



SDLP2.0

Developer's manual

Date:	March19, 2014
Version:	0.2
Issued by:	

This document being intended for internal needs of the Company is circulated by Quality Manager according to the Controlled Distribution List and is marked as "CONTROLLED COPY." The content of this document must not be reproduced either partly or entirely or disclosed to any person not being an employee of the Company without prior consent of Management Representative for Quality. The copies of the document being handed over to a third party by consent of Management Representative for Quality should be marked as "UNCONTROLLED COPY."



	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014


Table of contents

Table of contents	2
Revision History	4
Introduction	5
Definitions, acronyms and abbreviations	5
SDLP technology overview.....	5
SDLP high-level architecture	6
SDLCORE	7
Building and installation	7
Profile management layer.....	7
Profile	8
Profile manager proxy.....	10
Sample profile	11
SDLP benefits	11
SDLP Android proxy.....	12
Android permissions required for SDLP-enabled mobile applications	12
TCP/IP Connectivity.....	12
TCP/IP: Wi-Fi, USB	12
Device discovery	12
Using overview.....	13
Profile Manager proxy listener	13
SmartDeviceLinkProxyALM class.....	14
Responses on profile requests	14
SDLP iOS proxy	16
Main classes overview	16
How to use	16
Building application with the SmartDeviceLink framework.....	16

Copyright © 2013, Luxoft	Confidential	Page: 2 of 36
--------------------------	--------------	---------------


	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

Proxy creation – connectivity options	16
Proxy usage	17
Simple sample	19
Appendix A.....	21
Build instructions	21
Appendix B.....	23
Sample profile header file.....	23
Sample profile source file	26
Appendix C.....	30
iOS example	30
Appendix D.....	35
Android and iOS proxies cheat sheet.....	35

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

Revision History

Version	Date	Author	Description
0.1	December19, 2013	Vyacheslav Plachkov, E. Bratanova	Initial version
0.2	March 19, 2014	Bratanova Elena	SDLP2.0 update

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

Introduction

SDLP developer's manual provides a description of the SDLP2.0 technology and instructions for profiles and mobile applications development.


Definitions, acronyms and abbreviations

API	Application programming interface
DLL	Dynamic-link library
HMI	Human-machine interface
HU	Head unit
MD	Mobile device
PM	Profile Manager
SDL	SmartDeviceLink
SDLP	Smart Device Link Profiles (SDL with profile management layer)

SDLP technology overview

SDLP – universal multifunctional connectivity technology designed to connect mobile and automotive (in-vehicle infotainment - IVI) systems. It is based on the Smart Device Link (SDL) – the open-source technology developed by Ford Motor Company and released in GENIVI repository¹. SDLP2.0 is based on SDL2.0. SDL provides access to the HMI layer of IVI systems through JSON RPC interface. SDLP extends this interface to provide extra functionality. This functionality is designed to add an extensible profile management layer. The layer allows extending of the head unit HMI with profiles. A profile is an updateable dynamic-link entity that contains executable code and has message-based communication interface for mobile applications and HMI.

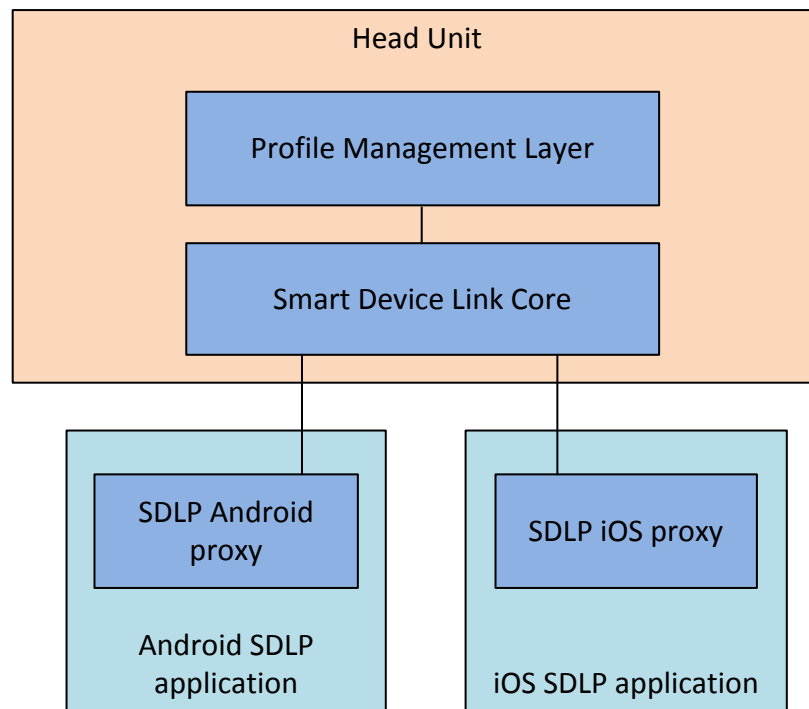
¹<http://git.projects.genivi.org/?p=smartdevicelink.git>

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

SDLP high-level architecture


In a very high-level point of view the SDLP technology consists of 4 modules:

- Smart Device Link core;
- Profile management layer with extensible profiles;
- SDLP Android proxy;
- SDLP iOS proxy.



Smart Device Link Core provides functionality of head unit part of SDL technology. This module is responsible for:

- Transport management;
- Mobile device connection handling;
- Connection with HMI to provide interface of HMI (SDL HMI RPC) for mobile devices;
- Connection with profile manager to provide interface of profile manager for mobile devices;
- Smart device Link Core contains demo HMI component based on HTML.

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

SDLCore

Building and installation

SDLP2.0 source code is available on [GitHub](#).

Available open source code includes all the functionality of SDL2.0, its demo HMI, profile management layer and a set of sample profiles. Build instructions are available².

Profile management layer

Profile management layer is extensible module of SDLP. It has profile manager component and a set of profiles, which could be extended. Head unit system has a means to add new profiles without necessity to flash head unit software. This feature is not mandatory - technology only provides a mechanism that implements it.

