


MIR SUPER SDK - Version 1.0.0

Instruction For Use

History

Rev.	Date	Author	Change/reason
00	2023.12.06	Arman Sarybayev	Initial version
01	2024.02.09	Arman Sarybayev	<p>MDR_IFU_MIR SUPER SDK_PC_rev00 was renamed as MDR_IFU_MIR SUPER SDK_rev01</p> <p>Clause 1.1 - updated Device Name, Model, and Version</p> <p>New clause - 7.1 Lifetime was added.</p> <p>Paragraph 2 - was edited, new cybersecurity instructions added</p> <p>Paragraph 4, 5 - were edited and reference to Annexes done</p> <p>Chapter 8.4 -Label and Symbols was created</p> <p>Chapter 7.2 technical specifications was edited, requirements for API were added</p> <p>Chapter 7.3 - Performance Characteristics updated</p> <p>Following Annexes were created:</p> <ul style="list-style-type: none"> • Annex A – MIR SSDK iOS • Annex B – MIR SSDK Windows • Annex C – MIR SSDK Web (Cloud/API) • Annex D – MIR SSDK MacOS • Annex E – MIR SSDK Android <p>Information on e-IFU was added</p>

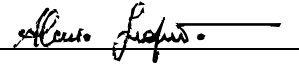
Revision Trial

	Name:	Date:	Signature:
Prepared	Arman Sarybayev	2024.02.09	

Approved

Alessio Segreto

2024.02.09



Manufacturer's address:

<p>MIR Medical International Research S.p.A. Viale Luigi Schiavonetti 270 00173 Rome (ITALY) Tel + 39 0622754777 Fax + 39 0622754785 Web site: www.spirometry.com Email: mir@spirometry.com</p>	<p>MIR USA, Inc. 5462 S. Westridge Drive New Berlin, WI 53151 - USA Tel + 1 (262) 565 – 6797 Fax + 1 (262) 364 – 2030 Web site: www.spirometry.com Email: mirusa@spirometry.com</p>
--	---

MIR has a policy of continuous product development and improvement. MIR reserves the right to modify and update the information in this User's Manual as deemed necessary. Any suggestions and or comments regarding this product are appreciated and may be sent via email to: mir@spirometry.com.

MIR accepts no responsibility for any loss or damage caused by the user of the device due to not follow of instructions contained in this Manual.

Please note that due to printing limitations, the screenshots shown in this manual may differ from the display of the machine and/or from the keyboard icons.

Copying this manual in whole or in part is strictly forbidden.

Electronic eIFU can be accessed through the following web page: <https://ssdk.spirometry.com/1.0/ifu-labels#>

Notice

You must report any serious incidents occurring in relation to the device to the manufacturer and the competent authority of the Member State where the user and/or patient is established, in accordance with Regulation 2017/745.



Contents

Contents	4
1. General Information	8
1.1. Device Name, Model, and Version.....	8
1.2. Purpose and Intended Use	8
1.3. Intended Users	8
1.4. Intended population.....	8
2. Safety Precautions and Warnings	9
3. Contraindications	9
4. Getting Started	11
Activation and Licensing	11
5. SW Integration	12
5.1. Correct Use of Features and Functions	12
5.2. Interpretation and Analysis of Output or Results	12
6. Maintenance	12
6.1. Routine Maintenance	12
6.2. Troubleshooting.....	13
7. Technical Specifications and Performance	13
7.1. Lifetime	13
7.2. Technical Specifications.....	13
7.3. Performance Characteristics	16
7.4. Validation and Testing Information.....	16
8. Legal and Regulatory Information	16
8.1. Compliance Statements	16
8.2. EU Declaration of Conformity.....	16
8.3. Post-Market Surveillance and Vigilance Procedures	17
8.4. Label and Symbols.....	18
8.4.1. Identification label and symbols	18

8.4.2. Used symbols	18
9. Manufacturer's Responsibility and Liability	18
9.1. Manufacturer's Commitment to Product Quality	19
9.2. Warranty Information	19
9.3. Limitations of Liability	19
9.4. Indemnification	19
9.5. Regulatory Compliance	19
9.6. Conclusion	19
9.7. List of MIR approved MD class I accessories	19
Annex A. Instruction for Use - SSDK iOS	20
Introduction	20
Prerequisites	20
Integration (How to implement?)	20
MIR SMART DEVICES	20
The SVC Test Guide	48
The FVC Plus Test Guide	53
SAMPLE DEMO APPLICATION	61
Additional Resources	65
Troubleshooting	66
Annex B. Instruction for Use - MIR SUPER SDK Windows	67
Introduction	67
Prerequisites	67
Integration (How to implement?)	67
Sample Demo Application	67
Integration into a Microsoft Visual Studio project	71
Main features	77
Downloads the archive from a MIR device via USB or BLE (Bluetooth Low Energy)	78
Conduct the FVC test, generate the graph during the test and receive the results	81
Conduct the VC test, generate the graph during the test and receive the results	84

Load the tests from the .mir or .mirx file	85
Generate the .mirx file downloading the data (not interpreted) by the device	86
Perform the Firmware upgrade of the MIR device	88
Additional Resources	89
Troubleshooting	89
Annex C. Instruction for Use - MIR SUPER SDK WEB (CLOUD/API)	91
Introduction.....	91
Vocabulary	91
Authentication.....	92
Principles	92
Credentials	92
Permissions	92
Scopes	92
How to get a Bearer Token	92
Rate limiting.....	95
Pagination	95
Errors.....	95
Content-Type.....	96
Routes	97
Data Types	97
Imports	97
Convert	97
Interpretation	99
Oximeter Analysis	99
Predicted Values	99
Print	100
Troubleshooting	101
Annex D. Instruction for Use - MIR SUPER SDK MacOS102	
Introduction.....	102

Prerequisites	102
How to implement	103
Integration into a XCode project	103
Main features.....	107
Connecting to a device	107
Conduct the FVC test, generate the graph during the test and receive the results	109
Conduct the VC test, generate the graph during the test and receive the results	110
Retrieving an archive and generating the .mirx file	111
Perform the Firmware upgrade of the MIR device	112
Additional Resources	113
Annex E. Instruction for Use - SSDK Android	114
Introduction.....	114
Prerequisites	114
Integration (How to implement?).....	114
Spirobank Smart SDK Android Guide	114
Sample Demo Application	127
Additional Resources	129
Troubleshooting	130

1. General Information

1.1. Device Name, Model, and Version

The MIR SUPER SDK - Spirometry and Oximetry Software Development Kit (SDK) is a medical device designed to provide comprehensive spirometry and oximetry functionalities. It is intended for use by healthcare professionals and developers in the medical field.

Ref	Description	Version / configuration	UDI_DI Packaging levels: 1
920230	MIR SUPER SDK	1.0.0	8052990322091

1.2. Purpose and Intended Use

The intended purpose of the MIR SUPER SDK is to serve as a black box software within a larger SUPER SDK Accessories. Its main function is to receive raw medical data from external sources, such as medical devices or other software applications, and process and analyze this data to generate treated medical information.

The MIR SUPER SDK is specifically designed to provide accurate and reliable results by applying advanced algorithms and methodologies. It performs data cleansing, signal processing, configuration of associated devices and interpretation to transform raw data into meaningful medical insights. The treated medical data includes parameters such as lung function measurements and oximetry measurements.

1.3. Intended Users

The intended users of the MIR SUPER SDK are software developers, system integrators, and other professionals involved in the development and integration of software applications in the healthcare domain. These individuals and organizations may include:

Medical software developers: Developers who are creating software applications for medical purposes, such as electronic health records (EHRs), telemedicine platforms, clinical decision support systems, or research tools.

System integrators: Professionals responsible for integrating various software components and systems within a healthcare environment, including hospitals, clinics, or research institutions.

Healthcare IT professionals: IT personnel working in healthcare settings who are responsible for implementing, configuring, and maintaining software solutions.

Medical device manufacturers: Companies involved in the production of medical devices that may require integration with software applications to enhance their functionality and data analysis capabilities.

Research institutions: Academic institutions or organizations engaged in medical research and data analysis that require reliable and accurate medical data for their studies.

It is important for the intended users to have a strong understanding of medical software development, data processing, and compliance with regulatory requirements. They should also have the necessary technical skills to integrate the MIR SUPER SDK into their software applications effectively.

1.4. Intended population

MIR SUPER SDK indirectly benefits patients and healthcare providers by enabling the delivery of accurate and treated medical data, its primary focus is on facilitating the development and integration of software solutions rather than

targeting a specific population. Also, intended population depends on Non-medical SW and spirometry, oximetry tests performed by patients.

2. Safety Precautions and Warnings

- Only qualified and trained software developers and healthcare professionals should have access to / and use the MIR SUPER SDK.
- Before utilizing the MIR SUPER SDK, it is imperative to read and fully understand Instructions for Use (IFU) to ensure proper software handling and usage.
- Unauthorized modification or alteration of the MIR SUPER SDK is strictly prohibited to maintain its intended performance and safety.
- Ensure that any connected medical devices and software applications adhere to relevant safety standards and regulations.
- For patients whose ability to perform spirometry or oximetry maneuvers is compromised or limited.
- In cases where patients cannot cooperate or follow the instructions required for proper use – do not use it.
- In environments where the specified operating conditions of the MIR SUPER SDK cannot be met – do not use it.
- The MIR SUPER SDK is exclusively intended for the measurement of lung function parameters and peripheral oxygen saturation. Any other uses are not recommended.
- Use only MIR-approved accessories and compatible software components to ensure accurate and reliable results (those are mentioned in Paragraph 9 of this IFU).
- Regularly inspect the MIR SUPER SDK software for any signs of issues or abnormalities. If any problems are detected, refrain from using the software and contact the manufacturer's technical support for assistance.
- In case the MIR SUPER SDK does not perform as expected, cease using it and seek technical support from the manufacturer.
- Cybersecurity: Access and use the MIR SUPER SDK software only from secure and authorized devices to prevent unauthorized access and potential security breaches.
- Cybersecurity: Avoid sharing the software or any associated authentication credentials with unauthorized individuals.
- Cybersecurity: Avoid using MIR SSDK on unknown and public Networks, at minimum those networks must be protected with WPA2 password protection protocol.
- Cybersecurity: Regularly update the software to the latest version provided by the manufacturer, including any security patches or enhancements.
- Cybersecurity - Prohibition of Abuse: The user agrees not to engage in brute force activities, attempt to bypass security measures, or use the API for purposes other than those explicitly authorized by MIR.
- GDPR: MIR SUPER SDK does not use and proceed any personal user information.

3. Contraindications

The MIR SUPER SDK, does not have direct contraindications for use. It is designed to assist in the development of software applications for telespirometry and respiratory data analysis.

However, it's important to note that the software applications developed using the MIR SUPER SDK may have specific contraindications depending on their intended use and the medical context in which they are deployed. These

contraindications would be determined by the software application developer or healthcare provider and may vary depending on factors such as the patient population, medical condition, and specific clinical requirements.

Typically, contraindications for the use of software applications incorporating the MIR SUPER SDK would align with contraindications for the underlying medical procedures or interventions being performed. For example, if the software application is intended for use in patients with specific respiratory conditions, the contraindications may mirror the contraindications associated with those conditions or with traditional spirometry testing.

It is essential for the software application developer and healthcare provider to clearly define and communicate the contraindications associated with the specific software application, ensuring that it is used appropriately and in accordance with relevant medical guidelines and regulations.

In case of MIR SUPER SDK Accessories:

MIR SUPER SDK ACCESSORIES, similar to spirometry, is a lung function testing SW, that measures the amount of air a person can breathe in and out and how quickly they can exhale. As with spirometry, a detailed clinical history and other tests suggested by a doctor should be considered to make an accurate diagnosis, and the test results, interpretation, and treatment suggestions should be provided by a doctor.

It is important that the patient fully cooperates during the test to ensure accuracy. MIR SUPER SDK ACCESSORIES also has relative contraindications, which could compromise the accuracy of the results or put the patient at risk. Therefore, it is necessary to follow guidelines and recommendations regarding when and how to perform telespirometry to minimize risks and obtain accurate results.

It is worth noting that telemedicine has expanded the possibilities of lung function testing and made it more accessible to people who cannot attend a healthcare facility. With MIR SUPER SDK ACCESSORIES, the patient can perform the test at home, and the results can be sent to a healthcare provider for interpretation and diagnosis. However, it is crucial to ensure that the patient understands the test's requirements and procedures and to provide appropriate guidance throughout the process to obtain reliable results.

Spirometry has relative contraindications, as reported in the 2019 update of the ATS/ERS guideline:

Due to increased myocardial demand or changes in blood pressure

- Acute myocardial infarction within 1 week
- Systemic hypotension or severe hypertension
- Significant atrial/ventricular arrhythmia
- Uncompensated heart failure
- Uncontrolled pulmonary hypertension
- Acute pulmonary heart
- Clinically unstable pulmonary embolism
- History of syncope related to forced expiration/cough

Due to increased intracranial/intraocular pressure

- Cerebral aneurysm
- Brain surgery within 4 weeks

- Recent concussion with persistent symptoms

- Eye surgery within 1 week

Due to increased sinus and middle ear pressure

- Sinus or middle ear surgery or infection within 1 week

Due to increased intrathoracic and intraabdominal pressure

- Presence of pneumothorax

- Thoracic surgery within 4 weeks

- Abdominal surgery within 4 weeks

- Pregnancy beyond term

Due to infection control problems

- Active or suspected transmissible respiratory or systemic infection, including tuberculosis

- Physical conditions predisposing to transmission of infection, such as haemoptysis, significant secretions or oral lesions or oral bleeding.

