APSDK USER MANUAL

Artificial Pancreas Software Developers Kit V2.4.1

Authors:
Patrick Keith-Hynes, Ph.D.
Boris Kovatchev, Ph.D.
Marc Breton, Ph.D.
Benton Mize
Antoine Robert

Corresponding Author:
Patrick Keith-Hynes, Ph.D., UVA Health System, P.O. Box 400888, Charlottesville, VA 22908
email: ptk4m@virginia.edu, phone: 434-243-1587, fax: 434-982-6765.

I. APSDK OVERVIEW

The Artificial Pancreas Software Developers Kit (APSDK) includes the following:

1. Android Medical OS (AMOS)
2. DiAs System Modules in binary form
3. Source code for DiAs Developer Shell modules
4. APSDK Manual

The procedure for developing, testing and deploying APMMP applications in clinical trials is shown in **Figure 1 - Regulatory Path to APMMP Clinical Trial**.

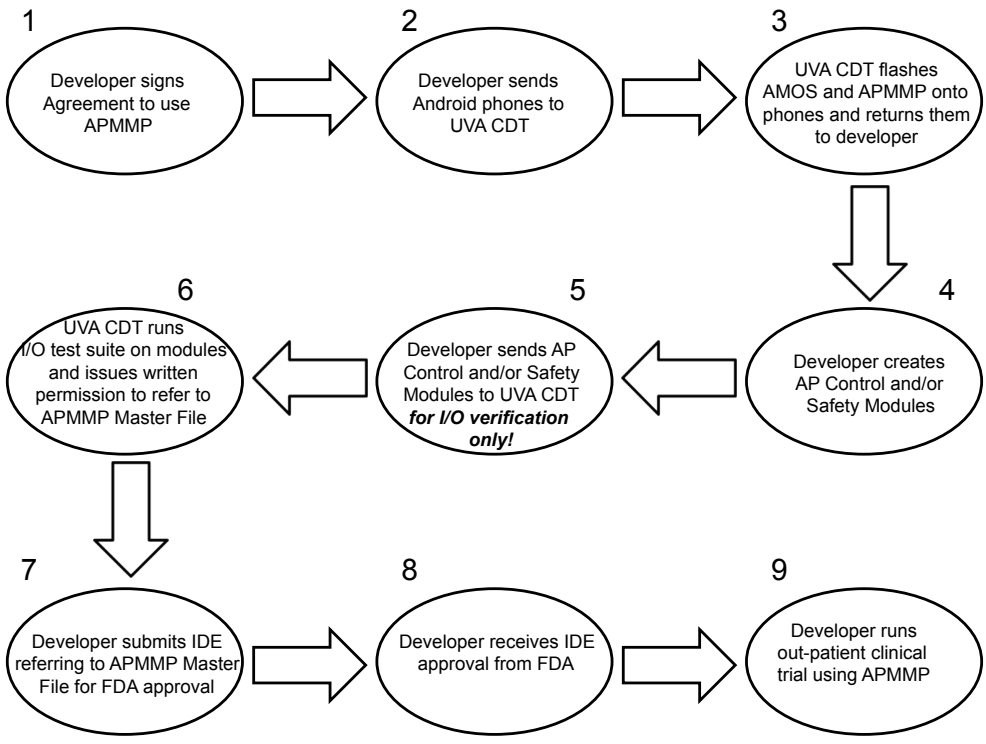


Figure 1 - Regulatory Path to APMMP Clinical Trial

After signing an agreement to use APMMP the developer ships phones (step 2) to UVA Center for Diabetes Technology (UVA CDT) to be flashed with AMOS and the APMMP binary files (step 3). This enables UVA CDT to be sure that the investigator is starting development with properly configured

phones. In certain cases UVA may provide the investigator with Android and other binary images so that the investigator may flash the phones themselves. In steps 5 and 6 UVA CDT verifies that the developer applications are interacting with the DiAs software in a safe manner. UVA CDT does not need to review the application source code and will not verify that the algorithms implemented in the applications will function properly. The tests performed by the UVA CDT are simple checks of the module inputs and outputs based upon test code that each module is required to have.

II. ARTIFICIAL PANCREAS MOBILE MEDICAL PLATFORM (APMMP)

II.A. APMMP Overview

The Artificial Pancreas Mobile Medical Platform (APMMP) is a modular software stack consisting of :

- The Android Medical Operating System (AMOS) – a build of the Android mobile operating system modified to make it suitable for use in a medical device. We refer to an approved Android cell phone with AMOS installed as a Cell Phone Medical Platform (CPMP).
- The Diabetes Assistant (DiAs) – a suite of Android applications operating in a coordinated fashion to create a research platform for outpatient testing of Artificial Pancreas (Closed Loop) blood glucose control and CGM sensor-only monitoring.

When loaded onto an approved Android cell phone and connected with a supported Continuous Glucose Monitor (CGM) and insulin pump the APMMP provides a complete mobile solution for diabetes researchers to test Closed Loop control, monitoring and Safety algorithms in an outpatient setting.

The APMMP does not include any closed loop treatment or safety algorithms. Instead it provides a user tested touch based graphical interface, connections to CGM sensors, insulin pumps and remote monitoring databases and simple, well-documented interfaces to permit developers to install and run their own closed loop and safety algorithms.

The APSDK includes the APMMP binary files, source code to sample versions of the developer applications and documentation.

II.B. Android Medical Operating System (AMOS) Description

DiAs is built on top of a modified version of the Android mobile operating system. Android includes the Linux kernel plus software extensions for process management, interprocess communication and interfacing with cell phone hardware. Specific modifications to Android include:

- Phone functions disabled
- SMS/MMS text message functions disabled
- Web browser removed
- Android Market removed
- All non-essential system applications removed

- "White-list" of applications which may be installed on system. All applications not appearing on this list are automatically blocked from installation by Android (additional run-time Configuration Management including package and version checking is performed by the DiAs software at startup).
- "Recent screens" button removed
- "Home" button pushes intercepted and redirected
- Audio volume keys disabled during audible alarms
- Speaker audio not muted by insertion of headphone jack into audio port
- Modifications to Android screen status line for use by DiAs applications

The following Android cell phones are supported by APMMP v2.0 (Table 1 - Android Phones Supported by APMMP):

Table 1 - Android Phones Supported by APMMP

Name	Mfg Model Number	Manufacturer	Distributor
Nexus 5	D820	LG Electronics	Google

II.C. Supported Peripheral Devices

Peripheral devices that have been verified for use with the APMMP are listed in Table 2.

Table 2 - Supported Peripheral Devices

Name	Type	Manufacturer	Comments
Dexcom Gen 4™	Continuous Glucose Monitor	Dexcom Inc., San Diego, CA	Bluetooth Low Energy (BLE) connection to CPMP via BTLE G4 Relay Device
Roche Accu-Chek Combo Insulin Pump	Insulin pump	Roche Diagnostics	Bluetooth (BT) connection to CPMP
Tandem AP Pump	Insulin Pump	Tandem Diabetes, San Diego, CA	BTLE connection to CPMP
Zephyr BioHarness 3	Heart Rate Monitor and Accelerometer	Zephyr Technology, Annapolis, MD	Bluetooth (BT) connection to CPMP
Zephyr HxM	Heart Rate Monitor	Zephyr Technology, Annapolis, MD	Bluetooth (BT) connection to CPMP
BTLE G4 Relay Device for Dexcom G4	Relay device	Currently distributed through the JAEB Center, Tampa, FL	Relay device which connects to a Dexcom G4 receiver and transmits CGM and meter values to CPMP via Bluetooth Low Energy (BLE)
Dexcom Share AP	Continuous Glucose Monitor	Dexcom Inc., San Diego, CA	Direct Bluetooth Low Energy (BLE) conneciton to the CPMP

II.C.1. Dexcom G4 Sensor

DexCom Gen 4™ is a CGM sensor device manufactured by DexCom, Inc., San Diego, CA. The CGM sensor will be used in combination with the Dexcom receiver. The receiver converts raw current data from the G4 CGM sensor into a stream of CGM values which it reports to the Cell Phone Medical Platform (CPMP) via a Bluetooth link provided by the BTLE_G4 relay device (described below). Calibration of the sensor is performed directly on the receiver and is reported to DiAs. Estimated blood glucose and calibration data is reported to the **CGM Device Driver** via the determined communication method. We have extensively tested the communication interface between the G4 receiver and the CPMP in several studies. During over 1,300 hours of use in challenging outpatient conditions, the data transmission between G4 receiver and the DiAs has been found accurate for use in closed loop and monitoring studies.

Investigators wishing to use the Dexcom G4 sensor with the APMMP should request permission from Dexcom to refer to the Dexcom G4 Master File.

II.C.2. Roche Accu-Chek Combo Insulin Pump

Roche Accu-Chek Combo Insulin Pump: Insulin will be delivered by an unmodified, FDA approved insulin pump, the Accu-Chek Combo Insulin Pump manufactured by Roche Diagnostics, Indianapolis, IN. The APMMP smartphone communicates wirelessly with the Accu-Chek Combo Insulin Pump through a Bluetooth interface, enabling the APMMP to request insulin delivery and to receive insulin delivery history and status information from the pump.

Investigators wishing to use the Roche Accu-Chek Combo pump with the APMMP should request permission from Roche Diagnostics.

II.C.3. BTLE_G4 Relay Device

BTLE_G4 Relay Device: The BTLE_G4 relay is a device containing a printed circuit board mounted inside a plastic box with a USB connector that mates with a Dexcom G4 receiver and transmits CGM values to the CPMP via Bluetooth Low Energy. The BTLE_G4 contains rechargeable batteries and provides power to the Dexcom G4 receiver. It may be charged via a micro-USB connector.

Figure 2 – BTLE_G4 Relay Device with Dexcom G4 Receiver Inserted

II.C.4. Dexcom Share AP

Dexcom Share™ AP is a CGM sensor device manufactured by Dexcom Inc., San Diego, CA. This version of the Dexcom system has a built-in BTLE radio to allow direct communication with the CPMP. Values are read and handled using an Android service that then stores the data on the CPMP in a database.

II.C.5. Zephyr HxM Heart Rate Monitor

The **Zephyr HxM Heart Rate Monitor** is a compact, wearable device that measures a subject's heart rate and wirelessly transmits this information to the CPMP via Bluetooth. The HxM contains a rechargeable battery which is charged using a docking station.

II.C.6. Zephyr BioHarness 3

The **Zephyr BioHarness 3** is a wearable device that measure's a subject's heart rate, acceleration and other data and wirelessly transmits this information to the CPMP via Bluetooth. The BioHarness electronics contain a battery which is charged using a charging cradle.

II.C.7. Tandem AP Insulin Pump

Insulin will be delivered by a modified FDA cleared insulin pump, the Tandem Artificial Pancreas Insulin Pump (t:AP) manufactured by Tandem Diabetes Care, San Diego, CA. The Artificial Pancreas Insulin Pump removes the ability for users to give insulin through its touch screen and instead provides a Bluetooth Low Energy (BLE) interface that permits the APMMP to request insulin delivery and to receive delivery status from the pump. Investigators wishing to use the Tandem pump with the APMMP

should request permission from Tandem Diabetes Care to refer to the Tandem Artificial Pancreas Insulin Pump Master File.

II.D. Diabetes Assistant (DiAs) Description

Figure 3 is a block diagram of the APMMP showing the relationship between the Android Medical OS and the Diabetes Assistant (DiAs). The DiAs software is a network of fourteen ***System Applications*** (of which DiAsUI, MealActivity, MCMservice and ExerciseService are ***Developer-Replaceable***) plus up to four ***Developer Applications.*** Each of the Developer-Applications and Developer-Replaceable System Applications has a **Public Application Programming Interface (API)** that is supplied to developers using the system. Each application is composed of one or more modules distributed in an "apk" file (extension **.apk**) which are installed on the APMMP system prior to operation. When an application is installed in the APMMP it is copied into the cell phone memory and its capabilities are made known to the Android operating system. The APMMP controls what modules may be installed through a set of Android modifications. In **Figure 3** ***System Applications*** are shown as orange blocks***, Developer-Replaceable System Applications*** appear as blue blocks, ***Developer Applications*** are colored green and the ***Remote Monitoring Application*** is shown as an orange cloud lying outside of the CPMP. In order for the system to start operation, the developer must supply the **SSMservice** (even if no insulin pump is included in the system) and at least one of either the **APCservice** or **BRMservice** modules (if Closed Loop operation is required). The **MealActivity** is an System Application which supports basic

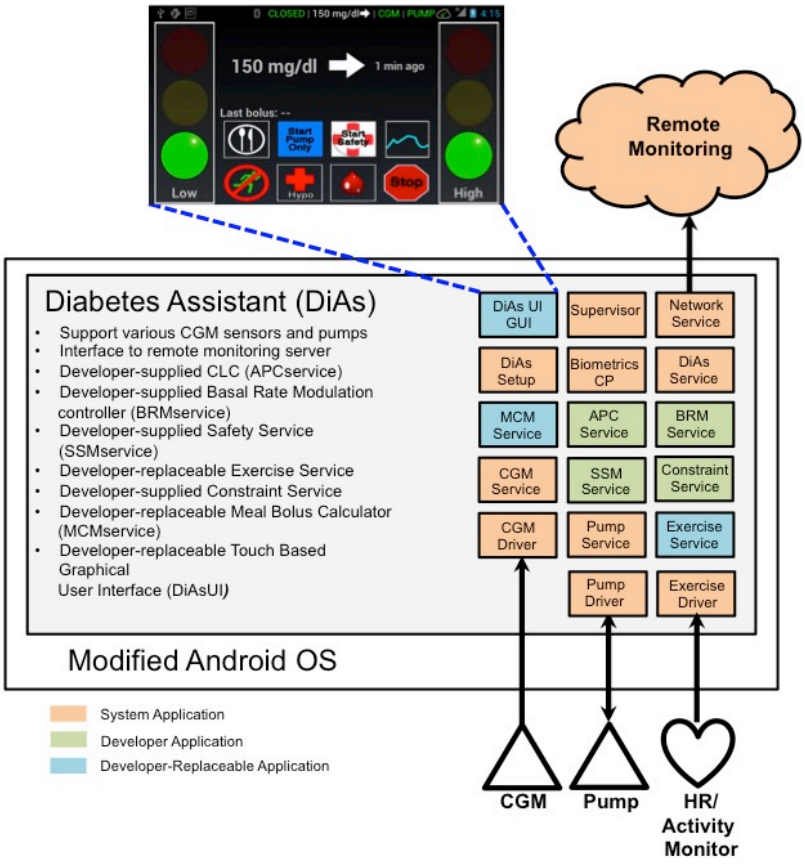


Figure 3 – APMMP Block Diagram

functionality to provide manual correction insulin. It may be replaced by developers who wish to provide their own meal screen user interface. The **MCMservice** is an application that supports the **MealActivity** by passing its output to DiAsService and optionally performing meal bolus calculations upon request. The **Basal Rate Modulation Service (BRMservice)** may be used as a secondary Closed Loop controller that is activated according to a daily schedule, or it may be used to set an constraint on the minimum amount of insulin to be delivered by the **APCservice**. The **ExerciseService** is an optional, Developer-Replaceable module which is installed if the system uses heart rate or accelerometer data. This service reads data from the EXERCISE_SENSOR_TABLE (and possibly other system data) and generates a entries in the EXERCISE_STATE_TABLE which are used to inform other DiAs modules.

II.D.1. DiAs Applications

Each DiAs application may contain multiple elements such as *Activities* (screen elements), *Services* (long running background processes), *Broadcast Listeners* (routines that run upon receipt of system-wide messages) and *Content Providers* (data sharing processes) that work together – and with elements from other applications – in order to provide a coherent working system. The DiAs uses three main communication mechanisms to coordinate the operation of the various applications:

1. *Broadcast Messages* – Some DiAs modules send broadcast *Intents* that all applications receive. An example is the system 'heartbeat' - an Intent that is broadcast by the Supervisor every 5 minutes. If an application registers a *Broadcast Listener* to receive this Intent, a code module within the application runs when the 'heartbeat' Intent arrives. Broadcast messages are also used to report changes in the state of connected devices.
2. *Directed Messages* – DiAs modules can also be *bound* to one another and send private messages back and forth between sender and receiver. For example the Safety Service maintains a bound connection with the Pump Service. Intents sent from the Safety Service to the Pump Service (and vice versa) are not received by other system applications.
3. *Central Database* – Access to the DiAs SQLite database is controlled by the biometricsContentProvider, which permits transparent access from multiple applications. CGM, insulin history, system status, SMBG, hardware and configuration data is shared among all applications through the database. This mechanism insures that all modules have access to the most up-to-date information while minimizing data transfers and associated processing overhead. It also controls which modules have **query**, **update**, **insert** and **delete** access to individual database tables. For a discussion of database contents and protections see section IV.A. Developer Applications may use the Android *ContentObserver* class to receive notification and control when a particular database table is modified. This can be particularly useful in observing the database 'cgm', 'insulin' and 'system' tables for changes in system mode or device connectivity (see Table 9, Table 10 and Table 34).

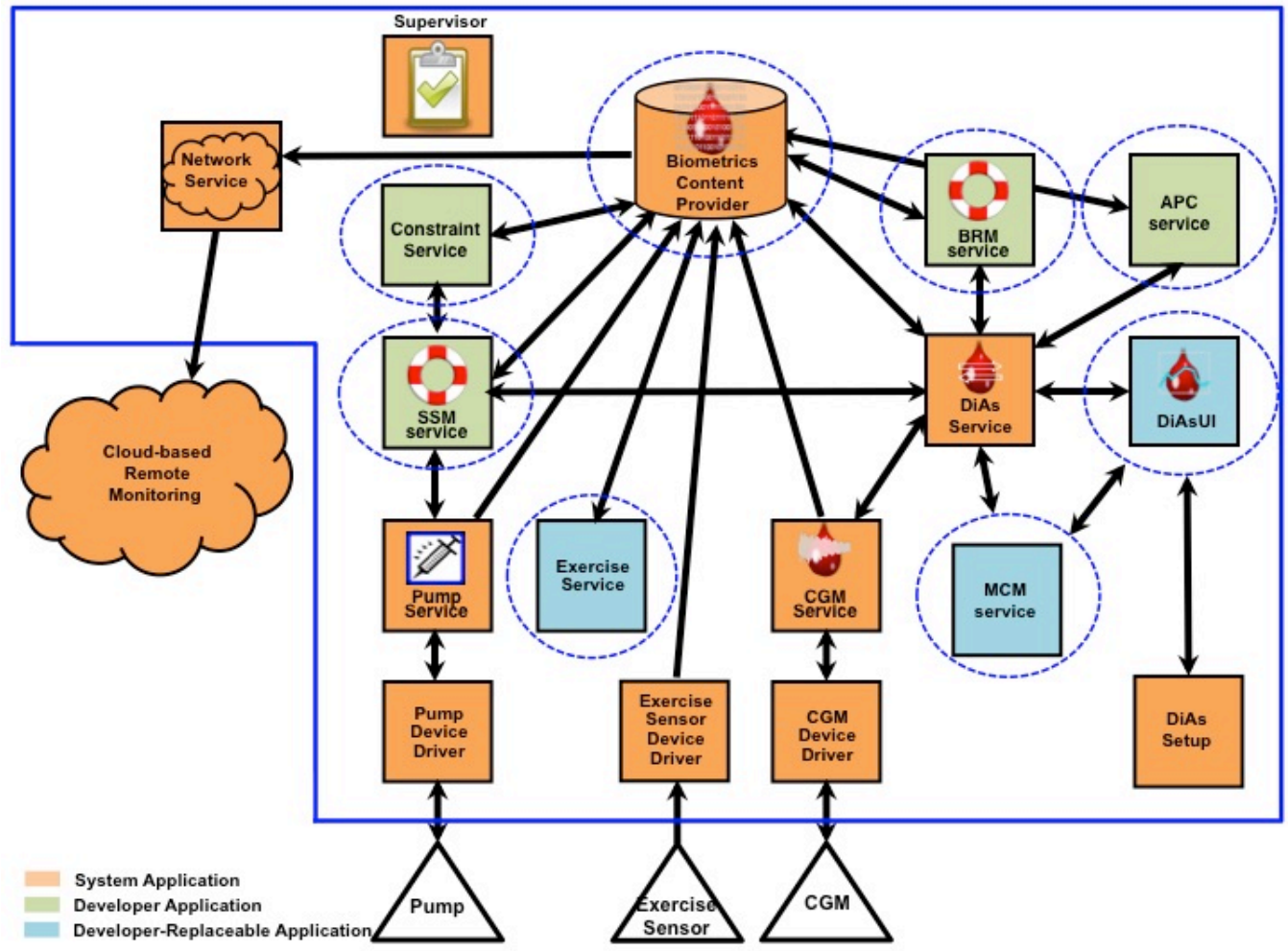


Figure 4 – APMMP Architectural Diagram

Figure 4 is a DiAs architectural diagram indicating the flow of Intent and Database messages between modules. Broadcast Intents are not represented in the diagram as they can be received by all modules. The blue dotted circles indicate modules with Public APIs. For details see section III.

II.D.1.1. Supervisor (system)

The supervisor module is responsible for system startup, configuration management and timing. Upon creation the Supervisor reads the system parameter file and stores its contents in the database. Next it reads the system configuration file, verifies that all required applications with appropriate version numbers are installed in the APMMP and starts the core modules. The Supervisor sends "heartbeat" broadcast Intents at 1 minute and 5 minute intervals which are used to coordinate activities across the system.

System parameters are stored in the file *parameters.xml* in the root level of the smartphone. This file contains a list of parameter elements of the form:

<parameter name="param_name" value="value" type="type"> </parameter>

The string "param_name" is used to access the parameter in the database. The string "type" must be one of the following:

- int
- double
- string
- long
- boolean

The string "value" must consist of characters which can be parsed into a value of the appropriate "type". Comments may be enclosed withinin <!-- and -->.