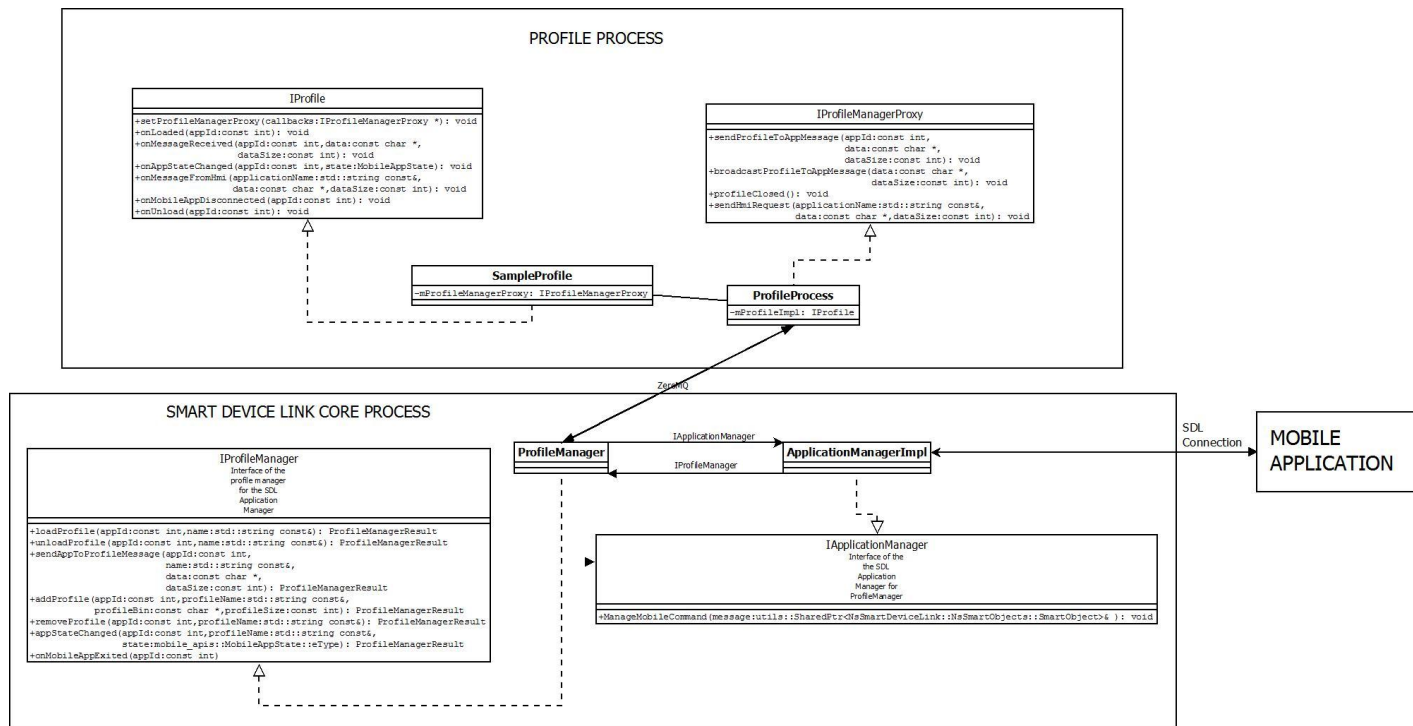
Profile is a key concept of SDLP designed to extend head unit functionality. Profile is a dynamic-link library that implements an interface to handle events from the mobile phone and the HMI. Public SDLP repository contains a set of sample profiles³.

Profile Manager is a system component responsible for profiles loading/unloading, updating and handling connection with mobile applications provided by the SDL core transport mechanisms. Profile manager creates a separate process for execution of each profile. This technical solution allows developers to add new functionality in a safer way. The SDLP framework provides an interface for profile developers to interact with mobile applications and the HMI.

Profile management layer class diagram is shown on the following diagram:

² See SDL_Core/doc/install.txt

³ See the "profiles" folder.




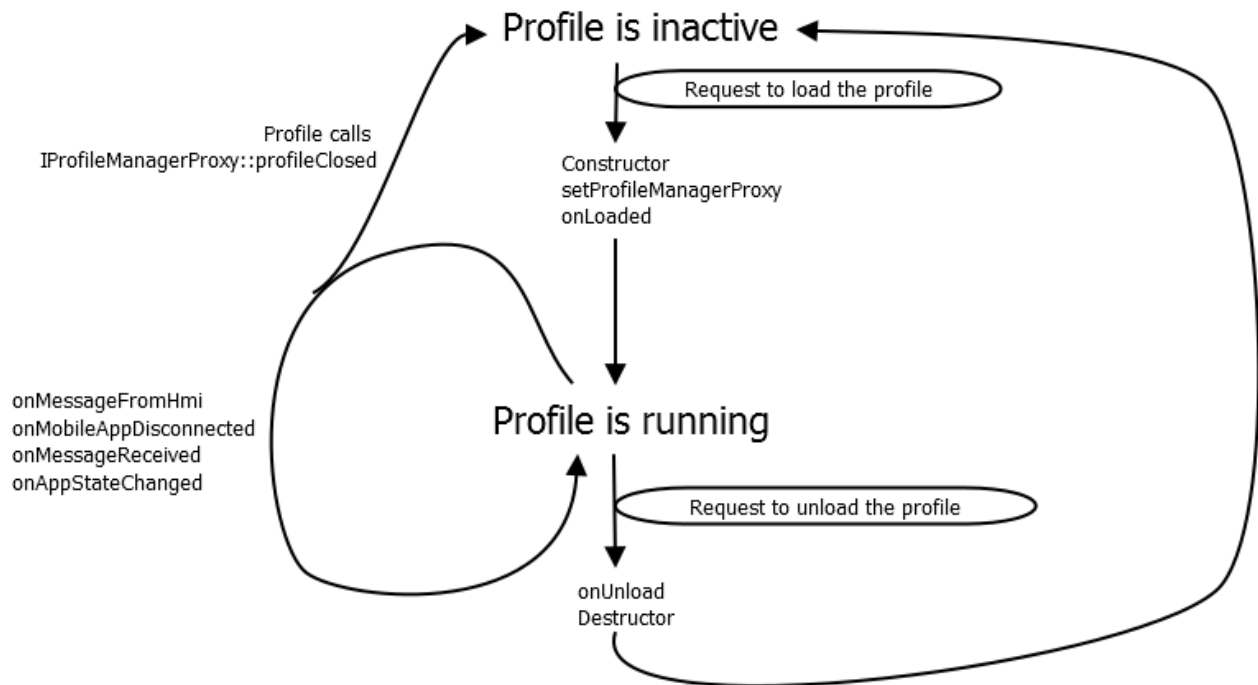
Profile

General profile lifecycle has the following stages:

1. Profile is not installed on the head unit (optional, it could be preinstalled).
2. A request to add profile is sent by a mobile application (optional, installs profile onto head unit).
3. A request to load profile is sent by a mobile application.

See the diagram below.

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014



Each profile inherits the `IProfile` interface provided by the SDLP⁴.


This class is a base interface for profiles. It provides a set of callbacks for incoming actions handling.

`onMessageReceived` – a message has been sent from a mobile application. Parameters: `appID` – the id of the application that has sent the message (can be used to send a response); `data` and `dataSize` – contents and length of the message.

`onAppStateChanged` – mobile application state has been changed. Mobile application states: foreground, background and lock screen. This message is **not** sent from the mobile app automatically, so mobile app developers should send it manually if their profile-app logic requires so.