It is important to note that spirometry testing is generally safe and complications are rare. However, potential risks should always be weighed against the potential benefits of the test, and the test should be performed according to established guidelines and recommendations.

4. Getting Started

To get started, please refer to the corresponding Annexes of each platform:

- Annex A – iOS
- Annex B – Windows
- Annex C – Web (Cloud/API)
- Annex D – MacOS
- Annex E - Android

Activation and Licensing

API Integration

The API requires having a developer account to obtain login credentials. To create an account, the integrator must contact our technical team. They will provide the integration with the necessary information to communicate with the API.

The API operates on a 'Pay As You Go' model. Each API request incurs a specific cost. At the end of each month, a weekly report is generated to bill the integrator only for what he has consumed.

Other integration

The use of SDKs is subject to a commercial offer and acceptance of a Non-Disclosure Agreement (NDA).

The integrator should reach out to our sales team to request a quote for obtaining the SDK. Once the commercial offer is accepted, the integration will gain access to our developer platform. On this platform, they can review and accept the NDA and proceed to download the SDK(s) they require.

5. SW Integration

For integration please refer to the corresponding Annexes:

- Annex A – iOS
- Annex B – Windows
- Annex C –Web (Cloud/API)
- Annex D – MacOS
- Annex E - Android

5.1. Correct Use of Features and Functions

To utilize the features and functions of the MIR SUPER SDK correctly, follow these guidelines:

- Familiarize yourself with the SDK's user documentation, including this user manual and its annexes.
- Ensure that you have the necessary permissions and authorizations to access and utilize specific features or functions of the SDK.
- Follow the recommended guidelines and best practices provided in the documentation to optimize the performance and accuracy of the SDK.
- Use the provided APIs and interfaces to interact with the SDK programmatically, adhering to the defined syntax and conventions.

5.2. Interpretation and Analysis of Output or Results

The MIR SUPER SDK generates output or results based on the performed spirometry and oximetry measurements. To interpret and analyze these results accurately, consider the following:

- Consult the Annexes for detailed information on the parameters and measurements provided by the SDK.
- Understand the clinical significance and relevance of the specific measurements or parameters in the context of the intended use and clinical condition.
- Compare the generated results with established reference values or guidelines provided by reputable medical associations or regulatory bodies.
- Exercise professional judgment and clinical expertise when interpreting the output or results, taking into account other relevant patient data and clinical observations.

6. Maintenance

6.1. Routine Maintenance

Application SDKs

For each new version of an SDK, our developer portal sends a notification to various integrators, keeping them informed of the updates. Each SDK includes documentation detailing the changes made and the compatible versions for both devices and operating systems.

Web API

Communications sent to integrators when new API versions are released. A changelog is also generated for each new version, and in some cases, a "Migration Guide" is provided for significant route changes.

6.2. Troubleshooting

Follow these instructions to address and resolve issues effectively:

- Consult the troubleshooting section of the Annexes.
- Follow the recommended troubleshooting steps, which may involve checking connections, restarting the software or hardware components, or contacting technical support if necessary.
- Document any troubleshooting steps taken and the outcomes, including any error messages or system logs, for reference and future assistance.
- If the issue persists or cannot be resolved through troubleshooting, contact technical support or the manufacturer for further assistance.

7. Technical Specifications and Performance

7.1. Lifetime

Software lifetime of MIR SUPER SDK can vary, and a priori decided minimum as 10 years of use because MD software code cannot degrades, but there are limitations of this period such as:

- Decision of the manufacturer;
- Loss of essential performances depending on change of use framework (MacOS, iOS, Windows, Android).
- Decision of development languages used in MIR SSDK to stop further development.
- Decision of Regulatory Authority to stop distribution of the MD.

In case when MIR decides to stop distribution and use of the MDSW following steps are done:

- Advisory Notice sent to all distributors
- Users are informed via accessory MDSW that in pre-determined period MDSW will be checked from the market.
- MIR SUPER SDK doesn't save any sensible data

7.2. Technical Specifications

The MIR SUPER SDK offers multiple interfaces and connectivity options to facilitate smooth integration with various medical devices and applications:

MIR SUPER SDK is a multiplatform MDSW that can be installed on physical devices. MIR SUPER SDK on web is not installable and can be used only through REST-API. The Source code and programming part of WEB configuration of MIR SUPER SDK is not resold and only used by third parties. Infrastructure for web is always the same as for API of MIR.

API (Application Programming Interface): The SDK exposes a robust API that allows developers to interact with the spirometry and oximetry functionalities programmatically. Through well-defined functions and methods, developers can access measurement data, calibration settings, and other relevant parameters.

Connectivity Protocols: The MIR SUPER SDK supports standard connectivity protocols such as USB, Bluetooth This enables seamless communication between the SDK-enabled medical device and the host system, facilitating data transfer and real-time monitoring.

Data Export: The SDK allows developers to export measurement data and analysis results in various formats, including CSV, XML, and JSON. This data export feature enhances interoperability and enables integration with electronic health records (EHR) systems and other medical software.

Platform Compatibility: The MIR SUPER SDK is designed to be compatible with multiple operating systems, including Windows, Android, iOS and macOS, enabling developers to create cross-platform applications.

For API (Data Centers):

The **SSDK Web** is available as an **API** for **clients**. Therefore, it requires the following prerequisites for usage

For Web:

- **Internet Connection:** 512kbps minimum.
- **Access to HTTPS Port (443):** Support for TLS 1.2 or higher for secure connections.
- **Access Credentials:** API keys or authentication tokens specific to the API.
- **HTTP Client Software:** Such as curl, Postman, or programming libraries that support HTTP(S) requests.
- **Network access:** - Authorization to send or receive payloads larger than 5 MB (without firewall / antivirus or security tools limitations).

For Windows:

- Windows Seven (32 bit/64 bit), Windows 8 (32 bit/64 bit), Windows 10 (32 bit/64 bit), Windows 11 (32 bit/64 bit)
- RAM: 1 gigabyte (GB) for 32 bit or 2 GB for 64 bits.
- 1 gigahertz (GHz) or faster processor, with two or more cores in a 64-bit processor
- Display resolution XGA at 1024 × 768 pixels or higher
- 1Gb of free hard disk space
- Administrative privileges for the operating system
- USB port
- Support for Bluetooth Low Energy (Smart Bluetooth) to connect medical devices with Bluetooth Low Energy connection.

For MacOS:

- Operating system 10.13
- 2 Gb RAM (recommended 4 Gb)
- 1Gb of free hard disk space
- Administrative privileges for the operating system
- USB port
- Support for Bluetooth Low Energy (Smart Bluetooth) to connect medical devices with Bluetooth Low Energy connection.

For iOS:

- iOS version 11.0 minimum
- Bluetooth Permissions
- Compatible Hardware for BLE Support
- 1GB RAM

- Sufficient Free Storage Space
- Full Access to Internet Connection (with at least 1 Mbit/s)

For Android:

- Minimum Android 4.3 (Spirobank II Smart) or 5.0 (Spirobank Smart/SmartOne) version
- Compatible Hardware for BLE Support
- Bluetooth Permissions (BLUETOOTH, BLUETOOTH_ADMIN, and ACCESS_FINE_LOCATION (for Android 6.0 and above)).
- 1GB RAM
- Free Space of the device
- Full access to internet Connection (with at least 1 Mbits/s)

Before proceeding and developing with the MIR SUPER SDK, ensure that your hardware computer system meets the minimum requirements for proper functionality:

Windows Computer (for Windows and Android developments)

Minimum system requirements:

- OS: Windows 8/8.1/10/11 (64-bit)
- CPU: 2nd generation Intel CPU (Sandy Bridge) or newer, AMD CPU with support for a Windows Hypervisor
- Memory: 8 GB RAM
- Free storage: 8 GB
- Screen resolution: 1280 x 800
- Android Studio installed.

Recommended system requirements:

- OS: Windows 10/11 64-bit
- CPU: Intel Core i5-8400 3.0 GHz or better
- Memory: 16 GB RAM
- Free storage: 30 GB (SSD is strongly recommended)
- Screen resolution: 1920 x 1080
- Android Studio installed.

MacOS computers (For MacOS and iOS developments)

Minimum system requirements:

- OS: macOS 10.14 (Mojave) or newer
- CPU: ARM-based chips, or 2nd generation Intel Core or newer with support for Hypervisor.Framework
- Memory: 8 GB RAM

- Free storage: 8 GB
- Screen resolution: 1280 x 800
- Xcode installed.

Recommended specifications:

- OS: macOS 10.15 (Catalina)
- CPU: Intel Core i5-8400 3.0 GHz or better
- Memory: 8 GB RAM
- Free storage: 30 GB (SSD is strongly recommended)
- Screen resolution: 1920 x 1080
- Xcode installed.

7.3. Performance Characteristics

The MIR SUPER SDK exhibits the following performance characteristics:

- **Accuracy:** MIR SUPER SDK does perform calculation based on input data, so accuracy of its output data fully rely on accuracy of input data, whether it comes from spirometers or any other data source.
- **Precision:** MIR SUPER SDK does perform calculation based on input data, so precision of its output data fully rely on precision of input data, whether it comes from spirometers or any other data source.
- **Measurement Range:** The data received and given as output always is exact value. The range vary based on input data given by spirometer or any other data source.
- **Response Time:** Range may vary between 1 ms up to 500ms in the WEB

7.4. Validation and Testing Information

The MIR SUPER SDK's performance claims are supported by relevant validation and testing information. This includes:

- Comparative Studies: Before releasing this version of MIR SUPER SDK the company performed clinical evaluation that proves clinical safety and benefits of our product.
- Validation Testing: All modules of MIR SSDK were tested many times before being sent into production.

8. Legal and Regulatory Information

8.1. Compliance Statements

The MIR SUPER SDK is designed and developed in compliance with relevant standards and regulations to ensure its safety and performance. The following compliance statements are provided:

Standards Compliance: The MIR SUPER SDK conforms to applicable international standards and guidelines, including but not limited to IEC 62304, ISO 14971, and ISO 13485.

Regulatory Compliance: The MIR SUPER SDK complies with the requirements outlined in the Medical Device Regulation (EU) 2017/745. It meets the essential requirements for medical devices specified in Annex I and has undergone the necessary conformity assessment procedures.

8.2. EU Declaration of Conformity

The MIR SUPER SDK is accompanied by an EU Declaration of Conformity. This declaration serves as a formal statement by the manufacturer, confirming that the device complies with the applicable legal requirements and standards. It includes information such as the device's identification, manufacturer's details, and references to relevant directives and regulations.

8.3. Post-Market Surveillance and Vigilance Procedures

The manufacturer of the MIR SUPER SDK has established comprehensive post-market surveillance and vigilance procedures to monitor the performance and safety of the device throughout its lifecycle. These procedures involve systematic collection and analysis of data related to the device's performance, adverse events, and user feedback.

Post-Market Surveillance: The manufacturer regularly collects and reviews data from various sources, including feedback from users, healthcare professionals, and adverse event reporting systems. This information is used to identify and evaluate any potential issues or risks associated with the MIR SUPER SDK.




Vigilance Procedures: In compliance with regulatory requirements, the manufacturer maintains a robust vigilance system to promptly identify, assess, and report any adverse events or incidents related to the MIR SUPER SDK. This includes the timely reporting of serious incidents to the competent authorities and taking appropriate corrective actions, if necessary.

8.4. Label and Symbols

8.4.1. Identification label and symbols



8.4.2. Used symbols

SYMBOL	DESCRIPTION
	Manufacturer symbol
REF	Indicates the manufacturer's catalogue number so that the medical device can be identified.
UDI	The symbol indicates the Unique Device Identification
MD	The symbol indicates that the product is a medical device
	This product is certified CE to conform to the Class IIa requirements of the European Regulation (EU) 2017/745.
	eIFU indicator

9. Manufacturer's Responsibility and Liability

9.1. Manufacturer's Commitment to Product Quality

MIR is committed to delivering a high-quality MIR SUPER SDK that meets rigorous standards of safety and performance.

9.2. Warranty Information

The MIR SUPER SDK is accompanied by a warranty that covers during normal use. We will repair or replace the software as specified in the warranty documentation.

9.3. Limitations of Liability

MIR accepts no responsibility for loss, damage, or injury resulting from improper use of the MIR SUPER SDK. Users are responsible for using the software correctly and interpreting results accurately.

9.4. Indemnification

Users agree to hold MIR harmless from any claims or liabilities arising from the use or misuse of the MIR SUPER SDK.

9.5. Regulatory Compliance

The MIR SUPER SDK complies with the requirements of the Medical Device Regulation (EU) 2017/745 and meets essential requirements for medical devices.

9.6. Conclusion

MIR takes responsibility for the quality and compliance of the MIR SUPER SDK. Users should adhere to instructions and understand their role in ensuring the proper use and interpretation of the software.

9.7. List of MIR approved MD class I accessories

MIR Spiro

MIR Spiro Expert

Pneumotel Web

Pneumotel Mobile

Annex A. Instruction for Use - SSDK iOS

Introduction

The purpose of this guide is to facilitate use of the tool **SUPER SDK (Software Development Kit)** for the rapid development of applications running on iOS for monitoring and storing patients, archives of spirometry and oximetry tests obtained through MIR Bluetooth devices.

MIR SSDK iOS allows you to quickly perform all the necessary operations to use Spirometry or Oximetry.

The SSDK is only compatible with Spirobank Smart, Smart One and Spirobank II Smart devices.

Prerequisites

Before you begin integrating the Android iOS into your application, make sure you have the following in place:

- iOS version 11.0 minimum
- Bluetooth Permissions
- Compatible Hardware for BLE Support
- 1GB RAM
- Sufficient Free Storage Space
- Full Access to Internet Connection (with at least 1 Mbit/s)

Integration (How to implement?)

