Depending on implementation | | |

**Test environment:**

1. Operating system: Linux 2.6.21
2. CPU: MIPS32 24K, 360 MHz
3. Physical memory: 32 MB

**System Library Scope**

All libraries used in **DINC** are **System V/POSIX/standard C libraries**. All data received is from **HAL** module and only **HAL** can reach the firmware. Here is the list of headers used in every module.

|  |  |  |  |
| --- | --- | --- | --- |
| stdio.h | string.h | sys/time.h | arpa/inet.h |
| stdlib.h | errno.h | sys/syscall.h | arpa/nameser.h |
| stdint.h | float.h | sys/types.h |  |
| stddef.h | ctype.h | sys/socket.h |  |
|  | math.h | sys/stat.h |  |
|  | limits.h |  |  |
|  | time.h |  |  |
|  | pthread.h |  |  |
|  | unistd.h |  |  |
|  | getopt.h |  |  |
|  | syslog.h |  |  |
|  | netdb.h |  |  |
|  | netinet/in.h |  |  |
|  | resolv.h |  |  |
|  | fcntl.h |  |  |
|  | dlfcn.h |  |  |
|  | signal.h |  |  |

**Directory structure**



|  |  |
| --- | --- |
| **Path** | **Description** |
| dinc/ | The DINC source package. |
| dinc/doc/ | The document directory. |
| dinc/profile/ | The device profile directory. |
| dinc/profile/template/ | The device profile template folder. |
| dinc/profile/template/config/ | The directory stores DCs and DINC’s configuration files. |
| dinc/profile/template/dinc\_util.sh | All collection of script functions that can be included in your scripts |
| dinc/profile/template/makecfg.mk | The makefile used to choose a specific device profile. |
| dinc/generic/ | The directory stores DCs and DINC’s source code. |
| dinc/hal/ | The HAL module directory. |
| dinc/hal/$module/ | The directory for each data collection module ($module). |
| dinc/hal/$module/template/ | The directory stores source file of HAL function skeletons. |
| dinc/hal/$module/template/data\_$module.c | The HAL function skeletons. |
| dinc/hal/$module/data\_$module.h | The header file of HAL functions. |
| dinc/hal/$module/utest\_$moduel.c | The unit test source file. |
| dinc/hal/$module/template.mk | The makefile template used to declare additional object files or static libraries. |
| dinc/hal/Makefile | The makefile used to build HAL main program. |
| dinc/hal/module.mk | The makefile used to build Data Dispatcher and unit test program. |
| dinc/output | The output directory. |
| dinc/common.mk | The makefile used to define some common variables. |
| dinc/main.c | main entry point. |
| dinc/Makefile | The main make file used to build DINC, DCs and HAL program. |
| dinc/README | Readme file |
| dinc/dins.exe | The server emulator. |

**Porting steps**

Porting procedure can be divided into the following steps:

1. **Profile**
   1. **Create device profile (configure the toolchain settings and build environment)**
2. **HAL**
   1. **Implement HAL Functions.**
   2. **Compile HAL Modules.**
   3. **Test HAL Modules.**
3. **Script**
   1. **Integrating Script (Implement functions in dinc\_util.sh and add dinc\_util.sh calls in rc.d/init.d and network/service hook)**
4. **Compilation**
   1. **DINC Package Compilation (sudo make)**
5. **Installation**
   1. **DINC Package Installation (sudo make install)**
6. **Testing**
   1. **Manual test with server emulator**
   2. **Trouble shooting**
   3. **Full test with server emulator**

**Quick Start**

To those who are not familiar with the whole **DINC** package and are at the first time to portit, we provide a quick start practice for you to start porting with the very important data collector **dc\_dev**. You may follow the instructions and refer to the descriptions below this section for details.

1. **Profile:**

You need to create a new profile for your own product, assume the product name is **cpe\_xyz**, for creating a profile for it please type-in command line:

cd dinc/

sudo make create-profile NAME=cpe-xyz

1. **HAL:**

**2.1** In directory **dinc/hal/dev/**, you will see there are new profile folder and corresponding configuration file which both named as **cpe\_xyz**. Now open file **data\_dev.c** and implement each function.

**2.2** After 2.1, you can build only **HAL** module for debugging, please type-in the following command:

cd dinc/hal/

sudo make PROFILE=cpe-xyz

**2.3** Go to directory **dinc/hal/dev/vpe\_xyz/**, you can execute **utest\_dev** to verify the accuracy of your implementation.