`onLoaded` – informs the profile that it has been created and initialized (given `alProfileManagerProxy` instance via `setProfileManagerProxy`). The `appID` parameter is the ID of the application that has requested the load. `onLoaded` is called only once; if a profile has

⁴SDL_Core/profileApi/IProfile.h

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

been loaded and another application tries to load it again (while it is still loaded), the `onLoaded` callback will not be called, and application will receive the **IN_USE** result code.

`onUnload` –informs that a mobile application (its ID provided) has requested the profile to be unloaded. After this callback the `ProfileProcess` is closed. To close the `ProfileProcess` without a request from a mobile app, use the `profileClosed` callback of the `IProfileManagerProxy`.

`onMessageFromHmi` – called when the Headunit HMI sends a message to the profile. Parameters are: application name (can be used to send a response); data and `dataSize` (contents and size of the message). Contents of the message depend on the HMI implementation⁵.

`onMobileAppDisconnected` – callback called when system loses a link with a mobile application. In general it means that mobile application

`setProfileManagerProxy` – system uses this callback to pass profile pointer to profile manager proxy. This proxy allows profile developer to send messages into mobile applications and get access to standard HMI interface provided by the SDLP technology.


Profile manager proxy

Profile manager proxy class is an interface provided by the SDLP system to the profile developer to interact with mobile applications and the HMI layer. This interface is located in `IProfileManagerProxy.h`⁶; it provides the following methods:

`sendProfileToAppMessage` –send a message to aconnected mobile application (destination mobile application is determined by the `appID` parameter).

⁵HMI developers may use the provided zmq Sender and Receiver sockets (see `SDL_Core/src/components/utis/zmq/zmq_socket/` and `SDL_Core/src/components/profile_manager/include/profile_manager/appman_hmi_protocol`) for communication with profiles. A `SenderZmqSocket` should be connected to the `FROM_APPMAN` address, and a `ReceiverZmqSocket` – to the `TO_APPMAN` address. `NsProfileManager::ProfilesAppsMessages` should be used to serialize/deserialize communication.

⁶`SDL_Core/profileApi/IProfileManagerProxy.h`

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

`broadcastProfileToAppMessage` – send a message to all connected mobile applications that have communicated (sent a message, a load request, a mobile app state notification) with this profile instance.

`profileClosed` – stop theProfileProcess (the IProfile instance will be destroyed). After the ProfileProcess has stopped, all connected mobile applications that have communicated with this profile instance will be notified.

`sendHmiRequest` – send a message to the head unit HMI (contents depend on profile and HMI implementations).


Sample profile

A sample profile is given in appendix B. This code shows basic steps to develop a profile. This code uses functions described above and has comments that explain chief points.

SDLP benefits

SDL provides fixed HMI RPC API for mobile application developers, but the profile management mechanism introduced in SDLP allows creating additional API of the head unit.

This technical solution makes it possible for OEMs to provide access to some additional functions, for example: access to the internal file storage for media or another data, additional diagnostic information, free-form communication with the HMI (and not a fixed set of requests like in basic SDL), etc. Another point is that a profile could contain some part of internal business logic of the mobile application – if mobile apps for both platforms (iOS and Android) are developed, some parts of their internal logic may be moved to the profile, thus removing the necessity of coding similar things twice – in Java and Objective-C.

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

SDLP Android proxy

The aim of this section is to help Android-developers create their own applications that use SDLP-profiles with full support of Wi-Fi transport adapter. It'll contain main steps and tips necessary to do this. SDLP Android proxy code is available at theGithub public repository⁷.

Android permissions required for SDLP-enabled mobile applications

Developers need to add the following permissions to their application's AndroidManifest.xml.

- To use Bluetooth connectivity: android.permission.BLUETOOTH and android.permission.BLUETOOTH_ADMIN;
- Wi-Fi basic functionality (without device discovery): android.permission.INTERNET;
- Wi-Fi with discovery: android.permission.INTERNET, android.permission.ACCESS_WIFI_STATE and optionally (some devices require it) android.permission.CHANGE_WIFI_MULTICAST_STATE.

TCP/IP Connectivity

TCP/IP: Wi-Fi, USB

SDLP provides a means to use Wi-Fi/USB to connect a mobile device to head unit.USB works when the phone has personal hotspot turned on and plugged into the headunit.


Device discovery

To start device discovery, create an instance of `com.smartdevicelink.tcpdiscovery.TcpDiscoverer`, provide a listener and call `performSearch` (non-blocking, will return immediately)

TcpDiscoverer's listeners:

- `com.smartdevicelink.tcpdiscovery.TcpDiscovererCallbackDefaultImplDelegate`— when a listener implementing this interface is provided to the `TcpDiscoverer`, upon finishing discovery, if any devices are found, it will display a popup with list of found devices and will wait for user to pick one of them or dismiss the popup.
- `com.smartdevicelink.tcpdiscovery.TcpDiscovererCallback` could be used to get information about found devices, if the mobile app developer wants some non-standard behavior (when he doesn't want a popup to be shown).

⁷SDL_Android/SmartDeviceLinkProxyAndroid

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

Using overview

Profile Manager proxy listener

Interface `com.smartdevicelink.proxy.interfaces.IProfileManagerListener` allows applications that use profile mechanisms instead of standard SDL HMI JSON RPC to get notifications about link up and link down events. These events are called instead of standard `onHmiStatus`, `onProxyClosed` and `onError` events. A developer can use the standard `com.smartdevicelink.proxy.interfaces.IProxyListenerALM` instead, if he wants to use the standard SDL RPC as well as communicate with profiles.

```
public interface IProfileManagerListener extends IProxyListenerProfileManager {
    /**
     * Called instead of the onOnHmiStatus callback
     */
    public void onLinkUp();

    /**
     * Called instead of onProxyClosed or onError
     */
    public void onLinkDown();
}
```

`IProfileManagerListener` inherits a set of events from the

`com.smartdevicelink.proxy.interfaces.IProxyListenerProfileManager` interface:


```
// ***** responses *****//
public void onAddProfileResponse(AddProfileResponse response);
public void onRemoveProfileResponse(RemoveProfileResponse response);
public void onLoadProfileResponse(LoadProfileResponse response);
public void onUnloadProfileResponse(UnloadProfileResponse response);
public void onSendMessageToProfileResponse(SendAppToProfileMessageResponse response);
public void onAppStateChangedResponse(AppStateChangedResponse response);

// ***** notifications *****//

public void onProfileClosed(OnProfileClosed notification);
public void onReceiveMessageFromProfile(OnSendProfileToAppMessage notification);
```

See table below for description of requests, responses and notifications.

Copyright © 2013, Luxoft	Confidential	Page: 13 of 36
--------------------------	--------------	----------------

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

SmartDeviceLinkProxyALM class

`com.smartdevicelink.proxy.SmartDeviceLinkProxyALM` class is the main class used to call SDLP RPC methods.