The *parameters.xml* file must at a minimum contain the parameters listed in Table 3 - Elements of *parameters.xml* which are used by core DiAs system functions. Developers may add additional parameters to the *parameters.xml* file and may access these parameters from the central database in order to customize system operation. The APSDK ships with a sample copy of *parameters.xml* which was used in verification testing of the developer's kit. **It is the responsibility of the developer of the system which makes use of DiAs to modify *parameters.xml* appropriately for their application and to verify that the overall system – including interactions with *parameters.xml* – operates correctly.**

Table 3 - Elements of *parameters.xml*

Name	Type	Value Range	Notes
allowed_modes	int	0-7	Allowed modes of operation ('Stop' and 'Sensor Mode' are always available) 0: No operating mode. 1: Pump Mode only. 2: Safety Mode only. 3: Pump and Safety Modes. 4: Closed Loop Mode only. 5: Pump and Closed Loop Modes. 6: Safety and Closed Loop Modes. 7: Pump, Safety and Closed Loop Modes available
safetyDistinctFromCL	boolean	false, true	Indicates whether Safety is considered as a distinct operating mode or a sub-mode of Closed Loop. If set to False, *allowed_modes* values 2, 3, 6 and 7 are effectively equivalent to 0, 1, 4 and

			5.
traffic_lights	int	0-3	Which module controls traffic lights 0 = None (Lights Off) 1 = SSMservice 2 = APCservice 3 = BRMservice
blood_glucose_display_units	int	0,1	Controls units in which CGM and BG are displayed. 0 = mg/dl 1 = mmol/L CGM and BG values are always stored in the Biometrics Content Provider database in units of mg/dl
meal_activity_bolus_calculation_mode	int	0-2	Controls the bolus calculation behavior of the MAF version of MealActivity. 0 = Standard bolus calculator in all modes 1 = Standard bolus calculator in Pump mode, custom calculator in Closed Loop and Safety modes. 2 = Custom calculator regardless of DiAs operating mode.
exercise_detection_mode	int	0, 1	Sets whether the Exercise button can be automatically turned ON/OFF 0 = disabled 1 = can be turned ON/OFF by ExerciseService
bolus_missed_threshold	int	[0,…]	If more than this number of boluses is missed the system will transition to Sensor or Stopped mode (depending upon CGM availability), stopping all DiAs insulin delivery. If the pump supports TBR then TBR will be cancelled so that the pump is responsible for delivering insulin according to its pre-programmed basal

			profile.
acc_enabled	boolean	false, true	Enable/disable DiAs smartphone accelerometer
gps_enabled	boolean	false, true	not used
gps_interval	long	[0-Long.MAX_VALUE]	not used
apc_enabled	int	0-3	APCservice 0: Disabled 1: Enabled (at all time) 2: Enabled within Night Profile 3: Disabled within Night Profile
brm_enabled	int	0-3	BRMservice 0: Disabled 1: Enabled (at all time) 2: Enabled within Night Profile 3: Disabled within Night Profile
tbr_enabled	boolean	false, true	Enable/disable Temporary Basal Rate functionality if supported by insulin pump
setup_screen_font_size	int	1-20	Font size in setup screens
mdi_requested_at_startup	boolean	false, true	If true DiAsService will always prompt for manually delvered insulin when DiAs transitions from Stopped mode to any insulin dosing mode (Pump, Closed Loop, Safety)
audible_alarms	boolean	false, true	Enable/disable audible alarms
vibrate_alarms	boolean	false, true	Enable/disable vibration during alarms
dwm_address_default	string	string	Default URI of secure web monitoring server
dwm_address_2	string	string	Alternate URI of secure web monitoring server
dwm_address_3	string	string	Alternate URI of secure web monitoring server
backup_password	string	string	Password for access to DiAs main menu. This password may be used in addition to a password selected by the subject.
bolus_interceptor_enabled	boolean	false, true	not used
center	string	string	Identifier of study center for use by secure web monitoring server (DWM)

protocol	string	string	Identifier of study protocol for use by secure web monitoring server (DWM)
cgm_history_hours	int	[0,…]	Maximum number of hours of CGM history to pull from the Dexcom receiver.
bg_threshold_1	int		Used by hypoglycemia handler. If hypo not treated and SMBG lower than this value the alarm is muted for hypo_mute_low_bg minutes. If hypo not treated and SMBG greater than this value but <= bg_threshold_2 then alarm is muted for hypo_mute_middle_bg minutes
bg_threshold_2	int		Used by hypoglycemia handler. If hypo not treated and SMBG greater than this value then alarm is muted for hypo_mute_high_bg minutes
hypo_mute_treated	int		Time in minutes that hypo alarm is muted if patient reports treating hypo
hypo_mute_no_bg	int		Time in minutes that hypo alarm is muted if patient does not report SMBG
hypo_mute_low_bg	int		Time in minutes that hypo alarm is muted if patient does not report treating hypo but patient reports SMBG <= bg_threshold_1
hypo_mute_middle_bg	int		Time in minutes that hypo alarm is muted if patient does not report treating hypo but patient reports SMBG > bg_threshold_1 and <= bg_threshold_2
hypo_mute_high_bg	int		Time in minutes that hypo alarm is muted if patient does not report treating hypo but patient

			reports SMBG > bg_threshold_2
enableIO	boolean	false, true	Used to enable IO logging for module compatibility testing
temporaryBasalRateEnabled	int	0-3	The Start/Cancel Temp Basal buttons will appear on the DiAsUI screen: 0: Never. ('Cancel' still may appear if Temp Basal initiated by a controller) 1: In Pump Mode only. 2: In Closed Loop Mode only. 3: Both in Pump and Closed Loop Modes.
collectBatteryStats	boolean	false, true	If true then statistics are collected about the top 20 highest consumers of smartphone battery energy
collectBatteryStatsInterval	int	[1,…]	Interval in minutes at which the battery statistics are collected
hba1c	double	[0.0, …]	Subjects HBA1C value to be stored in the parameter table

At startup the Supervisor reads the file *configurations.xml*, which contains one or more acceptable *configurations* of packages in the following format:

```
<config name="configuration 1">
  <package name="edu.virginia.dtc.packageName"   minVersion="nnn" maxVersion="mmm">
  </package>
</config>
```

Each configuration lists the packages and package version numbers that must be present. If packages corresponding to one or more valid configurations are found, the Supervisor shall start the DiAs system. If one or more of these applications are missing, or if the version number of one or more installed packages does not fall within the range [minVersion, maxVersion] from *configurations.xml*, the Supervisor shall display an error screen without starting the DiAs system. The APSDK ships with a copy of *configurations.xml* which was used in verifying the operation of the developer's kit. **It is the responsibility of the developer of the system which makes use of DiAs to generate an appropriate *configurations.xml* which references the packages and versions that were used in the developer's system testing.**

II.D.1.2. biometricsContentProvider (system)

The biometricsContentProvider maintains the DiAs SQLite database and provides an interface which allows simultaneous database access from multiple processes. It enforces permissions for each application to **query**, **update**, **insert** and **delete** data on a table by table basis. The DiAs database includes subject biometric and daily operating profile information, time stamped CGM and SMBG data, delivered insulin history, state estimation, log and other data. Details of the database and the API used to access the tables are provided in biometricsContentProvider Database Description and API, section IV.A.

II.D.1.3. DiAsSetup (system)

DiAs Setup presents the graphical user interface when the UI is in **Configuration Mode**. It enables the operator to input and edit subject biometric information, daily operating profiles (for Correction Factor, Carbohydrate to Insulin Ratio, Basal insulin and Night mode) and to configure software drivers for peripheral devices. If the DiAs system is started and no valid database exists DiAs Setup will be activated. It may also be activated from the DiAs Home Screen while the system is operating.

II.D.1.4. DiAsUI (system – developer replaceable)

The DiAsUI application provides the graphical user interface when the system is in **Normal Operation**. DiAsUI presents **Status Indicators** to inform the user about hypo- and hyperglycemia risk, CGM and insulin history. It also permits the user to change the system operating mode, trigger the MealActivity to manually request insulin delivery, display historical plots of CGM and insulin, calibrate the CGM (if supported by the CGM device) and activate the **Configuration Mode** user interface. Although DiAsUI is part of the DiAs system and is described in this document it may be replaced by an equivalent module as long as this module conforms to the DiAsUI API. The University of Virginia Center for Diabetes Technology will verify (IV.I.2) that the developer's DiAsUI Activity communicates properly with the rest of the system but will **not** verify that it operates correctly.

II.D.1.5. DiAs Service (system)

DiAsService performs the central coordinating functions of the DiAs system. It handles system operating mode transitions, monitors peripheral device status, makes periodic calls to the APCservice and BRMservice and receives insulin delivery requests from MCMservice. It sends insulin delivery requests to, and receives state estimate and bolus intercepts from, the SSMservice and updates the biometricsContentProvider with system state information. DiAsService coordinates the operation of the two different automated controller modules which may reside within the system: APCservice and BRMservice. DiAsService shall call APCservice and BRMservice every heartbeat interval using the interface (Table 53 - BRMservice Public API and Table 54 – APCservice Public API) to receive insulin dosing recommendations from each module. DiAsService shall disregard the recommendations of APCservice and BRMservice unless the system is operating in Closed Loop mode. If in Closed Loop mode it will send insulin requests to SSMservice based on the outputs from APCservice and BRMservice according to the rules describes in Table 4.

II.D.1.6. Network Service (system)

The Network Service is responsible for sending a stream of de-identified status information to a remote monitoring database using a secure link (SSL protocol). It reads subject biometrics, profiles, CGM history, insulin history, state estimate, log, Events and other data from the biometricsContentProvider and securely transmits it - via a cellular network or WiFi - to the Remote Monitoring Application which resides on a secure UVA server. Developers who wish to access this data for monitoring or other applications may do so through a secure RESTful API. This API is described in a separate document.

II.D.1.7. CGM Service (system)

The CGM Service module receives data from the CGM Device Driver and writes it to the biometricsContentProvider. If the particular CGM device connected to the system requires the DiAs to supervise calibration then this module receives Self Monitoring Blood Glucose (SMBG) data from the user interface and sends it to the CGM Device Driver.

II.D.1.8. Exercise Service (system – developer replaceable)

The Exercise Service module reads exercise sensor data from the EXERCISE_SENSOR_TABLE and possibly other database tables. It processes this data and stores its output in the EXERCISE_STATE_TABLE.

II.D.1.9. Pump Service (system)

The Pump Service module receives insulin delivery requests from the Safety Service and communicates them to the Pump Device Driver. It also receives insulin delivery confirmation from the Pump Device Driver, associates it with the corresponding insulin delivery request and stores the delivered insulin data in the database using the biometricsContentProvider.

II.D.1.10. CGM Device Drivers (system)

If the DiAs system needs to operate in a mode that requires CGM data (Closed Loop, Safety Only or Sensor Only modes) then a CGM Device Driver corresponding to the attached CGM device must be installed. Each CGM device connected to the DiAs must have its own specific device driver. It is the job of the device driver to take care of the details of communication with the CGM device and present a standard interface to the CGM Service. Depending upon the requirements of the particular CGM device the CGM Device Driver may consist of multiple applications working together.

II.D.1.11. Exercise Sensor Device Driver (system)

The Exercise Sensor Device Driver is a device driver which receives heart rate, accelerometer or other biometric information from the system and stores it in the EXERCISE_SENSOR_TABLE. This table permits storage of data in a flexible format in order to accommodate a variety of possible input signals which may be used to detect physical exertion, inform an algorithm or other system module about details of a subject's physical state or simply store data for later analysis.

II.D.1.12. Pump Device Drivers (system)

If the DiAs system requires an insulin pump for operation (Pump, Closed Loop or Safety Only modes) then a Pump Device Driver corresponding to the attached insulin pump must be installed. Each insulin pump connected to the DiAs must have its own specific device driver. It is the job of the device driver to take care of the details of communication with the insulin pump and present a standard interface to the Pump Service. Depending upon the requirements of the particular insulin pump the Pump Device Driver may consist of multiple applications working together.

II.D.1.13. BRMservice (Developer)

The BRMservice is a *developer-supplied* application which, along with the APCservice, determines the amount and timing of insulin delivery when the DiAs is operating in Closed Loop mode. The BRMservice is called by the DiAsService every Supervisor "heartbeat" (5 minutes) using a predefined software interface (see Table 53 - BRMservice Public API) The BRMservice uses data from the biometricsContentProvider to calculate how much insulin should be delivered to the user and returns this information to DiAsService. DiAsService checks the operating mode to verify that the insulin request should be sent. If valid, the insulin request is sent to the SSMservice which may intercept or modify the request if it determines that it is unsafe. The DiAs system is supplied to APSDK developers with a "template" version of the BRMservice which exchanges data with other system modules in accordance with the interface specification (including making insulin requests ***for testing only***) but **does not** contain a control algorithm. It is the responsibility of the APSDK developer to make certain that the BRMservice used in their clinical trials is functional and safe. The University of Virginia Center for Diabetes Technology will verify (IV.I.2) that the developer's BRMservice communicates properly with the rest of the system but will **not** verify that it operates correctly. In the situation in which both APCservice and BRMservice modules are installed and active within the DiAs system, the DiAsService module will determine how to interpret their output and provide insulin delivery commands to SSMservice using the rules listed in Table 4.

II.D.1.14. APCservice (Developer)

The APCservice is a *developer-supplied* application which determines the amount and timing of insulin delivery when the DiAs is operating in Closed Loop mode. The APCservice is called by the DiAsService every Supervisor "heartbeat" (5 minutes) using a predefined software interface (see Table 54 – APCservice Public API) regardless of operating mode. The APCservice uses data from the biometricsContentProvider to calculate how much insulin should be delivered to the user and returns this information to DiAsService. DiAsService checks the operating mode to verify that the insulin request should be sent. If valid, the insulin request is sent to the SSMservice which may intercept or modify the request if it determines that it is unsafe. The DiAs system is supplied to APSDK developers with a "template" version of the APCservice which exchanges data with other system modules in accordance with the interface specification (including making insulin requests ***for testing only***) but **does not** contain a control algorithm. It is the responsibility of the APSDK developer to make certain that the APCservice used in their clinical trials is functional and safe. The University of Virginia Center for

Diabetes Technology will verify (IV.I.2) that the developer's APCservice communicates properly with the rest of the system but will **not** verify that it operates correctly

II.D.1.15. MCMservice (system – developer replaceable)

MCMservice is a background service application which receives commands from MealActivity. The MealActivity can request that MCMservice calculate a bolus based upon meal size, SMBG and other factors. When the MealActivity wants to send a bolus to the subject it issues a bolus command to MCMservice which passes the bolus to DiAsService. Although MCMservice is part of the DiAs system and is described in this document it may be replaced by a different module as long as this module conforms to the MCMservice API. The University of Virginia Center for Diabetes Technology will verify (IV.I.2) that the developer's MCMservice communicates properly with the rest of the system but will **not** verify that it operates correctly.

II.D.1.16. SSMservice (Developer)

The SSMservice is a *developer-supplied* application which examines all insulin requests for safety when the DiAs is operating in Closed Loop or Safety Only modes. DiAsService makes insulin requests to the SSMservice using a predefined software interface (**Table 51 – SSMservice Public API**). The SSMservice then communicates insulin requests to the Pump Service using a second software interface (

1. SAFETY_SERVICE_CMD_REGISTER_CLIENT

   This command is used by the SSMservice to resolve the Messenger for status information replies to DiAsService:

```
Public Messenger mMessengerToClient = null;
mMessengerToClient = msg.replyTo;
```

2. SAFETY_SERVICE_CMD_START_SERVICE

   DiAsService sends this command to initialize the SSMservice and cause it to start and establish a connection with PumpService:

```
//  Command inputs
paramBundle = msg.getData();
boolean enableIOtest = (boolean)paramBundle.getBoolean(" enableIOtest");
int IOB_curve_duration_hours = paramBundle.getInt("IOB_curve_duration_hours");      // IOB curve duration (hours)
simulatedTime = paramBundle.getLong("simulatedTime", -1);                           // Contains the current Unix
                                                                                    // time stamp or -1 if clock

// Bind to the Insulin Pump Service
Intent intent = new Intent();
intent.setClassName("edu.virginia.dtc.PumpService", "edu.virginia.dtc.PumpService. PumpService");
bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
```

3. SAFETY_SERVICE_CMD_REQUEST_BOLUS

   DiAsService sends this command to direct the SSMservice to deliver insulin to the subject or to request or spend insulin "credit".

i. Requesting insulin

The DiAs classifies all insulin delivered to the subject into three categories: basal, correction and meal. The SSMservice can operate in an asynchronous manner and must include a basal bolus based upon the daily basal profile whenever DiAsService makes an insulin request with the **asynchronous** flag set to **false**. The **bolusRequested** argument contains the total amount of insulin requested in units and may be zero if only basal insulin delivery is desired (in this case asynchronous=false).

The **differential_basal_rate** is an argument that permits APCservice to modulate the basal delivery rate with respect to the daily basal profile. It can be positive, negative or zero and is limited to the range -6U/hour to +6U/hour. This is added to the basal rate as determined by the SSMservice (which may be attenuated below the profile rate if there is a hypoglycemia risk).

ii. credit_request and spend_request

The **credit_request** and **spend_request** arguments are used if the SSMservice is managing an insulin "credit pool". In this case the DiAsService can request "credit" insulin from the SSMservice which can require that the user immediately pre-approve the amount of credit requested. If approved by the user the credit can later be "spent" by the APCservice or BRMservice without any further confirmation required by the Safety System. This mechanism may be used by the DiAsUI to announce a meal (issue a "**credit_request**") and for the APCservice/BRMservice to deliver insulin to treat the meal at the appropriate time (issue a "**spend_request**").

iii. DIAS_STATE and status flags

- **DIAS_STATE** is the current state of the DiAs system and may be Stopped, Pump, Closed Loop, Safety Only or Sensor Only.
- **calFlagTime** is the Unix time of the last CGM calibration
- **hypoFlagTime** is the Unix time of the last hypoglycemic episode reported by the user by pressing the Hypoglycemia Button on the Home Screen of DiAsUI.
- **exercise** is a Boolean reporting whether the DiAsUI Exercise Button indicates that the subject is currently exercising.

```
//  Command inputs
paramBundle = msg.getData();                                                    // Fetch parameter bundle
enableIOtest=paramBundle.getBoolean("enableIOtest", false);
simulatedTime = paramBundle.getLong("simulatedTime", -1);
bolusRequested =  paramBundle.getDouble("bolusRequested", 0);
bolusMeal =  paramBundle.getDouble("bolusMeal", 0);
bolusCorrection =  paramBundle.getDouble("bolusCorrection", 0);
double differential_basal_rate = paramBundle.getDouble("differential_basal_rate", 0);      // [-6U/hour:6U/hour]
credit_request = paramBundle.getDouble("credit_request", 0);
spend_request = paramBundle.getDouble("spend_request", 0);
asynchronous = paramBundle.getBoolean("asynchronous", false);
long calFlagTime = (long)paramBundle.getLong("calFlagTime", 0);
long hypoFlagTime = (long)paramBundle.getLong("hypoFlagTime", 0);
boolean exercise = paramBundle.getBoolean("currentlyExercising", false);
DIAS_STATE = paramBundle.getInt("DIAS_STATE", DIAS_STATE_OPEN_LOOP);
```

When the SSMservice completes normal processing of the deliver bolus command it sends the status SAFETY_SERVICE_STATE_NORMAL to DiAsService along with additional data:

```
response = Message.obtain(null, SAFETY_SERVICE_STATE_NORMAL, 0, 0);
responseBundle = new Bundle();
responseBundle.putInt("stoplight", hypolight);
responseBundle.putInt("stoplight2", hyperlight);
responseBundle.putBoolean("isMealBolus", isMealBolus);     // true if this is a meal bolus
responseBundle.putDouble("brakes_coeff", brakes_coeff);    // Value between 0 and 1
                                                                    // 0: brakes on, no basal
                                                                    // 1: brakes off, full basal
responseBundle.putDouble("IOB", IOB);                      // Latest IOB estimate
response.setData(responseBundle);
mMessengerToClient.send(response);
```

4. SAFETY_SERVICE_CMD_CALCULATE_STATE

DiAsService sends this command to direct the SSMservice to calculate the state estimate based on CGM, insulin and possibly other data from the database. The state estimate is written into the 'stateestimate' database table.

```
// Command inputs
enableIOtest = (boolean)paramBundle.getBoolean("enableIOtest");
simulatedTime = paramBundle.getLong("simulatedTime", -1);
bolus_meal =  0.0;
bolus_correction =  0.0;
bolusRequested = 0.0;
differential_basal_rate = 0.0;
credit_request = 0.0;
spend_request = 0.0;
asynchronous = false;
calFlagTime = (long)paramBundle.getLong("calFlagTime", 0);
hypoFlagTime = (long)paramBundle.getLong("hypoFlagTime", 0);
exercise = paramBundle.getBoolean("currentlyExercising", false);
DIAS_STATE = paramBundle.getInt("DIAS_STATE", DIAS_STATE_OPEN_LOOP);
```

When the SSMservice completes normal processing of the calculate state estimate command it sends the status SAFETY_SERVICE_STATE_CALCULATE_RESPONSE to DiAsService:

```
response = Message.obtain(null, SAFETY_SERVICE_STATE_CALCULATE_RESPONSE, 0, 0);
mMessengerToClient.send(response);
```

Table 52 – Pump Service Public API). The SSMservice uses data from the biometricsContentProvider to add basal insulin to any request per the daily basal profile. It also may use insulin constraints written to the database by the developer supplied ConstraintService in order to inform its decisions. The SSMservice may take the following actions to ensure the safety of the subject:

- Intercept and pause insulin delivery pending user confirmation
- Reduce the size of an insulin request to avoid hypoglycemia
- Block insulin requests entirely

The SSMservice also calculates the current value of Insulin on Board (IOB) and the current hypo- and hyperglycemia risk levels which are used to control the traffic lights. It writes state estimate data to the

database and returns to DiAsService a data packet containing appropriate status values. The DiAs system is supplied to third party developers with a "template" version of the SSMservice which exchanges data with other system modules in accordance with the interface specification but **does not contain a safety algorithm**. This template SSMservice passes insulin requests to the Pump Service - and returns values for IOB, hypo- and hyperglycemia risk - **for testing purposes only**. It is the responsibility of the third part developer to make certain that the SSMservice used in their clinical trials is functional and safe. The University of Virginia Center for Diabetes Technology will verify (IV.I.2) that the developer's SSMservice communicates properly with the rest of the system but will **not** verify that it operates correctly.

II.D.1.17. ConstraintService (Developer)

The ConstraintService is a *developer-supplied* application used to calculate an upper constraint on the amount of insulin which may be delivered to the user. The ConstraintService is activated by the SSMservice, reads necessary data from the biometricsContentProvider and writes time stamped insulin constraints into a special table within the database reserved for this purpose (Table 28 – CONSTRAINTS_TABLE ("constraints")). This table may be used by the SSMservice as an additional input when determining how much insulin should be delivered to the user. The ConstraintService operates in an advisory capacity only. It is up to the SSMservice to determine how to make use of constraints calculated by this module. The SSMservice has the final say concerning how much insulin will be delivered to the user. The details of the interface between the ConstraintService and SSMservice is to be determined by the developers of these modules.