**Note: You may iterate step 2 till all functions are implemented for the accuracy of HAL part.**

1. **Script:**

Adding script callings within your product, for instance, two mandatory events must call script to activate **DINC** and **DC**.

Activate **DINC**: Add following scripts in system folder **init.d** or **rc.d**:

dinc\_util.sh system\_up

Activate **DC**:

dinc\_util.sh network\_change lte

1. **Compilation**

Execute the following command line to build **DINC** package for product profile **cpe\_xyz**:

cd dinc/

sudo make PROFILE=cpe\_xyz

1. **Installation**

To install **DINC** package, we type in the following command:

cd dinc/

sudo make install PROFILE=cpe\_xyz

1. **Testing:**

For testing the functionality of **DINC** and **dc\_dev**, a server emulator must be launched in the first place. Please type-in the following command line to activate server emulator:

cd dinc/

mono dins

Open file **/etc/dinc/din.cfg**, and assign your host IP address to **server\_ip\_addr**. For instance, if **DINC** and **server emulator** will be deployed on the same machine, the IP address will be assigned as **127.0.0.1**.

**Note: DC dev would not report data itself, a network DC must launch as well. You can iterate the above steps to porting a network DC for your own product. After a DC implementation is done please type-in the following command to activate that DC for network interface, assume it is LTE.**

dinc\_util.sh system\_up

dinc\_util.sh network\_change lte

**After a while, the data packet will be list on your screen and the data contains device information and network interface LTE.**

**1.1 Create Device Profile**

To create a device profile, please type in the following command:

sudo make create-profile NAME=device-name

The script will copy all necessary folder and files in correct directory and rename them according to what device name you input, followed this step you might need to modify configuration file “makecfg.mk” if you have specific development environment settings:

################################################################################

#

# The cross compiler prefix of the toolchain.

#

# For example, if the binary path is "/opt/gcc/bin/mips-linux-gcc",

# the prefix should be "/opt/gcc/bin/mips-linux-".

#

################################################################################

**CROSS\_COMPILE\_PREFIX = /usr/bin/**

################################################################################

#

# The header directory of the toolchain.

#

################################################################################

**C\_INCLUDE\_PATH = /usr/include**

################################################################################

#

# The library directory of the toolchain.

#

################################################################################

**LIBRARY\_PATH = /usr/lib**

################################################################################

#

# The directories containing the required libraries by the toolchain.

#

################################################################################

**LD\_LIBRARY\_PATH = /usr/lib**

################################################################################

#

# The data collection modules supported by your device.

#

# The example shows all possible choices.

#

# For details, see dinc/hal/$module/template/data\_$module.c.

#

################################################################################

**MODULES = wifi\_sta wifi\_ap wmx 2g 3g lte voip ethernet**

################################################################################

#

#The ROOT path of your device (very important!!!)

#

# For example, if you want to do local test, please set it to "",

#but if you are going to install this package into your device,

#please set it to "your firmware root"

#

################################################################################

**INSTALL\_ROOT =**

################################################################################

#

#The directories for DINC in your device

#

# For example: local test is "/usr/bin"

# your device is "your firmware root/usr/bin"

#

################################################################################

**INSTALL\_DINC\_PATH = $(INSTALL\_ROOT)/usr/bin**

################################################################################

#

#The directories for DINC Shared Library (HAL) in your device

#

#For example:local test is "/usr/lib"

# your device is "your firmware root/usr/lib"

#

################################################################################

**INSTALL\_DINC\_LIB\_PATH = $(INSTALL\_ROOT)/usr/lib**

################################################################################

#

#ATTENTION!!!

#

# DO NOT CHANGE this setting, this directory is necessary of DINC

#

################################################################################

**INSTALL\_DINC\_CFG\_PATH = $(INSTALL\_ROOT)/etc/dinc**

The first four variables are related to the toolchain. Set them to proper paths.

The fifth variable is a list of supported data collections modules. In fact, the module names are the same as the directory names under **hal** directory, except **hal/include**. You don’t need to specify **dev** module since it is included by default.