It implements and inherits all main methods that are necessary to interact with smart device link core including SDLP RPC. So it provides profile management functionality for mobile apps developers:

- **loadProfile**(StringprofileId,IntegercorrelationID
- **unloadProfile**(StringprofileId,IntegercorrelationID)
- **addProfile**(StringprofileId,byte[]profileBinData,IntegercorrelationID)
- **removeProfile**(StringprofileId,IntegercorrelationID)
- **appStateChanged**(StringprofileId,MobileAppStatestate,IntegercorrelationID)
- **sendAppToProfileMessage**(StringprofileId,byte[]message,IntegercorrelationID)

`loadProfile` requests the Profile Manager to load a DLL that matches given profileID.

`unloadProfile` requests the Profile Manager to terminate this profile's process.

`addProfile` method is designed to extend head unit dynamically by sending a new profile library as binary data.

`removeProfile` request to remove this profile's DLL from the head unit.

`appStateChanged` method is designed profile about application state change.

`sendAppToProfileMessage` request is a way to transmit messages from mobile application to running profile. This method is the most frequently used until the profile is active.

Responses on profile requests


`LoadProfileResponse` codes:

- `INVALID_ID` – no profile with given ID in system;
- `IN_USE` – profile with givenID has already been loaded;
- `GENERIC_ERROR` – other error on load profile;
- `SUCCESS` – profile has been loaded.

`UnloadProfileResponse` codes:

- `INVALID_ID` – profile is not active or does not exist

Copyright © 2013, Luxoft	Confidential	Page: 14 of 36
--------------------------	--------------	----------------

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

- SUCCESS – profile has been unloaded

AddProfileResponse codes:

- IN_USE – Profile with given ID is currently active;
- INVALID_DATA – wrong order of received data frames with profile;
- SUCCESS – profile has been added to the system.

RemoveProfileResponse codes:

- IN_USE – Profile with given ID is currently active;
- INVALID_ID – profile with given ID does not exist;
- SUCCESS – profile has been removed from the system.


SendAppToProfileResponse codes:

- INVALID_ID – profile is not active or does not exist;
- SUCCESS – message received by profile

AppStateChangedResponse codes:

- INVALID_ID – profile is not active or does not exist;
- SUCCESS – message received by profile.

See the table in Appendix D for comparison with iOS and some additional comments.

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

SDLP iOS proxy

Main classes overview

SDLProxy – proxy of SmartDeviceLinkCore running in the Headunit; is used for establishing connection, sending requests to the HU, and receiving responses and notifications from the HU

SDLProxyFactory – can be used to instantiate a *SDLProxy* (depending on the method that was used means of connecting to the HU will differ)

SDLProxyListener – protocol for callbacks (notifications and responses) from SDL Core running in the Headunit. All callbacks are posted to the main queue.

SDLRpcRequestFactory – provides convenience methods for creating and initializing RPC requests that then may be send to the SDL Core

SDLTcpDiscoverer – may be used to obtain list of IP addresses of devices with SDL Core processes running in the same WiFi network

How to use

Building application with the SmartDeviceLinkframework

Add the library project to the same XCode workspace where your application is. Drag and drop the SmartDeviceLink onto your project. In your project's Target Settings (Build Phases) add the following libraries to "Link Binary With Libraries": *QuartzCore*, *ExternalAccessory*. Generate the framework files (follow this [link](#) for instructions). Add the generated `SmartDeviceLink.framework` to your project's dependencies.

Proxy creation – connectivity options


If you intend to use USB as transport, use *SDLProxyFactory's* `buildProxyWithListener:` (HU must have the Apple authentication chip; our implementation of SDL Core does not have the capabilities to accept such connection).

If you intend to use TCP/IP overWiFi or USB as transport, use

SDLProxyFactory's `buildProxyWithListener:tcpIPAddress:tcpPort:`. To obtain IP address and port of the HU *SDLTcpDiscoverer* may be used (see below). To use USB for the TCP/IP connection, mobile phone's personal hotspot should be enabled and the phone should be plugged into the headunit. This:

- Does not require the head unit to have the Apple auth chip
- Won't work with devices running iOS7 and newer (until libimobiledevice has been fixed)

Copyright © 2013, Luxoft	Confidential	Page: 16 of 36
--------------------------	--------------	----------------

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

TCP/IP discovery

SDLTcpDiscoverer may be used to obtain list of IP addresses of devices with SDL Core processes running in the same WiFi network. To use it, you should create a class conforming to one of the protocols:

1. ***SDLTcpDiscovererListener*** – one of the following callbacks will be received after the observed *SDLTcpDiscoverer* instance has finished searching: `onFoundDevices`: with list of *SDLTcpDiscoveredDevice* instances (ip, port, hostname and unique id of the device running SDL Core may be obtained from them), or `onFoundNothing` if no headunits had been found.
2. ***SDLTcpDiscovererDefaultListenerDelegate*** – a standard way of handling device discovery is provided: after searching, if some devices have been found, a popup window with list of hostnames and a dismiss button is displayed, and user may either pick one of them or dismiss the popup. `onUserSelectedDeviceWithIP:tcpPort:` is called if the user has clicked one of the items in the displayed list of discovered devices. If the user has clicked the dismiss button, `onUserCanceledAlert` will be called (then you may, for example, stop device discovery, or restart it). If the observed *SDLTcpDiscoverer* instance has found nothing, `onFoundNothing` will be called. Additionally, the method `dismissButtonTitle` may be implemented – the value returned from it will be displayed in the dismiss button and may reflect what will happen when the user click it (like “Stop searching” or “Search again”)


To perform device discovery, one should create an instance of *SDLTcpDiscoverer* (either with `initWithListener:` and an implementation of *SDLTcpDiscovererListener*, or `initWithDefaultListener:` and an implementation of *SDLTcpDiscovererDefaultListenerDelegate*) and call `performSearch`. Discovery is done on a background thread (`performSearch` returns immediately, the discovery process itself takes about two seconds). All callbacks are posted to the main queue.

Proxy usage

After an instance of *SDLProxy* has been created, it will try to establish connection with an SDL Core process. After establishing connection `onProxyOpened` will be called in all registered listeners (in addition to the *SDLProxyListener* implementation that is mandatory to create an instance of *SDLProxy*, more listeners may be registered with *SDLProxy*'s `saddDelegate:`). At that moment one of the listeners should send *aSDLRegisterAppInterface* request (IMPORTANT: if it has not been sent, one second after establishing connection it will be closed). After your listeners have received `aonRegisterAppInterfaceResponse:` callback, you may use SDLP fully (load profiles, bind buttons, etc.). If connection between the phone and HU has been broken, `onProxyClosed` will be called.

Responses are associated with requests by a correlation ID (int number).