MIR SMART DEVICES

Import the Framework

To import the framework inside your project just take the following steps: 1- Open Xcode and drag the `MirSmartDevice.framework` bundle inside the project 2- Select the project in the TARGETS list. In the General tab click the add button (+) in the Embedded Binaries section to add the `MirSmartDevice.framework` If in the section "Linked Frameworks and Libraries" the `MirSmartDevice.framework` is listed twice, just remove one item using the remove button (-) see the video tutorial at: [vimeo](#)

After that use the following instruction: `#import < MirSmartDevice / MirSmartDevice.h>`

Starting up a Device Manager

The first step to take to use this framework is to get a `SODeviceManager` object. The `SODeviceManager` has been implemented according to the singleton pattern so you would get a same instance of it to be used in every context of your app.

The `SODeviceManager` class also exposes the `addDelegate` and `RemoveDelegate` methods in order to supports multiple delegates.

Initialise the Bluetooth

Before starting any bluetooth communication the [bluetoothState](#) method of the `SODeviceManager` must be called. The [bluetoothState](#) method returns the [CBCentralManagerState](#).

[CBCentralManagerState](#) tells you current bluetooth state and can assume values:
[CBCentralManagerStateUnknown](#)
[CBCentralManagerStatePoweredOn](#) [CBCentralManagerStateUnsupported](#)
[CBCentralManagerStatePoweredOff](#)

[CBCentralManagerStateResetting](#) [CBCentralManagerStateUnauthorized](#)

If the returned state is [CBCentralManagerStateUnknown](#), you have to wait for `SODeviceManager` to call its delegate method [didUpdateBluetoothWithState](#) with the updated state (before perform any scan or connection). Otherwise call `SODeviceManager` [initBluetooth](#) method to get updated bluetooth state through [didUpdateBluetoothWithState](#) method

If the returned state is different from [CBCentralManagerStateUnknown](#), it means that the bluetooth has already been initialised. (IMPORTANT!) In this case the [didUpdateBluetoothWithState](#) won't be called and you can use the returned [CBCentralManagerState](#) immediately to perform discovery, connection

`SODeviceManager` also call the [didUpdateBluetoothWithState](#) method each time the bluetooth status is modified (i.e.. the user switches off the bluetooth, the user switches it on again)

Perform a scan (discovery)

With an instance of `deviceManager` and after having initialised the Bluetooth, the client app may call the [startDiscovery](#) method. The discovery will retrieves all the Spirobank Smart devices in range. For each device discovered, the `deviceManager` call its delegate method [didDiscoverDeviceWithInfo](#).

Perform a “direct connection” to a Device with an instance of `deviceManager` and after having initialised the Bluetooth, the client app may call the `connect` method passing the `deviceId` (UUID of the device) even if the device has not been discovered during the current app life cycle.

When `deviceManager` calls its delegate method [didConnectDevice](#) it means that the device is connected and all its services and characteristics have been read. If the same device is already connected the `connect` method will disconnect and reconnect the device. if a different device is already connected, the `connect` method disconnect it before connecting that new device.

Start a test in the “multitest mode” environment

IMPORTANT NOTE:

For the scope of this framework “test” means a complete expiratory manoeuvre

The framework supports the “multitest mode”. This means that different kind of tests can be started.

At present the tests supported are:

Test supported	Description	From version	Note
FVC test	a forced expiratory maneuver		
PeakFlow/Fev 1	a forced expiratory maneuver that lasts 1 second		works only with SpirobankSmart with firmware >= 3.0
Flow Monitoring Test		2.8.0	works only with the new SpirobankSmart Oxi
Oximetry test		2.8.0	works only with the new SpirobankSmart Oxi
FVC PLUS Test	a forced expiratory and inspiratory maneuver	2.9.0	works only with ENABLED device: SpirobankSmart with firmware >= 3.1 and Spirobank OXI with firmware >= 1.0
SVC	A slow expiratory and inspiratory maneuver	3.0.0	works only with ENABLED device: SpirobankSmart with firmware >= 4.3 and Spirobank OXI with firmware >= 4.3

Please note that if you are using this framework with spirobank smart device equipped with firmware versions that do not support the multitest mode, the only valid command is that one to start the FVC test. the command to start the PeakFlow/Fev1 test would be just ignored.

Only the Spirobank Smart with firmware version >= 1.7 (protocol 005) supports the multitest mode.

To require a specific test to be started by the device, the framework provides a new method with a parameter. With an instance of SOdevice the

[StartTestWithTestType:\(SOTestType\)testType](#) method has to be invoked to

start one of the supported test type. Pass to this method the parameter

- [SOTestType.TestFVC](#) to start the FVC test
- [SOTestType.TestPeakFlowFev1](#) to start the PeakFlow/Fev1 test

- [SOTestType.TestFTmonitor](#) to start the FVC test
- [SOTestType.TestFVCPlus](#) to start the FVC PLUS test
- [SOTestType.TestSVC](#) to start the SVC test
- [SOTestType.TestMVV](#) to start the MVV test (only Spirobank II >= 5.3)
-

The method `StartTest` is **deprecated** from version 2.0 of the framework but is still working for the backward compatibility. This method starts the FVC test which is the "default" test for this framework (FVC in case of Spirobank Smart/ Spierobank Oxi)

Starting a test with a customized "End of Test timeout" (from version 2.3.0) Turbine Type (from version 2.6.0) Ambient Temperature (for version 3.0.0)

Overloads of the "start test" method has been implemented by this framework.

In version 2.3 a new argument [endOfTestTimeout](#) was added in order to set the End of Test timeout (see below: End Of Test Timeout). The new method (available from version 2.3 of this framework and from the version 2.4 of SpirobankSmart internal software) is the following:

[startTestWithTestType:\(SOTestType\)testType endOfTestTimeout:\(Byte\) timeoutInSeconds;](#)

This method is invoked to start the test specified by the parameter "testType" with the EndOfTest timeout specified by the parameter "timeoutInSeconds". The EOT timeout is the number of seconds after which the test is automatically ended by the spirometer (if the user was not been blowing at all since the test started).

The valid range for the "timeout" parameter is 15s - 120s.

If a value < 15 is passed, the spirometer sets the timeout to 15s.

If a value > 120 is passed, the spirometer sets the timeout to 120s. Whatever value is passed to a SpirobankSmart with internal software < 2.4, the default value of 15 seconds will be applied **EXCEPT FOR OXIMETRY where this parameter will be ignored**

We recommend to avoid any unnecessary increasing of the timeout to preserve the Mir Smart Device's battery life

Multiturbine management

In the version 2.6.0 a new argument **turbine** was added in order to specify the turbine type (reusable or disposable) is in use on the device.

```
(void)startTestWithType:(SOTestType)testType      endOfTestTimeout:(Byte)
timeoutInSeconds turbineType:(SOTurbineType)turbine
```

The turbine type can be reusable (default) or disposable. The client app should ask the user which turbine is he/she using (Orange one = reusable or White one = disposable). It should be a good idea to show to the user the pictures of the 2 turbines to make the selection easier for him/her.

This instruction is supported only by spiobankSmart equipped with **firmware**

```
>
=
2.
7
```

This instruction does affects the reading made by the device because different algorithms are used for each turbine type.

This parameters is ignored in case of OXIMETRY test will be ignored

method (whatever is the
StartTest handshaking When the client app sends used
requested) the framework, from version
2.6.0 sends a new delegate method: - (
id)
DeviceDidStartTest:(SO
*)soDevice: This method is invoked when the device is a
vice
measurements. For this reason, this new method is the right
actually READY to take
the user can START EXHALING (blowing).

A new argument [ambientTemperatureCelsius](#) was added for the compliance to the ATS 2019 guidelines.

If passed this parameter is used by the device to calculate the BTPS. If not passed the device calculates the BTPS at the "default" temperature of 25°C (77° F)

During the test the SOdevice object calls its delegate method [didUpdateFlowValue:isFirstPackage](#) to pass the measured flow values **in case of FVC or PEF-FEV1 test or FVCPlus test**

[didUpdateFlowTimeMonitoringValue](#) to pass the measured flow values **in case of Flow Time Monitoring test**

[didUpdateVcVolumeTimePoint](#) to pass the measured volume and time values **in case of SVC test**

(FVCPlus test) [didReceiveEndOfForcedExpirationIndicator](#) to notify that one

EOFE criteria has been achieved during the **FVCPlus test**

(SVC test) [didPerformVentilatoryProfile](#) to notify the END of the tidal breathing phase and the beginning of the SVC test phase (deep and slow expiration / inspiration) during the **SVC test**

`didUpdateOximetryRealTimeValuesWithSignal:spO2Value:bpmValue:warning:`

`isDataValid` to pass the measured oximetry values in case of oximetry test

`didUpdateOximetryPletismographicValue` to pass the point values of the plethysmographic curve

in case of oximetry test `heartBeatDetected` to pass the detection of an heart beat in case of

oximetry test

For each flow point received the current volume can be get by:

- (expiring phase) adding the `volumeStep` (SODevice attribute) to the current volume
Current Volume += volumeStep mnhju
- (inspiring phase) subtracting the `volumeStep` (SODevice attribute) to the current volume
Current Volume -= volumeStep

At the end of FVC or PEF-FEV1 test , SOdevice object usually calls its delegate method to provide the test's results: `soDevice:didUPdateResults:(SOResults *)results`

At the end of FVCPlus test , SOdevice object calls

`soDevice:didUPdateFvcPlusResults:(SOResultsFvcPlus *)results;`

At the end of SVC test , SOdevice object calls `soDevice:didUPdateVcResults:(SOResultsVc *)results;`

If a parameter is not provided by the performed test type, it is passed with value = -1 by the Results object.

Note that if the test is performed very bad (too poor information detected by the device sensors)

the device is not able to calculate the results and therefore the delegate method `soDevice:didUpdateResults` IS NOT CALLED AT ALL. the Results provided (by the different test types) are the following

PARAMETER

PROVIDED BY

<code>pef_cLs</code> (Peakflow cL sec)	FVC-PEAKFLOW/FEV1
<code>fev1_cL</code> (Forced Exp Vol at 1th sec in cL)	FVC-PEAKFLOW/FEV1
<code>quality</code> (acceptability calculation)	FVC-PEAKFLOW/FEV1
<code>fvc_cL</code> (Forced Exp Capacity in cL)	FVC
<code>fev1_fvc_pcmt</code> (Fev1% in percentage)	FVC
<code>fev6_cL</code> (Forced Exp Volume at 6th sec in cL)	FVC
<code>fef2575_cLs</code> (Max mid-expiratory flow in cL sec)	FVC
<code>eVol_mL</code> (Extrapolated volume in mL)	PEAKFLOW/FEV1
<code>pefTime_sec</code> (time to reach Peakfow in sec)	PEAKFLOW/FEV1

FVC PLUS TEST

<code>pef_Ls</code> (Exp Peakflow L sec)
<code>fev1_L</code> (Forced Exp Vol at 1st sec in L)
<code>quality</code> (acceptability calculation)
<code>fvc_L</code> (Forced Exp Capacity in L)
<code>fev1_fvc_pcmt</code> (Fev1% in percentage)
<code>fev6_L</code> (Forced Exp Volume at 6th sec in L)
<code>fef2575_Ls</code> (Max mid-expiratory flow in L sec)
<code>eVol_mL</code> (Extrapolated volume in mL)
<code>pefTime_ms</code> (time to reach Peakfow in milliseconds)
<code>fef75_L</code> (Max mid-expiratory flow in L sec)

fet_cs (flow expiratory time in sec * 100)

fef25_Ls (Max mid-expiratory flow in L sec)

fef50_Ls (Max mid-expiratory flow in L sec)

fivc_L (Forced Insp Capacity in L)

fiv1_L (Forced Insp Vol at 1st sec in L)

pif_Ls (Insp Peakflow L sec)

fev3_L (Forced Insp Vol at 3rd sec in L)

fev05_L (Forced Insp Vol at 0.5th sec in L)

fev075_L (Forced Insp Vol at 0.75th sec in L)

fev2_L (Forced Insp Vol at 2nd sec in L)

fef7585_Ls (Max mid-expiratory flow in L sec)

fif25_Ls (Max mid-expiratory flow in L sec)

fif50_Ls (Max mid-expiratory flow in L sec)

fif75_Ls (Max mid-expiratory flow in L sec)

fev1_fev6_perc

fev6_fvc_perc

fiv1_fivc_perc

fev3_fvc_perc

fev05_fvc_perc

fev075_fvc_perc fev2_fvc_perc

hesitationTime_s

SVC TEST

PARAMETER

evc_L (Exp Vital Capacity in L) **ivc_L** (Insp Vital Capacity in L) **ic_L** (Inspiratory Capacity in L)
slowExpInspTime_s (exp or insp time)

_!

At the end of Oximetry test, SOdevice object usually calls its delegate method to provide the test's results:

`soDevice:didUpdateOximetryResults:(SOResultsOximetry *)oximetryResults`

PARAMETER

PROVIDED BY

spO2Mean (%)	OXIMETRY
spO2Max (%)	OXIMETRY
spO2Min (%)	OXIMETRY
HeartRateMean (bpm)	OXIMETRY
HeartRateMax (bpm)	OXIMETRY
HeartRateMin (bpm)	OXIMETRY
spo2Points (Array)	OXIMETRY
heartRatePoints (Array)	OXIMETRY

MVV TEST

At the end of Mvv test, SOdevice object usually calls its delegate method to provide the test's results:

`soDevice didUpdateMvvResults:(SOResultsMvv *)results;`

PARAMETER

PROVIDED BY

MVV_Lm	MVV
--------	-----

How the test is stopped/restarted in the “multitest mode” environment

The test are stopped in two ways:

- A. when the [stopTest](#) method is invoked
- B. automatically, when the device/framework detects the

EOT (**End Of Test**) criteria. See above the chapter **End Of Test Criteria**.