You may have several profiles. When you want to build **DINC** using my-device.mk, please execute the following command:

sudo make PROFILE=device-name

**2.1 Implement HAL Functions**

Now we have created the supported modules in the profile. The next task is to implement each of them. In each module directory **hal/$module**, you can see a directory which is named as your device, a default folder is **template**. (For creating a profile in **HAL** please refer to previous subject: **Create device profile**)

Open the directory, It contains one file: **data\_$module.c**. Below is a part of **data\_dev.c**:

/\*

\* MOBRIC CONFIDENTIAL

\* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*

\* 2013 Mobric Incorporated

\* All Rights Reserved.

\*

\* NOTICE:

\* All information contained herein is, and remains the property of

\* Mobric Incorporated and its suppliers, if any. Dissemination of this

\* information or reproduction of this material is strictly forbidden unless

\* prior written permission is obtained from Mobric Incorporated.

\*

\*/

#include <stdio.h>

#include <string.h>

#include <stdint.h>

#include "msg\_hal.h"

#include "data\_dev.h"

/\*

\* Description : Get the device unique ID of the device.

\*

\* Use gen\_uid\_from\_mac\_addr() / gen\_uid\_from\_imsi() to generate an

\* unique ID from MAC address / IMSI.

\*

\* Return Code : 0 for success, -1 for failure.

\*

\* Data Valid Range : Text Format. Only accept [0-9], [A-F], [a-f].

\*

\* The string length should be DEV\_UID\_LEN.

\*

\* Reference :

\*

\* Remarks :

\*

\*/

int dev\_get\_uid(char uid[DEV\_UID\_LEN + 1]) {

#ifdef SAMPLE\_CODE

uint8\_t mac\_addr[] = {

0x12,

0x23,

0x12,

0x23,

0x12,

0x23

};

gen\_uid\_from\_mac\_addr(uid, mac\_addr);

return 0;

#else

return -1;

#endif

}

Each type of data we want to collect has a corresponding skeleton function. It also comes with comments of description, valid data range and etc. We provide some sample code for your reference. If the data is available, please add your code in the **#else** part, otherwise just return -1.

**2.2 Compile HAL Modules**

Open file **hal/$module/device-name.mk**:

################################################################################

#

# The additional source files for data\_$module.c.

#

################################################################################

**ADD\_SRCS +=**

################################################################################

#

# The additional header files for data\_$module.c.

#

################################################################################

**ADD\_DEPS +=**

################################################################################

#

# The additional library (path) for data\_$module.c.

#

# Example: -ladd -Ladd/lib/path

#

################################################################################

**ADD\_LDLIBS +=**

Sometimes you may add your own source files. For example, **my-device/data\_lte.c** requires **my-device/foo.c** and **my-device/foo.h**. Add them to the make file:

ADD\_SRCS += my-device/foo.c

ADD\_DEPS += my-device/foo.h

Finally, build the LTE module:

cd dinc/hal

make lte

Or you can clean the module:

cd dinc/hal

make lte\_clean

**2.3 Test HAL Modules**

We have prepared unit test for each module. In the previous step, it also generates an executable **utest\_lte** at **hal/lte/my-device** directory. Copy this file to your device and run it, it should output the test result like below:

[ LTE ]

----------------------------------------

[ SUPPORTED\_FREQ\_BAND ]

---------------------------------------- #1: PASS 1511828488192 #2: PASS 1511828488192 #3: PASS 1511828488192 ---------------------------------------- Result : PASS Rounds : 3 Run Time (max) : 0.023 ms Run Time (min) : 0.003 ms Run Time (avg) : 0.010 ms Run Time (total) : 0.030 ms ----------------------------------------

[ IP\_ADDRESS ] ---------------------------------------- #1: PASS 2002:0:0:0:0:0:7b42:d285 #2: PASS 2002:0:0:0:0:0:7b42:d285 #3: PASS 2002:0:0:0:0:0:7b42:d285 ---------------------------------------- Result : PASS Rounds : 3 Run Time (max) : 0.028 ms Run Time (min) : 0.003 ms Run Time (avg) : 0.012 ms Run Time (total) : 0.035 ms ----------------------------------------

(skip)