Copyright © 2013, Luxoft	Confidential	Page: 17 of 36
--------------------------	--------------	----------------

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

The following requests may be used for profile management purposes:

- ***SDLLoadProfile*** – request to load a profile by its name
- ***SDLUnloadProfile*** – request to unload a profile by its name
- ***SDLAddProfile*** – request to add a profile *.so to the system. Use only *SDLRPCRequestFactory's* `buildAddProfileWithName:rawData:correlationID:` or `buildAddProfileForEmbeddedPath:profileName:correlationID:` (the first parameter is the name of a *.so file that is included into your Xcode project as a resource) to obtain an array of *SDLAddProfile* requests containing data of the profile. The first *SDLAddProfile* in the array will have a correlation ID that has been passed to the building method, and the last one will have (passedCorrelationID + ([resultArray count] – 1)). You should store the correlation ID of the last request part, and when you receive a response to it, you may load the profile (if it has been added successfully).
- ***SDLRemoveProfile*** – request to delete a profile library file from the system
- ***SDLSendAppToProfileMessage*** – request to send some data to a profile
- ***SDLAppStateChanged*** – request to inform a profile that the mobile application state has changed (for example when it went to foreground or background)

Now we will take a closer look at the part of the *SDLProxyListener* protocol that is related to profile management:

– (void) onOnProfileToAppMessage: (SDLOnProfileToAppMessage*) notification – called when a profile sends a message to your application (from the notification you can get the name of the profile and message either as *NSData* or *NSString*)


– (void) onOnProfileUnloaded: (SDLOnProfileUnloaded*) notification – called when a profile process dies (you will receive this notification only if you have sent some requests (like a load request, or a message) related to this profile beforehand)

– (void) onAddProfileResponse: (SDLAddProfileResponse*) response – received as a response to a *SDLAddProfile* request. Possible result codes: *IN_USE* (profile with the same name has been already loaded), *INVALID_DATA* (*SDLAddProfile* requests has been sent in incorrect order), *SUCCESS*.

– (void) onAppStateChangedResponse: (SDLAppStateChangedResponse*) response – response to a *SDLAppStateChanged* request. Possible result codes: *INVALID_ID* (profile with the name specified in request has not been loaded), *SUCCESS*

– (void) onLoadProfileResponse: (SDLLoadProfileResponse*) response – response to *SDLLoadProfile*. Possible result codes: *INVALID_ID* (no *.so file for profile with this name in system – time to send some *SDLAddProfile* requests), *IN_USE* (profile has already been loaded prior to this request and may be used), *GENERIC_ERROR* (when a profile process cannot be

Copyright © 2013, Luxoft	Confidential	Page: 18 of 36
--------------------------	--------------	----------------

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

started due to some system error in HU), SUCCESS (profile has been loaded and may be used)

– (void) onRemoveProfileResponse: (SDLLoadProfileResponse*) response – response to *SDLRemoveProfile*. Result codes: IN_USE (profile is currently loaded, cannot remove its *.so file), INVALID_ID (there was no file for this profile), SUCCESS (profile library has been removed)

– (void) onUnloadProfileResponse: (SDLUnloadProfileResponse*) response – response to *SDLUnloadProfile*. Results: INVALID_ID (the profile has not been loaded, so nothing has been unloaded), SUCCESS (unload request has been passed to this profile's process). Please note that receiving this response with SUCCESS code does not mean that the profile has stopped – it may take some time for it to unload, and you will receive onOnProfileUnloaded: when that happens.

– (void) onSendAppToProfileMessageResponse: (SDLSendAppToProfileMessageResponse*) response – response to *SDLSendAppToProfile* message. Result codes: INVALID_ID (a profile with the name specified has not been loaded), SUCCESS (message has been passed to the profile). Please note that it is not a “response” per se – it does not carry any payload, and if the profile sends you any response to your message, you will receive it with onOnProfileToAppMessage:. See the table in appendix D for comparison with the Android proxy.

Simple sample


```
#import <Foundation/Foundation.h>
#import "SDLProxy.h"
#import "SDLTcpDiscoverer.h"

@protocol SDLLogicListener
- (void) onProfilesReady;
- (void) onReceiveData:(NSData*)data fromProfile:(NSString*)profileName;
@end

@interface SDLLogicBase :NSObject<SDLProxyListener,
    SDLTcpDiscovererDefaultListenerDelegate>
@property (weak) id<SDLLogicListener> delegate;
+ (id) sharedInstance;
    (NSArray*)profilesToLoadwithAppInterfaceName:(NSString*)appInterfaceName;
+ (id) getInstance;
- (void) sendData:(NSData*)data toProfile:(NSString*)profileName;
@end
```


Upon creation (createInstance:withAppInterfaceName: - the first parameter should be a *NSArray* with *NSStrings* containing names of profiles that should be loaded, the second one is the name of your application that may be displayed in headunit HMI's list of apps) *SDLLogic* tries to discover a HU and connect to it. After connection has been established, it will try to load all profiles from the list. If some profiles are not present in the headunit, it is expected that their

Copyright © 2013, Luxoft	Confidential	Page: 19 of 36
--------------------------	--------------	----------------

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

binaries are present in the phone, and *SDLLogic* will try to send them and then attempt to load again. After all profiles have been loaded, delegate's `onProfilesReady` will be called. If one of the profiles sends data to the application, delegate's `onReceiveData:fromProfile:` will be invoked with the message from the profile and that profile's name.

An example of use is in appendix C.

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

Appendix A

Build instructions

* Introduction

=====

smartDeviceLinkCore is an application which manages the transport, connection and communication between a head unit and mobile device.

Software version 1.0

* OS and Hardware

=====

Ubuntu 12.04.01 LTS 32-bit OS on the PC with USB-dongle

Application has been tested using 2 types of USB-dongle:

D-Link DBT-122

STLab B-121mini

* External components

=====

Run the setup_env.sh script to install all necessary dependencies.

* Build application

=====

We support "out of sources" concept for build.

It means all generated by build tools files will be stored in separate folder.

1. Create directory outside of SDLP project directory.

For example "build" folder in the same folder with SDLP git repo folder which has a name "git_repo":

```
mkdir build
```

You will have folders structure like this:

```
/home/projects/SDLP
```

```
|--build
```

```
|--git_repo
```

```
    |--SDL_Core
```

```
        |--doc
```

```
        |--src
```

```
        |--DoxyFile
```

```
    \--CMakeLists.txt
```

```
        |--SDL_Android
```

```
        |--SDL_iOS
```

```
        |--profiles
```

Enter build folder:

```
cd build
```

2. Create build configuration using cmake:

2.1 For Debug configuration

```
cmake ../git_repo
```

2.2 For Release configuration, run:

```
cmake -DCMAKE_BUILD_TYPE=Release ../git_repo
```

3. Make project:


```
make
```

Ready to use release application smartDeviceLinkCore will be in build/src/appMain/smartDeviceLinkCore

4. Additionally it could be installed into build/bin folder

```
make install
```

Copyright © 2013, Luxoft	Confidential	Page: 21 of 36
--------------------------	--------------	----------------

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

Ready to use release application will be in build/bin/

Running application
=====

If you want to use Bluetooth:
Plug USB-dongle in.