In the SVC test when the test is stopped (automatically or not) the device always quits from “test mode” and a new command “startTest” needs to be sent to start a new test the sequence of the delegate methods called during an SCV test are the following:

[soDevice:didUpdateVcVolumeTimePoint](#) [soDevice:didStopTest](#) (always called, even if the test was stopped by the invocation of the stopTest method) [soDevice:didUpdateVcResults](#) (which might not be called in case there aren't the conditions to return the Results)

In the FVC Plus test when the test is stopped (automatically or not) the device always quits from “test mode” and a new command “startTest” needs to be sent to start a new test the sequence of the delegate methods called during an FCV test are the following:

[soDevice:didUpdateFlowValue](#)

[soDevice:didStopTest](#) (always called, even if the test was stopped by the invocation of the stopTest method) [soDevice:didUpdateFvcPlusResults](#) (which might not be called in case there aren't the conditions to return the Results)

Note that the FVC Plus test automatically quits after 60 seconds

In the FVC test when the test is stopped (automatically or not) the device always quits from “test mode” and a new command “startTest” needs to be sent to start a new test the sequence of the delegate methods called during an FCV test are the following:

[soDevice:didUpdateFlowValue](#)

[soDevice:didStopTest](#) (always called, even if the test was stopped by the invocation of the stopTest method) [soDevice:didUpdateResults](#) (which might not be called in case there aren't the conditions to return the Results)

In the Peakflow/Fev1 test

There is a different behavior depending how the stop the test has occurred.

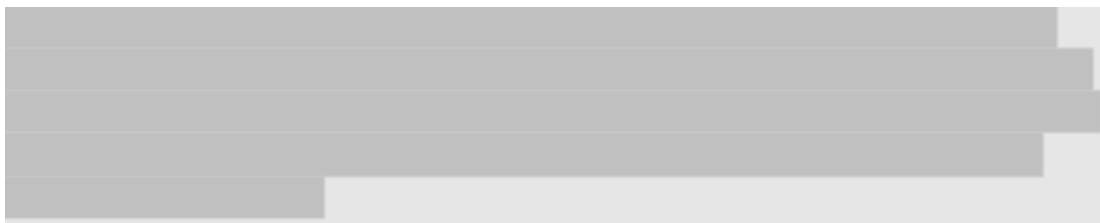
1) when the test is stopped:

- by the invocation of the stopTest method
- by the expiration of the timeouts spiobanksmart quits from the "test mode" and a new command "startTest" needs to be sent to start a new test.

In this case the sequence of the delegate methods called are the following:
[soDevice:didUpdateFlowValue](#)

[soDevice:didUpdateResults](#) (which might not be called in case there aren't the conditions to return the Results)

[soDevice:didStopTest](#) (always called, even if the test was stopped by the invocation of the stopTest method)



IMPORTANT NOTE:

It is strongly recommended to avoid placing the call to the StartTest method into the soDevice:didStopTest delegate method for two main reason: Because this might activate a loop with a negative impact on the battery life and because the test stopping and restarting would take place almost simultaneously.

2) when the test is stopped:

- because the device has automatically detected the end of the expiratory manoeuvre (See above the chapter **End Of Test Criteria**)

spirobanksmart stops the test but it DOES NOT quit from the “test mode” and RESTART AUTOMATICALLY a new test (no need to call the startTest method to start a new test)

In this case the sequence of the delegate methods called are the following:
[soDevice:didUpdateFlowValue](#)

[soDevice:didUpdateResults](#) (which might not be called in case there aren't the conditions to return the Results) [soDevice:didRestartTest](#) (always called. It means that a new test is started, the user can blow)

This behavior, called **AutomaticTestRestarting**, has been designed for the Peakflow test where the patient can perform the 3 tests, recommended for a valid session, without any rest interval because of the short duration of each manoeuvre (1 second).

Thought the AutomaticTestRestarting approach is recommended, the developer can decide to handle the Peakflow/Fev1 test using the **Start&Stop** approach: with this approach the developer should invoke the [stopTest](#) method as soon as the delegate method [soDevice:didRestartTest](#) is called and then use the [StartTestWithTestType](#) method to start a new test.

See Best Practices section for more detailed info.

In the Flow Time Monitoring test when the test is stopped (automatically or not) the device always quits from “test mode” and a new command “startTest” needs to be sent to start a new test the sequence of the delegate methods called during an Flow Time Monitoring test are the following:
[soDevice: didUpdateFlowTimeMonitoringValue](#)

[soDevice:didStopTest](#) (always called, even if the test was stopped by the invocation of the stopTest method)

NO RESULTS ARE SENT WITH THIS KIND OF TEST

In the OXIMETRY test when the test is stopped the device always quits from “test mode” and a

new command “startTest” needs to be sent to start a new test the sequence of the delegate methods called during an FCV test are the following:

[didUpdateOximetryRealTimeValuesWithSignal:spO2Value:bpmValue:warning:](#)

[isDataValid didUpdateOximetryPletismographicValue heartBeatDetected](#)

[soDevice:didUpdateResults](#) (which might not be called in case there aren't the conditions to return the Results)

[soDevice:didStopTest](#) (always called, even if the test was stopped by the invocation of the stopTest method)

Real Time Animation in the PEAKFLOW-FEV1 test the SOPatient class can be instantiated to get some important information during the test to be used to display the animated feed back of the user's expiration.

The model of animation proposed by SOPatient, is based on the concept of the Predicted Area (calculated from user's personal data). Two graphic objects have to move inside the Predicted Area: One graphic object (target object) is moved by the user's expired (and inspired in the FVC Plus test) volume with a preset speed (based on the predicted flow). The other graphic object (user object) is moved according to the user's expired volume (and inspired in the FVC Plus test) and at the speed of the user's flow (measured flow).

The Predicted Area is based on patient's FVC and PeakFlow in case of FVC test and FVC Plus test. In case of PeakFlow/Fev1 test, instead, the Predicted Area is based on patient's FEV1 and the PeakFlow.

The method [actualPercentageOfTargetWithFlow:volumeStep:isFirstPackage:](#) retrieves the percentage of the Predicted Area which has been covered by the

“user object”

This percentage value can be asked to SOPatient for each flow retrieved by the method [soDevice:didUpdateFlowValue:isFirstPackage:](#)

set the difficulty of the test in percentage (from 20 to 200). If the value passed is < 100 is easier to reach the target, if it is > 100 is harder. 100 is normal. if you don't know the value to set, pass 100 to this parameter

The method [predictedPercentageOfTargetWithFlow:volumeStep:isFirstPackage:](#) retrieves the percentage of the Predicted "AREA" which has been covered by the "target object"

This percentage value can be asked to SOPatient for each flow retrieved by the method [soDevice:didUpdateFlowValue:isFirstPackage:](#)

Difficulty Level

Before starting the test, a difficulty level can be set to make easier or harder for the "user object" to reach the "target object". Use the SOPatient property [difficultyLevel](#).

The difficulty level has only a psychological effect on the user (i.e. avoid his / her frustration when the user object can never reach the speed of the target object): It does not affect in any way the results of the manoeuvre (Peak flow or FEV1 values).

The difficulty level is expressed in percentage and can be set with values from 20 to 200 (any values out of range will be set to the nearest threshold).

= 100 -> expected difficulty to reach the target

< 100 -> easier than normal to reach the target

> 100 -> harder than normal to reach target

Real Time Animation in the FVC-FVCPLUS test

During the FVC and the **FVC Plus** test the Flow Volume loop can be plotted. The flow points are provided by the delegate method [soDevice:didUpdateFlowValue:isFirstPackage:](#)

These flow points are provided at a constant volume step ([volumeStep](#)). During the expiration, for each flow point the current volume increase of 1 [volumeStep](#)

During the inspiration (**FVC Plus test only**), for each flow point the current volume decrease of 1 [volumeStep](#)

Real Time Animation in the Flow Time Monitoring test

During Flow Time Monitoring test the Flow Time loop (expired and inspired points) can be plotted.

The Flow points are provided (by the delegate method `soDevice:`

`didUpdateFlowTimeMonitoringValue`) at a constant time of 10 milliseconds. The value of flow is an int type that is positive for expiration and negative for expiration. It is provided in cL/s.

Real Time Animation in the Oximetry test

During an Oximetry test the following curves can be plotted:

- the plethysmographic curve, using `didUpdateOximetryPletismographicValue`

`(int) ppmSignal`

- the sPO2 curve and/or the Pulse Rate curve, using

`soDevice:(SODevice *)soDevice didUpdateOximetryRealTimeValuesWithSignal:(int)signal`

`spO2Value:(int)spO2`

`bpmValue:(int)bpm`

`warning:(SOOximetryWarnings)warning`

`isDataValid:(BOOL)isdatavalid;`

Value range is 70 → 99 bpm

Value range is 30 → 300

the above method can also be used for displaying other info to the user such as `signal` (range 0 to 8)

`warnings` (`SOOximetryWarnings`)

The following values can be assigned to the parameter [warning](#)

NoWarning

DefectiveSensor

BatteryLow

NoFinger

PulseSearching

PulseSearchingTooLong

LossOfPulse

LowSignalQuality

LowPerfusion

ArtifactDetected

CAUTION: when the [warning](#) parameter = [BatteryLow](#), the device will stop the test and the delegate method [SODeviceDidStopTest](#) is called.

the parameter [IsValidData](#) specifies if the value of SpO2 ([spO2Value](#)) and Pulse Rate ([bpmValue](#)) are valid.

When [IsValidData](#) = NO, those values should be displayed with the symbol “—” and the [test duration timer](#) (if used) should be paused.

Usually when warning value is different from [NoWarning](#), [isDataValid](#) =NO.

This does not happen when warning = [LowPerfusion](#): in that case [IsValidData](#) = YES

the delegate method [heartBeatDetected](#) can be used to perform a beat if an icon with a beating

heart is displayed

Real Time Animation in the SVC TEST

The methods provided by the SOPatient class during the FVC real time test in order to build an animated feedback (actualPercentageOfTargetWithFlow and predictedPercentageOfTargetWithFlow) cannot be used for the SVC test because they are based on flow.

Quality report - Acceptability (ONLY FOR FVC, FVC PLUS AND PEAKFLOW-FEV1 TEST)

Device ATS 2019

- Boot ID = SE And PROTOCOL \geq 11 (SPIROBANK OXI)
- Boot ID = SM And PROTOCOL \geq 9 (SPIROBANK SMART)
- Boot ID = SX And PROTOCOL \geq 10 (SMARTONE OXI)
- Boot ID = SO And PROTOCOL \geq 8 (SMARTONE)
- Boot ID = BK And PROTOCOL \geq 00 (SPIROBANK II)
- Boot ID = BK And PROTOCOL \geq 13 (SPIROBANK II)

Device ATS 2015

- Boot ID = SE And PROTOCOL $<$ 11 (SPIROBANK OXI)
- Boot ID = SM And PROTOCOL $<$ 9 (SPIROBANK SMART)
- Boot ID = SX And PROTOCOL $<$ 10 (SMARTONE OXI)
- Boot ID = SO And PROTOCOL $<$ 8 (SMARTONE)
- Boot ID = BN And PROTOCOL = 255 (SPIROBANK II)

The SOPatient class provides information about the acceptability of each single manoeuvre in terms of quality

Starting from the framework version 3.0.0 the information about the Acceptability are provided for both ATS2015 and ATS2019 standards, depending on the standard supported by the connected mir

device.