Table 4 - DiAsService Closed Loop Insulin Delivery Rules

Controllers setting	***APC Disabled***	***APC Enabled***	***APC Enabled within Night Profile***	***APC Disabled within Night Profile***
BRM Disabled	Closed Loop unavailable	APC does Corr and Diff Basal	APC does Corr and Diff Basal if within profile, nothing otherwise	APC does Corr and Diff Basal if outside profile, nothing otherwise
BRM Enabled	BRM does Corr and Diff Basal	APC does Corr and BRM does Diff Basal	APC does Corr if within profile, nothing otherwise. BRM does Diff Basal at all time	APC does Corr if outside profile, nothing otherwise. BRM does Diff Basal at all time
BRM Enabled within Night Profile	BRM does Corr and Diff Basal if within profile, nothing otherwise	BRM does Diff Basal if within profile, nothing otherwise. APC does Corr at all time	BRM does Diff Basal and APC does Corr if within profile, nothing otherwise.	BRM does Diff Basal if within profile, nothing otherwise APC does Corr if outside profile, nothing otherwise
BRM Disabled within Night Profile	BRM does Corr and Diff Basal if outside profile, nothing otherwise	BRM does Diff Basal if outside profile, nothing otherwise. APC does Corr at all	APC does Corr if within profile, nothing otherwise BRM does Diff Basal	BRM does Diff Basal and APC does Corr if outside profile, nothing otherwise.

		time	if outside profile, nothing otherwise	

II.D.1.18. Remote Monitoring Application (remote, system)

The Remote Monitoring Application is a system application external to the APMMP which permits engineers and clinicians to remotely monitor the state of the subject and the performance of the system. It receives data from the biometricsContentProvider via a secure link, stores it in a database on the server and a provides password-protected web interface for real-time remote viewing and monitoring of APMMP activity. Developers who wish to access their data from the secure database for monitoring or other applications may do so through a RESTful API. This API is described in a separate document

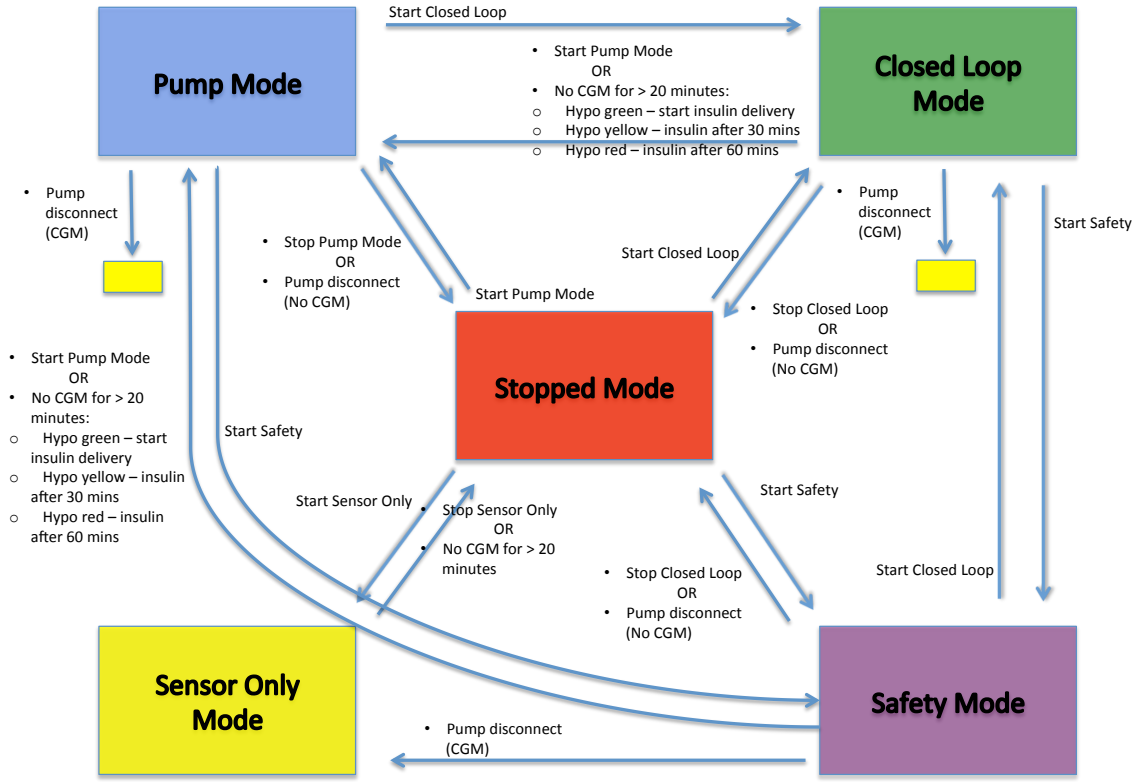


Figure 5 - DiAs Operating Modes

II.D.2. Modes of Operation

The DiAs system supports five operating modes. Which modes are available depends upon what peripheral devices are attached to the system, their current status and how the system has been configured. The current operating mode determines what actions the DiAs can perform and what

options are presented to the user. **Figure 5** graphically describes state transitions between operating modes.

II.D.2.1. Stopped

In Stopped mode the system is not delivering insulin to the user. If the CGM device driver is configured then CGM data may be displayed on the user interface and stored in the database. If the DiAs has not yet been configured with Subject Information (see II.D.3.2.1) it will be in Stopped mode. When the APMMP smartphone is powered on the DiAs will always be in Stopped mode.

II.D.2.2. Pump

In Pump mode the DiAs must be configured with Subject Information as well as CF, CR and Basal profiles. The system delivers insulin to the user based upon the Basal profile with no deviation regardless of blood glucose level or IOB. The user is able to request insulin delivery manually using the MealActivity and requested insulin is delivered without any additional safety checks. The insulin pump device driver must be configured and connectivity must be confirmed by the DiAs software. Traffic lights are operational if CGM data is available.

II.D.2.3. Closed Loop

In Closed Loop mode the DiAs must be configured with Subject Information as well as CF, CR and Basal profiles. The system delivers insulin to the user according to the Basal profile. The APCservice and/or BRMservice is active and may modulate insulin delivery to be greater or less than the pre-programmed Basal profile depending upon the predicted blood glucose level, IOB and other factors. The APCservice and/or BRMservice may also choose to deliver correction boluses to mitigate hyperglycemic risk. The user can announce meals to the system which causes an immediate insulin request to the SSMservice. Closed Loop mode requires that the insulin pump device driver be configured and connectivity be confirmed by the DiAs software. The system must also confirm that CGM data has been received from the sensor within the past 20 minutes. Traffic lights are operational and indicate the estimated risk of hypo- and hyperglycemia as determined by the developer supplied SSMservice.

II.D.2.4. Safety

In Safety mode the DiAs must be configured with Subject Information as well as CF, CR, Basal and Safety Only profiles. The system delivers insulin to the user based upon the Basal profile. The Closed Loop control algorithm is ***not*** used but the Safety System continues to operate and insulin delivery can be less than the pre-programmed Basal profile based upon the predicted blood glucose level and IOB. The user can announce meals to the system which causes an immediate insulin request to the SSMservice. The insulin pump device driver must be configured and connectivity must be confirmed by the DiAs software. The system must have confirmed that CGM data has been received from the sensor within the past 20 minutes. Traffic lights are operational and indicate the estimated risk of hypo- and hyperglycemia as determined by the developer supplied SSMservice.

II.D.2.5. Sensor Only

In Sensor Only mode the DiAs must be configured with Subject Information (see II.D.3.2.1). The system is not delivering insulin to the user. The CGM device driver must be configured and a valid CGM data point must have been received by the system within the past 20 minutes. Traffic lights are operational and indicate risk for hypo- and hyperglycemia based upon CGM data.

II.D.3. Graphical User Interface

II.D.3.1. Overview

The DiAs system provides a touch-based graphical interface that enables the user to configure and operate the system. Configuration involves setting up subject specific quantities such as height and weight and entering or modifying standard pump therapy daily profiles such as correction factor, carbohydrate to insulin ratio and basal rate. Operation involves changing the system mode (Stopped, Pump, Closed Loop, Safety Only, Sensor Only), announcing meals, delivering boluses and displaying plots. The DiAs stores user profile, CGM and delivered insulin information in an internal database which is initialized when a new subject begins their clinical trial. Information from this database is periodically sent to an external remote monitoring system

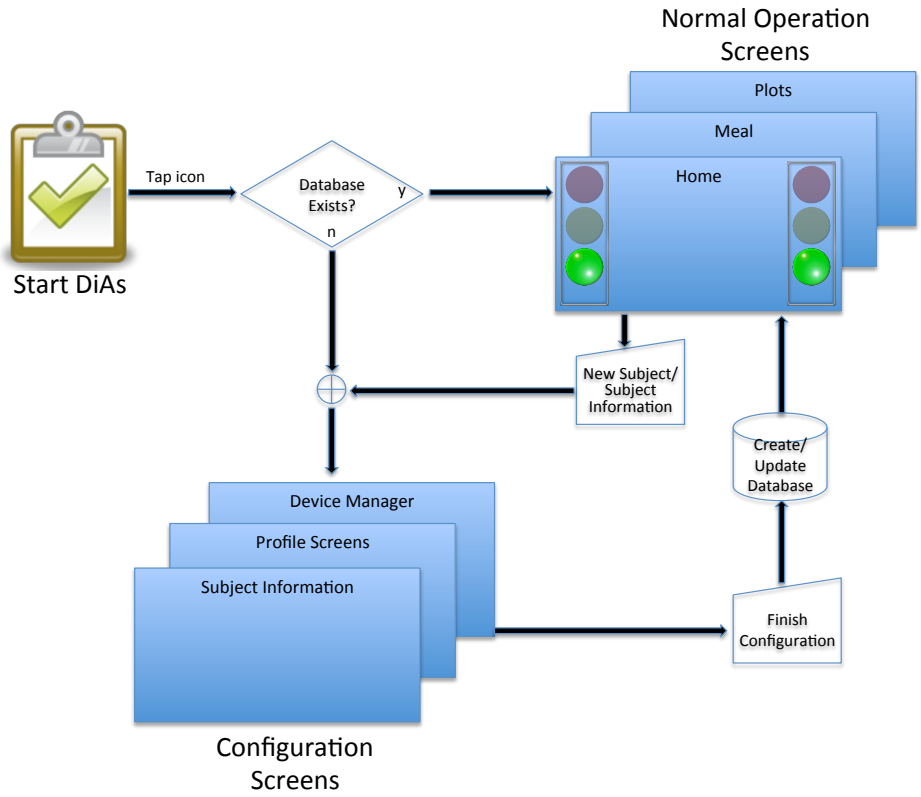


Figure 6

The user interface consists of two groups of screens (Figure 6): Configuration Screens and Normal Operation Screens. Configuration Screens are accessed when a subject is initialized or when the user wishes to view or modify settings and are managed by the DiAsSetup system application as well as by device driver Activities. The rest of the time the Normal Operation Screens are displayed. Normal Operation Screens enable the user to transition between different modes (Stopped, Pump, Closed Loop, Safety Only, Sensor Only), announce meal boluses and display plots and are presented and managed by the DiAsUI and MealActivity developer-replaceable applications.

II.D.3.2. Configuration

The DiAs Setup application manages the user interface during Configuration. There are six main configuration screens as well as one or more screens which may be presented by each device driver. The user moves between the Subject Information screen, profile screens and Device Manager screen by swiping or by tapping the **Subject Info**, **Insulin Profiles** or **Device Manager** tabs at the top of the screens. Activating the CGM sensor or insulin pump device driver may cause additional screens to appear to enable the user to enter communication parameters appropriate to the device. Figure 7 shows a schematic of the configuration work flow.

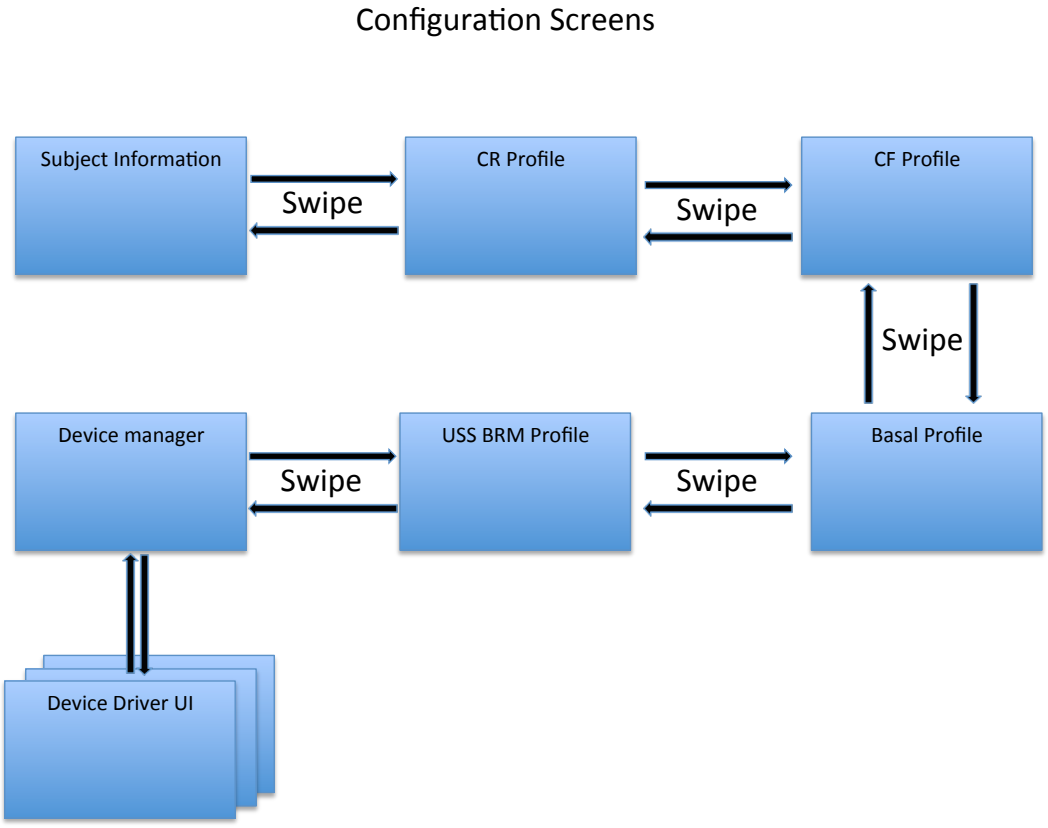


Figure 7

Configuration screens are required for system initialization and may also be accessed during normal operation. The user must enter Subject Information and profile data before normal operation can commence. Once the system has been initialized and normal operation begins the user can access the Configuration Screens and make changes to items such as daily profiles, system password and the device operation. Subject biometric information, subject number and ID cannot be changed

without reinitializing the system. Once started, the DiAs will continue to operate, regardless of whether the user interface is displaying Configuration Screens or Normal Operating Screens.

II.D.3.2.1. Subject Information Screen

The Subject Informtion screen is used to enter and display basic information about the user and the clinical trial. It contains the following fields, all of which must be initialized:

Name	Units, range	Comments
Subject ID	Alphanumeric string	Used to identify the trial subject, becomes part of the database file name.
Subject Number	Numeric, 1-999999999	Numeric identifier, used by remote monitoring systems.
Weight	kg, 27-136	
Height	cm, 127-221	
Age	years, 1-100	
Total Daily Insulin	Units, 10-100	
Gender	Male/Female	
System password	Alphanumeric, 4-16 characters	Password required to start Pump and Closed Loop modes. Tap the key icon at the top of the screen to configure.

Table 5 – Subject Information Screen

To configure the system password tap the key-shaped icon at the top of the screen. The subject gender is indicated by the state of a radio button. All other fields contain text or numeric data and are entered by tapping the associated box. An example Subject Information screen is shown in Figure 8. At the bottom of the screen is a progress bar labeled ***Available Modes*** indicating when configuration has

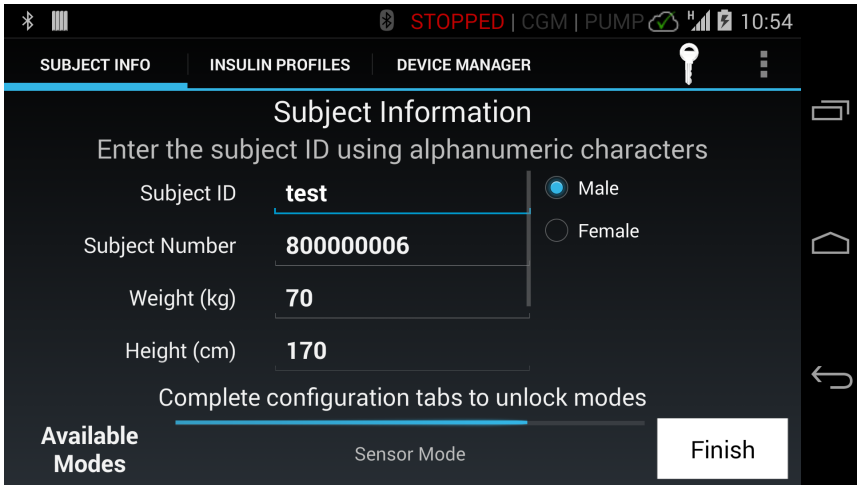


Figure 8

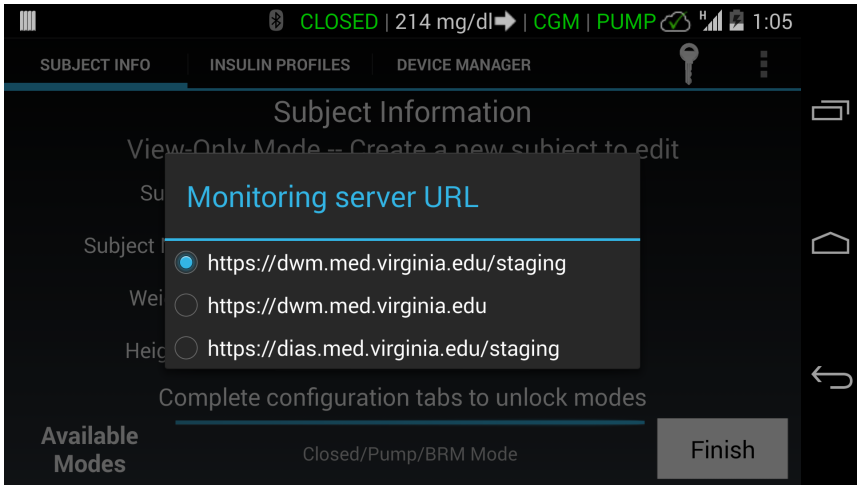


Figure 9

progressed sufficiently to support operation of a particular mode. For example ***Closed Loop*** and ***Safety*** modes require that Subject Information, CR, CF and basal profiles, insulin pump and CGM devices all be initialized before they become available while ***Pump*** mode does not require a CGM device. Table 6 describes the subject profile information and device connectivity requirements for each of the five operating modes.

Table 6 – Configuration and Available Operating Modes

Operating Mode	Subject Information	CF Profile	CR Profile	Basal Profile	Controller Profile	CGM	Pump
Stopped							
Sensor Only	X					X	
Pump	X	X	X	X			X
Closed Loop	X	X	X	X	Must be active if in Mode 4	X	X
Safety	X	X	X	X	Must be active if in Mode 4	X	X

II.D.3.2.2. Correction Factor Profile Screen

This screen is used to enter, view or edit the Correction Factor daily profile. The Correction Factor (CF) is a patient-specific parameter containing an estimate of the lowering of blood glucose in mg/dl that is expected when one unit of insulin is injected. The profile consists of a series of segments made up of a starting time in HH:MM (24 hour format) along with a corresponding CF value (mg/dl/U). The CF profile must contain at least one active segment corresponding to the time within the day at which the corresponding CF value becomes active. If CF is fixed for a subject then the profile time is unimportant as the associated CF value will always be active.

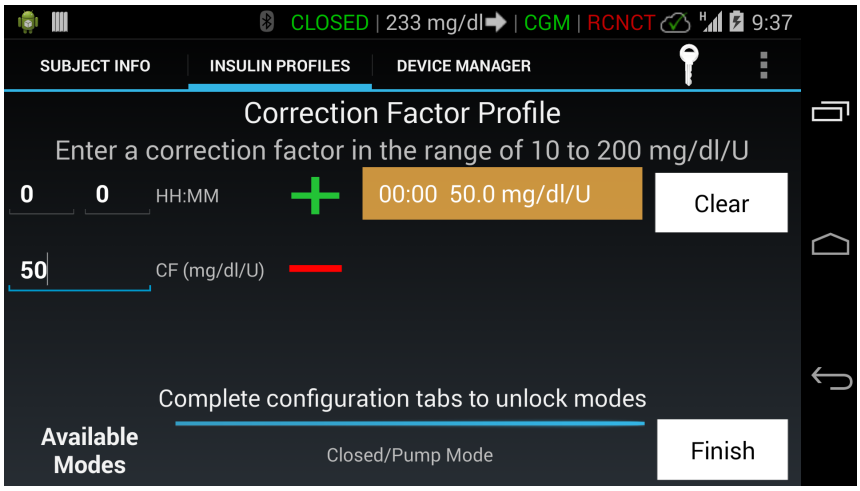


Figure 10

Figure 10 shows an example of a Correction Factor Profile containing a single segment with a value of 50 mg/dl/U valid for the entire day. The HH:MM and CF elements of each segment are entered using the fields at the left hand side of the screen and the segment is confirmed and entered into the profile by tapping the "+" button. Individual segments may be deleted by highlighting them and pressing the "-" button while all segments may be deleted by tapping "Clear". The profile may contain a maximum of 12 segments.

II.D.3.2.3. Carbohydrate to Insulin Ratio Profile Screen

This screen is used to enter, view or edit the Carbohydrate to Insulin Ratio daily profile. The Carbohydrate Ratio (CR) is a patient-specific parameter containing an estimate of the amount of insulin in U that is required to metabolize one gram of carbohydrate. The profile consists of a series of segments made up of a starting time in HH:MM (24 hour format) along with a corresponding CR value (g/U). The CR profile must contain at least one active segment corresponding to the time within the day at which the corresponding CR value becomes active. If CR is fixed for a subject then the profile time is unimportant as the associated CR value will always be active.

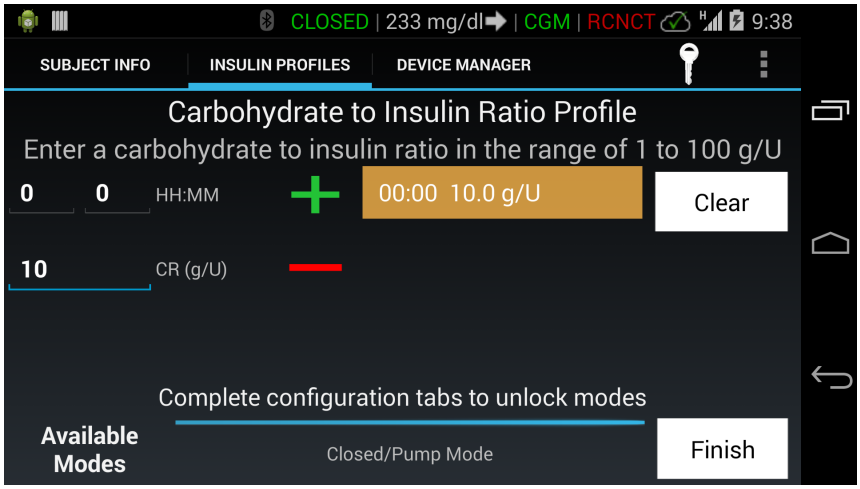


Figure 11

Figure 11 shows an example of a Carbohydrate to Insulin Ratio Profile containing a single segment with a value of 10 g/U valid for the entire day. The HH:MM and CR elements of each segment are entered using the fields at the left hand side of the screen and the segment is confirmed and entered into the profile by tapping the "+" button. Individual segments may be deleted by highlighting them and pressing the "-" button while all segments may be deleted by tapping "Clear". The profile may contain a maximum of 12 segments.

II.D.3.2.4. Basal Rate Profile Screen

This screen is used to enter, view or edit the Insulin Basal Rate daily profile. The Basal Rate is a patient-specific parameter containing the rate in U/hour at which insulin should be delivered to the patient. The profile consists of a series of segments made up of a starting time in HH:MM (24 hour format) along with a corresponding basal rate (U/hour). The Basal Rate profile must contain at least one active segment corresponding to the time within the day at which the corresponding Basal Rate value becomes active. If Basal Rate is fixed for a subject then the profile time is unimportant as the associated Basal Rate value will always be active.

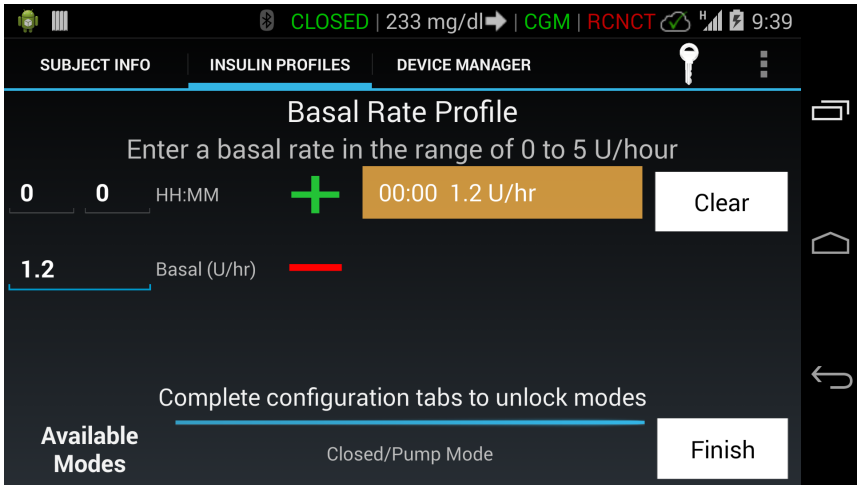


Figure 12

Figure 12 shows an example of a Basal Rate Profile containing a single segment with a value of 1.0 U/hour valid for the entire day. The HH:MM and Basal Rate elements of each segment are entered using the fields at the left hand side of the screen and the segment is confirmed and entered into the profile by tapping the "+" button. Individual segments may be deleted by highlighting them and pressing the "-" button while all segments may be deleted by tapping "Clear". The profile may contain a maximum of 12 segments.