[ NEIGHBORING\_CELL\_INFO ] ---------------------------------------- #1: PASS CID:20216, MCC:208, MNC:01, LAC:21212, RSSI:-90, RSRP:-80, RSRQ:-17.500000 CID:20238, MCC:208, MNC:02, LAC:21215, RSSI:-75, RSRP:invalid, RSRQ:invalid #2: PASS CID:20216, MCC:208, MNC:01, LAC:21212, RSSI:-90, RSRP:-80, RSRQ:-17.500000 CID:20238, MCC:208, MNC:02, LAC:21215, RSSI:-75, RSRP:invalid, RSRQ:invalid #3: PASS CID:20216, MCC:208, MNC:01, LAC:21212, RSSI:-90, RSRP:-80, RSRQ:-17.500000 CID:20238, MCC:208, MNC:02, LAC:21215, RSSI:-75, RSRP:invalid, RSRQ:invalid ---------------------------------------- Result : PASS Rounds : 3 Run Time (max) : 0.028 ms Run Time (min) : 0.014 ms Run Time (avg) : 0.019 ms Run Time (total) : 0.058 ms ----------------------------------------

---------------------------------------- Result : PASS Run Time : 0.819 ms----------------------------------------

If the test failed, please fix the code and re-run the test.

**3.1 Integrating Script**

As mentioned before, **DINC** and **DCs** will be launched (or terminated) when **specific event** occurs. We provide a utility script file called **dinc/profile/$module/dinc\_util.sh**. Here is the usage:

dinc\_util.sh <event type> [parameters ...]

There are two event types: mandatory and optional.

**Mandatory Events:**

system\_up

The event should be sent on system startup. No parameter is needed.

dinc\_util.sh system\_up

network\_change

The event should be sent when the connectivity status of the 2g, 3g, lte, wmx and wifi\_sta changed (connect or disconnect). The parameters are the current connected network aliases. For example:

dinc\_util.sh network\_change lte

If all networks are disconnected, just keep the parameters empty.

If your device is connected by an external adaptor such as a USB wireless adaptor, then you need to use “extern\_src” to indicate network status changed:

dinc\_util.sh network\_change extern\_src

In this way, three **DC** will be activated; **wifi\_ap**, **voip** and **ethernet**, in addition, you also need to fill out a script function **get\_usb\_dev\_ip()** to indicate the IP address of the adaptor.

**Note:** Please ensure that the USB device has already connected with network before **network\_change extern\_src** is called, a very common way to ensure that is to ping a pre-defined and always exists host such as Google’s DNS (8.8.8.8).

**Optional Events Hooks <Highly recommended>:**

network\_search

The event can be called optionally when searching a certain network. We use this event to determine the network entry latency. This function can be ignored if network entry latency is supported by your **HAL** implementation.

This event should be sent with a network alias such as “wmx” to indicate what network is being searching for:

dinc\_util.sh network\_search wmx

handover\_begin

handover\_success

handover\_fail

These events can be sent optionally when handover occurs. We use these events to determine the handover success/failure count and latency.

They should be sent with a network alias:

dinc\_util.sh handover\_begin lte

You can ignore them if handover data are all supported by your **HAL** implementation.

**Steps to Integrate Script**

**Step 1:**

Implementing following functions in **dinc\_util.sh**:

get\_domain\_name\_from\_dhcp()

Description: Return the optional domain name in the DHCP server set by network service provider.

get\_wmx\_nap\_id ()

Description: Return the NAP-ID (the most significant 24-bits of the base station ID). You don’t need to implement this function if your device doesn’t have WiMAX interface.

get\_imsi\_mcc()

get\_imsi\_mnc()

Description: Return the MCC/MNC of the IMSI. You don’t need to implement this function if your device doesn’t have 2G, 3G and LTE interfaces.

get\_usb\_dev\_ip ()

Description: For the case of device is connected by USB wireless adaptor. Return the IP address of that USB device.

🟊get\_activated\_nif ()

Description: Return current activated network interface(s) which could be multiple values such as “2g 3g lte”. The first one (“2g” in this case) will be the currently used network type.

Remark: This function is a required item which is needed to be filled in order to provide server auto lookup in case the IP address has been changed. The lookup interval is set to be 24 hours.

🟊get\_cmd\_pid ()

Description: Get the currently running process ID by command name.

Remark: This function is a required item which is needed to be filled in order to be used in function “kill\_watchdog\_loop” to kill a process created in function “start\_watchdog\_loop”.

In order to make sure **DINC** works correctly, understand the usage of **Integration Script** is not enough; please add some scripts in your system folder like **init.d** or **rc.d** or any other folder related to call this integrating script.