ATS Satandard supported by mir devices

Device ATS 2019

- Boot ID = SE And PROTOCOL >= 11 (SPIROBANK OXI)
- Boot ID = SM And PROTOCOL >= 9 (SPIROBANK SMART)
- Boot ID = SX And PROTOCOL >= 10 (SMARTONE OXI)
- Boot ID = SO And PROTOCOL >= 8 (SMARTONE)
- Boot ID = BK And PROTOCOL >= 00 (SPIROBANK II)
- Boot ID = BK And PROTOCOL >= 13 (SPIROBANK II)

Device ATS 2015

- Boot ID = SE And PROTOCOL < 11 (SPIROBANK OXI)
- Boot ID = SM And PROTOCOL < 9 (SPIROBANK SMART)
- Boot ID = SX And PROTOCOL < 10 (SMARTONE OXI)
- Boot ID = SO And PROTOCOL < 8 (SMARTONE)
- Boot ID = BN And PROTOCOL = 255 (SPIROBANK II)

In the frameworks >= 3.0.0 the methods *QualityMessageForResults*

The new methods to calculate the Acceptability are:

For FVC and peakflow-fev1 test

-(SOQualityReport *_Nullable) QualityReportForResults:(SOResults *_Nonnull) results;

For FVC Plus test

-(SOQualityReport *_Nullable) QualityReportForResultsFvcPlus:(SOResultsFvcPlus *_Nonnull) results;

Or the overload version

For FVC and peakflow-fev1 test

-(SOQualityReport * _Nullable) QualityReportForResults:(SOResults * _Nonnull) results
 WithSessionLargestFvcValue_L: (float)bestSessionFvc_L; -(SOQualityReport * _Nullable)

For FVC Plus test

-(SOQualityReport * _Nullable) QualityReportForResultsFvcPlus:(SOResultsFvcPlus * _Nonnull)results
 WithSessionLargestFvcValue_L: (float) bestSessionFvc_L;

The object retrieved by the above functions, [SOQualityReport](#), includes the following info:

standardUsedByCurrentDevice	The standard in use by the device
trialAcceptability	Calculated only if the standard in use by the device is = ATS2015
fvcAcceptability	Calculated only if the standard in use by the device is = ATS2019
fev1Acceptability	Calculated only if the standard in use by the device is = ATS2019
fev075Acceptability	Calculated only if the standard in use by the device is = ATS2019
QualityIndications	SOQualityMessage SOQualityInstruction

in the FVC/**FVC Plus** maneuver the SOQualityMessage provided are:

SOQualityMessageDontEsitate

SOQualityMessageBlowOutFaster

SOQualityMessageBlowOutLonger

SOQualityMessageAbruptEnd

SOQualityMessageGoodBlow

SOQualityMessageDontStartTooEarly SOQualityMessageAvoidCoughing

(the following provided only if the device in use supports ATS2019 standard)

SOQualityMessageHesitationAtMaxVolume

SOQualityMessageSlowFilling

SOQualityMessageLowFinalInspiration

SOQualityMessageIncompleteInspirationPriorToFvc

SOQualityMessageLowForcedExpirationVolume

in the PeakFlow/Fev1 maneuver the following messages are provided:

SOQualityMessageDontEsitate

SOQualityMessageBlowOutFaster

SOQualityMessageBlowOutLonger (*) SOQualityMessageGoodBlow

SOQualityMessageDontStartTooEarly

SOQualityMessageAvoidCoughing

(*) The message "BlowOutLonger" is available, in the PeakFlow/Fev1 test, only if you use this

FRAMEWORK with a SpirobankSmart equipped with internal software version >= 2.3

Get the FVC curve points at the highest resolution

From SDK version 3.1.0, when connected to a Spirobank smart supporting this feature (firmware \geq 4.6), you can get the **expired only** curve points of the last FVC PLUS test performed at the resolution of 100Hz (one point each 10 milliseconds).

From the SDK version 4.1.0 when connected to a Spirobank smart supporting this feature (firmware \geq 4.7), you can get the **expired and the inspired** curve points of the last FVC PLUS test. Note that the high resolution curve points ARE NOT automatically retrieved with the `SOResultsFvcPlus` object passed by the `didUpdateFvcPlusResults` delegate method.

To get the high resolution expired curve points the method [getHighResolutionCurveForLastSpirometryTest](#) of the `SODevice` object must be called. It must be called AFTER receiving the `didUpdateFvcPlusResults` call back

To get the high resolution expired and **inspired** curve points the method [getHighResolutionCurveForLastSpirometryTestWithInspiration](#) of the `SODevice` object must be called. It must be called AFTER receiving the `didUpdateFvcPlusResults` call back

This last method (`withInspiration`) is slower than the previous method (only expiration) so if inspiration points are not needed it is recommended to call the previous one (only expiration)

When one of the above-mentioned methods is called (only expiration or expiration and inspiration) the high resolution curve points are provided by the delegate method `didUpdateHighResolutionCurvePoints` (`SODevice` class)

The above the delegate method retrieves a collection of `CurvePoint` *objects* having, each one, as a properties Flow, Volume and Time

Update device internal software

CAUTION: THIS FUNCTION ONLY ALLOWS THE FIRMWARE UPDATE FOR THE FOLLOWING DEVICES;

device supported	firmware supported
Smart One	< 4.0
Spirobank Smart	< 4.0

With an instance of SoDevice call the method `startSoftwareUpdate:(NSData *) newSoftware`. The newSoftware argument is a .bin file provided by MIR.

During the update the SODevice object notifies its subscribers by calling the delegate method `soDevice:(SODevice *)soDevice didReceiveSoftwareUpdateProgress:(NSUInteger)progress withStatus:`

`(UpdateStatus)status error:(NSString *)description; soDevice`

soDevice is the device providing this information.

progress is the percentage value of the update progress (0 to 100) status is the update status code (UpdateIdle, UpdateInProgress, UpdateError, UpdateComplete)

description is the description of the cause of the failure in case of error. If no error has occurred it is = nil. The error descriptions can be:

@"Update start timed out" when the time it takes to start the firmware loading procedure exceeds the timeout
@"Communication timed out"

- when the time it takes to load one of the "packets" (of firmware) exceeds the timeout
- this message can also appear after 100% if the firmware doesn't match

the device.

Run the Project

ATTENTION: the projects with the MirSmartDevice.framework embedded can only be compiled and run in a iOS device. Simulator is not supported.

Best Practices

The best practice to handle Mir Spirometer and avoid instability and malfunctioning is the following:

PEAK-FLOW / FEV1 TEST (AutomaticTestRestarting approach)

1. Get an instance of the [SODeviceManager](#) Class
2. Perform a scan and connect OR perform a "direct connection" to a spirometer
3. Start the test (user must start blowing within the EOT timeout: default =15 sec)
4. Use a visual feedback to prevent that user start blowing before the app has received the delegate method [soDeviceDidStartTest](#)

[soDeviceDidStopTest](#) delegate method is called if user doesn't blow within the EOT timeout (in this case give to the user the ability to restart the test manually)

It is strongly NOT recommended to call the Start test on the [soDeviceDidStopTest](#) delegate method as a workaround to contrast the End Of Test timeout effect. Instead call the start test only when the patient is ready to blow or increase the End of Test Timeout (see above: [Starting a test with a customized "End of Test timeout"](#))

5. Use [didUpdateFlowValue](#) delegate method to show the animated feedback

(also in connection with the SOPatient class's dedicated methods)

6. Use [didUpdateResults](#) to show the result (this method might not be called if the device was not able to calculate the results)
7. use [didRestartTest](#) delegate method to detect that the current expiratory manoeuvre is ended and advice the user to start blowing and perform a new expiratory manoeuvre
8. Repeat from step 4 (3 times at least)
9. send the [stopTest](#) command to quit from test and wait for the [didStopTest](#) delegate method to be called.

It is strongly NOT recommended to disconnect the device

([SODeviceManager:disconnect](#)) at the end of each session or test. The bluetooth disconnection of the device should be called only if no more spirometry test need to be performed in a short time (less than 1 minute).

Even better is to disconnect when the app became inactive

PEAK-FLOW / FEV1 TEST (Start&Stop approach)

1. Get an instance of the [SODeviceManager](#) Class
2. Perform a scan and connect OR perform a "direct connection" to a spirometer
3. Start the test (user must start blowing within the EOT timeout: default = 15 sec)

[soDeviceDidStopTest](#) delegate method is called if user doesn't blow within the EOT timeout -default = 15 sec-

The app should give to the user the ability to restart the test manually whenever this timeout expires

It is strongly NOT recommended to call the Start test on the [soDeviceDidStopTest](#) delegate method as a workaround to contrast the End Of Test timeout effect. Instead call the start test only when the patient is ready to blow or increase the End of Test Timeout (see above: **Starting a test with a customized "End of Test timeout"**)

4. Use a visual feedback to prevent that user start blowing before the app has received the delegate method [soDeviceDidStartTest](#)

5. Use [didUpdateFlowValue](#) delegate method to show the animated feedback

(also in connection with the [SOPatient](#) class's dedicated methods)

6. Use [didUpdateResults](#) to show the result (this method might not be called if the device was not able to calculate the results)

7. use [didRestartTest](#) delegate method to detect that the current expiratory manoeuvre is ended and use the [stopTest](#) command to quit from test (then wait for the [didStopTest](#) delegate method to be called).

8. Repeat from step 3 (3 times at least)

It is strongly NOT recommended to disconnect the device

([SODeviceManager:disconnect](#)) at the end of each session or test. The bluetooth disconnection of the device should be called only if no more spirometry test need to be performed in a short time (less than 1 minute).

Even better is to disconnect when the app became inactive

FVC / FVC PLUS TEST

1. Get an instance of the [SODeviceManager](#) Class
2. Perform a scan and connect OR perform a "direct connection" to a spirometer
3. Start the test (user must start blowing within the EOT timeout: default 15 sec) [soDeviceDidStopTest](#) delegate method is called if user doesn't blow within the EOT timeout (give to the user the ability to restart test manually) It is strongly NOT recommended to call the Start test on the [soDeviceDidStopTest](#) delegate method as a workaround to contrast the End Of Test timeout effect. Instead call the start test only when the patient is ready to blow or increase the End of Test Timeout (see above: **Starting a test with a customized "End of Test timeout"**)
4. Use a visual feedback to prevent that user start blowing before the app has received the delegate method [soDeviceDidStartTest](#)
5. Use [didUpdateFlowValue](#) delegate method to show the animated feedback
(also in connection with the [SOPatient](#) class's dedicated methods)
6. If the [receivedEndOfForcedExpirationIndicator](#) delegate method is called, the user can be advised to stop blowing, since a plateau or the end of expiratory time was reached
7. Use [didUpdateResults](#) / [didUpdateFvcPlusResults](#) to show the result (this method might not be called if the device was not able to calculate the results)

It is strongly NOT recommended to disconnect the device

([SODeviceManager:disconnect](#)) at the end of each session or test. The bluetooth disconnection of the device should be called only if no more spirometry test need to be performed in a short time (less than 1 minute). Even better is to disconnect when the app became inactive

SVC TEST

(*Spirobank Smart only from firmware version 4.3+*)

- a. Get an instance of the [SODeviceManager](#) Class
- b. Perform a scan and connect OR perform a "direct connection" to a spirometer
- c. Start the test (user must start blowing within the EOT timeout: default 15 sec)
- d. [soDeviceDidStopTest](#) delegate method is called if user doesn't blow within the EOT timeout (give to the user the ability to restart test manually)

It is strongly NOT recommended to call the Start test on the [soDeviceDidStopTest](#) delegate method as a workaround to contrast the End Of Test timeout effect. Instead call the start test only when the patient is ready to blow or increase the End of Test Timeout (see above: [Starting a test with a customized "End of Test timeout"](#))

- e. Use a visual feedback to prevent that user start blowing before the app has received the delegate method [soDeviceDidStartTest](#)

- f. Use [didUpdateVcVolumeTimePoint](#) delegate method to show the animated feedback
- g. Before the [ventilatoryProfilePerformed](#) delegate method is called prompt the user to breath normally (tidal breathing / ventilatory profile)
- h. After the [ventilatoryProfilePerformed](#) delegate method is called, (patient's ventilatory profile is acquired) prompt the user to start a deep and slow insp/exp maneuver.
- i. Use [ResultsVcUpdated](#) to show the result (this method might not be called if the device was not able to calculate the results)
- j. [soDeviceDidStopTest](#) delegate method is called: the test is over

It is strongly NOT recommended to disconnect the device ([DeviceManager disconnect](#)) at the end of each session or test. The bluetooth disconnection of the device should be called only if no more spirometry test need to be performed in a short time (less than 1 minute). Even better is to disconnect when the app became inactive

End Of Test Criteria

During a spirometry manoeuvre the **End of Test** is detected by the spirometer

(and thus propagated by the FRAMEWORK) according to the following criteria:

FVC TEST

The FVC manoeuvre AUTOMATICALLY ends:

1. when an expiratory PLATEAU has been reached. The expiratory plateau is detected, by the spirobankSmart, when no significant volumes (< 20mL) have been measured within a timeframe of 3 seconds
2. when a significant inhaled volume is detected AND an exhalation has been performed.
3. when a timeout is expired. A timeout expires in the following conditions:
 - a. when the user has never been blowing for n seconds since the test was started. The value of n can be set by the app developer from 15 (which is the default) to 120 seconds
 - b. when the user stop blowing for 3 sec
 - c. when the user keep on blowing for 60 seconds and no plateau has been reached

FVC PLUS TEST

The FVC PLUS maneuver AUTOMATICALLY ends:

1. when a timeout is expired. A timeout expires in the following conditions:
 - a. when the user has never been blowing for n seconds since the test was started. The value of n can be set by the app developer from 15 (which is the default) to 120 seconds
 - b. when the user stop blowing for 3 sec
 - c. After 60 seconds despite the user is blowing

Note: According to ATS/ERS guidelines the framework, during the **FVCPlus** maneuver, calls the delegate method [didReceiveEndOfForcedExpirationIndicator](#) to notify that one EOFE criteria has been achieved

PEAKFLOW/FEV1 test

The PEAKFLOW/FEV1 manoeuvre AUTOMATICALLY ends:

1. when the spirometrySmart detects a volume < 200mL AND a flow < 300mL/s within a timeframe of 2 seconds [here the test is AUTOMATICALLY RESTARTED]
2. when a significant inhaled volume is detected [here the test is AUTOMATICALLY RESTARTED]
3. when a timeout is expired. A timeout expires in the following conditions:
 - a. when the user has never been blowing for n seconds since the test was started. The value of n can be set by the app developer from 15 (which is the default) to 120 seconds
 - b. when the user stop blowing for 3 sec [here the test is AUTOMATICALLY RESTARTED]
 - c. when the user keep on blowing for 60 seconds AND none of the previous condition has been met

Note: To have an acceptable PEAKFLOW/FEV1 manoeuvre, the exhalation must last no less than 1 seconds

The SVC Test Guide

OVERVIEW

The SVC Plus, includes the following features:

- Slow Vital Capacity with both expiratory and inspiratory maneuvers
- Disposable and reusable turbine supported
- Measured volume and time in real time
- Volume-Time curve points (at the end of each test)
- Measured, predicted, LLn and zScore values for the following parameters:
 - o *evc*
 - o *ivc*
 - o *ic*
 - o *slow Expiratory /Inspiratory Time_s*

Depending on the Author of the reference equations the predicted, LLN and zScore values may not be provided for some parameters

*Depending on how the maneuver is performed, one parameter among EVC and IVC is always = 0
Also the IC can be = 0 when the SCV test is performed without the initial Tidal breathing (inhalation and exhalation during restful breathing)*

Spirometers involved

The spirometers that support the SVC are the following

- Spirobank OXI running firmware version ≥ 4.3
- Spirobank Smart running firmware version ≥ 4.3

Enabling the SVC PLUS test

The firmware ≥ 4.3 (for spirobank Smart and spirobank Oxi) can be provided as already enabled to the SVC or as disable to the SCV plus. An "SVC disabled" device can be enabled using a dedicated method (see below)

The firmware < 4.3 for spirobankSmart cannot be enabled (update is necessary)

Note: the Spirobank Smart running firmware version < 4.0 cannot be updated to firmware version ≥ 4.3

The framework provides the following methods:

A method to check if the SVC PLUS test is enabled on the connected spirometer

`-(void) isDeviceEnabledToSvc:(void (^)(CheckState checkState)) checkStateCompletion`

A method to enable the connected spirometer to the FVC PLUS test

`-(void) enableSvcWithPassCode:(NSString *)passCode completeBlock:(void (^)(BOOL isSuccess, NSError *error))boolCompletion;`

The passcode (or password) is provided by MIR and is based on the spirometer's serial number.