II.D.3.2.5. Night Profile Screen

This screen (Figure 13) is used to determine the "Night" time interval during which particular control rules are in effect. The control rules depend upon the state of the "mode" parameter in *parameters.xml*, which may have the following values:

- 0 = No Pump, Safety or Closed Loop modes (Sensor and Stopped modes available)
- 1 = Pump and Stopped modes available
- 2 = Pump mode not available. Closed Loop and Safety modes available. If Closed Loop selected and within Controller schedule use BRMservice only. If Closed Loop selected and outside Controller schedule then BRMservice and APCservice used.

- 3 = Pump mode available. Closed Loop and Safety modes available. If Closed Loop selected and within Controller schedule use BRMservice only. If Closed Loop selected and outside Controller schedule then BRMservice and APCservice used.
- 4 = Pump mode available. If within Controller schedule then Closed Loop available (with BRMservice only) otherwise Closed Loop not available.

Table 4 - DiAsService Closed Loop Insulin Delivery Rules contains a summary of the rules for insulin dosing algorithm use in Closed Loop mode.

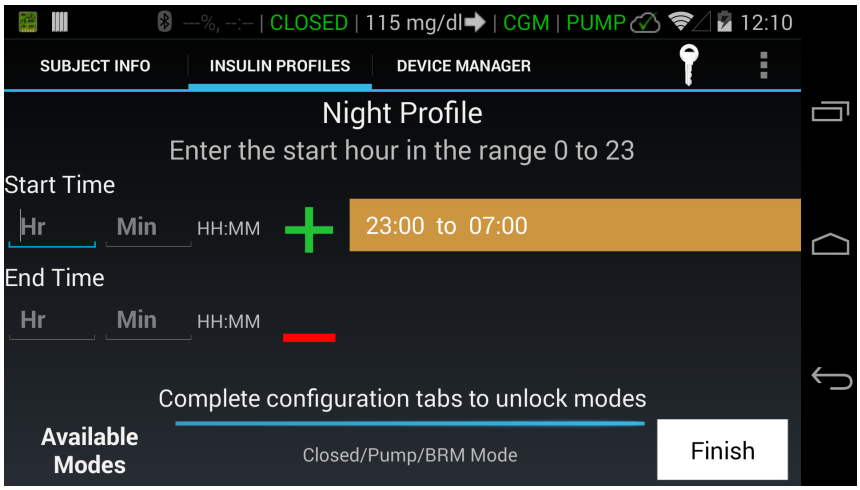


Figure 13

II.D.3.2.6. Device Manager Screen

The Device Manager Screen (**Figure 14**) shall enable the user to activate CGM sensor, insulin pump, heart rate monitor and other device drivers and set up their configuration parameters. Using the "Settings" button will cause the application information screen to appear for the device. This screen will allow users to "Force Close" or "Clear Data" for devices, as this is a common troubleshooting method. Once the drivers have been started and connection established with devices the user can tap the "Finish" button to conclude initialization.

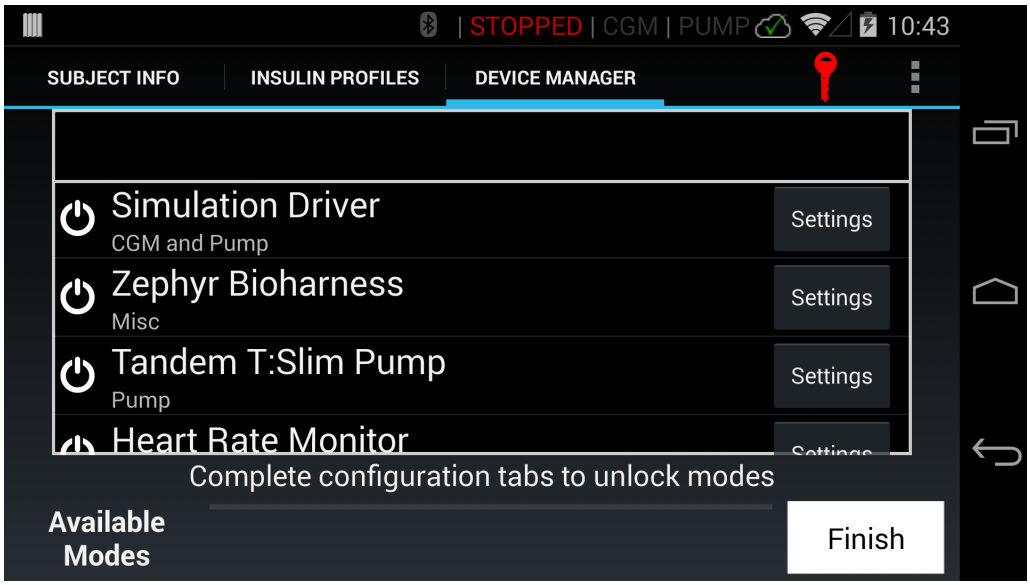


Figure 14

II.D.3.3. Device Drivers

The DiAs APMMP includes the following device drivers:

- Standalone driver (software defined CGM and insulin pump with up to 10X speedup)
- Roche Accu-Chek Combo insulin pump (via Bluetooth)
- Tandem T:Slim Pump (via BTLE)
- Dexcom G4 Continuous Glucose Monitor (via BTLE relay device)
- Dexcom G4 Share AP
- Zephyr Bioharness

II.D.3.3.1. Standalone Driver

It is implemented in the **standaloneDriver** package. It provides a generic CGM Device Driver and Pump Device Driver in one package. It is used for testing other components of the DiAs software without the need for a physical pump or CGM device. The CGM portion of the StandaloneDriver provides a series of time stamped CGM values at approximately 5 minute intervals. The Pump portion of the StandaloneDriver accepts bolus commands and reports each bolus as delivered with the

appropriate delivery time stamp. The standaloneDriver should always be removed from the system during trials involving human subjects.

To activate the standaloneDriver CGM:

i. Tap the "Add CGM" button. The text "CGM Registered" will appear.
ii. Press and hold the "CGM Registered" text to reveal a menu.
iii. "Start" will initiate a canned CGM trace.
iv. "Input Value"+"Start" permits entry of a BG value in mg/dl which may be updated while the system is operating.
v. "DB read" "+"Start" reads CGM values from an existing DiAs CPMP database.
vi. "CSV Read" "+"Start" reads CGM values from a CSV file.
vii. Time stamps are always set by the DiAs APMMP.

To activate the standaloneDriver Pump:

i. Tap the "Add Pump" button. The text "Started" will appear.
ii. Press and hold "Pump Registered" to reveal a menu.
iii. "Start" will initiate pump operation.

II.D.3.3.2. Roche Accu-Chek Combo Driver

Fill the Roche pump with insulin and place on the subject according to directions from the manufacturer. When the pump is ready for use navigate to the pump **Bluetooth Settings** submenu and delete any paired devices. On the CPMP select **Roche** under **Available Drivers**, tap the button marked "Discoverable", press "Yes" to confirm and initiate pairing on the pump. When the CPMP chimes and the name of the CGMP device appears on the pump select "Add Device" on the pump. If you see a pairing request at the top of the screen select and accept the request. A popup will appear on the CPMP screen and the pump will display a 10 digit numeric code. Enter this code on the CPMP popup and confirm. The pump will display a message indicating that pairing is complete. Keep the Roche device driver screen open on the CPMP and wait for the pump display to time out. A Bluetooth symbol should appear on the pump display and the checkboxes next to "Start", "Time" and "History" should be marked on the CPMP. The Roche driver is now running and the pump is ready to deliver insulin. See Figure 16 to Figure 20 for examples of the RocheDriver User Interface screens that you should see during the pairing procedure.

Figure 15 - RocheDriver User Interface

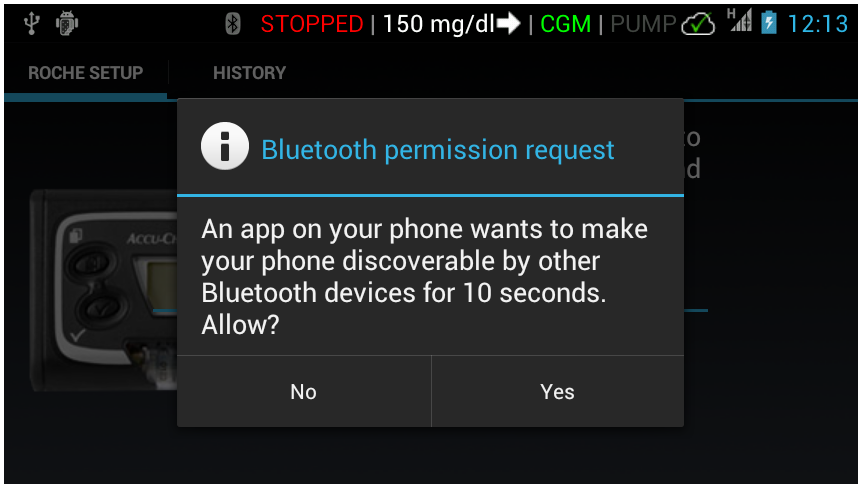


Figure 16 – Enable Bluetooth pairing on CPMP

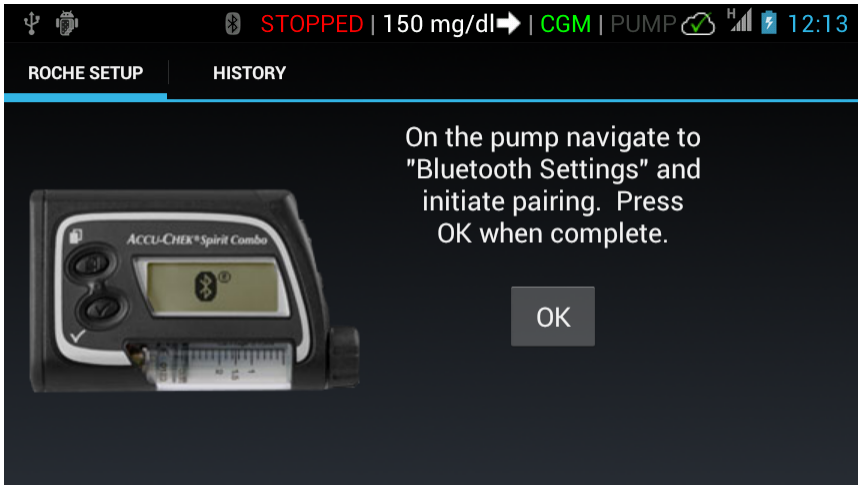


Figure 17 – Initiate pairing on Pump

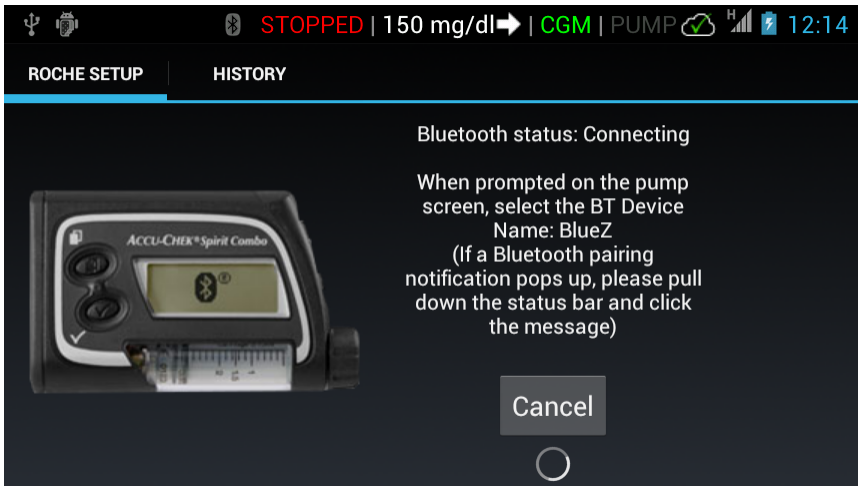


Figure 18 – Pairing in process

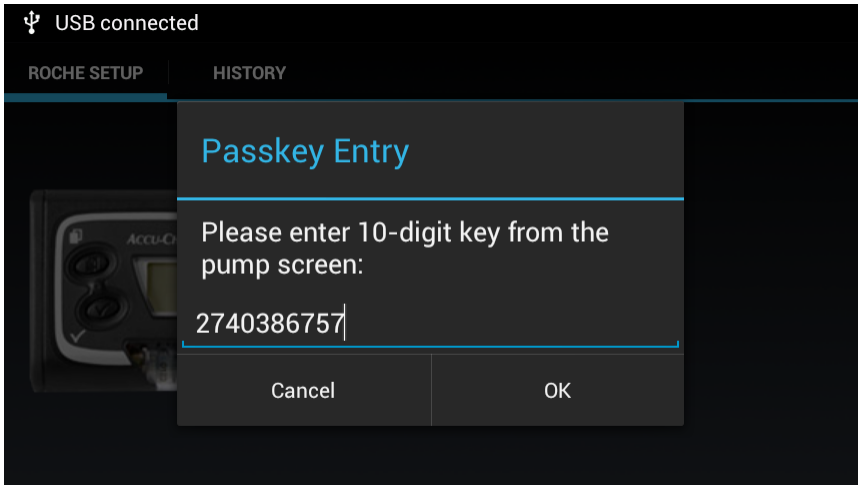


Figure 19 – Enter 10-digit key from pump

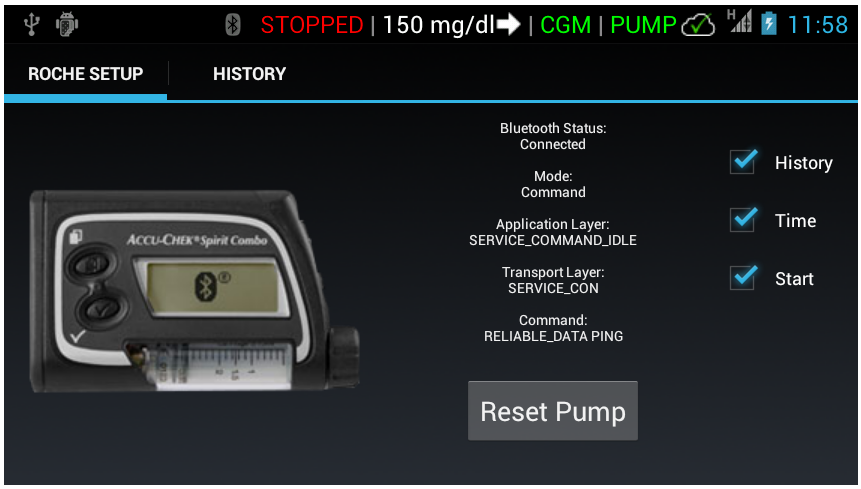


Figure 20 – Pump paired with CPMP

II.D.3.3.3. Tandem T:Slim AP Pump

The Tandem Artificial Pancreas Insulin Pump device driver is implemented in the application BTLE_Tandem. Before starting the device driver fill the Tandem pump with insulin and place on the subject according to directions from the manufacturer. When the pump is ready for use navigate to the pump **Bluetooth** submenu. by selecting **Options > Bluetooth** from the pump main menu. If the "Unpair Wireless Device" menu item is illuminated then the pump is already paired with another device. Tap "Unpair Wireless Device" and turn "Bluetooth" OFF and then ON once again. On the CPMP tap the "Tandem T:Slim Pump" label in the device manager. The Tandem driver UI screen will appear. Tap "Scan" and wait for a MAC address to appear in the upper left corner. Tap the MAC address of the desired pump and select "Connect" from the menu which will appear at the center of the screen. The pump will beep and display a pairing request. Accept the pairing request and the pump will beep again and display a pairing code. Enter the pairing code into the pairing field that appears on the CPMP. Tap "Get Status" and verify that the pump status appears at top center of the screen. Press the "Back" button to return to the "Device manager" screen.

II.D.3.3.4. BTLE_G4 Device Driver

The BTLE_G4 Bluetooth Low Energy relay device connects to the Dexcom G4 receiver and provides CGM values from the receiver to the Nexus 5 CPMP via a BLE wireless connection. It may be used with the Nexus 5 only. The BTLE_G4 Device Driver makes use of the Android 4.4.2 Bluetooth software stack to establish a BLE connection with the relay device and to read CGM values and meter records from the Dexcom G4 receiver.

Make sure that the Dexcom G4 sensor is inserted and warmed up and that valid CGM data is reading on the receiver display and make sure that Bluetooth is enabled on the Nexus 5 CPMP. Go to the DiAs **Device Manager** and select "Dexcom G4 (BTLE)" to display the device driver user interface. Tap "Scan for Devices" on the lower left corner and when the MAC address of the BTLE_G4 relay device appears tap it, and then select "Connect" on the Pairing dialog box that appears. Within 60-90 seconds the "Registered", "Receiver Connected" and "Setup Complete" boxes should have blue check marks and some time later new CGM readings will appear under the "CGM Values" column. When the Dexcom G4 receiver is calibrated a reading corresponding to the SMBG value will appear under the "Meter Values" column. Any time after pressing "Connect" on the Pairing dialog box you may exit the device driver UI by pressing the back key. When the device driver has established a connection with the Dexcom G4 the "CGM" status indicator at the top of the screen will light up green.

II.D.3.3.5. Dexcom Share AP

Dexcom Share™ AP is a CGM sensor device manufactured by Dexcom Inc., San Diego, CA. This version of the Dexcom system has a built-in BTLE radio to allow direct communication with the CPMP. Values are read and handled using an Android service that then stores the data on the CPMP in a database.

Setup the CPMP and navigate to the Device Manager. Now swipe the phone over the NFC tag on the receiver and the phone should make a positive chime. Within the next five minutes the CPMP should populate the most recent CGM value in the status bar. The UI for the Dexcom Share is only used for debugging and does not have any function for setting up the connection.

II.D.3.3.6. Zephyr HxM Driver

The Zephyr HxM Heart Rate Monitor driver is implemented by the application HR_Driver. This must be used to establish a Bluetooth connection between the BT-enabled Heart Rate Monitor device and the CPMP. Before starting the device driver place the Heart rate device's strap properly on the subject's body with the HxM attached to it. The Bluetooth radio will start automatically and it will be ready for pairing. Navigate to the device manager and press the Heart Rate Monitor driver. In the UI, press Scan and the heart rate device will appear named "HXM<6-digit-number>", hit it once to start the pairing process. Enter the pairing code 1234 when prompted. Once the pairing process is successful, a list of heart rate values will be displayed in real time. The Heart Rate Monitor device is now running.

II.D.3.3.7. Zephyr BioHarness 3 Device Driver

The Zephyr BioHarness device driver is implemented by the application Bioharness_Driver. Before starting the device driver establish Bluetooth pairing between the Bioharness and the CPMP. Navigate to the device manager and select the Zephyr Bioharness driver. Press Scan and select the correct Bioharness. Enter the pairing code 1234 when prompted. Once the pairing process is successful the UI should start displaying data. A blue LED will illuminate on the Bioharness device indicating BT connectivity.

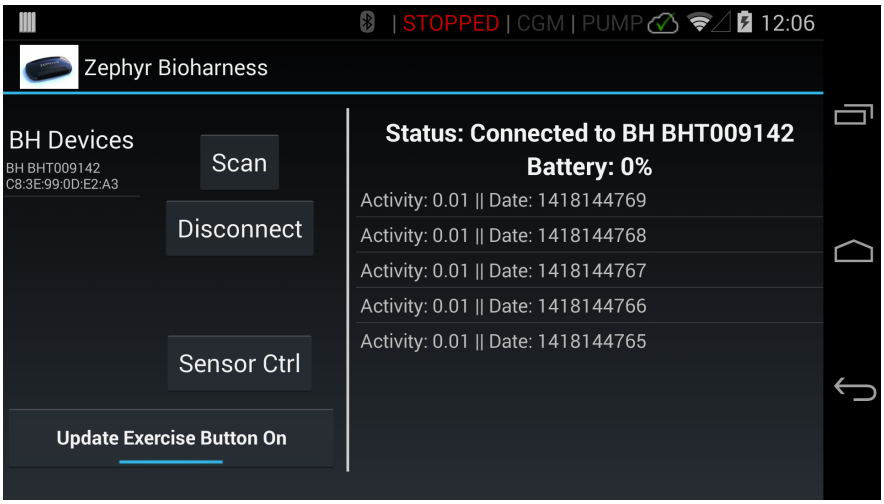


Figure 21 – Bioharness_Driver User Interface

II.D.3.4. Normal Operation

There are three Normal Operating Screens: the Home Screen, Plots, Calibration and the MealActivity Screen. Through these screens the user can display the current CGM, risk status and current and historical information. They also provide a means to control system operation (see Figure 22). The following sections describe the Normal Operating Screens provided by the DiAsUI and MealActivity developer-replaceable modules provided with the DiAs APMMP. The appearance of these screens will change if they are replaced by the developer.

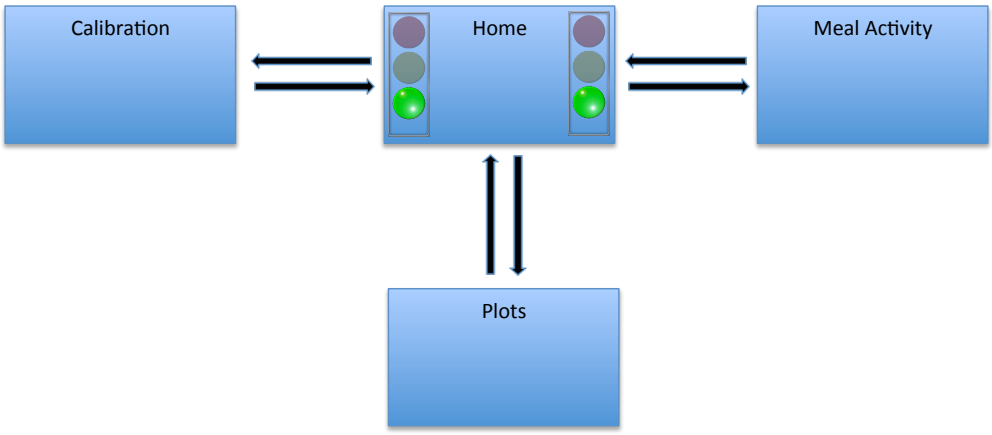


Figure 22 – Normal Operation Control Flow

II.D.3.4.1. System Status Line

At the top of all APMMP screens is a status line which is maintained by the Android Operating System which is visible regardless of which Activity is currently displayed (see the top line in Figure 8 through **Figure 28**). This status line is present in all Normal and Configuration screens and even persists if the user closes all DiAs APMMP Activities and accesses the Android launch screen. The status line displays the following information (see Figure 23 from right to left):

- Current CPMP time
- CPMP remaining battery charge and charging status
- Cellular/WiFi data connectivity
- Remote Monitoring Server connectivity
- Insulin Pump status
- CGM status
- Current CGM value and trend

- DiAs operating mode
- Bluetooth status
- CPMP "speedup" mode (standaloneDriver only – disabled if any other device drivers installed)
- USB connectivity

II.D.3.4.2. Home Screen

The Home Screen displays the current CGM value with a trend arrow, the hypo- and hyperglycemia traffic lights, a row of status indicators, the current time and buttons for system control.