**Step 2:**

Add following scripts in system folder **init.d** or **rc.d**:

dinc\_util.sh system\_up

**Step 3:**

Add following scripts to network start:

dinc\_util.sh network\_change $(network\_type)

**4.1 DINC Package Compilation**

After finishing the implementation of HAL program, please input these command to compile all **DINC** packages. All binary files will be built to in the **generic/dinc/**.

cd dinc

sudo make (sample)

If you leave no arguments behind **“sudo make”**, then **DINC** builds default device **“template”** and it will not define **SAMPLE\_CODE** for **hal**, if adding argument **sample,** for instance **“sudo make sample”** **dinc** will build sample code part in **hal** modules.

In addition, in order to build a specific device, you can add device name after make command.

cd dinc

sudo make PROFILE=device-name

Start from this version, **DINC** can be built with output debug messages. Please add a parameter **DEBUGMODE** and indicate value as **ON** while make project. You can also using **OFF** or remove this parameter to build a version without outputting debug messages.

cd dinc

sudo make DEBUGMODE=ON

**DINC** can be built with output HAL raw data. Please add a parameter **HALDATA** and indicate value as **SHOW** while make project. You can also using **HIDE** (default) or remove this parameter to build a version without outputting HAL raw data.

cd dinc

sudo make HALDATA=SHOW

**5.1 DINC Package Installation**

To install **DINC** package, we type in the following command:

cd dinc

sudo make install PROFILE= device-name

DINC components will be installed in correct location within device.

* **din/dinc\_util.sh/lib:** Will be installed in **/sbin** or **/usr/bin**.
* **Configuration file**s: Will be installed in the **/etc/dinc**. (Optional)

**Note: Service Insight**, an internal testing emulator, **not a part** of your product/device, any OS with .Net Framework is necessary. For example, like Windows with .Net Framework 3.5 or Linux (Ubuntu/Fedora/openSUSE…) with mono. Please **DO NOT** put this module into your product/device.

**6.1 Manual Test with Server Emulator**

**Step 1:**

Open file **/etc/dinc/din.cfg**, and assign your host IP address to **server\_ip\_addr**.

**server\_ip\_addr=**

**Step 2:**

Execute the server emulator program **dinc/dins.exe** on a Windows machine. It requires .NET framework 3.5.

Please use the following command line:

cd dinc

mono dins fd

or

cd dinc

mono dins

The argument **fd** means **fast debug** which will let server emulator only print out header part of each data packet in order to verifying the accuracy of packet report interval. Without this argument, server emulator will do data de-serializing and print out each data field name and data value.

**Step 3:**

By design, **DINC** will be launched while system starts up, and following **Device DC** will be launched to collect general device information and send it to **DINC**.

You can do this manually by running:

dinc\_util.sh system\_up

When one of the WAN interfaces is connected, the system should send **DINC** a message to start connecting to the server and launch the correspondent DC. For instance, if it is a LTE interface:

dinc\_util.sh network\_change lte

After a while, the server emulator should display the collected data similar as below in the console. It will receive new data every 1 - 3 minutes.

1/23/2045 1:23:45 PM

--- Category: Device, Type: DcDevice

Device ID: 32ECA13877A36F96

[vendor]Mobric[model]MIFX-N2-009[firmwareVersion]1.0.6[serialNumber]857234600[chipRevision]1.0.0[cpuCoreCount]1[cpuClock]600[ramSize]256[flashMemorySize]1024[batteryCapacity]3000[usbDevices]

1(0x01) 6(0x06)

[networkInterfaces]

[type]

80

[macAddress]

18(0x12) 35(0x23) 52(0x34) 69(0x45) 86(0x56) 103(0x67)

[networkInterfaces]

[type]

96

[macAddress]

35(0x23) 52(0x34) 69(0x45) 86(0x56) 103(0x67) 120(0x78)

[imsi]

46000123456789

--- Decode End

(skip)

1/23/2045 1:23:45 PM

--- Category: Lte, Type: DcNetworkEntry

Device ID: 32ECA13877A36F96

Longitude: 123

Latitude: 123

[ipAddress]

18(0x12) 52(0x34) 35(0x23) 69(0x45) 52(0x34) 86(0x56) 69(0x45) 103(0x67) 0(0x00) 0(0x00) 86(0x56) 120(0x78) 103(0x67) 137(0x89) 120(0x78) 154(0x9A)

[cellGlobalId]

[cellId]

250

[mcc]

208

[mnc]

01

(skip)

**6.2 Trouble shooting**

**Case 1:**

The server emulator doesn’t show any message. Please check if your PC blocks incoming connection on port 52888 for server emulator.