Switch Bluetooth on a mobile device ON and make the device discoverable.

Pair mobile device with PC using Ubuntu tools.

Device should contain SDLP compatible application installed.

If you want to use Wi-Fi:

Connect device to the head unit Wi-Fi network.

Start application with command:

./smartDeviceLinkCore

Application starts to search devices and starts HMI in chromium-browser.

In case HMI has not been started please start web-based HMI manually in browser opening SDL_Core/src/components/HMI/index.html.

SmartDeviceLinkCore is searching Bluetooth devices with a correspondibg service.

Go to info menu in HMI and press App button.

Press change Devices button.

Select the device from a list.

Application opens all available ports on devices and starts communication.

Returning to the App menu all applications will be shown in a list.

* SDLP Profiles examples

=====

Open directory git_repo/profiles/.

This folder has structure like this:

profiles:

|--TestEchoProfile

|--TestSendingProfile

TestEchoProfile folder contains profile designed to test profile functionality.

This profile with mobile test application provide echo client-server test system.

TestSendingMsgProfile folder contains profile designed to test message transmissionbetween HU and mobile application functionality.

* Android

=====

SDLP Android part is located in SDL_Android folder. It contains SDLP Android proxy library and SDLP tester application.

SDL_Android

|--SmartDeviceLinkProxyAndroid - contains SDLP Android proxy

|--SmartDeviceLinkTester - contains SDLP tester application

SDLP Android proxy library is designed to be used in SDLP-enabled application to interact with SDLP core.SDLP tester application is designed to test SDLP functionality.

* iOS

===

SDLP iOS part is located in SDL_iOS folder. It contains SDLP iOS proxy and SDLP iOS tester application and SDLP Profiles Tester.


SDL_iOS

|--SmartDeviceLink - contains SDLP iOS proxy

|--SmartDeviceLinkTester - contains SDLP tester application

|--ProfilesTester - contains SDLP Profiles tester application

Copyright © 2013, Luxoft	Confidential	Page: 22 of 36
--------------------------	--------------	----------------

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

Appendix B

Sample profile header file

SampleProfile.h⁸

```
#ifndef TESTECHOPROFILE_H
```

```
#define TESTECHOPROFILE_H
```

```
#include "Logger.hpp"
```

```
#include "IProfile.h"
```

```
classTestEchoProfile : public NsProfileManager::IProfile
```

```
{
```

```
public:
```

```
TestEchoProfile();
```

```
~TestEchoProfile();
```

```
// From IProfile
```


```
virtual void onMessageReceived(constintappId, const char * data, constintdataSize);
```

```
/**
```

```
 * @brief onAppStateChanged Mobile application state changes handling
```

```
 * @param state Mobile application state
```

⁸ Sample profile header file: profiles/TestEchoProfile/include/TestEchoProfile.h

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

```

    * @paramappId Application id

    */

virtual void onAppStateChanged(constintappId, constNsProfileManager::MobileAppState state);

/**
 * @brief onLoaded - first callback - will be called right after creation with app id
 * that requested the load. ProfileManagerProxy is set already and can be used.
 * @paramappId Application id
 */
virtual void onLoaded(constintappId);

/**
 * @brief onUnload Unloading profile handling
 * @paramappId Application id
 */
virtual void onUnload(constintappId);


/**
 * Called when a notification from the HMI comes.
 * Pointer to the data shouldn't be held onto, as it is deallocated after method exit
 * @return true if the data received may be deleted
 */

virtual void onMessageFromHmi(std::string const&applicationName, const char * data,
constintdataSize) {}

/**
 * @brief setProfileManagerProxy Set profile callbacks
 * @param callbacks Profile callbacks

```

Copyright © 2013, Luxoft	Confidential	Page: 24 of 36
--------------------------	--------------	----------------

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

*/

```
virtual void onMobileAppDisconnected(const int appID) {}
```

```
/**
```

```
 * @brief setProfileCallbacks Set profile callbacks
```

```
 * @param callbacks Profile callbacks
```

```
*/
```

```
virtual void setProfileManagerProxy(NsProfileManager::IProfileManagerProxy * callbacks);
```

```
private:
```

```
NsProfileManager::IProfileManagerProxy * mCallbacks;
```


```
std::string mName;
```

```
static Logger mLogger;
```

```
int mAppId;
```

```
};
```

```
#endif //TESTECHOPROFILE_H
```

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

Sample profile source file

SampleProfile.cpp⁹

```
#include "TestEchoProfile.h"
```

```
using namespace NsProfileManager;
```

```
Logger TestEchoProfile::mLogger = Logger::getInstance(
    LOG4CPLUS_TEXT("SDL.TestEchoProfile"));
```


```
#define LOADMSG "KONICHIWA! WATASHI WA EKO!!!"
```

```
extern "C" IProfile *CreateProfile()
{
    return new TestEchoProfile;
}
```

```
extern "C" void DestroyProfile(IProfile *profile)
{
    delete profile;
}
```

```
TestEchoProfile::TestEchoProfile()
    : mCallbacks(NULL),
    mName("TestEchoProfile"),
    mAppId(0)
```

⁹Sample profile source file: profiles/TestEchoProfile/src/TestEchoProfile.cpp

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

```

{
    LOG4CPLUS_TRACE_METHOD(mLogger, __PRETTY_FUNCTION__);
}

TestEchoProfile::~TestEchoProfile()
{
    LOG4CPLUS_TRACE_METHOD(mLogger, __PRETTY_FUNCTION__);
}

void TestEchoProfile::onMessageReceived(const int appId, const char *data, const int dataSize)
{
    LOG4CPLUS_TRACE_METHOD(mLogger, __PRETTY_FUNCTION__);

    LOG4CPLUS_INFO(mLogger, "Message received : " + std::string(data, dataSize));
    LOG4CPLUS_INFO(mLogger, "Message received : dataSize = " << dataSize);
    LOG4CPLUS_INFO(mLogger, "Message received : appId = " << appId);

    mCallbacks->sendProfileToAppMessage(appId, data, dataSize);
}


void TestEchoProfile::onAppStateChanged(const int appId, const MobileAppState state)
{
    LOG4CPLUS_TRACE_METHOD(mLogger, __PRETTY_FUNCTION__);

    std::string response;

    if (mCallbacks)
    {
        switch (state)

```

Copyright © 2013, Luxoft	Confidential	Page: 27 of 36
--------------------------	--------------	----------------