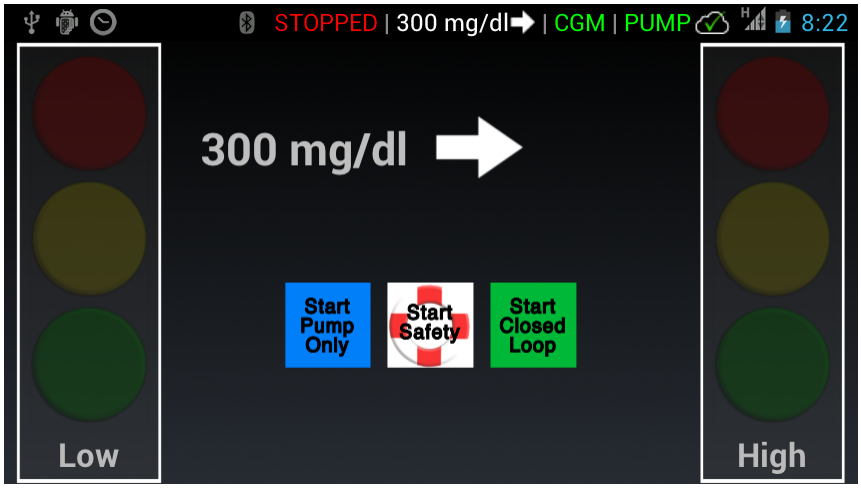


Figure 23

Figure 23 shows the Home Screen when the system is in **Stopped** mode such as immediately after system initialization. At the center of the display (if CGM is connected) is the most recent CGM value in mg/dl (or mmol/L if selected in *parameters.xml*) with the trend arrow and the number of minutes since the reading was generated. Below this is the size in Units and delivery time of the Last Bolus delivered by the system.

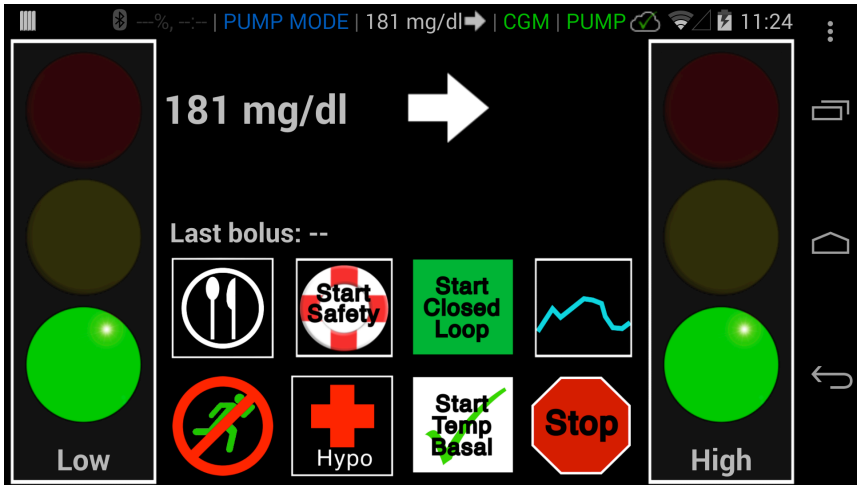


Figure 24

In **Pump** mode the Home Screen appears as in Figure 24. The status line now indicates **Pump Mode** and several buttons have appeared in the central portion of the screen. From left to right and top to bottom these are:

- MealActivity – tapping this button launches the MealActivity. The operation of the standard version of this application is discussed in a section that follows.
- Start Safety - tapping this button triggers a confirmation dialog. Confirming will cause the system to switch to **Safety** mode. This button will only appear if the pump is connected and at least one valid CGM value has been received by the DiAs within the past 20 minutes.
- Start Closed Loop – tapping this button triggers a confirmation dialog. Confirming will cause the system to switch to **Closed Loop** mode. This button will only appear if the pump is connected and at least one valid CGM value has been received by the DiAs within the past 20 minutes.
- Plots – pressing this button causes the Plots Screen to appear. The operation of this screen is discussed in a section that follows.
- Not Exercising/Exercising – This is a toggle button which changes appearance to indicate when the user indicates that he is exercising. Pressing the button and confirming causes a time stamped event to be stored in the DiAs database and updates the 'exercising' value in the 'system' table.
- Hypo Treatment – This button should be pressed when the user treats herself for hypoglycemia. Confirming hypoglycemia treatment stops the hypoglycemia audible alarm (if active) for 15 minutes and cause a time stamped Event to be stored in the DiAs database.
- Start/Stop Temp Basal – Pressing Start Temp Basal has different effect depending upon the system operating mode:
    - **Pump mode:** triggers a window enabling the user to set the basal rate to from 0% to 200% of the current profile value in 30 minute increments from 30 minutes to 24 hours. The % of current basal rate and the time remaining is displayed on the status line.

- **Closed Loop or Safety Mode**: causes the broadcast Intent edu.virginia.dtc.intent.action.TEMP_BASAL with the Extra ("command", TempBasal.TEMP_BASAL_START). This Intent may be used to trigger an Activity or Service within SSMservice, BRM_service or APC_service in order to trigger temporary basal rate during Safety and Closed Loop modes.
- **Stop Temp Basal**: triggers a confirmation menu allowing the user to cancel the temporary basal rate. Pump mode temporary basal rate is also canceled on any operating mode switch such as

- Stop – pressing this button will cause a confirmation dialog to appear. If confirmed the system will immediately enter **Stopped** mode.

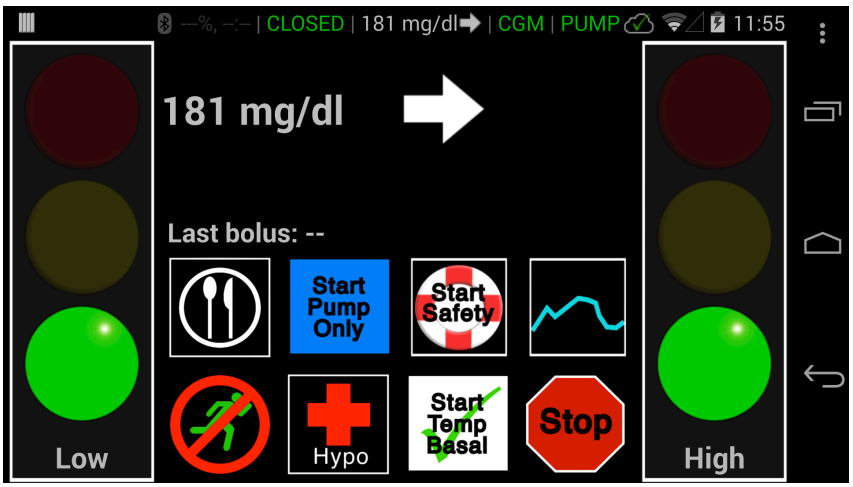


Figure 25

Figure 25 shows the Home Screen in **Closed Loop** mode. The new mode is shown in the upper left corner. Other than this all buttons are the same except that there is a direct path from **Closed Loop** to **Pump** mode.

If the user wishes to reach the Configuration Screens in order to view the current values or edit a profile she can press the **Options** button which appears as 3 vertical dots at the upper right corner of the screen.

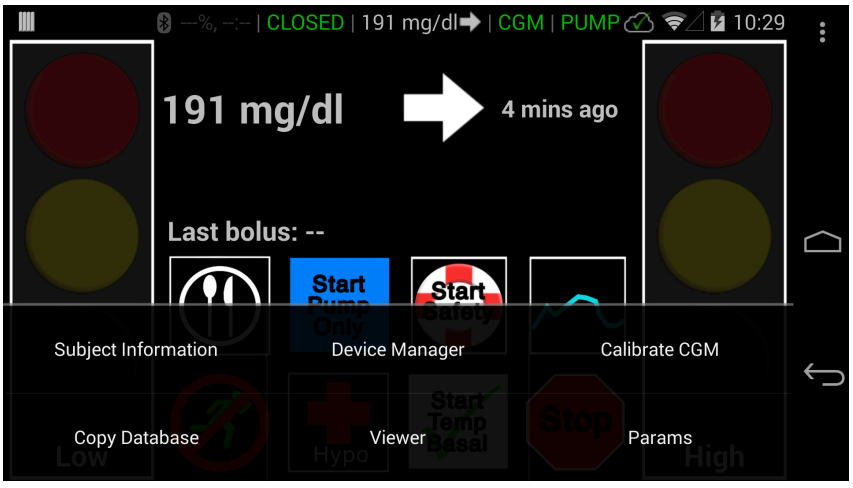


Figure 26

This causes the Options menu to appear at the bottom of the screen with elements that link to the Subject Information, Device Manager and Viewer screens. The Viewer permits quick access to the contents of the 'events' and 'insulin' tables from the database.

II.D.3.4.3. Plots Screen

The Plots Screen displays historical CGM and delivered insulin data from the DiAs database. The center panel of the Home Screen is replaced with a CGM display on its upper part and below that a

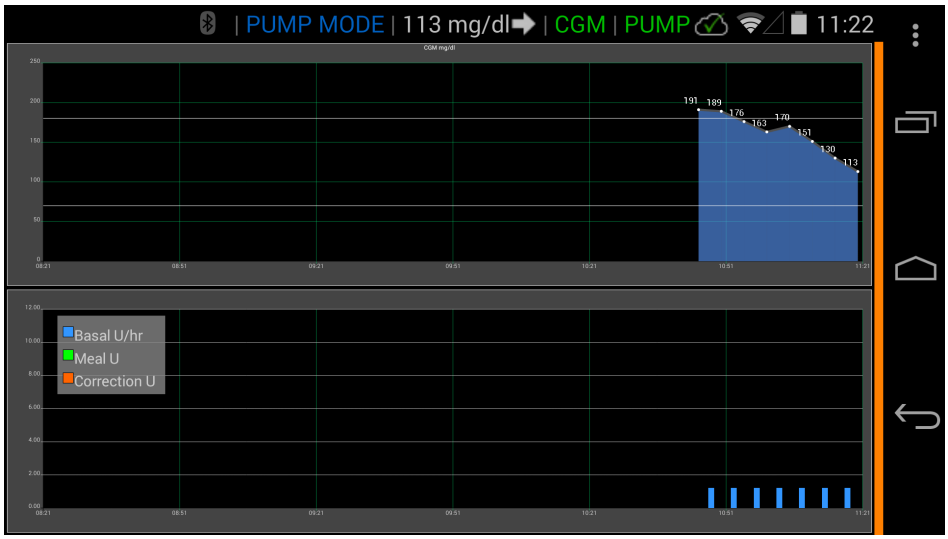


Figure 27

display of delivered insulin. The CGM plot is filled with a color that indicates the operating mode of the system at the time the CGM value was received: green for **Closed Loop**, blue for **Pump,** purple for **Safety**, yellow for **Sensor Only** and red for **Stopped**. The primary insulin display shows delivered insulin in U/hour. Vertical blue bars show the insulin actually delivered which may be different from the basal profile depending upon the action of the APCservice, BRMservice and Safety System. Meal boluses are indicated by vertical green bars and corrections are indicated by vertical orange bars.

The time scale is common to both plots and can be changed either by pinching and spreading two fingers directly on the display or by pressing the **Options** button and selecting a time range.

II.D.3.4.4. Calibration

Certain CGM devices may require calibration through the APMMP platform. The Standalone Driver can also be calibrated through the APMMP platform in order to test Calibration. The **Calibrate CGM** menu item will appear in the APMMP Main Menu Options (**Figure 26**). If the CGM device supports DiAs calibration then pressing the calibration menu item will cause the Calibration Activity (see Figure 28) to appear in the center of the screen. The user measures their blood glucose with an SMBG, enters the value in the box and presses the "Calibrate" button. The DiAs APMMP will indicate when calibration is complete, the Calibration Activity will disappear and control will return to the DiAsUI main screen.

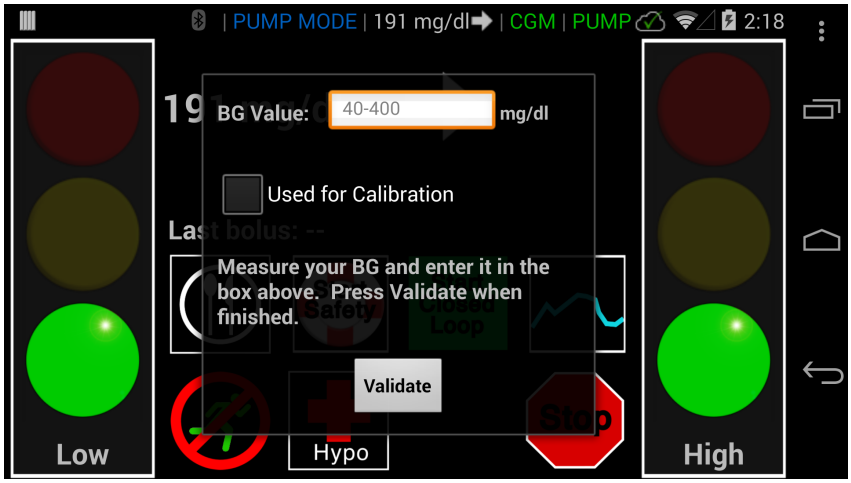


Figure 28

II.D.3.4.5. MCM Meal Screen

When the DiAs system is in either **Pump, Safety** or **Closed Loop** mode the **Meal** button will appear in the central part of the Home Screen. Tapping this button launches the meal screen. As this is a developer-replaceable application its appearance may vary. The following sections describe the default screen provided with the DiAs APMMP.

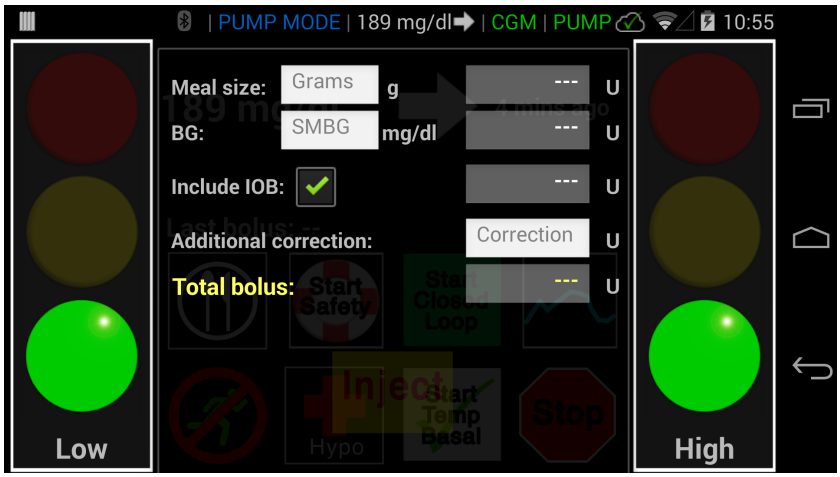


Figure 29

The operation varies depending upon the system operating mode and the status of the *meal_activity_bolus_calculation_mode* parameter (see Table 3 - Elements of *parameters.xml*). If the MealActivity is configured to calculate the meal bolus (*meal_activity_bolus_calculation_mode==0* or *meal_activity_bolus_calculation_mode == 1* and Pump mode) then the screen appears as in Figure 29. The user can enter a **Meal size** (grams of carbohydrate), their current **Blood Glucose** (mg/dl), can choose whether to subtract the **IOB** estimate and can include an **Additional correction** (Units) which

may be positive or negative. Each of these inputs generates a corresponding insulin contribution in the right hand column which are tallied to generate **Total bolus** (see Table 7).

Input	Description	Insulin
Meal size	grams of carbohydrate	(Meal size)/CR
BG	Mg/dl	(BG-110 mg/dl)/CF
Include IOB	Boolean	-IOB
Additional correction	Units of insulin	Positive or negative correction

Table 7

When the Inject button is pressed, and the bolus is confirmed by the user, the resulting **Total bolus** is injected just as if the subject was using their pump to directly send a bolus. Pressing the **Home** button causes the bolus to be canceled and the **Home** screen to display.

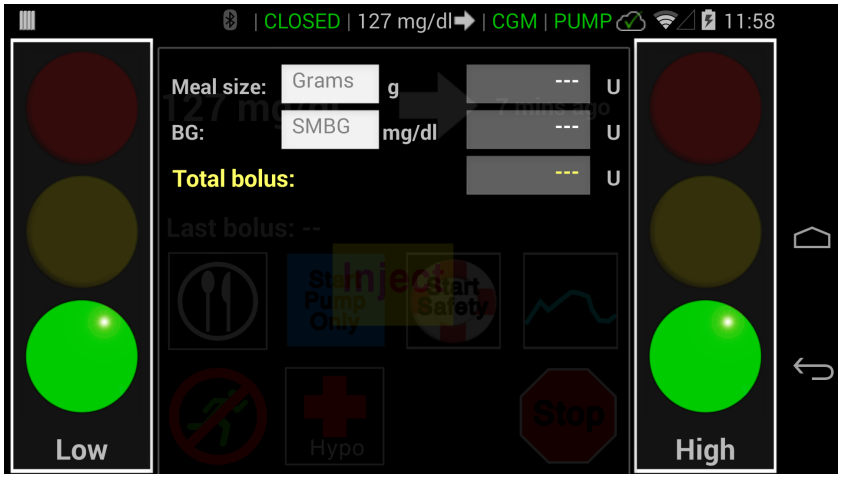


Figure 30

In **Closed Loop** mode only the **Meal size**, **BG** and **Total bolus** fields are enabled (see Figure 30). When both the **Meal size** and **BG** fields contain valid inputs the DiAs sends a message to the MCMservice requesting that it calculate the appropriate bolus. This calculation is done based upon whatever optimal bolus calculation algorithm is embedded in MCMservice. The bolus size is returned to meal screen which displays the total bolus as shown in Figure 30. When the Inject button is pressed, and the bolus is confirmed by the user, the resulting **Total bolus** is injected just as if the subject was using their pump to directly send a bolus. Pressing the **Home** button causes the bolus to be canceled and the **Home** screen to display.

III. SOFTWARE DISTRIBUTION

The APSDK is distributed in the file APSDK-v2.4-archive.zip. The archive contains this manual, a python script for installing files on the cell phone (*maf_install.py* which requires the utility program *adb*

from the Android project), *parameters.xml*, *configurations.xml* and several source code folders in Eclipse project format:

- APCserviceShell
- BRMserviceShell
- MCMservice
- ConstraintServiceShell
- SSMserviceShell
- ExerciseServiceShell
- sysman.jar library

It also contains the following binary (.apk and .xml) files:

- APCserviceShell.apk
- biometricsContentProvider.apk
- BRMserviceShell.apk
- BTLE_G4.apk
- BTLE_Tandem.apk
- BTLE_Share.apk
- CgmService.apk
- configurations.xml
- ConstraintServiceShell.apk
- DiAsService.apk
- DiAsSetupNew.apk
- DiAsUI.apk
- ExerciseServiceShell.apk
- HR_Driver.apk
- Bioharness_Driver.apk
- MCMservice.apk
- networkService.apk
- parameters.xml
- PumpService.apk
- RocheDriver.apk
- SSMserviceShell.apk
- standaloneDriver.apk
- supervisor.apk

IV. PUBLIC APPLICATION PROGRAMMING INTERFACES

IV.A. biometricsContentProvider Database Description and API

At creation the biometricsContentProvider builds a database that includes the following tables which are described in detail in Table 9 through Table 36:

- CGM_TABLE (cgm)
- INSULIN_TABLE (insulin)
- INSULIN_CREDIT_TABLE (insulincredit)
- STATE_ESTIMATE_TABLE (stateestimate)
- HMS_STATE_ESTIMATE_TABLE (hmsstateestimate)
- MEAL_TABLE (meal)
- LOG_TABLE (log)
- SUBJECT_DATA_TABLE (subjectdata)
- CF_PROFILE_TABLE (cfprofile)
- CR_PROFILE_TABLE (crprofile)
- BASAL_PROFILE_TABLE (basalprofile)
- HARDWARE_CONFIGURATION_TABLE (hardwareconfiguration)
- USER_TABLE_1 (user1)
- USER_TABLE_2 (user2)
- USER_TABLE_3 (user3)
- USER_TABLE_4 (user4)
- SMBG_TABLE (smbg)
- PASSWORD_TABLE (password)
- CGM_DETAILS_TABLE (cgmdetails)
- PUMP_DETAILS_TABLE (pumpdetails)
- DEVICE_DETAILS_TABLE (devicedetails)
- SAFETY_PROFILE_TABLE (safetyprofile)
- CONSTRAINTS_TABLE (constraints)
- GPS_TABLE (gps)
- EXERCISE_SENSOR_TABLE (exercise_sensor)
- ACCELEROMETER_TABLE (acc)
- SYSTEM_TABLE (system)
- EVENT_TABLE (events)
- PARAMETER_TABLE (params)
- SERVER_TABLE (server_url)
- TEMP_BASAL_TABLE (temporary_basal_rate)
- STATE_TABLE (state)
- TIME_TABLE (time)

- EXERCISE_STATE_TABLE (exercise_state)
- CONTROLLER_PARAMETERS (controller_parameters)
- MISC_TABLE (misc)

Data is read from and written to these tables through the biometricsContentProvider, which supports the following methods to permit applications to read, write, insert and delete records:

- query(Uri uri, String [] projection, String selection, String selectionArgs, String sortOrder)
- update(Uri uri, ContentValues values, String selection, String [] selectionArgs)
- insert(Uri URI, ContentValues values)
- delete(Uri URI, String selection, String selectionArgs[])

Data tables managed by the biometricsContentProvider are accessed using Uri elements defined in the Biometrics class within the SysMan project. A summary of the Uri elements is provided in Table 8:

Table 8 - Uri Definitions

```
// Interface definitions for the biometricsContentProvider
public static final String PROVIDER_NAME = "edu.virginia.dtc.provider.biometrics";
public static final Uri CGM_URI = Uri.parse("content://"+ PROVIDER_NAME + "/cgm");
public static final Uri INSULIN_URI = Uri.parse("content://"+ PROVIDER_NAME + "/insulin");
public static final Uri INSULIN_CREDIT_URI = Uri.parse("content://"+ PROVIDER_NAME + "/insulincredit");
public static final Uri STATE_ESTIMATE_URI = Uri.parse("content://"+ PROVIDER_NAME + "/stateestimate");
public static final Uri HMS_STATE_ESTIMATE_URI = Uri.parse("content://" + PROVIDER_NAME + "/hmsstateestimate");
public static final Uri MEAL_URI = Uri.parse("content://"+ PROVIDER_NAME + "/meal");
public static final Uri LOG_URI = Uri.parse("content://"+ PROVIDER_NAME + "/log");
public static final Uri SUBJECT_DATA_URI = Uri.parse("content://"+ PROVIDER_NAME + "/subjectdata");
public static final Uri CF_PROFILE_URI = Uri.parse("content://" + PROVIDER_NAME + "/cfprofile");
public static final Uri CR_PROFILE_URI = Uri.parse("content://" + PROVIDER_NAME + "/crprofile");
public static final Uri BASAL_PROFILE_URI = Uri.parse("content://" + PROVIDER_NAME + "/basalprofile");
public static final Uri HARDWARE_CONFIGURATION_URI = Uri.parse("content://" + PROVIDER_NAME + "/hardwareconfiguration");
public static final Uri USER_TABLE_1_URI = Uri.parse("content://"+ PROVIDER_NAME + "/user1");
public static final Uri USER_TABLE_2_URI = Uri.parse("content://"+ PROVIDER_NAME + "/user2");
public static final Uri SMBG_URI = Uri.parse("content://"+ PROVIDER_NAME + "/smbg");
public static final Uri PASSWORD_URI = Uri.parse("content://"+ PROVIDER_NAME + "/password");
public static final Uri CGM_DETAILS_URI = Uri.parse("content://"+ PROVIDER_NAME + "/cgmdetails");
public static final Uri PUMP_DETAILS_URI = Uri.parse("content://"+ PROVIDER_NAME + "/pumpdetails");
public static final Uri DEV_DETAILS_URI = Uri.parse("content://"+ PROVIDER_NAME + "/devicedetails");
public static final Uri SAFETY_PROFILE_URI = Uri.parse("content://" + PROVIDER_NAME + "/safetyprofile");
public static final Uri CONSTRAINTS_URI = Uri.parse("content://"+ PROVIDER_NAME + "/constraints");
public static final Uri GPS_URI = Uri.parse("content://" + PROVIDER_NAME + "/gps");
public static final Uri EXERCISE_SENSOR_URI = Uri.parse("content://" + PROVIDER_NAME + "/exercise_sensor");
public static final Uri ACC_URI = Uri.parse("content://" + PROVIDER_NAME + "/acc");
public static final Uri USER_TABLE_3_URI = Uri.parse("content://"+ PROVIDER_NAME + "/user3");
public static final Uri USER_TABLE_4_URI = Uri.parse("content://"+ PROVIDER_NAME + "/user4");
public static final Uri SYSTEM_URI = Uri.parse("content://"+ PROVIDER_NAME + "/system");
public static final Uri EVENT_URI = Uri.parse("content://"+ PROVIDER_NAME + "/events");
public static final Uri PARAM_URI = Uri.parse("content://"+ PROVIDER_NAME + "/params");
public static final Uri SERVER_URI = Uri.parse("content://"+ PROVIDER_NAME + "/server_url");
public static final Uri TEMP_BASAL_URI = Uri.parse("content://" + PROVIDER_NAME + "/temporary_basal_rate");
public static final Uri STATE_URI = Uri.parse("content://" + PROVIDER_NAME + "/state");
public static final Uri TIME_URI = Uri.parse("content://" + PROVIDER_NAME + "/time");
public static final Uri EXERCISE_STATE_URI = Uri.parse("content://" + PROVIDER_NAME + "/exercise_state");
public static final Uri CONTROLLER_PARAMETERS_URI = Uri.parse("content://" + PROVIDER_NAME + "/controller_parameters");
public static final Uri TABLE_URI = Uri.parse("content://" + PROVIDER_NAME + "/"+MISC_TABLE_NAME);
```

The biometricsContentProvider enforces "QUID" (Query, Update, Insert, Delete) permissions for all DiAs applications on a table by table basis. The Developer and Developer-Replaceable Applications have QUID access permissions as described in **Error! Reference source not found.** through Table 49:

Table 9 – CGM_TABLE ("cgm")

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
diasState	integer	State of DiAs at "time"	DiAs operating state
cgm	double	not null	CGM value in mg/dl
trend	integer		2: Arrow straight up 1: Arrow angled up 0: Arrow flat -1: Arrow angled down -2: Arrow straight down
state	integer	State of CGM device	0: Normal 1: data error 2: Not active 3: None 4: Noise 5: Warmup 6: Calibration needed 7: Dual Calibration needed 8: Calibration Low 9: Calibration high 10: Sensor failed
time	long	not null	Unix time stamp from CGM device
recv_time	long		Unix time stamp from phone at moment that CGM value was received
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 10 – INSULIN_TABLE ("insulin")

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
req_time	long		timestamp bolus was requested
req_total	double		
req_basal	double		basal insulin requested
req_meal	double		meal insulin requested
req_corr	double		correction insulin requested
deliv_time	long		timestamp bolus was delivered
deliv_total	double		
deliv_basal	double		basal insulin delivered
deliv_meal	double		meal insulin delivered

deliv_corr	double		correction insulin delivered
recv_time	long		Unix time stamp from phone at moment that insulin was delivered
running_total	double		
identifier	long		
status	int		status of bolus
num_retries	int		
type	int		type of bolus (system or user generated)
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 11 – STATE_ESTIMATE_TABLE (“stateestimate”)

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
enough_data	boolean		
CGM	double		
IOB	double		Insulin on board (U)
IOBlast	double		
IOBlast2	double		
Gpred	double		Estimated BG 1 (mg/dl)
Gbrakes	double		Estimated BG 2 (mg/dl)
Gpred_light	double		Estimated BG 3 (mg/dl)
Xi00	double		State estimate element 0
Xi01	double		State estimate element 1
Xi02	double		State estimate element 2
Xi03	double		State estimate element 3
Xi04	double		State estimate element 4
Xi05	double		State estimate element 5
Xi06	double		State estimate element 6
Xi07	double		State estimate element 7
isMealBolus	boolean		
Gpred_bolus	double		
CHOpred	double		
Abrakes	double		
CGM_corr	double		
IOB_controller_rate	double		
SSM_amount	double		
State	int		
stoplight	int		hypo traffic light
stoplight2	int		hyper traffic light
SSM_state	double		
SSM_state_timestamp	long		
exerciseFlagTimeStart	long		
exerciseFlagTimeStop	long		
hypoFlagTime	long		
calFlagTime	long		
corrFlagTime	long		

DIAS_state	int		
UTC_offset_seconds	int		
time	long	not null	Unix time stamp
Umax_IOB	double		
brakes_coeff	double		
asynchronous	int		
Gpred_1h	double		
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 12 – MEAL_TABLE ("meal")

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long	not null	Announcement time
smbg	double		User SMBG
carbs	double		Bolus size (U)
meal_bolus	double		Meal insulin
corr_bolus	double		Correction insulin
status	int		
json	text		Field for adding extra data to the meal
send_attempts_server	int		Number of attempts to send to server
received_server	boolean		Boolean that signifies server reception

Table 13 – LOG_TABLE ("log")

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
status	text	not null	Log text
service	text	not null	Name of service generating log entry
time	long	not null	log time
priority	int		Can be used to establish relative importance of entries
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 14 – SUBJECT_DATA_TABLE ("subjectdata")

_id	integer	primary key autoincrement	
subjectid	text	not null	Subject identifier
session	text	not null	Session identifier

weight	int	not null	Weight in kg
height	int	not null	Height in cm
age	int	not null	Age in years
TDI	int	not null	TDI in U
isfemale	int	not null	0 if male 1 if female
AIT	int	not null	Active Insulin Time in hours
enableIOtest	int	not null	1 if IO tests enabled 0 if IO tests disabled
insulinSetupComplete	int	not null	Used during setup
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 15 – CF_PROFILE_TABLE (“cfprofile”)

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long	not null	time in seconds relative to start of current day
value	double	not null	Correction Factor value which becomes active at the current time (mg/dl/U)
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 16 – CR_PROFILE_TABLE (“crprofile”)

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long	not null	time in seconds relative to start of current day
value	double	not null	Carbohydrate to Insulin Ratio value which becomes active at the current time (g/U)
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 17 – BASAL_PROFILE_TABLE (“basalprofile”)

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long	not null	time in seconds relative to

			start of current day
value	double	not null	Basal rate value which becomes active at the current time (U/hour)
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 18 – HARDWARE_CONFIGURATION_TABLE (“hardwareconfiguration”)

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
running	int	not null	1 if system running 0 otherwise
running_pump	text		Name of pump
running_cgm	text		Name of CGM
running_misc	text		
last_state	int		Last known DiAs state
ask_at_startup	int		
send_attempts_server	int		
received_server	boolean		boolean that signifies server reception

Table 19 – USER_TABLE_1 (“user1”)

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long		User defined
l0	long		User defined
l1	long		User defined
d0	double	not null	User defined
d1	double		User defined
d2	double		User defined
d3	double		User defined
d4	double		User defined
d5	double		User defined
d6	double		User defined
d7	double		User defined
d8	double		User defined
d9	double		User defined
d10	double		User defined
d11	double		User defined
d12	double		User defined
d13	double		User defined
d14	double		User defined
d15	double		User defined
send_attempts_server	int		

received_server	boolean		boolean that signifies server reception

Table 20 – USER_TABLE_2 ("user2")

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long		User defined
l0	long		User defined
l1	long		User defined
d0	double	not null	User defined
d1	double		User defined
d2	double		User defined
d3	double		User defined
d4	double		User defined
d5	double		User defined
d6	double		User defined
d7	double		User defined
d8	double		User defined
d9	double		User defined
d10	double		User defined
d11	double		User defined
d12	double		User defined
d13	double		User defined
d14	double		User defined
d15	double		User defined
send_attempts_server	int		
received_server	boolean		boolean that signifies server reception

Table 21 – INSULIN_CREDIT_TABLE ("insulincredit")

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
credit	double	not null	Insulin credit request in U
spent	double	not null	Insulin spent in U
time	long	not null	Unix time stamp for CGM
net	double	not null	credit – spent
send_attempts_server	int		
received_server	boolean		boolean that signifies server reception

Table 22 – SMBG_TABLE ("smbg")

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long	not null	Unix time stamp for CGM
smbg	double	not null	
isCalibration	int	not null	0: not calibration 1: calibration

isHypo	boolean	not null	
didTreat	boolean		
carbs	int		
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 23 – PASSWORD_TABLE (“password”)

Name	Data Type	Notes	Description
password	text		DiAs system password
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 24 – CGM_DETAILS_TABLE (“cgmdetails”)

Name	Data Type	Notes	Description
details	text		
min_cgm	double		lowest CGM value that can be reported by this device in mg/dl
max_cgm	double		highest CGM value that can be reported by this device in mg/dl
phone_calibration	int		If >0 then CGM calibration from the DiAs is permitted, otherwise not permitted
state	int		
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 25 – PUMP_DETAILS_TABLE (“pumpdetails”)

Name	Data Type	Notes	Description
details	text		
low_reservoir_threshold_U	double		threshold in U below which the pump reservoir is considered low
reservoir_size_U	double		pump reservoir size in U
infusion_rate_U_sec	double		pump infusion rate U/sec
unit_conversion	double		
unit_name	text		
min_bolus_U	double		min bolus in U

max_bolus_U	double		max bolus in U
min_quanta_U	double		min bolus quantization unit (U)
queryable	int		0: not queryable 1: queryable
temp_basal	int		
temp_basal_time	int		
retries	int		
max_retries	int		
state	int		
service_state	int		
set_temp_basal_time	int		
set_temp_basal_target	int		
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 26 – SAFETY_PROFILE_TABLE (“safetyprofile”)

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long	not null	starting time within the current day of a safety only time segment
endtime	long	not null	ending time within the current day of a safety only time segment
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 27 – DEVICE_DETAILS_TABLE (“devicedetails”)

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long	not null	
battery	text		
plugged	int		
network_type	text		
network_strength	text		
ping_attempts_last_hour	int		
ping_success_last_hour	int		
battery_stats	text		
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 28 – CONSTRAINTS_TABLE ("constraints")

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long		Unix time stamp
status	integer		-1: Constraint requested by SSMservice 0: Constraint written by ConstraintService 1: Constraint read by SSMservice -2: Constraint timed out
constraint1	double		User defined insulin constraint in U
constraint2	double		User defined insulin constraint in U
constraint3	double		User defined insulin constraint in U
constraint4	double		User defined insulin constraint in U
constraint5	double		User defined insulin constraint in U
constraint6	double		User defined insulin constraint in U
constraint7	double		User defined insulin constraint in U
constraint8	double		User defined insulin constraint in U
constraint9	double		User defined insulin constraint in U
constraint10	double		User defined insulin constraint in U
constraint11	double		User defined insulin constraint in U
constraint12	double		User defined insulin constraint in U
constraint13	double		User defined insulin constraint in U
constraint14	double		User defined insulin constraint in U
constraint15	double		User defined insulin constraint in U
constraint16	double		User defined insulin constraint in U
constraint17	double		User defined insulin constraint in U
constraint18	double		User defined insulin constraint in U
constraint19	double		User defined insulin constraint in U
constraint20	double		User defined insulin constraint in U

send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 29 – GPS_TABLE (“gps”)

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long		Unix time stamp
gpsLat	double		GPS latitude value
gpsLong	double		GPS longitude value
gpsAlt	double		GPS altitude value
gpsBearing	float		GPS bearing
gpsSpeed	float		GPS speed
gpsSysTime	long		GPS time
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 30 – EXERCISE_SENSOR_TABLE (“exercise_sensor”)

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long		Unix time stamp
json_data	text		Sensor data stored in json format
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 31 – ACC_TABLE (“acc”)

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long		Unix time stamp
x	float		X acceleration value
y	float		Y acceleration value
z	float		Z acceleration value
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 32 – USER_TABLE_3 (“user3”)

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long		User defined
l0	long		User defined
l1	long		User defined
d0	double	not null	User defined
d1	double		User defined
d2	double		User defined
d3	double		User defined
d4	double		User defined
d5	double		User defined
d6	double		User defined
d7	double		User defined
d8	double		User defined
d9	double		User defined
d10	double		User defined
d11	double		User defined
d12	double		User defined
d13	double		User defined
d14	double		User defined
d15	double		User defined
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 33 – USER_TABLE_4 (“user4”)

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long		User defined
l0	long		User defined
l1	long		User defined
d0	double	not null	User defined
d1	double		User defined
d2	double		User defined
d3	double		User defined
d4	double		User defined
d5	double		User defined
d6	double		User defined
d7	double		User defined
d8	double		User defined
d9	double		User defined
d10	double		User defined
d11	double		User defined
d12	double		User defined
d13	double		User defined
d14	double		User defined
d15	double		User defined
send_attempts_server	int		number of attempts to send

			to server
received_server	boolean		boolean that signifies server reception

Table 34 - SYSTEM_TABLE (“system”)

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long		
sysTime	long		
safetyMode	int		
diasState	int		
enableIOTest	int		
battery	int		
cgmValue	double		
cgmTrend	int		
cgmLastTime	long		
cgmState	int		
cgmStatus	text		
pumpLastBolus	double		
pumpLastBolusTime	long		
pumpState	int		
pumpStatus	text		
iobValue	double		
hypoLight	int		
hyperLight	int		
apcBolus	double		
apcStatus	int		
apcType	int		
apcString	string		
exercising	int		
alarmNoCgm	int		
alarmHypo	int		
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 35 - EVENTS_TABLE ("events")

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
time	long not null		
code	int		
json	text		
settings	int		
popup_displayed	int		
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server

			reception

Table 36 - PARAMS_TABLE ("params")

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
name	text		Used to access parameter
value	text		Parameter value stored as a text string
type	text		int double string long boolean
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 37 - SERVER_TABLE ("server_url")

Name	Data Type	Notes	Description
server_url	text		DWM server URI text string
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 38 - TEMP_BASAL_TABLE ("temporary_basal_rate")

Name	Data Type	Notes	Description
_id	integer	primary key autoincrement	
start_time	long		Start of temp basal interval – Unix time
scheduled_end_time	long		Scheduled end of temp basal interval – Unix time
actual_end_time	long		Actual end of temp basal interval – Unix time
percent_of_profile_basal_rate	int		
status_code	int		
owner	int		
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 39 - STATE_TABLE ("state")

Name	Data Type	Notes	Description
_id	int	primary key autoincrement	
sync_state	int		
async_state	int		

tbr_state	int		
dev_req	int		
dev_resp	int		
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 40 - TIME_TABLE ("time")

Name	Data Type	Notes	Description
_id	int	primary key autoincrement	
simTime	long		
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 41 - EXERCISE_STATE_TABLE ("exercise_state")

Name	Data Type	Notes	Description
_id	int	primary key autoincrement	
time	long	not null	
currentlyExercising	int		
json_data	text		
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 42 – CONTROLLER_PARAMETERS_TABLE ("controller_parameters")

Name	Data Type	Notes	Description
_id	int	primary key autoincrement	
time	long	not null	
json_data	text		
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

Table 43 – MISC_TABLE ("misc")

Name	Data Type	Notes	Description
_id	int	primary key autoincrement	
time	long	not null	
id	long		
json	text		
send_attempts_server	int		number of attempts to send to server
received_server	boolean		boolean that signifies server reception

The *Developer* and *Developer-Replaceable* Applications have the following access permissions to database tables:

Table 44 - MCMservice database Permissions

Table	Permissions
CGM_TABLE	Q
INSULIN_TABLE	Q
STATE_ESTIMATE_TABLE	Q
MEAL_TABLE	Q
LOG_TABLE	QI
SUBJECT_DATA_TABLE	Q
CF_PROFILE	Q
CR_PROFILE	Q
BASAL_PROFILE	Q
HARDWARE_CONFIGURATION_TABLE	Q
USER_TABLE_1	Q
USER_TABLE_2	Q
INSULIN_CREDIT_TABLE	Q
SMBG_TABLE	Q
HMS_STATE_ESTIMATE_TABLE	Q
PASSWORD_TABLE	Q
CGM_DETAILS_TABLE	Q
PUMP_DETAILS_TABLE	QUI
SAFETY_PROFILE_TABLE	Q
DEVICE_DETAILS_TABLE	Q
CONSTRAINTS_TABLE	Q
GPS_TABLE	-
EXERCISE_SENSOR_TABLE	-
ACCELEROMETER_TABLE	-
USER_TABLE_3	QI
USER_TABLE_4	QI
SYSTEM_TABLE	Q
EVENT_TABLE	QI
PARAMETER_TABLE	Q
SERVER_TABLE	Q
TEMP_BASAL_TABLE	Q
STATE_TABLE	Q
TIME_TABLE	Q
EXERCISE_STATE_TABLE	Q
SERVICE_OUTPUTS_TABLE	QI
MISC_TABLE	Q

Table 45 – SSMservice Database Permissions

Table	Permissions
CGM_TABLE	Q

INSULIN_TABLE	QI
STATE_ESTIMATE_TABLE	QUI
MEAL_TABLE	QU
LOG_TABLE	QI
SUBJECT_DATA_TABLE	Q
CF_PROFILE	Q
CR_PROFILE	Q
BASAL_PROFILE	Q
HARDWARE_CONFIGURATION_TABLE	Q
USER_TABLE_1	QUI
USER_TABLE_2	QUI
INSULIN_CREDIT_TABLE	QUI
SMBG_TABLE	Q
HMS_STATE_ESTIMATE_TABLE	Q
PASSWORD_TABLE	Q
CGM_DETAILS_TABLE	Q
PUMP_DETAILS_TABLE	Q
SAFETY_PROFILE_TABLE	Q
DEVICE_DETAILS_TABLE	Q
CONSTRAINTS_TABLE	QUID
GPS_TABLE	Q
EXERCISE_SENSOR_TABLE	Q
ACCELEROMETER_TABLE	Q
USER_TABLE_3	QUI
USER_TABLE_4	QUI
SYSTEM_TABLE	Q
EVENT_TABLE	QI
PARAMETER_TABLE	Q
SERVER_TABLE	Q
TEMP_BASAL_TABLE	Q
STATE_TABLE	Q
TIME_TABLE	Q
EXERCISE_STATE_TABLE	Q
SERVICE_OUTPUTS_TABLE	QI
MISC_TABLE	Q

Table 46 – ConstraintService Database Permissions

Table	Permissions
CGM_TABLE	Q
INSULIN_TABLE	Q
STATE_ESTIMATE_TABLE	Q
MEAL_TABLE	Q
LOG_TABLE	QI
SUBJECT_DATA_TABLE	Q
CF_PROFILE	Q
CR_PROFILE	Q
BASAL_PROFILE	Q
HARDWARE_CONFIGURATION_TABLE	Q
USER_TABLE_1	QUI
USER_TABLE_2	Q
INSULIN_CREDIT_TABLE	Q

SMBG_TABLE	Q
HMS_STATE_ESTIMATE_TABLE	Q
PASSWORD_TABLE	Q
CGM_DETAILS_TABLE	Q
PUMP_DETAILS_TABLE	Q
SAFETY_PROFILE_TABLE	Q
DEVICE_DETAILS_TABLE	Q
CONSTRAINTS_TABLE	QUID
GPS_TABLE	Q
EXERCISE_SENSOR_TABLE	Q
ACCELEROMETER_TABLE	Q
USER_TABLE_3	Q
USER_TABLE_4	Q
SYSTEM_TABLE	Q
EVENT_TABLE	QI
PARAMETER_TABLE	Q
SERVER_TABLE	Q
TEMP_BASAL_TABLE	Q
STATE_TABLE	Q
TIME_TABLE	Q
EXERCISE_STATE_TABLE	Q
SERVICE_OUTPUTS_TABLE	Q
MISC_TABLE	Q

Table 47 – ExerciseService Database Permissions

Table	Permissions
CGM_TABLE	Q
INSULIN_TABLE	Q
STATE_ESTIMATE_TABLE	Q
MEAL_TABLE	Q
LOG_TABLE	QI
SUBJECT_DATA_TABLE	Q
CF_PROFILE	Q
CR_PROFILE	Q
BASAL_PROFILE	Q
HARDWARE_CONFIGURATION_TABLE	Q
USER_TABLE_1	Q
USER_TABLE_2	Q
INSULIN_CREDIT_TABLE	Q
SMBG_TABLE	Q
HMS_STATE_ESTIMATE_TABLE	Q
PASSWORD_TABLE	Q
CGM_DETAILS_TABLE	Q
PUMP_DETAILS_TABLE	Q
SAFETY_PROFILE_TABLE	Q
DEVICE_DETAILS_TABLE	Q
CONSTRAINTS_TABLE	Q
GPS_TABLE	-
EXERCISE_SENSOR_TABLE	QUI
ACCELEROMETER_TABLE	-
USER_TABLE_3	Q

USER_TABLE_4	Q
SYSTEM_TABLE	Q
EVENT_TABLE	QI
PARAMETER_TABLE	Q
SERVER_TABLE	Q
TEMP_BASAL_TABLE	Q
STATE_TABLE	Q
TIME_TABLE	Q
EXERCISE_STATE_TABLE	QUI
SERVICE_OUTPUTS_TABLE	Q
MISC_TABLE	Q

Table 48 – APCservice and BRMservice Database Permissions

CGM_TABLE	Q
INSULIN_TABLE	QUI
STATE_ESTIMATE_TABLE	Q
MEAL_TABLE	QU
LOG_TABLE	QI
SUBJECT_DATA_TABLE	Q
CF_PROFILE	Q
CR_PROFILE	Q
BASAL_PROFILE	Q
HARDWARE_CONFIGURATION_TABLE	Q
USER_TABLE_1	QUI
USER_TABLE_2	Q
INSULIN_CREDIT_TABLE	Q
SMBG_TABLE	Q
HMS_STATE_ESTIMATE_TABLE	QUID
PASSWORD_TABLE	Q
CGM_DETAILS_TABLE	Q
PUMP_DETAILS_TABLE	Q
SAFETY_PROFILE_TABLE	Q
DEVICE_DETAILS_TABLE	Q
CONSTRAINTS_TABLE	Q
GPS_TABLE	Q
EXERCISE_SENSOR_TABLE	Q
ACCELEROMETER_TABLE	Q
USER_TABLE_3	QUID
USER_TABLE_4	QUID
SYSTEM_TABLE	Q
EVENT_TABLE	QI
PARAMETER_TABLE	Q
SERVER_TABLE	Q
TEMP_BASAL_TABLE	Q
STATE_TABLE	Q
TIME_TABLE	Q
EXERCISE_STATE_TABLE	Q
SERVICE_OUTPUTS_TABLE	QI
MISC_TABLE	Q

Table 49 - DiAsUI Database Table Permissions

CGM_TABLE	Q
INSULIN_TABLE	Q
STATE_ESTIMATE_TABLE	Q
MEAL_TABLE	Q
LOG_TABLE	QI
SUBJECT_DATA_TABLE	Q
CF_PROFILE	Q
CR_PROFILE	Q
BASAL_PROFILE	Q
HARDWARE_CONFIGURATION_TABLE	Q
USER_TABLE_1	Q
USER_TABLE_2	Q
INSULIN_CREDIT_TABLE	Q
SMBG_TABLE	Q
HMS_STATE_ESTIMATE_TABLE	Q
PASSWORD_TABLE	Q
CGM_DETAILS_TABLE	Q
PUMP_DETAILS_TABLE	Q
SAFETY_PROFILE_TABLE	Q
DEVICE_DETAILS_TABLE	Q
CONSTRAINTS_TABLE	Q
GPS_TABLE	Q
EXERCISE_SENSOR_TABLE	Q
ACCELEROMETER_TABLE	Q
USER_TABLE_3	Q
USER_TABLE_4	Q
SYSTEM_TABLE	Q
EVENT_TABLE	QI
PARAMETER_TABLE	Q
SERVER_TABLE	Q
TEMP_BASAL_TABLE	QUI
STATE_TABLE	Q
TIME_TABLE	Q
EXERCISE_STATE_TABLE	Q
SERVICE_OUTPUTS_TABLE	Q
MISC_TABLE	Q

IV.B. DiAsUI

The DiAsUI Developer-Replaceable Application is responsible for providing the Graphical User Interface to the APMMP system when it is in Normal Operating Mode. DiAsUI shall support the *DiAsUI to DiAsService API* (Table 50) in order that it may provide signals to DiAsService indicating the user's intent to switch operating modes or perform other functions. DiAsUI reads from biometricsContentProvider (particularly the SYSTEM_TABLE, CGM_TABLE and INSULIN_TABLE) to access the current operating state of the APMMP system. DiAsUI is also responsible for launching toe MealActivity when requested by the user (through a button press, for example). It does this using the *MealActivity to DiAsUI API* (**Error! Reference source not found.**).