**Case 2:**

The server emulator is able to receive device information but no DC-specific information. It’s because DINC always checks the timestamp in the **Service Insight** response. The packets will be ignored if the time difference between them is too large. Normally, the time setting of your PC is automatically synchronized with time server. Please check the setting of your device.

**6.3 Full Test with Server Emulator**

**Step 1:**

Open file **/etc/dinc/din.cfg**, and assign your host IP address to **server\_ip\_addr**.

**server\_ip\_addr=**

**Step 2:**

Execute the server emulator program **dinc/dins.exe** on a Windows machine. It requires .NET framework 3.5.

**Step 3:**

Let all network interfaces connect with internet and check packet data values in server emulator. Not like **Manual test with server emulator**, it does not need to execute each script manually but let device executes them automatically.

After a while, the server emulator should display the collected data similar as below in the console. It will receive new data every 1 - 3 minutes.

1/23/2045 1:23:45 PM

--- Category: Device, Type: DcDevice

Device ID: 32ECA13877A36F96

[vendor]Mobric[model]MIFX-N2-009[firmwareVersion]1.0.6[serialNumber]857234600[chipRevision]1.0.0[cpuCoreCount]1[cpuClock]600[ramSize]256[flashMemorySize]1024[batteryCapacity]3000[usbDevices]

1(0x01) 6(0x06)

[networkInterfaces]

[type]

80

[macAddress]

18(0x12) 35(0x23) 52(0x34) 69(0x45) 86(0x56) 103(0x67)

[networkInterfaces]

[type]

96

[macAddress]

35(0x23) 52(0x34) 69(0x45) 86(0x56) 103(0x67) 120(0x78)

[imsi]

46000123456789

--- Decode End

(skip)

1/23/2045 1:23:45 PM

--- Category: Lte, Type: DcNetworkEntry

Device ID: 32ECA13877A36F96

Longitude: 123

Latitude: 123

[ipAddress]

18(0x12) 52(0x34) 35(0x23) 69(0x45) 52(0x34) 86(0x56) 69(0x45) 103(0x67) 0(0x00) 0(0x00) 86(0x56) 120(0x78) 103(0x67) 137(0x89) 120(0x78) 154(0x9A)

[cellGlobalId]

[cellId]

250

[mcc]

208

[mnc]

01

(skip)

**Customization**

**Create a Data Collector**

* Before create a data collector, here introduces two **DC** related structures:

1. **dc\_routine\_definition**:

A structure defines all parts of a **DC** which contains a set of function pointers and all functions will be called during a life cycle of a **DC**. You can implement your own **DC** functions and assign them to these pointers as a callback.

Moreover, this mechanism allows customers to define their own events which can be handled in their own callback function.

1. **dc\_routine\_context**:

A structure provides utility functions which can help **DC** to report data or retrieve specific information which is only stored in **DINC** mainframe

* To simplify creation steps, we provide a template **DC** project and **HAL** project in **dinc/doc/sample\_dc.tar**, please decompress **sample\_dc.tar** and follow the steps below:

**Note:** All **DC** related functions, files or folders follow a naming rule which is all **DC** will be recognized by a prefix “dc\_”. For instance, if creating a geolocation **DC** you name the **DC** as “**dc\_geolocation**”.

**Step 1: Copy template projects**

The template projects we provided have a special string “**{###}**” this is which you need to replace with your **DC** name.

* + - 1. Copying template project “**dc\_{###}**” into **dinc/generic/** and rename all “**{###}**” to the name of **DC** (in this case “geolocation”). All necessary modifications including folder name, file names and function names.
      2. Copying template project “**{###}**” into **dinc/hal/** and rename all “**{###}**” to the name of **DC** (in this case “geolocation”). All necessary modifications including folder name, file names and function names.

In function **dc\_{###}\_id()**, you need to define the id of **DC**, All customized **DC** start id from **DIN\_DC\_CUSTOM\_BASE** and increased by one.