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

```

        {

case MOBILE_APP_BACKGROUND:

            LOG4CPLUS_INFO(mLogger, "MOBILE_APP_BACKGROUND received");

response = "MOBILE_APP_BACKGROUND";

break;

case MOBILE_APP_FOREGROUND:

            LOG4CPLUS_INFO(mLogger, "MOBILE_APP_FOREGROUND received");

response = "MOBILE_APP_FOREGROUND";

break;

case MOBILE_APP_LOCK_SCREEN:

            LOG4CPLUS_INFO(mLogger, "MOBILE_APP_LOCK_SCREEN received");

response = "MOBILE_APP_LOCK_SCREEN";

break;

default:

return;

        }

mCallbacks->sendProfileToAppMessage(appId, response.c_str(), response.size());

    }

}

voidTestEchoProfile::onUnload(constintappId)

{

    LOG4CPLUS_TRACE_METHOD(mLogger, __PRETTY_FUNCTION__);


}

voidTestEchoProfile::onLoaded(constintappId)

{

```

Copyright © 2013, Luxoft	Confidential	Page: 28 of 36
--------------------------	--------------	----------------

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

```


        LOG4CPLUS_TRACE_METHOD(mLogger, __PRETTY_FUNCTION__);

        if (mCallbacks)
        {
            mAppId = appId;
        }
    }

void TestEchoProfile::setProfileManagerProxy(IProfileManagerProxy *callbacks)
{
    LOG4CPLUS_TRACE_METHOD(mLogger, __PRETTY_FUNCTION__);

    if (callbacks)
    {
        mCallbacks = callbacks;
    }
}

```

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

Appendix C

iOS example

```
#import "SDLLogicBase.h"
#import "SDLProxy.h"
#import "SDLTcpDiscoverer.h"
#import "SDLProxyFactory.h"
#import "SBJSON.h"

staticSDLLogicBase * instance;


@interface SDLLogicBase ()
{
    SDLProxy * mProxy;
    // name of application that may be displayed in headuint's HMI
    NSString * mName;
    SDLTcpDiscoverer * mDiscoverer;
    // correlation ID of latest request sent to HU
    intmCorrelationID;
    // counter of profiles that have yet to be loaded - when it reaches 0 system is
    ready
    intmCounter;
    // list of names of profiles that should be loaded
    NSMutableArray * mProfilesToLoad;
    // maps correlation id of the last SDLAddProfile request to profile name
    NSMutableDictionary * mAddProfileRequests;
    // maps correlation id of SDLLoadProfile request to profile name
    NSMutableDictionary * mLoadProfileRequests;
}
- (id) initWithProfiles: (NSArray*) profiles name: (NSString*)name;
- (void) showPopUpMessage: (NSString*)msg;
@end

@implementation SDLLogicBase

- (void) showPopUpMessage: (NSString *)msg
{
    UIAlertView * alert = [[UIAlertViewalloc] initWithTitle:@"SDL"
        message:msgdelegate:nilcancelButtonTitle:@"Dismiss" otherButtonTitles:nil];
    [alert show];
}

- (id) initWithProfiles: (NSArray*)profiles name: (NSString*)name
{
    self = [super init];
    if (self)
    {
        mCorrelationID = 100;
        mDiscoverer = [[SDLTcpDiscovereralloc] initWithDefaultListener:self];
    }
}
```

Copyright © 2013, Luxoft	Confidential	Page: 30 of 36
--------------------------	--------------	----------------

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

```

mName = name;
mAddProfileRequests = [[NSMutableDictionaryalloc] initWithCapacity: [profiles
count]];
mLoadProfileRequests = [[NSMutableDictionaryalloc] initWithCapacity: [profiles count]
];
mProfilesToLoad = [[NSMutableArrayalloc] initWithArray: profiles];
mCounter = [mProfilesToLoad count];
[mDiscovererperformSearch];
}
return self;
}

+ (id) getInstance
{
@synchronized([SDLLogicBase class])
{
if (instance == nil)
{
[NSException raise: @"Instance not created!" format:nil];
}
return instance;
}
}

+ (id) createInstance:
(NSArray*)profilesToLoadwithAppInterfaceName: (NSString*) appInterfaceName
{
@synchronized([SDLLogicBase class])
{
if (instance == nil)
{
instance = [[SDLLogicBasealloc]
initWithProfiles:profilesToLoadname:appInterfaceName];
}
return instance;
}
}


-(void)onFoundNothing
{
[mDiscovererperformSearch];
}

-(void)onUserCanceledAlert
{
[mDiscovererperformSearch];
}

-(void)onUserSelectedDeviceWithIP: (NSString *) ipaddresstcpPort: (NSString *) port
{
NSLog(@"onUserSelectedDeviceWithIP: %@, %@", ipaddress, port);

```

Copyright © 2013, Luxoft	Confidential	Page: 31 of 36
--------------------------	--------------	----------------

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

```

mProxy = [SDLProxyFactorybuildProxyWithListener: self
tcpIPAddress: ipAddress
tcpPort: port];
}

-(NSString*) dismissButtonTitle
{
return @"Continue searching";
}

-(void) onRegisterAppInterfaceResponse: (SDLRegisterAppInterfaceResponse*) response
{
for (id object in mProfilesToLoad)
{
SDLLoadProfile *req = [SDLRPCRequestFactorybuildLoadProfileWithName:object
correlationID:@(mCorrelationID)];
[mLoadProfileRequests setObject: object forKey:@(mCorrelationID)];
mCorrelationID ++;
[mProxysendRPCRequest:req];
}
}


-(void) onProxyOpened
{
NSLog(@"onProxyOpened");
SDLRegisterAppInterface* regRequest =
[SDLRPCRequestFactorybuildRegisterAppInterfaceWithAppName:mName];
regRequest.isMediaApplication = @(NO);
[mProxysendRPCRequest:regRequest];
}

-(void) onProxyClosed
{
NSLog(@"onProxyClosed");
[mProxy dispose];
mProxy = nil;
mCounter = [mProfilesToLoad count];
[selfshowPopUpMessage: @"Connection with HU has been lost!"];
[mDiscovererperformSearch];
}

-(void) onOnProfileToAppMessage: (SDLOnProfileToAppMessage*) notification
{
[[self delegate] onReceiveData:[notification messageData]
fromProfile:[notification profileName]];
}

```

Copyright © 2013, Luxoft	Confidential	Page: 32 of 36
--------------------------	--------------	----------------