The SVC PLUS REAL TIME TEST

Once a supported spirometer has been discovered and connected use the following method of the SODevice class To start the FVC Plus test:

```
- (void)startTestWithTestType:(SOTestType)testType endOfTestTimeout:(Byte)timeoutInSeconds  
turbineType:(SOTurbineType)turbine ambientTemperatureCelsius:(Byte) celsiusDegree;
```

passing SOTestType.SVC as the testType

The EndOfTest timeout is specified by the parameter "timeoutInSeconds"

the EOT timeout is the number of seconds after which the test is automatically ended by the spirometer if the user was not been blowing at all since the test started

The valid range for the "timeout" is 15s - 120s.

- * If a value < 15 is passed, the spirometer sets the timeout to 15s.
- * If a value > 120 is passed, the spirometer sets the timeout to 120s

the turbine type (Reusable or Disposable) affects the values of the results.

The total test duration is 60 seconds. After that the device quits from the test (the user should be warned about it).

A new argument `ambientTemperatureCelsius` was added for the compliance with ATS2019 guidelines (If you don't want to pass the temperature just call the version of this method without this argument)

During the real time test

The SVC test usually starts with a tidal breathing (inhalation and exhalation during restful breathing)

After 3 similar breaths SODevice calls the delegate method
`(void)didPerformVentilatoryProfile:(SODevice *)soDevice;`

This

As soon as the above method is called, the app must advice the user to take a "Deep breath in, then breath all the way out".

A delegate method is called for each volume point detected by the device:

```
- (void)soDevice:(SODevice *)soDevice didUpdateVcVolumeTimePoint:(volumeTimePoint *)vcVtPoint isFirstPackage:(BOOL)isFirstPackage;
```

vcVtPoint includes volume_L and time_s.

the volume (in Liters) and the time (in seconds) ready to be plotted on the XY chart: vcVtPoint.volume_L is the Y point, vcVtPoint.time_s_L the X point

isFirstPackage is set to YES only when the first point of the test is passed.

The Animated Feedback

The methods provided by the **SOPatient** class (actualPercentageOfTargetWithFlow and predictedPercentageOfTargetWithFlow) in order to build an animated feedback during the FVC real time test cannot be used for the SVC test because they are based on flow.

The end of real time test

The test are stopped in two ways:

- when the stopTest method is invoked
- automatically, when the device/framework detects the EOT (End Of Test) criteria.

End Of Test Criteria

During a spirometry maneuver the **End of Test** is detected by the spirometer (and thus propagated by the FRAMEWORK) according to the following criteria: The FVC PLUS maneuver AUTOMATICALLY ends:

1. when a timeout is expired. A timeout expires in the following conditions:
 - a. when the user has never been blowing for n seconds since the test was started. The value of n can be set by the app developer from 15 (which is the default) to 120 seconds using the parameter `endOfTestTimeout` of the StartTest method
 - b. when the user stop blowing for 3 sec

Test Results

The real time SVC test ends automatically if a timeout expires (see above).

It can be ended anytime by calling the method `-(void)stopTest`

At the end of the test, the delegate method of the SODevice class is called to provide the results of the test

`-(void)soDevice:(SODevice *)soDevice didUpdateVcResults:(SODevice *)results;`

The SODevice class includes:

- The measured values of all supported parameters:
 - evc_L (Expiratory Vital Capacity in L), ivc_L_L (Insp Vital Capacity in L), ic (Insp capacity in L), slowExpInTime_s (expiratory time in sec)
- The Volume Time loop, including all volume points in Liters and time points in seconds

Predicted values

As soon as the SOPatient object is created, the predicted values and the LLN values are available to be read.

The zScore values are not available before the test result are provided. This because the measured values are needed to calculate the zScore.

So, to get the zScore values the following method of SOPatient class must be called AFTER the SODevice are received:

`-(void) CalculateZscore: (SODevice *_Nullable) results;`

The predicted (in L / Ls, s) , LLN (in L / Ls, s) and zScore (positive or negative number) values provided (as double) are the following:

double EVC_TargetValue;	double EVC_LLN;	double EVC_zScore;
double IVC_TargetValue;	double IVC_LLN;	double IVC_zScore;

double IC_TargetValue;	double IC_LLN;	double IC_zScore;
double SETSIT_TargetValue;	double fef2575LLN;	double fef2575_zScore;

The FVC Plus Test Guide

OVERVIEW

The FVC Plus, includes the following features:

- Forced Vital Capacity with both expiratory and inspiratory maneuvers
- Disposable and reusable turbine supported
- Measured flows and volumes in real time
- Flow-Volume points of the “best” loop (at the end of each test)
- Volume-Time curve points (at the end of each test)
- Measured, predicted, LLn and zScore values for the following parameters:

- o fvc
- o fev1
- o pef
- o fef2575
- o fev6
- o eVol (extrapolated volume)
- o pefTime (time to Peakflow)
- o fev1/fvc %
- o fef75
- o fet (flow expiratory time)
- o fef25
- o fef50
- o fivc
- o fiv1
- o pif
- o fev3
- o fev05
- o fev075
- o fev2
- o fef7585
- o fif25
- o fif50
- o fif75
- o fev1/fev6 %
- o fev6/fvc %
- o fiv1/fivc %
- o fev3/fvc %
- o fev05/fvc %
- o fev075/fvc %
- o fev2/fvc %
- o hesitationTime_s

depending on the Author of the reference equations the predicted, LLN and zScore values may not be provided for some parameters depending on how the maneuver is performed, the measured values for inspiratory parameters may not be provided

Spirometers involved

The spirometers that supports the FVC PLUS are the following

- Spirobank Smart running version ≥ 3.1
- Spirobank Smart OXI from version 1.0

Enabling the FVC PLUS test

The firmware 3.2 (for spirobankSmart) is natively enabled to the FVC PLUS

The firmware 1.0 for Spirobank OXI is natively enabled to the FVC PLUS

The firmware 3.1 (for spirobankSmart) is not natively enabled to the FVC PLUS but it can be enabled using a dedicated method

The firmware < 3.1 for spirobankSmart cannot be enabled (update is necessary)

The framework provides the following methods:

A method to check if the FVC PLUS test is enabled on the connected spirometer

`-(void) isDeviceEnabledToFvcPlus:(void (^)(CheckState checkState)) checkStateCompletion`

A method to enable the connected spirometer to the FVC PLUS test

`-(void) enableSvcWithPassCode:(NSString *)passCode completeBlock:(void (^)(BOOL isSuccess, NSError *error))boolCompletion;`

The passcode (or password) is provided by MIR and is based on the spirometer's serial number.

The FVC PLUS REAL TIME TEST

Once a supported spirometer has been discovered and connected use the following method of the SODevice class To start the FVC Plus test:

```
- (void)startTestWithTestType:(SOTestType)testType endOfTestTimeout:(Byte)timeoutInSeconds turbineType:(SOTurbineType)turbine ambientTemperatureCelsius:(Byte) celsiusDegree;
```

passing SOTestType.FVCPlus as the testType

The EndOfTest timeout is specified by the parameter "timeoutInSeconds"

the EOT timeout is the number of seconds after which the test is automatically ended by the spirometer if the user was not been blowing at all since the test started

The valid range for the "timeout" is 15s - 120s.

- * If a value < 15 is passed, the spirometer sets the timeout to 15s.
- * If a value > 120 is passed, the spirometer sets the timeout to 120s

the turbine type (Reusable or Disposable) affects the values of the results.

The total test duration is 60 seconds. After that the device quits from the test (the user should be warned about it).

A new argument ambientTemperatureCelsius was added for the compliance with ATS2019 guidelines (If you don't want to pass the temperature just call the version of this method without this argument)

During the real time test

During the test the user can perform as many inspired and expired loops as he/she likes.

A delegate method is called for each flow point detected by the device:

```
-(void)soDevice:(SODevice *)soDevice didUpdateFlowValue:(float)value isFirstPackage:(BOOL)isFirstPackage;
```

The flow value is provided in centiliters. isFirstPackage is set to YES only when the passed flow is the first one of the test.

The volume value can be calculated in the following way:

For each Expired Flow Point: Current Volume += volumeStep

For each Inspired Flow Point: Current Volume -= volumeStep

An overload of this method is available to get both the volume (in Liters) and the flow (in Liters per seconds) ready to be plotted on the XY chart.

```
-(void)soDevice:(SODevice *)soDevice didUpdateFvcPlusFlowVolumePoint:(flowVolmePoint *)
fvPoint isFirstPackage:(BOOL)isFirstPackage;
```

fvPoint.flow_Ls is the Y point, fvPoint.volume_L the X point

The Animated Feedback

For whom who wants to use an animated feedback “Smart One app’s style” - instead of the Flow-Volume curve – the following methods of the **SOPatient** class can be called.

```
-(float) actualPercentageOfTargetWithFlow:(float) flow volumeStep:(NSInteger) volumeStep
isFirstPackage:(BOOL)isFirstPackage
```

```
-(float) predictedPercentageOfTargetWithFlow:(float) flow volumeStep:(NSInteger) volumeStep
isFirstPackage:(BOOL)isFirstPackage;
```

CAUTION: THE FLOW MUST BE PASSED IN cL (centiliters)

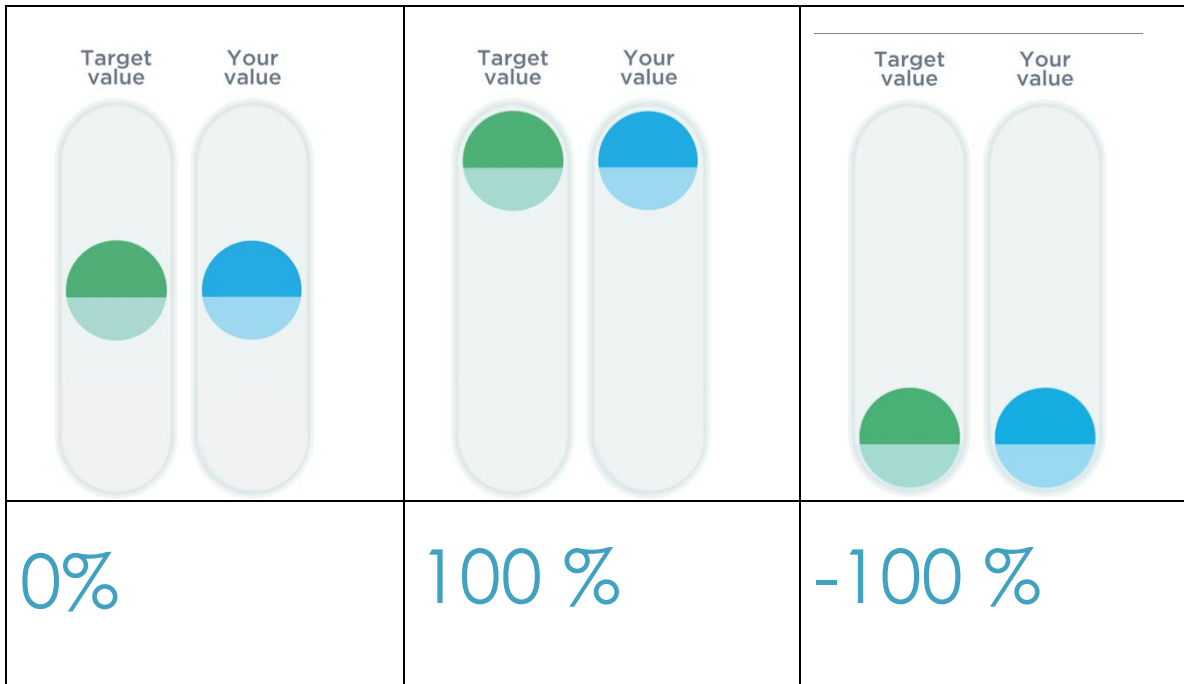
These methods retrieve -as percentage- the relative position of an object “moved” by the user’s blow into a container.

0% is the start position

The values between 1% and 100% indicate the relative positions of the object during the expiring maneuver

The values between -1% and -100% indicate the relative positions of the object during the

inspiring maneuver



Method [predictedPercentageOfTargetWithFlow](#) refers to the “Target value”: what the user should perform

Method [actualPercentageOfTargetWithFlow](#) refers to “Your value”: what the user is actually performing

Each method must be called for each flow point received

At the end of the test the balls should be positioned in the 0% position

The end of real time test

The test are stopped in two ways:

- when the stopTest method is invoked
- automatically, when the device/framework detects the EOT (End Of Test) criteria.