**The following table contains all targets which needed to be modified:**

|  |  |
| --- | --- |
| **Folders** | **Remark** |
| dinc/generic/dc\_{####}/ |  |
| dinc/hal/{###}/ |  |
| **Files** | **Remark** |
| dinc/generic/dc\_{####}/dc\_{###}.c |  |
| dinc/hal/data\_{###}.h |  |
| dinc/hal/template/data\_{###}.c |  |
| dinc/hal/utest\_{###}.c |  |
| **Functions** | **Remark** |
| uint32\_t dc\_{###}\_id(void) | dc\_{###}.c |
| int dc\_{###}\_main(int argc, char \*argv[]) | dc\_{###}.c |
| #include "data\_{###}.h" | data\_{###}.c |

**Step 2: Add DC into profile**

Modifying profile files to recognize your **DC** in order to let **DINC** build your source code. Open file **dinc/profile/template/makecfg.mk** append **DC** name to “**MODULES = wifi\_sta wifi\_ap wmx 2g 3g lte voip ethernet**”.

In this case, “**MODULES = wifi\_sta wifi\_ap wmx 2g 3g lte voip ethernet geolocation**”

**Step 3: Determine DC launching time**

In file **dinc/profile/template/dinc\_util.sh** add **DC** name in function “**stop\_all\_dcs()**”, In addition, if your **DC** is designed to launch separately, add into function “**start\_client\_event\_trigger()**” or if it will launch while other DC is launching then add into function “**start\_extra\_dcs()**”.

**Step 4: Add new Config file (Optional)**

All DC modules can associate with a configuration file which will be located in the path of **dinc/profile/$module/config/dc\_\*.cfg** in order to provide customized criteria values for **DC** during run time, or you can just ignore it to let **DC** run as default values (assigned in **DC** source code).

To implement your own configuration file, you will need to parse this file in **DC** function “**dc\_load\_config(const char\* path)**”. The naming rule for file is the same “**dc\_{###}.cfg**”.

**Step 5:**

Build project by the following command:

sudo make clean

sudo make PROFILE=cpe\_xyz

**Step 6:**

Install and deploy library and scripts:

sudo make install PROFILE=cpe\_xyz

Now you have installed **DINC** and **DCs** in your device and you may start to test it.

**Create a Packet Component in Server Emulator**

Each data collector packet in **DINC** has a corresponding data structure in server emulator which is used to extracted data value from packet. The server emulator is named **dins** in our package. It is a c# project and already has sets of packet classes which are used to de-serialize packets. Therefore when adding a new data collector packet you need to create a corresponding packet de-serializing object in **dins**.

**Step 1: In a new DinPacketCategories enum**

In sample case, when we create a new data collector **geolocation** in **DINC** we need to add a new category ID in **dins** in order to be recognized. Therefore add a new category named **Geolocation** and assign id number as **0x1100**.

**Step 2: In a new DinPacketTypes enum**

Each DC packet has a unique message ID to help dins to recognize them, all customized message ID starts from **0x0081** (**DIN\_PACKET\_DC\_MSG\_BASE**).

In sample case, we add a new message called **ReportGeolocation** in **DinPacketTypes** and assign ID number as **0x0081.**

**Step 3: Create new data class**

All data packet inherit a parent class called **DinPacket**. We can copy an existed packet class and modify to yours. In sample case, class geolocation will be as following:

[PacketId(DinPacketCategories.Geolocation, DinPacketTypes.ReportGeolocation)]

public class GeolocationPacket : DinPacket< GeolocationPacket, GeolocationPacket.BodyFormat>

{

public class BodyFormat {

[Order(0)]public float longitude;

[Order(1)]public float Latitude;

}

public GeolocationPacket(){

}

protected override bool Decode(BodyFormat format)

{

Throw new NotImplementedException();

}

}

**Note:** All data in **BodyFormat** must have the same order as in **DC** packet.

**DINC Commands**

In the following chapter, we are going to list current supported command by **dinc;** along with example code and result output, in order to provide information while **dinc** is running during the development period.

**Command 1: Get Device UID**

din –request\_dev\_uid=127.0.0.1

This command is used to ask **dinc** to report its own unique device ID which can be used to verify ID listed in **MobileInsight** server.

**Command 2: Get Server Domain Name**

din –request\_full\_domain=127.0.0.1

***[sample output]*** g460-11

This command is used to ask **dinc** to report current domain name for **MobileInsight** server. A domain is composed by a certain rule according to which type of network interface is using such as LTE, WiMAX, WiFi station mode…

**Command 3: Get Server Latency Test Failed Count**

din –latency\_fail\_cnt=127.0.0.1

***[sample output]*** 0

This command is used to ask **dinc** to report current latency failed count with server. This value could be accumulated if failure consistently happening and could be reset to zero once a successful testing occured.