Table 50 - DiAsUI to DiAsService API

DIAS_SERVICE_COMMAND_STOP_CLICK

Transition to Stopped mode.

```
Intent intent1 = new Intent();
intent1.setClassName("edu.virginia.dtc.DiAsService", "edu.virginia.dtc.DiAsService.DiAsService");
intent1.putExtra("DiAsCommand", DIAS_SERVICE_COMMAND_STOP_CLICK);
startService(intent1);
```

DIAS_SERVICE_COMMAND_START_OPEN_LOOP_CLICK

Transition to Pump mode.

```
Intent intent1 = new Intent();
intent1.setClassName("edu.virginia.dtc.DiAsService", "edu.virginia.dtc.DiAsService.DiAsService");
intent1.putExtra("DiAsCommand", DIAS_SERVICE_COMMAND_START_OPEN_LOOP_CLICK);
startService(intent1);
```

DIAS_SERVICE_COMMAND_START_SAFETY_CLICK

Transition to Safety mode.

```
Intent intent1 = new Intent();
intent1.setClassName("edu.virginia.dtc.DiAsService", "edu.virginia.dtc.DiAsService.DiAsService");
intent1.putExtra("DiAsCommand", DIAS_SERVICE_COMMAND_START_SAFETY_CLICK);
startService(intent1);
```

DIAS_SERVICE_COMMAND_START_CLOSED_LOOP_CLICK

Transition to Closed Loop mode.

```
Intent intent1 = new Intent();
intent1.setClassName("edu.virginia.dtc.DiAsService", "edu.virginia.dtc.DiAsService.DiAsService");
intent1.putExtra("DiAsCommand", DIAS_SERVICE_COMMAND_START_CLOSED_LOOP_CLICK);
startService(intent1);
```

DIAS_SERVICE_COMMAND_START_SENSOR_ONLY_CLICK

Transition to Sensor Only mode.

```
Intent intent1 = new Intent();
intent1.setClassName("edu.virginia.dtc.DiAsService", "edu.virginia.dtc.DiAsService.DiAsService");
intent1.putExtra("DiAsCommand", DIAS_SERVICE_COMMAND_START_SENSOR_ONLY_CLICK);
startService(intent1);
```

DIAS_SERVICE_COMMAND_SET_EXERCISE_STATE

Set exercise state.

```
Intent intent1 = new Intent();
intent1.setClassName("edu.virginia.dtc.DiAsService", "edu.virginia.dtc.DiAsService.DiAsService");
intent1.putExtra("DiAsCommand", DIAS_SERVICE_COMMAND_SET_EXERCISE_STATE);
```

```
intent1.putExtra("currentlyExercising", currentlyExercising);
startService(intent1);
```

DIAS_SERVICE_COMMAND_SET_HYPO_FLAG_TIME

Set flag marking hypoglycemia event.

```
Intent intent1 = new Intent();
intent1.setClassName("edu.virginia.dtc.DiAsService", "edu.virginia.dtc.DiAsService.DiAsService");
intent1.putExtra("DiAsCommand", DIAS_SERVICE_COMMAND_SET_HYPO_FLAG_TIME);
intent1.putExtra("didTreatHypo", didTreatHypo);
startService(intent1);
```

IV.C. Safety Service

The Safety Service supports an API to DiAs Service (Table 51 – SSMservice Public API) and another to Pump Service (

5. SAFETY_SERVICE_CMD_REGISTER_CLIENT

   This command is used by the SSMservice to resolve the Messenger for status information replies to DiAsService:

```
Public Messenger mMessengerToClient = null;
mMessengerToClient = msg.replyTo;
```

6. SAFETY_SERVICE_CMD_START_SERVICE

   DiAsService sends this command to initialize the SSMservice and cause it to start and establish a connection with PumpService:

```
//  Command inputs
paramBundle = msg.getData();
boolean enableIOtest = (boolean)paramBundle.getBoolean(“ enableIOtest”);
int IOB_curve_duration_hours = paramBundle.getInt(“IOB_curve_duration_hours”);      // IOB curve duration (hours)
simulatedTime = paramBundle.getLong(“simulatedTime”, -1);                          // Contains the current Unix
                                                                                   // time stamp or -1 if clock

// Bind to the Insulin Pump Service
Intent intent = new Intent();
intent.setClassName(“edu.virginia.dtc.PumpService”, “edu.virginia.dtc.PumpService. PumpService”);
bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
```

7. SAFETY_SERVICE_CMD_REQUEST_BOLUS

   DiAsService sends this command to direct the SSMservice to deliver insulin to the subject or to request or spend insulin “credit”.

iv. Requesting insulin

The DiAs classifies all insulin delivered to the subject into three categories: basal, correction and meal. The SSMservice can operate in an asynchronous manner and must include a basal bolus based upon the daily basal profile whenever DiAsService makes an insulin request with the

asynchronous flag set to **false**. The **bolusRequested** argument contains the total amount of insulin requested in units and may be zero if only basal insulin delivery is desired (in this case asynchronous=false).

The **differential_basal_rate** is an argument that permits APCservice to modulate the basal delivery rate with respect to the daily basal profile. It can be positive, negative or zero and is limited to the range -6U/hour to +6U/hour. This is added to the basal rate as determined by the SSMservice (which may be attenuated below the profile rate if there is a hypoglycemia risk).

v. credit_request and spend_request

The **credit_request** and **spend_request** arguments are used if the SSMservice is managing an insulin "credit pool". In this case the DiAsService can request "credit" insulin from the SSMservice which can require that the user immediately pre-approve the amount of credit requested. If approved by the user the credit can later be "spent" by the APCservice or BRMservice without any further confirmation required by the Safety System. This mechanism may be used by the DiAsUI to announce a meal (issue a "**credit_request**") and for the APCservice/BRMservice to deliver insulin to treat the meal at the appropriate time (issue a "**spend_request**").

vi. DIAS_STATE and status flags

- **DIAS_STATE** is the current state of the DiAs system and may be Stopped, Pump, Closed Loop, Safety Only or Sensor Only.
- **calFlagTime** is the Unix time of the last CGM calibration
- **hypoFlagTime** is the Unix time of the last hypoglycemic episode reported by the user by pressing the Hypoglycemia Button on the Home Screen of DiAsUI.
- **exercise** is a Boolean reporting whether the DiAsUI Exercise Button indicates that the subject is currently exercising.

```
//  Command inputs
paramBundle = msg.getData();                                                    // Fetch parameter bundle
enableIOtest=paramBundle.getBoolean("enableIOtest", false);
simulatedTime = paramBundle.getLong("simulatedTime", -1);
bolusRequested =  paramBundle.getDouble("bolusRequested", 0);
bolusMeal =  paramBundle.getDouble("bolusMeal", 0);
bolusCorrection =  paramBundle.getDouble("bolusCorrection", 0);
double differential_basal_rate = paramBundle.getDouble("differential_basal_rate", 0);      // [-6U/hour:6U/hour]
credit_request = paramBundle.getDouble("credit_request", 0);
spend_request = paramBundle.getDouble("spend_request", 0);
asynchronous = paramBundle.getBoolean("asynchronous", false);
long calFlagTime = (long)paramBundle.getLong("calFlagTime", 0);
long hypoFlagTime = (long)paramBundle.getLong("hypoFlagTime", 0);
boolean exercise = paramBundle.getBoolean("currentlyExercising", false);
DIAS_STATE = paramBundle.getInt("DIAS_STATE", DIAS_STATE_OPEN_LOOP);
```

When the SSMservice completes normal processing of the deliver bolus command it sends the status SAFETY_SERVICE_STATE_NORMAL to DiAsService along with additional data:

```
response = Message.obtain(null, SAFETY_SERVICE_STATE_NORMAL, 0, 0);
responseBundle = new Bundle();
```

```
responseBundle.putInt("stoplight", hypolight);
responseBundle.putInt("stoplight2", hyperlight);
responseBundle.putBoolean("isMealBolus", isMealBolus);     // true if this is a meal bolus
responseBundle.putDouble("brakes_coeff", brakes_coeff);    // Value between 0 and 1
                                                                                    // 0: brakes on, no basal
                                                                                    // 1: brakes off, full basal
responseBundle.putDouble("IOB", IOB);                      // Latest IOB estimate
response.setData(responseBundle);
mMessengerToClient.send(response);
```

8. SAFETY_SERVICE_CMD_CALCULATE_STATE

DiAsService sends this command to direct the SSMservice to calculate the state estimate based on CGM, insulin and possibly other data from the database. The state estimate is written into the 'stateestimate' database table.

```
// Command inputs
enableIOtest = (boolean)paramBundle.getBoolean("enableIOtest");
simulatedTime = paramBundle.getLong("simulatedTime", -1);
bolus_meal =  0.0;
bolus_correction =  0.0;
bolusRequested = 0.0;
differential_basal_rate = 0.0;
credit_request = 0.0;
spend_request = 0.0;
asynchronous = false;
calFlagTime = (long)paramBundle.getLong("calFlagTime", 0);
hypoFlagTime = (long)paramBundle.getLong("hypoFlagTime", 0);
exercise = paramBundle.getBoolean("currentlyExercising", false);
DIAS_STATE = paramBundle.getInt("DIAS_STATE", DIAS_STATE_OPEN_LOOP);
```

When the SSMservice completes normal processing of the calculate state estimate command it sends the status SAFETY_SERVICE_STATE_CALCULATE_RESPONSE to DiAsService:

```
response = Message.obtain(null, SAFETY_SERVICE_STATE_CALCULATE_RESPONSE, 0, 0);
mMessengerToClient.send(response);
```

Table 52 – Pump Service Public API). It also reads and writes data from biometricsContentProvider. Examples of the use of these APIs can be seen in the **SSMserviceShell** sample application distributed with the APSDK.

9. SAFETY_SERVICE_CMD_REGISTER_CLIENT

This command is used by the SSMservice to resolve the Messenger for status information replies to DiAsService:

```
Public Messenger mMessengerToClient = null;
mMessengerToClient = msg.replyTo;
```

10. SAFETY_SERVICE_CMD_START_SERVICE

DiAsService sends this command to initialize the SSMservice and cause it to start and establish a connection with PumpService:

```
//  Command inputs
```

```
paramBundle = msg.getData();
boolean enableIOtest = (boolean)paramBundle.getBoolean(" enableIOtest");
int IOB_curve_duration_hours = paramBundle.getInt("IOB_curve_duration_hours");   // IOB curve duration (hours)
simulatedTime = paramBundle.getLong("simulatedTime", -1);                        // Contains the current Unix
                                                                                 // time stamp or -1 if clock

// Bind to the Insulin Pump Service
Intent intent = new Intent();
intent.setClassName("edu.virginia.dtc.PumpService", "edu.virginia.dtc.PumpService. PumpService");
bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
```

11. SAFETY_SERVICE_CMD_REQUEST_BOLUS

DiAsService sends this command to direct the SSMservice to deliver insulin to the subject or to request or spend insulin "credit".

vii. Requesting insulin

The DiAs classifies all insulin delivered to the subject into three categories: basal, correction and meal. The SSMservice can operate in an asynchronous manner and must include a basal bolus based upon the daily basal profile whenever DiAsService makes an insulin request with the **asynchronous** flag set to **false**. The **bolusRequested** argument contains the total amount of insulin requested in units and may be zero if only basal insulin delivery is desired (in this case asynchronous=false).

The **differential_basal_rate** is an argument that permits APCservice to modulate the basal delivery rate with respect to the daily basal profile. It can be positive, negative or zero and is limited to the range -6U/hour to +6U/hour. This is added to the basal rate as determined by the SSMservice (which may be attenuated below the profile rate if there is a hypoglycemia risk).

viii. credit_request and spend_request

The **credit_request** and **spend_request** arguments are used if the SSMservice is managing an insulin "credit pool". In this case the DiAsService can request "credit" insulin from the SSMservice which can require that the user immediately pre-approve the amount of credit requested. If approved by the user the credit can later be "spent" by the APCservice or BRMservice without any further confirmation required by the Safety System. This mechanism may be used by the DiAsUI to announce a meal (issue a "**credit_request**") and for the APCservice/BRMservice to deliver insulin to treat the meal at the appropriate time (issue a "**spend_request**").

ix. DIAS_STATE and status flags

- **DIAS_STATE** is the current state of the DiAs system and may be Stopped, Pump, Closed Loop, Safety Only or Sensor Only.
- **calFlagTime** is the Unix time of the last CGM calibration
- **hypoFlagTime** is the Unix time of the last hypoglycemic episode reported by the user by pressing the Hypoglycemia Button on the Home Screen of DiAsUI.
- **exercise** is a Boolean reporting whether the DiAsUI Exercise Button indicates that the subject is

currently exercising.

```
//  Command inputs
paramBundle = msg.getData();                                                    // Fetch parameter bundle
enableIOtest=paramBundle.getBoolean("enableIOtest", false);
simulatedTime = paramBundle.getLong("simulatedTime", -1);
bolusRequested =  paramBundle.getDouble("bolusRequested", 0);
bolusMeal =  paramBundle.getDouble("bolusMeal", 0);
bolusCorrection =  paramBundle.getDouble("bolusCorrection", 0);
double differential_basal_rate = paramBundle.getDouble("differential_basal_rate", 0);        // [-6U/hour:6U/hour]
credit_request = paramBundle.getDouble("credit_request", 0);
spend_request = paramBundle.getDouble("spend_request", 0);
asynchronous = paramBundle.getBoolean("asynchronous", false);
long calFlagTime = (long)paramBundle.getLong("calFlagTime", 0);
long hypoFlagTime = (long)paramBundle.getLong("hypoFlagTime", 0);
boolean exercise = paramBundle.getBoolean("currentlyExercising", false);
DIAS_STATE = paramBundle.getInt("DIAS_STATE", DIAS_STATE_OPEN_LOOP);
```

When the SSMservice completes normal processing of the deliver bolus command it sends the status SAFETY_SERVICE_STATE_NORMAL to DiAsService along with additional data:

```
response = Message.obtain(null, SAFETY_SERVICE_STATE_NORMAL, 0, 0);
responseBundle = new Bundle();
responseBundle.putInt("stoplight", hypolight);
responseBundle.putInt("stoplight2", hyperlight);
responseBundle.putBoolean("isMealBolus", isMealBolus);     // true if this is a meal bolus
responseBundle.putDouble("brakes_coeff", brakes_coeff);    // Value between 0 and 1
                                                                                // 0: brakes on, no basal
                                                                                // 1: brakes off, full basal
responseBundle.putDouble("IOB", IOB);                      // Latest IOB estimate
response.setData(responseBundle);
mMessengerToClient.send(response);
```

12. SAFETY_SERVICE_CMD_CALCULATE_STATE

DiAsService sends this command to direct the SSMservice to calculate the state estimate based on CGM, insulin and possibly other data from the database. The state estimate is written into the 'stateestimate' database table.

```
// Command inputs
enableIOtest = (boolean)paramBundle.getBoolean("enableIOtest");
simulatedTime = paramBundle.getLong("simulatedTime", -1);
bolus_meal =  0.0;
bolus_correction =  0.0;
bolusRequested = 0.0;
differential_basal_rate = 0.0;
credit_request = 0.0;
spend_request = 0.0;
asynchronous = false;
calFlagTime = (long)paramBundle.getLong("calFlagTime", 0);
hypoFlagTime = (long)paramBundle.getLong("hypoFlagTime", 0);
exercise = paramBundle.getBoolean("currentlyExercising", false);
DIAS_STATE = paramBundle.getInt("DIAS_STATE", DIAS_STATE_OPEN_LOOP);
```

When the SSMservice completes normal processing of the calculate state estimate command it sends the status SAFETY_SERVICE_STATE_CALCULATE_RESPONSE to DiAsService:

```
response = Message.obtain(null, SAFETY_SERVICE_STATE_CALCULATE_RESPONSE, 0, 0);
```

```
mMessengerToClient.send(response);
```

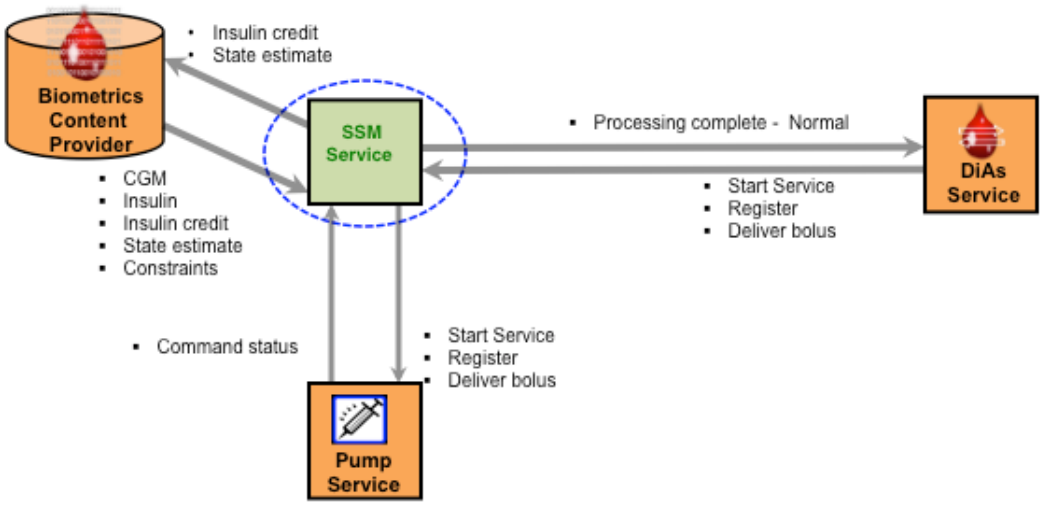


Table 51 – SSMservice Public API

13. SAFETY_SERVICE_CMD_REGISTER_CLIENT

    This command is used by the SSMservice to resolve the Messenger for status information replies to DiAsService:

```
Public Messenger mMessengerToClient = null;
mMessengerToClient = msg.replyTo;
```

14. SAFETY_SERVICE_CMD_START_SERVICE

    DiAsService sends this command to initialize the SSMservice and cause it to start and establish a connection with PumpService:

```
//  Command inputs
paramBundle = msg.getData();
boolean enableIOtest = (boolean)paramBundle.getBoolean(“ enableIOtest”);
int IOB_curve_duration_hours = paramBundle.getInt(“IOB_curve_duration_hours”);     // IOB curve duration (hours)
simulatedTime = paramBundle.getLong(“simulatedTime”, -1);                          // Contains the current Unix
                                                                                   // time stamp or -1 if clock

// Bind to the Insulin Pump Service
Intent intent = new Intent();
intent.setClassName(“edu.virginia.dtc.PumpService”, “edu.virginia.dtc.PumpService. PumpService”);
bindService(intent, mConnection, Context.BIND_AUTO_CREATE);
```

15. SAFETY_SERVICE_CMD_REQUEST_BOLUS

    DiAsService sends this command to direct the SSMservice to deliver insulin to the subject or to request or spend insulin “credit”.

x. Requesting insulin

The DiAs classifies all insulin delivered to the subject into three categories: basal, correction and meal. The SSMservice can operate in an asynchronous manner and must include a basal bolus based upon the daily basal profile whenever DiAsService makes an insulin request with the **asynchronous** flag set to **false**. The **bolusRequested** argument contains the total amount of insulin requested in units and may be zero if only basal insulin delivery is desired (in this case asynchronous=false).