End Of Test Criteria

During a spirometry maneuver the **End of Test** is detected by the spirometer (and thus propagated by the FRAMEWORK) according to the following criteria: The FVC PLUS maneuver AUTOMATICALLY ends:

1. when a timeout is expired. A timeout expires in the following conditions:
 - a. when the user has never been blowing for n seconds since the test was started. The value of n can be set by the app developer from 15 (which is the default) to 120 seconds using the parameter `endOfTestTimeout` of the StartTest method
 - b. when the user stop blowing for 3 sec
 - c. After 60 seconds despite the user is blowing or not

According to the last ATS 2019 guidelines, a new delegate method was added in order to advice that EndOfForcedExpiration has been achieved

The method is: `-(void)soDevice:(SODevice *)soDevice didReceiveEndOfForcedExpirationIndicator:(EndOfForcedExpirationIndicator`

```
typedef NS_ENUM(NSUInteger, EndOfForcedExpirationIndicator) {
    PlateauReached,
    ExpiratoryTimeReached
};
```

When this method is called the FVC plus trial can be ended.

Test Results

The real time FVC Plus test ends automatically after 60 seconds.

It can be ended anytime by calling the method `-(void)stopTest`

At the end of the test, the delegate method of the SODevice class is called to provide the results of the test

```
-(void)soDevice:(SODevice *)soDevice didUpdateFvcPlusResults:(SODevice *)results;
```

The SOResultsFVCPlus includes:

- The measured values of all supported parameters:

pef_Ls (Exp Peakflow L sec), fev1_L (Forced Exp Vol at 1st sec in L), quality (acceptability calculation), fvc_L (Forced Exp Capacity in L), fev1_fvc_pcnc (Fev1% in percentage), fev6_L (Forced Exp Volume at 6th sec in L), fef2575_Ls (Max mid-expiratory flow in L sec), eVol_mL (Extrapolated volume in mL), pefTime_ms (time to reach Peakfow in milliseconds), fef75_L (Max mid-expiratory flow in L sec), fet_cs (flow expiratory time in sec * 100), fef25_Ls (Max mid-expiratory flow in L sec), fef50_Ls (Max mid-expiratory flow in L sec), fivc_L (Forced Insp Capacity in L), fiv1_L (Forced Insp Vol at 1st sec in L), pif_Ls (Insp Peakflow L sec), fev3_L (Forced Insp Vol at 3rd sec in L), fev05_L (Forced Insp Vol at 0.5th sec in L), fev075_L (Forced Insp Vol at 0.75th sec in L), fev2_L (Forced Insp Vol at 2nd sec in L), fef7585_Ls (Max mid-expiratory flow in L sec), fif25_Ls (Max mid-expiratory flow in L sec), fif50_Ls (Max mid-expiratory flow in L sec), fif75_Ls (Max mid-expiratory flow in L sec), fev1_fev6_perc , fev6_fvc_perc, fiv1_fivc_perc, fev3_fvc_perc, fev05_fvc_perc, fev075_fvc_perc, fev2_fvc_perc

- The best Flow Volume loop, including all volume points in Liters and flow points in Liters per seconds
- The Volume Time loop, including all volume points in Liters and time points in seconds

Get the FVC curve points at the highest resolution

From SDK version 3.1.0, when connected to a Spirobank smart supporting this feature (firmware >= 4.6) , you can get the **expired only** curve points of the last FVC PLUS test performed at the resolution of 100Hz (one point each 10 milliseconds).

From the SDK version 4.1.0 when connected to a Spirobank smart supporting this feature (firmware >= 4.7), you can get the **expired and the inspired** curve points of the last FVC PLUS test

Note that the high resolution curve points ARE NOT automatically retrieved with the SOResultsFvcPlus object passed by the didUPdateFvcPlusResults delegate method .

To get the high resolution expired curve points the method [getHighResolutionCurveForLastSpirometryTest](#) of the SODevice object must be called. It must be called AFTER receiving the didUPdateFvcPlusResults call back

To get the high resolution expired and **inspired** curve points the method [getHighResolutionCurveForLastSpirometryTestWithInspiration](#) of the SODevice object must be called. It must be called AFTER receiving the didUPdateFvcPlusResults call back

This last method (withInspiration) is slower than the previous method (only expiration) so if inspiration points are not needed it is recommended to call the previous one (only

expiration)

When one of the above-mentioned methods is called (only expiration or expiration and inspiration) the high resolution curve points are provided by the delegate method `didUpdateHighResolutionCurvePoints` (`SODevice` class)

The above the delegate method retrieves a collection of `CurvePoint` objects having, each one, as a properties `Flow`, `Volume` and `Time`

Predicted values

As soon as the `SOPatient` object is created, the predicted values and the LLN values are available to be read.

The `zScore` values are not available before the test result are provided. This because the measured values are needed to calculate the `zScore`.

So, to get the `zScore` values the following method of `SOPatient` class must be called AFTER the `SOResultsFvcPlus` are received:

-(void) CalculateZscore: (SOResultsFvcPlus * _Nullable) results;

The predicted (in L / Ls, s) , LLN (in L / Ls, s) and `zScore` (positive or negative number) values provided (as double) are the following:

double peakFlowTargetValue;	double fev1LLN;	double FEV1_Zscore;
double fev1TargetValue;	double peakFlowLLN;	double peakFlow_zScore;
double fvcTargetValue;	double fvcLLN;	double fvc_zScore;
double fev1_fvc_TargetValue;	double fev6LLN;	double fev6_zScore;
double fev6TargetValue;	double fef2575LLN;	double fef2575_zScore;
double fef2575TargetValue;	double fev1_fvc_LLN;	double fev1_fvc_zScore;
double FEF75TargetValue;	double FEF75LLN;	double FEF75_zScore;
double FETTargetValue;	double FETLLN;	double FET_zScore;
double FEV3TargetValue;	double FEV3LLN;	double FEV3_zScore;
double FEV1_FEV6TargetValue;	double FEV1_FEV6LLN;	double FEV1_FEV6_zScore;

double FEF25TargetValue;	double FEF25LLN;	double FEF25_zScore;
double FEF50TargetValue;	double FEF50LLN;	double FEF50_zScore;
double FIVCTargetValue;	double FIVCLLN;	double FIVC_zScore;
double FIV1TargetValue;	double FIV1LLN;	double FIV1_zScore;
double FIV1_FIVCTargetValue;	double FIV1_FIVCLLN;	double FIV1_FIVC_zScore;
double PIFTargetValue;	double PIFLLN;	double PIF_zScore;
double FEV3_FVCTargetValue;	double FEV3_FVCLLN;	double FEV3_FVC_zScore

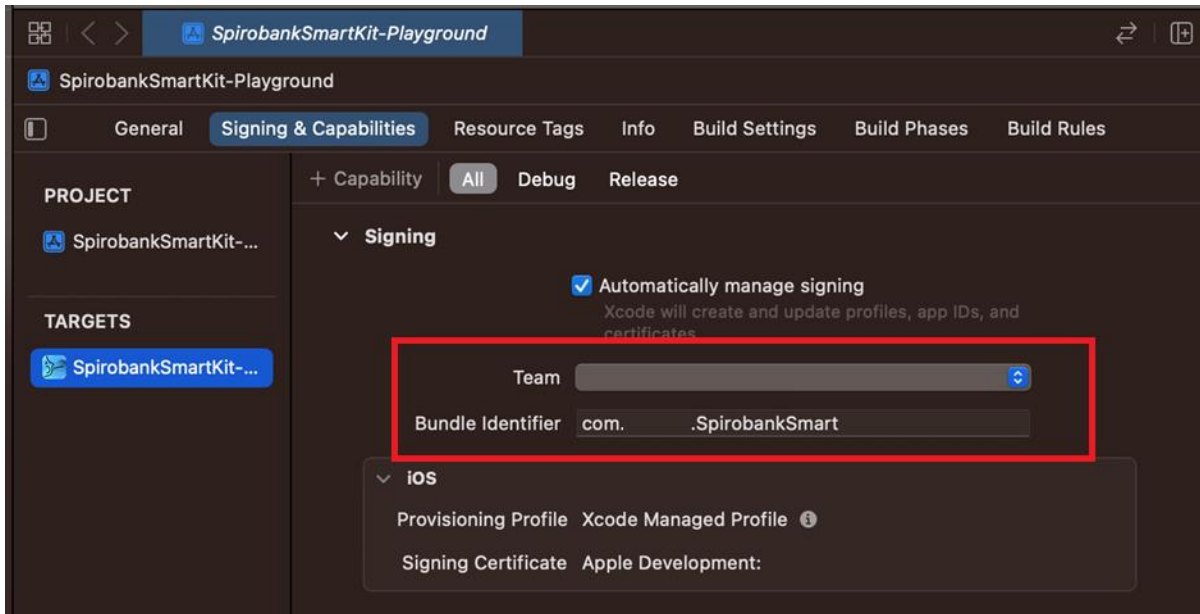
SAMPLE DEMO APPLICATION

The archive contains a subfolder named " Sample App with A.B.C UNIVERSAL FULLBITCODE".

This folder contains an XCode project.

To use it:

- a) Open the file SpirobankSmartKit-Playground.xcodeproj with Xcode.
- b) In Xcode, in the project settings, in the tab “Signing and Capabilities”, select your own Apple certificate:



Edit also the “Bundle identifier” to set a unique identifier. You can for example use “com.your-company-name.SpirobankSmart”.

c) Build the App and start it on a device.

The App allows you to scan the MIR Bluetooth Low Energy and connect to a MIR Device:

11:21 📶 📶 🔋

[Disable logs](#) **Scan** [Stop](#)

Spirobank Smart SM-009-Z115161

SPIROBANK OXI SE-011-E000368

Spirobank Smart SM-005-Z008091

Spirobank Smart SM-005-Z016908

11:22 📶 📶 🔋

[Reset Device \(disable ALL\)](#)

Device S/N: Z016908 G1

Volume step: 50

Firmware ver.: 3.3

Bluetooth ver.: 3.3

Battery level: 79

ConnectionTime(s): 0.0

No previously connected device

Device ID:
95C7B8BD-FAB9-3020-8E5D-C863CE89F502



d) The apps also allow you to perform FVC, PEF, FVC PLUS, VC and OXI tests:

11:22 📶 🔋

FVC PEF FVC PLUS VC Start

Reset Zoom
get HR curve

disposable turbine

33 175 80

PEF: 371 L/m | FEV1: 3.72 L
FVC: 4.31 L | FEV1/FVC: 86.31
FEV6: 4.31 L | FEF2575: 3.83 L
[SOQualityMessageBlowOutLonger](#)

0 0 0 0 0 1 1 1 1 1 1 0 0 1 1 0 0 1 1 0 0

Scan Connect Spiro Oxi / Ecg Update

11:22 📶 🔋

OXI ECG Start

Command Not Supported

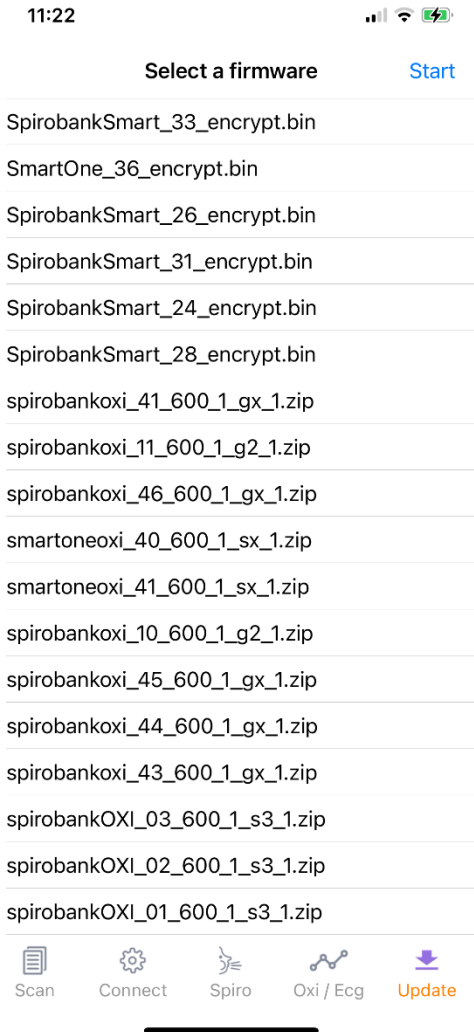
SpO2 (%) Heart Rate (bpm)

signal:

Filter:

Scan Connect Spiro Oxi / Ecg Update

e) The app also allows you to upgrade the firmware of the device:



Additional Resources

- MIR Website: <https://www.spirometry.com>

Troubleshooting

1. Provisioning Profile Error: No valid provisioning profiles found for this device.

Solution: Ensure you select the correct Provisioning Profile in your project settings.

Bundle Identifier Error:

2. Bundle Identifier 'com.spirometry.SpirobankSmart has already been used.

Solution: Modify the Bundle Identifier in your project settings to make it unique.

3. Undefined symbols for architecture x86_64..."

Solution: Ensure that the architectures of your project and dependencies are compatible.

Building for 'iOS-simulator', but linking in dylib (<your-path>/MIR_PLUS_SDK_SpirobankSmart_Smartone_IOS_A.B.C/Sample App with A.B.C.D UNIVERSAL FULLBITCODE/SpirobankSmartKit-Playground/MirSmartDevice.framework/MirSmartDevice) built for 'iOS'

Solution: Ensure you start the application on a real Apple device. The app cannot be started on a simulator.