1. **Return 0:** Value of Zero means no failure is caused during testing.
2. **Return 65535:** Value of 65535 means **dinc** is failed on handshake with server.

**Command 4: Get Currently Running Data Collectors**

din –who

***[sample output]*** 0x80BB0007

This command is used to ask **dinc** to report currently running data collectors. The return value will be collectors’ ID number in **hex** format.

1. **Data Collectors Name and ID Mapping**:

|  |  |
| --- | --- |
| ***Data Collector*** | ***ID*** |
| WIFI\_STA | 0x80BB0002 |
| WIFI\_STA | 0x80BB0003 |
| WIMAX | 0x80BB0004 |
| 2G | 0x80BB0005 |
| 3G | 0x80BB0006 |
| LTE | 0x80BB0007 |
| VOIP | 0x80BB0008 |
| ETHERNET | 0x80BB0009 |

1. **Please Note**: This command can only affective when **dinc** is not running in daemon mode. In order to using this command, **dinc** must be launched by ***system\_up\_for\_debug***.

**Questions and Answers**

In the following chapter, we listed several questions which are asking frequently with our answers.

**Q. How to launch and re-launch DIN client?**

|  |
| --- |
| **A.**  There are 2 steps to activate DIN client:  **Step 1**. **dinc\_util.sh system\_up**  This will launch DINC main process which includes a UDP server in charge to receiving packets from server via internet, receiving local commands such as **network\_change** and packets from other data collectors locally.  **Step 2**. **dinc\_util.sh network\_change ${network\_interface}**  Once you are sure the device has connected with internet, call this to do the followings:  a) Start handshake procedure with server  b) Activate a data collector based on what network interface you gave it such as 3G or LTE.  There is no re-activate command for DINC since we assume that the main process would live as long as the system, but you can make a data collector disable and enable by calling **dinc\_util.sh network\_change ${network\_interface}**. This will stop current running data collector process then launch another one. To stop the DINC main process you need to kill this process by yourself. |

**Q. Launching DIN client after device is online, however still cannot make connection with server.**

|  |
| --- |
| **A.** The possible reason would be:  1. Only calling **dinc\_util.sh system\_up** cannot make it doing handshake with server, you have to call **dinc\_util.sh network\_change ${network\_interface}** after you are sure the internet is working.  2. Check your din.cfg file it should be leave as empty.  3. Check your implementation on dinc\_util.sh.  4. For LTE check if **g$(get\_imsi\_mcc)-$(get\_imsi\_mnc)-dpm.mobric.com** can be resolved to a real IP address. |

**Q. How does handshake procedure begin? Is TCP or UDP DIN client using?**

|  |
| --- |
| **A.** Once you call **dinc\_util.sh network\_change ${network\_interface}**. DIN client will fork a process to do the handshake job will server. You can see a new process also named din in "ps" command. All packets sending between DINC, data collector and server are UDP. |

**Q. How to determine that the server handshake is failed?**

|  |
| --- |
| **A.** There are two ways to determine whether or not the handshaking is failing:  1. If DIN client is failed on doing handshaking with server, then all data collectors will be terminated by main process. So there will be only one process left.  2. Running command **din --latency\_fail\_cnt=127.0.0.1**, if it returns **65535** then it means handshake failed. |

**Q. If handshake is failing, do we need to trigger DIN client to doing handshake process again?**

|  |
| --- |
| **A.**  If handshake is failed. DINC will try to doing handshake again after **24** hours (**86400** seconds). Or you can force DINC to do handshake again by running command:  **dinc\_util.sh network\_change**  **dinc\_util.sh network\_change ${network\_interface}** |

**Q. When device is offline then back online. Do we need to info DIN client such information? Do we need to re-launch DIN client or it can do it on its own?**

|  |
| --- |
| **A.**  We strongly recommend you using **dinc\_util.sh network\_change** with no interface in the end of command indicates that the network is down when the device is offline and calling **dinc\_util.sh**  **network\_change ${network\_interface}** again when the network is back online. Since the connection activity is also an important data to MobileInsight. |

**Q. In the development period is there any debugging or log mechanism which can provide help?**

|  |
| --- |
| **A.** Yes, DINC can be built as a special version called "Debug" version. And it will output log to syslog for almost each steps that DINC has executed.  For the detail, please refer to the section ***4.1 DINC Package Compilation*** |