The **differential_basal_rate** is an argument that permits APCservice to modulate the basal delivery rate with respect to the daily basal profile. It can be positive, negative or zero and is limited to the range -6U/hour to +6U/hour. This is added to the basal rate as determined by the SSMservice (which may be attenuated below the profile rate if there is a hypoglycemia risk).

xi. credit_request and spend_request

The **credit_request** and **spend_request** arguments are used if the SSMservice is managing an insulin "credit pool". In this case the DiAsService can request "credit" insulin from the SSMservice which can require that the user immediately pre-approve the amount of credit requested. If approved by the user the credit can later be "spent" by the APCservice or BRMservice without any further confirmation required by the Safety System. This mechanism may be used by the DiAsUI to announce a meal (issue a "**credit_request**") and for the APCservice/BRMservice to deliver insulin to treat the meal at the appropriate time (issue a "**spend_request**").

xii. DIAS_STATE and status flags

- **DIAS_STATE** is the current state of the DiAs system and may be Stopped, Pump, Closed Loop, Safety Only or Sensor Only.
- **calFlagTime** is the Unix time of the last CGM calibration
- **hypoFlagTime** is the Unix time of the last hypoglycemic episode reported by the user by pressing the Hypoglycemia Button on the Home Screen of DiAsUI.
- **exercise** is a Boolean reporting whether the DiAsUI Exercise Button indicates that the subject is currently exercising.

```
//  Command inputs
paramBundle = msg.getData();                                                    // Fetch parameter bundle
enableIOtest=paramBundle.getBoolean("enableIOtest", false);
simulatedTime = paramBundle.getLong("simulatedTime", -1);
bolusRequested =  paramBundle.getDouble("bolusRequested", 0);
bolusMeal =  paramBundle.getDouble("bolusMeal", 0);
bolusCorrection =  paramBundle.getDouble("bolusCorrection", 0);
double differential_basal_rate = paramBundle.getDouble("differential_basal_rate", 0);      // [-6U/hour:6U/hour]
credit_request = paramBundle.getDouble("credit_request", 0);
spend_request = paramBundle.getDouble("spend_request", 0);
asynchronous = paramBundle.getBoolean("asynchronous", false);
long calFlagTime = (long)paramBundle.getLong("calFlagTime", 0);
long hypoFlagTime = (long)paramBundle.getLong("hypoFlagTime", 0);
boolean exercise = paramBundle.getBoolean("currentlyExercising", false);
DIAS_STATE = paramBundle.getInt("DIAS_STATE", DIAS_STATE_OPEN_LOOP);
```

When the SSMservice completes normal processing of the deliver bolus command it sends the status SAFETY_SERVICE_STATE_NORMAL to DiAsService along with additional data:

```
response = Message.obtain(null, SAFETY_SERVICE_STATE_NORMAL, 0, 0);
responseBundle = new Bundle();
responseBundle.putInt("stoplight", hypolight);
responseBundle.putInt("stoplight2", hyperlight);
responseBundle.putBoolean("isMealBolus", isMealBolus);     // true if this is a meal bolus
responseBundle.putDouble("brakes_coeff", brakes_coeff);    // Value between 0 and 1
                                                                    // 0: brakes on, no basal
                                                                    // 1: brakes off, full basal
responseBundle.putDouble("IOB", IOB);                      // Latest IOB estimate
response.setData(responseBundle);
mMessengerToClient.send(response);
```

16. SAFETY_SERVICE_CMD_CALCULATE_STATE

DiAsService sends this command to direct the SSMservice to calculate the state estimate based on CGM, insulin and possibly other data from the database. The state estimate is written into the 'stateestimate' database table.

```
// Command inputs
enableIOtest = (boolean)paramBundle.getBoolean("enableIOtest");
simulatedTime = paramBundle.getLong("simulatedTime", -1);
bolus_meal =  0.0;
bolus_correction =  0.0;
bolusRequested = 0.0;
differential_basal_rate = 0.0;
credit_request = 0.0;
spend_request = 0.0;
asynchronous = false;
calFlagTime = (long)paramBundle.getLong("calFlagTime", 0);
hypoFlagTime = (long)paramBundle.getLong("hypoFlagTime", 0);
exercise = paramBundle.getBoolean("currentlyExercising", false);
DIAS_STATE = paramBundle.getInt("DIAS_STATE", DIAS_STATE_OPEN_LOOP);
```

When the SSMservice completes normal processing of the calculate state estimate command it sends the status SAFETY_SERVICE_STATE_CALCULATE_RESPONSE to DiAsService:

```
response = Message.obtain(null, SAFETY_SERVICE_STATE_CALCULATE_RESPONSE, 0, 0);
mMessengerToClient.send(response);
```

Table 52 – Pump Service Public API

PumpService messages from SSMservice

1. PUMP_SERVICE_CMD_REGISTER_CLIENT

SSMservice uses this command to send a reference to its Messenger for status information replies from the Pump Service:

```
// Send a register-client message to the service with the client message handler in replyTo
Message msg = Message.obtain(null, PUMP_SERVICE_CMD_REGISTER_CLIENT, 0, 0);
mMessengerFromService  = msg.replyTo;
```

2. PUMP_SERVICE_CMD_START_SERVICE

   Initialize PumpService.

3. PUMP_SERVICE_CMD_DELIVER_BOLUS

   SSMservice uses this command to send a bolus command to the Pump Service.

```
lastBolusSimulatedTime = paramBundle.getLong("lastBolusSimulatedTime", getCurrentTimeSeconds());
asynchronous = paramBundle.getBoolean("asynchronous", false);
double pre_authorized = paramBundle.getDouble("pre_authorized", 0.0);
bolus_max = paramBundle.getDouble("bolus_max", -1.0);
double basal_bolus = paramBundle.getDouble(INSULIN_BASAL_BOLUS, 0.0);
double meal_bolus = paramBundle.getDouble(INSULIN_MEAL_BOLUS, 0.0);
double corr_bolus = paramBundle.getDouble(INSULIN_CORR_BOLUS, 0.0);
Tvector tvectorBasal = getTvector(paramBundle, "Basaltimes", "Basalvalues");
```

4. PUMP_SERVICE_CMD_STOP_SERVICE

   Stop the service.

```
stopSelf();
```

5. PUMP_SERVICE_CMD_SET_HYPO_TIME

   Get the time of the last hypoglycemia reported by APMMP.

```
paramBundle = msg.getData();
long hypoFlagTime = (long)paramBundle.getLong("hypoFlagTime", 0);
```

SSMservice messages from PumpService

6. PUMP_STATE_COMMAND_COMPLETE

   Report to SSMservice that the pump command has been processed

```
Message response = Message.obtain(null, PUMP_STATE_COMMAND_COMPLETE, 0, 0);
mMessengerFromService.send(response);
```

IV.D. BRMservice

BRMservice interfaces to DiAs Service (see Table 53 - BRMservice Public API) and biometricsContentProvider. Examples of the use of this API can be seen in the **BRMserviceShell** sample application.

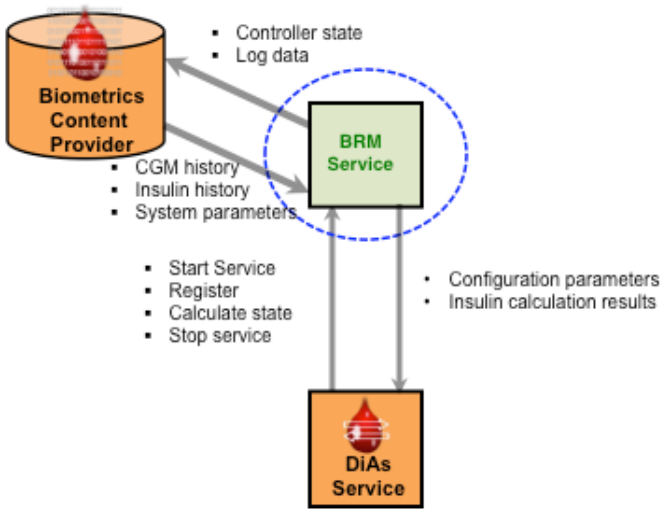


Table 53 - BRMservice Public API

1. APC_SERVICE_CMD_REGISTER_CLIENT

   Note: The commands from DiAsService to APCservice and BRMservice use the prefix APC_ when communicating with either controller.

   This command is used by the BRMservice to resolve the Messenger for status information replies:

```
Public Messenger mMessengerToClient = null;
mMessengerToClient = msg.replyTo;
```

2. APC_SERVICE_CMD_START_SERVICE

   DiAsService sends this command to initialize BRMservice with current system parameters that were entered by the user into the DiAs Setup Subject Information and profile screens. The simulatedTime parameter is used as a way to be able to run BRMservice in real time (as in a clinical trial) or in simulated time (for rapid testing). If the simulatedTime is >0 then it contains the time that should be used in all calculations. If it is <0 then BRMservice should use the current time as read from an Android system call.

```
//  Command inputs
paramBundle = msg.getData();
enableIOtest = (boolean)paramBundle.getBoolean("enableIOtest");
double TDI = (double)paramBundle.getDouble("TDI");                          // Get total daily insulin (U)
int IOB_curve_duration_hours = paramBundle.getInt("IOB_curve_duration_hours"); // IOB curve duration (hours)
simulatedTime = paramBundle.getLong("simulatedTime", -1);                   // Contains the current Unix
                                                                            // time stamp or -1 if clock
                                                                            // time should be used
```

   The BRMservice response is used to report its characteristics and capabilities to DiAsService.

```
// Command response
response = Message.obtain(null, APC_CONFIGURATION_PARAMETERS, 0, 0);          // Identify type of response
responseBundle = new Bundle();                                               // Make a bundle object
responseBundle.putInt("Timer_Ticks_Per_Control_Tick", Timer_Ticks_Per_Control_Tick); // How many heartbeat ticks
                                                                             // per control tick
response.setData(responseBundle);                                            // send the response
mMessengerToClient.send(response);                                           // to DiAsService
```

3. APC_SERVICE_CMD_STOP_SERVICE

   Stop the BRMservice service:

```
stopSelf();                                                    // Stop this service
```

4. APC_SERVICE_CMD_CALCULATE_STATE

   Calculate and report the recommended bolus, meal insulin credit and spend requests and modulation of the basal rate. In order to calculate the appropriate insulin therapy the BRMservice will read CGM, insulin, meal and SMBG history from the Biometrics Content Provider and may store state information to the Biometrics Content Provider "hmsstateestimate", "user1" and "user2" tables.

```
// Command inputs
paramBundle = msg.getData();                                   // Fetch parameter bundle
enableIOtest = (boolean)paramBundle.getBoolean("enableIOtest");
simulatedTime = paramBundle.getLong("simulatedTime", -1);      // Contains Unix time stamp or -1
boolean asynch = paramBundle.getInt("asynchronous", false);    //Asynchronous calculation flag
long corrFlagTime = (long)paramBundle.getLong("corrFlagTime", 0);   // Most recent correction bolus time
long hypoFlagTime = (long)paramBundle.getLong("hypoFlagTime", 0);   // Most recent hypo time
long calFlagTime = (long)paramBundle.getLong("calFlagTime", 0);     // Most recent calibration time
long mealFlagTime = (long)paramBundle.getLong("mealFlagTime", 0);   // Most recent meal flag time
                                                               // if using clock time
DIAS_STATE = paramBundle.getInt("DIAS_STATE", 0);              // Most recent DiAs operating state
tick_modulus = paramBundle.getInt("tick_modulus", 0);      // Timer_Ticks % Timer_Ticks_Per_Control_Tick

// Command response
response = Message.obtain(null, APC_PROCESSING_STATE_NORMAL, 0, 0);  // Set type of response
responseBundle = new Bundle();                                 // Create response bundle object
responseBundle.putBoolean("doesBolus", doesBolus);
responseBundle.putBoolean("doesRate", doesRate);
responseBundle.putBoolean("doesCredit", doesCredit);
responseBundle.putDouble("recommended_bolus", recommended_bolus);   // Bolus to deliver
responseBundle.putDouble("creditRequest", 0.0);                // Insulin credit request
responseBundle.putDouble("spendRequest", 0.0);                 // Insulin spend request
responseBundle.putBoolean("new_differential_rate", true);      // Indicates if a new rate has been set
responseBundle.putDouble("differential_basal_rate", differential_basal_rate);   // Modulation to basal therapy
responseBundle.putInt("stoplight", hypolight);                  // hypo traffic light value
responseBundle.putInt("stoplight2", hyperlight);                // hyper traffic light value
responseBundle.putDouble("IOB", IOB_estimate);                 // Estimated IOB
responseBundle.putBoolean("extendedBolus", false);             //Not currently used
responseBundle.putBoolean("asynchronous", asynch);            //Asynchronous calc. flag
responseBundle.putDouble("extendedBolusMealInsulin", 0.0);     //Not currently used
responseBundle.putDouble("extendedBolusCorrInsulin", 0.0);     //Not currently used
response.setData(responseBundle);                               // send response to
mMessengerToClient.send(response);                              // DiAsService
```

IV.E. APCservice

APCservice interfaces to DiAs Service (see Table 54 – APCservice Public API) and biometricsContentProvider. Examples of the use of this API can be seen in the **APCserviceShell** sample application.

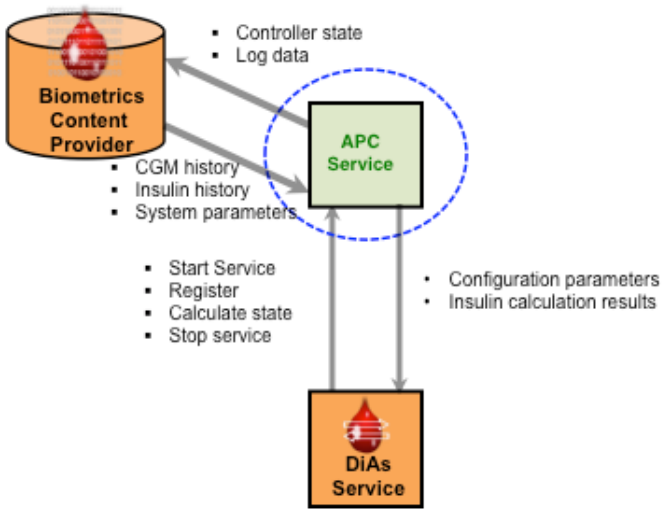


Table 54 – APCservice Public API

1. APC_SERVICE_CMD_REGISTER_CLIENT

This command is used by the APCservice to resolve the Messenger for status information replies:

```
Public Messenger mMessengerToClient = null;
mMessengerToClient = msg.replyTo;
```

2. APC_SERVICE_CMD_START_SERVICE

DiAsService sends this command to initialize APCservice with current system parameters that were entered by the user into the DiAs Setup Subject Information and profile screens. The simulatedTime parameter is used as a way to be able to run APCservice in real time (as in a clinical trial) or in simulated time (for rapid testing). If the simulatedTime is >0 then it contains the time that should be used in all calculations. If it is <0 then APCservice should use the current time as read from an Android system call.

```
//  Command inputs
paramBundle = msg.getData();
enableIOtest = (boolean)paramBundle.getBoolean("enableIOtest");
double TDI = (double)paramBundle.getDouble("TDI");                          // Get total daily insulin (U)
int IOB_curve_duration_hours = paramBundle.getInt("IOB_curve_duration_hours");  // IOB curve duration (hours)
simulatedTime = paramBundle.getLong("simulatedTime", -1);                   // Contains the current Unix
                                                                            // time stamp or -1 if clock
                                                                            // time should be used
```

The APCservice response is used to report its characteristics and capabilities to DiAsService.

```
// Command response
response = Message.obtain(null, APC_CONFIGURATION_PARAMETERS, 0, 0);          // Identify type of response
responseBundle = new Bundle();                                               // Make a bundle object
responseBundle.putInt("Timer_Ticks_Per_Control_Tick", Timer_Ticks_Per_Control_Tick);  // How many heartbeat ticks
                                                                             // per control tick
response.setData(responseBundle);                                            // send the response
mMessengerToClient.send(response);                                           // to DiAsService
```

3. APC_SERVICE_CMD_STOP_SERVICE

Stop the APCservice service:

```
stopSelf();                                                    // Stop this service
```

4. APC_SERVICE_CMD_CALCULATE_STATE

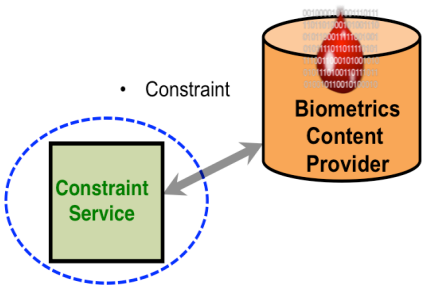
Calculate and report the recommended bolus, meal insulin credit and spend requests and modulation of the basal rate. In order to calculate the appropriate insulin therapy the APCservice will read CGM, insulin, meal and SMBG history from the Biometrics Content Provider and may store state information to the Biometrics Content Provider "hmsstateestimate", "user1" and "user2" tables.

```
// Command inputs
paramBundle = msg.getData();                                   // Fetch parameter bundle
enableIOtest = (boolean)paramBundle.getBoolean("enableIOtest");
simulatedTime = paramBundle.getLong("simulatedTime", -1);      // Contains Unix time stamp or -1
boolean asynch = paramBundle.getInt("asynchronous", false);    //Asynchronous calculation flag
long corrFlagTime = (long)paramBundle.getLong("corrFlagTime", 0);   // Most recent correction bolus time
long hypoFlagTime = (long)paramBundle.getLong("hypoFlagTime", 0);   // Most recent hypo time
long calFlagTime = (long)paramBundle.getLong("calFlagTime", 0);     // Most recent calibration time
long mealFlagTime = (long)paramBundle.getLong("mealFlagTime", 0);   // Most recent meal flag time
                                                               // if using clock time
DIAS_STATE = paramBundle.getInt("DIAS_STATE", 0);              // Most recent DiAs operating state
tick_modulus = paramBundle.getInt("tick_modulus", 0);      // Timer_Ticks % Timer_Ticks_Per_Control_Tick

// Command response
response = Message.obtain(null, APC_PROCESSING_STATE_NORMAL, 0, 0);  // Set type of response
responseBundle = new Bundle();                                 // Create response bundle object
responseBundle.putBoolean("doesBolus", doesBolus);
responseBundle.putBoolean("doesRate", doesRate);
responseBundle.putBoolean("doesCredit", doesCredit);
responseBundle.putDouble("recommended_bolus", recommended_bolus);   // Bolus to deliver
responseBundle.putDouble("creditRequest", 0.0);                // Insulin credit request
responseBundle.putDouble("spendRequest", 0.0);                 // Insulin spend request
responseBundle.putBoolean("new_differential_rate", true);      // Indicates if a new rate has been set
responseBundle.putDouble("differential_basal_rate", differential_basal_rate);   // Modulation to basal therapy
responseBundle.putInt("stoplight", hypolight);                  // hypo traffic light value
responseBundle.putInt("stoplight2", hyperlight);                // hyper traffic light value
responseBundle.putDouble("IOB", IOB_estimate);                 // Estimated IOB
responseBundle.putBoolean("extendedBolus", false);             //Not currently used
responseBundle.putBoolean("asynchronous", asynch);            //Asynchronous calc. flag
responseBundle.putDouble("extendedBolusMealInsulin", 0.0);    //Not currently used
responseBundle.putDouble("extendedBolusCorrInsulin", 0.0);    //Not currently used
response.setData(responseBundle);                              // send response to
mMessengerToClient.send(response);                             // DiAsService
```

IV.F. ConstraintService

The ConstraintService supports a public interface to the biometricsContentProvider. It reads data from the database, establishes constraints and writes them to the CONSTRAINT_TABLE. It may use tables in the database as a way to communicate with SSMservice, however details of such communication are to be determined by the developer. Examples of the use of such communication mechanisms can be seen in the **ConstraintService** sample application.



IV.G. MCMservice

MCMservice communicates with MealActivity to calculate and deliver boluses requested by the user. It also communicates with DiAsService to pass boluses initiated by MealActivity to DiAsService for delivery to SSMservice (see Table 55 - MCMservice Public API).

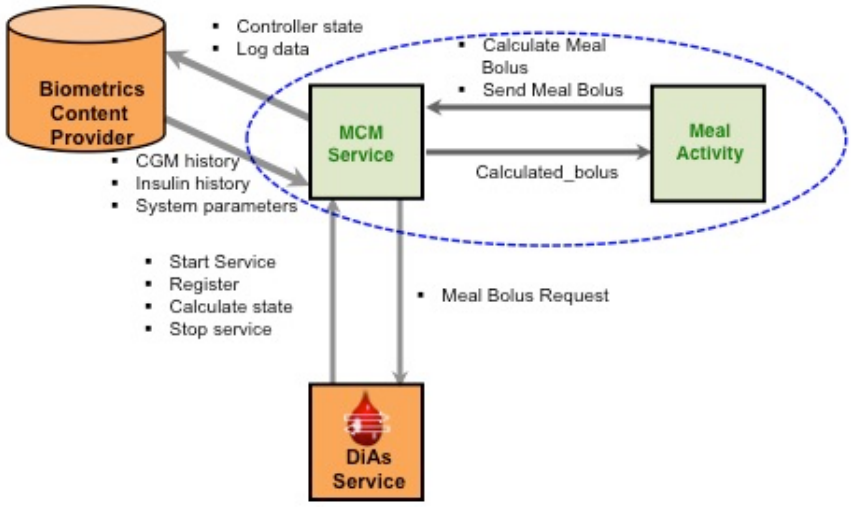


Figure 31 – MCMservice API

Table 55 - MCMservice Public API

Meal Screen to MCMservice Messages
Meal.UI_REGISTER Registers the activity for entering data
Meal.UI_CHANGE Notifies the MCM of changes in the UI
Meal.UI_CLOSED Notifies the MCM when UI closes
Meal.INJECT Message sent from UI to MCM when a bolus is requested to be injected

Meal.MCM_CALCULATED Message to notify the activity of when the MCM has changed output values
DiAsService to MCMservice Messages
Meal.REGISTER Registers the MCM with DiAs Service
Meal.SSM_CALC_DONE Message used to notify the MCM that the SSM is done updating IOB
Meal.UI_STARTED Message to DiAs Service from the MCM to indicate the activity for meal entry has been started
Meal.UI_CLOSED Message to DiAs Service from the MCM to indicate the user has closed the activity
Meal.MCM_BOLUS Message from the MCM with bolus information (meal and correction size) • Double "meal" • Double "correction"

IV.H. I/O Verification of Developer Applications

Permission to refer to the APMMP Master File for use of the APMMP and APSDK requires that any new developer applications be submitted to the UVA Center for Diabetes Technology for I/O Verification. The UVA Center for Diabetes Technology staff will run the I/O Verification Procedure for each developer module to make sure that it is proerly receiving and sending data to the rest of the DiAs system. **This procedure does not take the place of validation and verification testing**. The Center for Diabetes technology makes no claim that the application developed by the investigator will perfom properly or is suitable for any purpose. Rather the Center for Diabetes Technology will check that the module passes I/O tests which are necessary, but not sufficient, conditions for the application to work properly within the APMMP.

IV.H.1. Compliance with APMMP Public Interfaces

Permission to refer to the APMMP Master File for use of the APMMP and APSDK requires that the application respect the APMMP Public Application Programming Interfaces. Applications which do not respect the software interfaces are which attempt to use interfaces to DiAs modules not documented in the APSDK are not permitted to refer to the APMMP Master File.

IV.H.2. APMMP DMR Reference Permission

Investigators wishing to refer to the APMMP Master file should request a letter of permission from:

Dr. Patrick Keith-Hynes
Center for Diabetes Technology
University of Virginia
617 West Main Street, Fourth Floor
Charlottesville, VA 22902

IV.I. Verification Process

IV.I.1. Environment and Tools

Verification testing is performed in a lab environment using the following tools:

i. CPMP
ii. Personal computer running Eclipse with ADT plugin
iii. Peripheral devices
   - Tandem Pump
   - Dexcom Share AP
   - BTLE G4 Relay Box
   - Dexcom G4 receiver
   - Roche Pump
   - Zephyr HxM
   - Zephyr Bioharness
iv. Remote monitoring server supporting SSL with the **DiAs Web Monitoring** software installed.

IV.I.2. I/O Verification

IO verification is a semi-automated process. The "template" (or "shell") versions of the Developer Applications (SSMserviceShell, APCserviceShell, BRMserviceShell and ConstraintServiceShell) contain Event logging code which executes conditionally if the boolean parameter **enableIO** is set **true**. This flag is read by applications and used to determine whether IO test logging should be generated. The Event data stored when this boolean is enabled may be used to verify the commands, parameters and return values sent and received by the Developer Applications and the modules with which they communicate. Events associated with certain infrequent communication activities such as service registration and startup are always logged. The **enableIO** boolean is used only during bench testing in the lab – never during a clinical trial.

V. DEFINITIONS

Android Medical Operating System (AMOS) – a build of the Android mobile operating system modified to make it suitable for use in a medical device.

Cell Phone Medical Platform (CPMP) - An approved Android cell phone with AMOS installed.

Diabetes Assistant (DiAs) – a suite of Android applications operating in a coordinated fashion to create a research platform for outpatient testing of Artificial Pancreas (Closed Loop) blood glucose control, Safety System and CGM sensor-only monitoring.

Artificial Pancreas Mobile Medical Platform (APMMP) - a modular software stack consisting of the Android Medical Operating System (AMOS) and the Diabetes Assistant (DiAs).