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

```


-(void) onOnProfileUnloaded: (SDLOnProfileUnloaded *)notification
{
    NSLog(@"Profile Unloaded! trying to reload");
    mCounter = [mProfilesToLoad count];
    for (id object in mProfilesToLoad)
    {
        SDLLoadProfile *req = [SDLRPCRequestFactorybuildLoadProfileWithName:object
        correlationID:@(mCorrelationID)];
        [mLoadProfileRequests setObject: object forKey:@(mCorrelationID)];
        mCorrelationID ++;
        [mProxysendRPCRequest:req];
    }
}

-(void) onAddProfileResponse: (SDLAddProfileResponse*) response
{
    NSString * profileName = [mAddProfileRequestsobjectForKey: [response correlationID]];
    if (profileName != nil)
    {
        SDLLoadProfile *req = [SDLRPCRequestFactorybuildLoadProfileWithName:profileName
        correlationID:@(mCorrelationID)];
        [mLoadProfileRequests setObject: profileName forKey:@(mCorrelationID)];
        mCorrelationID ++;
        [mProxysendRPCRequest:req];
        [mAddProfileRequestsremoveObjectForKey: [response correlationID]];
    }
}

-(void) onLoadProfileResponse: (SDLLoadProfileResponse*) response
{
    NSString * profileName = [mLoadProfileRequestsobjectForKey: [response
        correlationID]];
    NSLog(@"profile: %@", profileName);
    if ([response isSuccess] || [[response resultCode] isEqual: [SDLResult IN_USE]])
    {
        mCounter --;
        if (mCounter == 0)
        {
            [selfshowPopUpMessage: @"Connection established!"];
            [[self delegate] onProfilesReady];
        }
    }
    else if ([[response resultCode] isEqual: [SDLResult INVALID_ID]])
    {
        NSLog(@"No profile binary on HU, sending");
    }
}

```

Copyright © 2013, Luxoft	Confidential	Page: 33 of 36
--------------------------	--------------	----------------

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014


```

NSArray * addProfile =
    [SDLRPCRequestFactorybuildAddProfileForEmbeddedPath:[NSString stringWithFormat:
        @"lib%.so", profileName]
    profileName:profileName
    correlationID:@(mCorrelationID)];
mCorrelationID += [addProfile count] - 1;
    [mAddProfileRequests setObject: profileName forKey:@(mCorrelationID)];
mCorrelationID ++;
for (id request in addProfile) {
    [mProxysendRPCRequest: request];
}
}
else
{
    NSLog(@"Unexpected result: %@", [response resultCode]);
    assert(false);
}
    [mLoadProfileRequests removeObjectForKey: [response correlationID]];
}

- (void) sendData:(NSData *)data toProfile:(NSString *)profileName
{
    if (mCounter != 0)
    {
        [selfshowPopUpMessage: @"No connection to HU"];
        return;
    }
    SDLSendAppToProfileMessage * req =
        [SDLRPCRequestFactorybuildSendAppToProfileMessageWithName:profileNamerawData:dat
        aCorrelationID:@(mCorrelationID++)];
    [mProxysendRPCRequest: req];
}

```

@end

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

Appendix D

Android and iOS proxiescheatsheet


Request description	Android proxy ¹⁰ method	Android response callback	iOS request class ¹¹	iOS response callback	Result codes description
Add profile – add profile's DLL to the system. The library's binary data should be split in multiple packets and sent in multiple requests.	public Integer addProfile (String profileId, byte[] profileBinData, Integer correlationID) ¹²	onAddProfileResponse	SDLAddProfile ¹³	onAddProfileResponse:	IN_USE – profile is currently loaded, unload it first and then try again INVALID_DATA – profile binary pieces are sent in incorrect order SUCCESS – profile binary piece has been appended successfully (if it was the last one, the profile may now be loaded)
Load profile – create this profile's instance and prepare it for communication.	public void loadProfile (String profileId, Integer correlationID)	onLoadProfileResponse	SDLLoadProfile	onLoadProfileResponse:	INVALID_ID – profile library is not present in system IN_USE – profile has been loaded already and is ready for communication GENERIC_ERROR – something went wrong during the ProfileProcess creation SUCCESS – profile has been loaded successfully and is ready for communication
Send message to profile	public void sendAppToProfileMessage (String profileId, byte[] message, Integer correlationID)	onSendMessageToProfileResponse	SDLSendAppToProfileMessage	onSendAppToProfileMessageResponse	SUCCESS – data has been delivered to the profile instance. The response does not carry any actual response data from the profile, data from the profile is received with the onProfileToAppMessage notification. INVALID_ID – the profile is not loaded
Send mobile app state to profile – this is not sent automatically.	public void appStateChanged (String profileId, MobileAppState state,	onAppStateChangedResponse	SDLAppStateChanged	onAppStateChangedResponse:	INVALID_ID – the profile is not loaded SUCCESS – notification has been delivered to the

¹⁰ SmartDeviceLinkProxy/SmartDeviceLinkProxyALM

¹¹ Pass instances to proxy's sendRPCRequest. May be instantiated not directly, but using SDLRPCRequestsFactory

¹² The returned value is the correlation ID of the last AddProfile request sent. After response to it, the profile can be loaded (in case of success).

¹³ Use ONLY SDLRPCRequestFactory's buildAddProfileWithName:rawData:correlationID: OR buildAddProfileForEmbeddedPath:profileName:correlationID: (the first parameter is the name of a *.so file that is included into your Xcode project as a resource) to obtain an NSArray of SDLAddProfile requests containing data of the profile. The first SDLAddProfile in the array will have a correlation ID that has been passed to the building method, and the last one will have (passedCorrelationID + ([resultArray count] - 1)).

	SDLP developer's manual	Document revision # 0.2
	Author(s): V. Plachkov, E. Bratanova	Revision Date: 19-Mar-2014

Developers should sent it manually if their profile-mobile app logic requires so.	Integer correlationID)				profile
Unload profile – destroy running instance of this profile	public void unloadProfile (String profileId, Integer correlationID)	onUnloadProfileResponse	SDLUnloadProfile	onUnloadProfileResponse:	SUCCESS – request to die has been delivered to this profile's process. Profile may stop not immediately, wait for the onProfileClosed notification to be sure. INVALID_ID - the profile is not loaded.
Remove profile – delete this profile's library from the system (after successful remove, all requests to load this profile will fail, until it has been loaded again)	public void removeProfile (String profileId, Integer correlationID)	onRemoveProfileResponse	SDLRemoveProfile	onRemoveProfileResponse:	IN_USE – the profile is currently active, unload it first and then try to remove again INVALID_ID – profile is not present in system, nothing has changed SUCCESS – profile library has been successfully removed

Notification description	Android callback	iOS callback	Comments
Profile closed – sent to all mobile applications that have ever communicated with this profile instance (tried to load it, sent messages or mobile app states)	public void onProfileClosed (OnProfileClosed notification);	-(void) onOnProfileUnloaded : (SDLOnProfileUnloaded*)	After this notification profile is considered inactive, and may be: <ul style="list-style-type: none"> Loaded Added Removed Such operations as: <ul style="list-style-type: none"> Send mobile app state changed notification Unload the profile Send message to the profile will fail while the profile is in this state.
Message from profile – profile may either sent it to one specific mobile application, or to all applications, that have ever communicated with it.	public void onReceiveMessageFromProfile (OnSendProfileToAppMessage notification);	-(void) onOnProfileToAppMessage : (SDLOnProfileToAppMessage*)	The notification contains data from the profile.