Annex B. Instruction for Use - MIR SUPER SDK Windows

Introduction

The purpose of this guide is to facilitate use of the tool **SUPER SDK (Software Development Kit)** for the rapid development of applications running on windows for monitoring and storing patients, archives of spirometry and oximetry tests obtained through MIR devices.

MIR SSDK Windows allows you to quickly perform all the necessary operations to use Spirometry or Oximetry.

It consists of various libraries as described below:

- **MirBLE:** Allows to establish a complete communication with the device through Bluetooth Low Energy (BLE) by sending it all the necessary commands (tests, calibration, etc.)
- **MirChartingCore:** Contains all the functions necessary for the creation and management of graphs.
- **MirCommunication:** Allows to manage communication with all devices, regardless of the protocol.
- **MirDataTypes:** Contains all the properties (in the form of Enums and methods) for all the elements necessary for Spirometry or for the Oximetry (for e.g., spirometry parameters, ethnic groups codes, etc.)
- **MirDeviceManager:** Oversees the management of the communication of MIR devices.
- **MirInterpretation:** Analyze a session and return its interpretation.
- **MirWpfUtilities:** Utilities for User Interface on the SampleApp Demo.
- **MirWspNET:** Manage the USB low level communication.

Prerequisites

Before you begin integrating the Windows SDK into your application, make sure you have the following in place:

.NET Core: Ensure you have a functional installation of Visual Studio on your system. The Windows SDK is primarily developed in C#, so the .NET runtime version 4.0+ is required.

Windows Version: Our SDK is compatible with Windows 7 (Version 1702) and later versions. Using an older version does not guarantee the functionality of all features.

Visual Studio installed on Windows

Integration (How to implement?)

To implement the SDK, you have 2 options:

- We provide the **WindowsSdkSampleApp** application which allows you to open a "Sample Demo Application" in Microsoft Visual Studio. This project enables you to perform the main operations and see how to implement the various classes and methods.
- You can follow the chapter "Integration into a Microsoft Visual Studio project" to learn the procedure to use it in your project."

Sample Demo Application

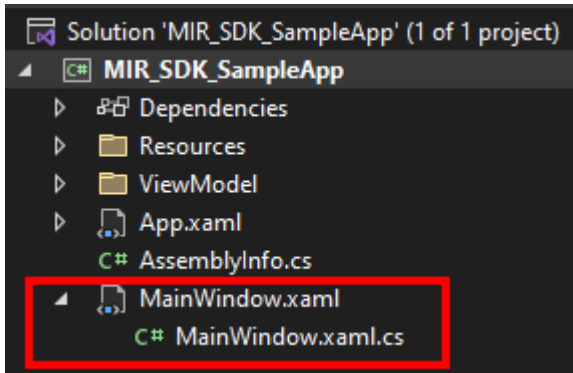
The "mir_sdk_windows_<version>" archive contains a subfolder named "MIR SDK <version>\MIR SDK - SampleApp".

This folder contains a solution named "MIR_SDK_SampleApp.sln".

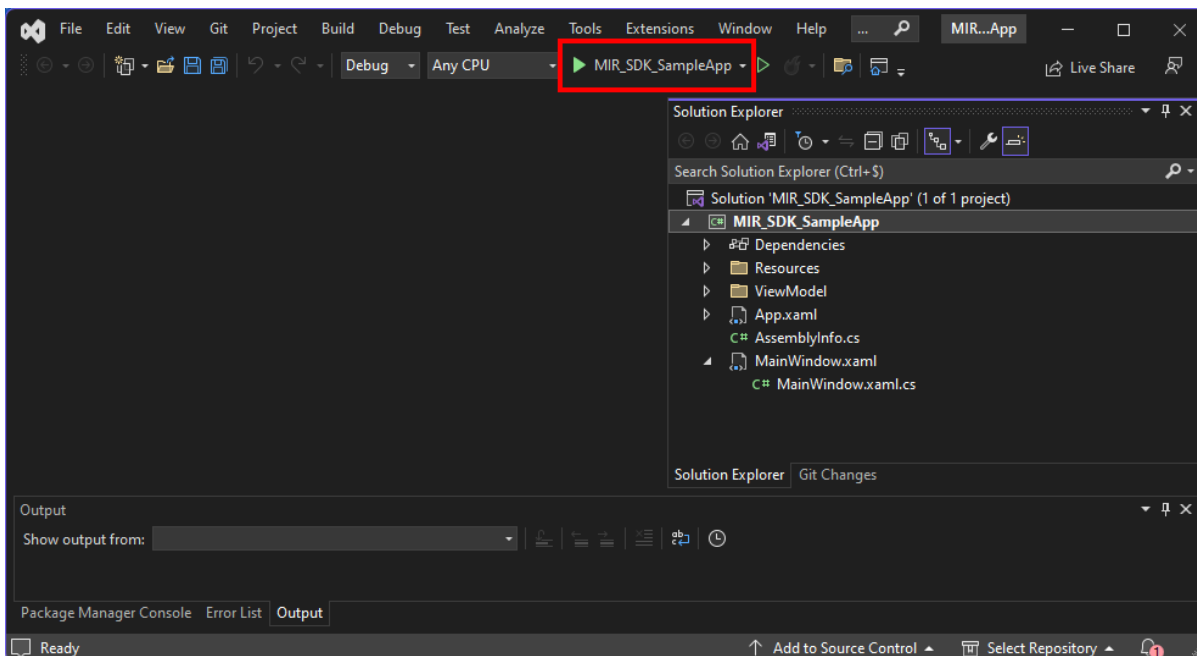
Open this file with Microsoft Visual Studio. Then, ensure you have .NET Runtime installed on your machine.

This is a .NET project with WPF that contains a single class named "MainWindow".

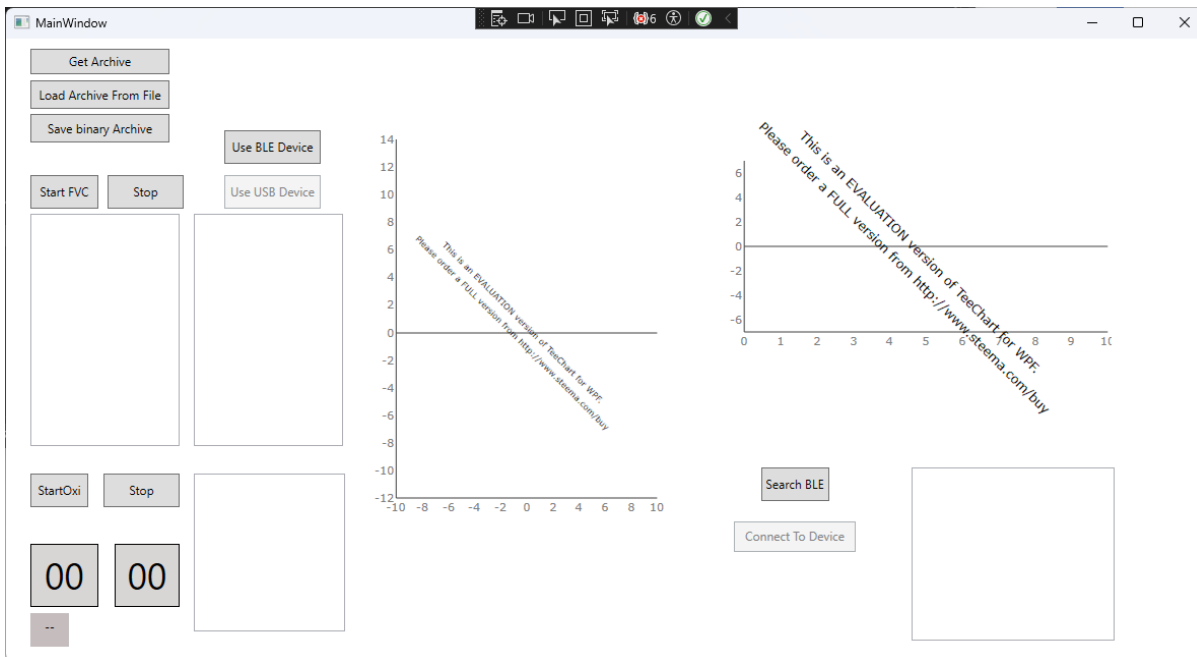
This class features a WPF graphical interface, allowing you to see the function called behind each button:



To run it, simply build the application using the "Play" icon:



After the build is executed, you'll see the following window:



Description of the various buttons:

Get Archive: Retrieves the archive (that is, the device's memory) and fills a progress bar to show the download progress.

Load Archive From File: Opens a window to select an archive and decode its content based on predefined theoretical authors.

Save Binary Archive: Saves the archive to a file.

Start FVC: Starts a Spirometry test.

Stop: Stops the FVC test already in progress.

Start Oxi: Starts an Oximetry test.

Stop: Stops the Oximetry test already in progress.

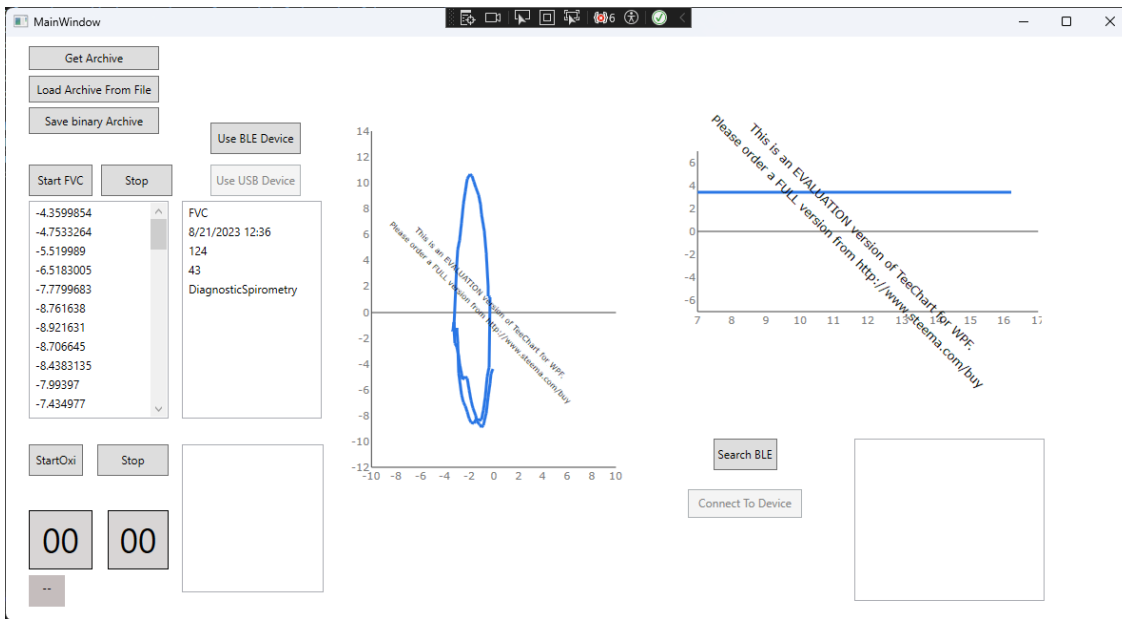
Labels 00 and 00: Values of current BPM and SpO2 when Oximetry test is running.

Use BLE Device / Use USB Device: "Allows you to select which communication method should be used.

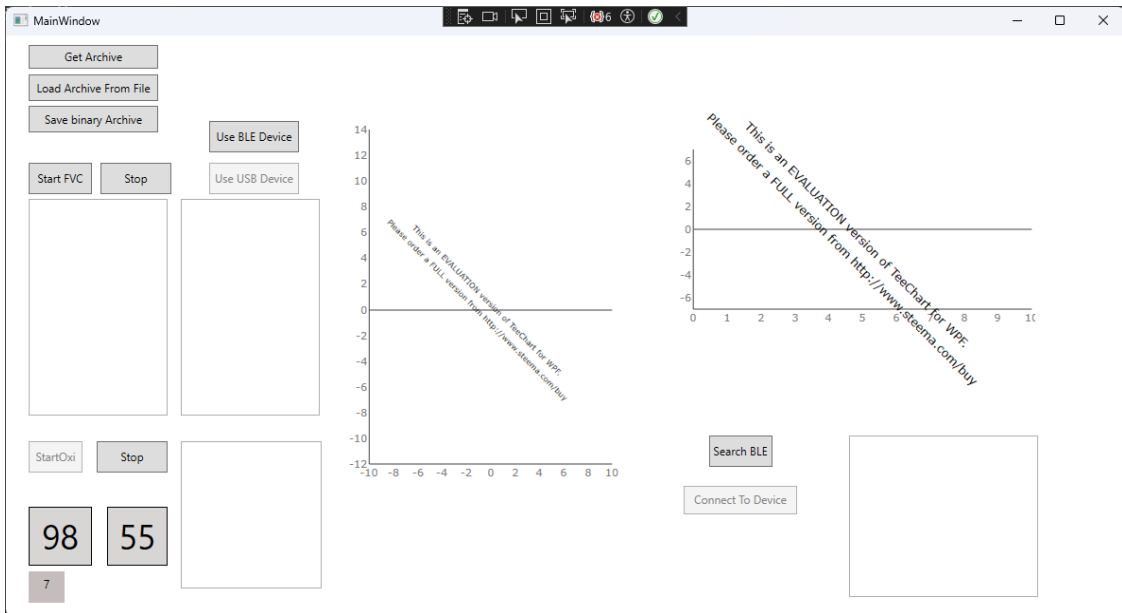
Search BLE: Searches the MIR BLE (Bluetooth Low Energy) devices available.

Connect To Device: Once a device is found and selected (by clicking on its name), the button will connect to the device.

Example of screen during a Spirometry session



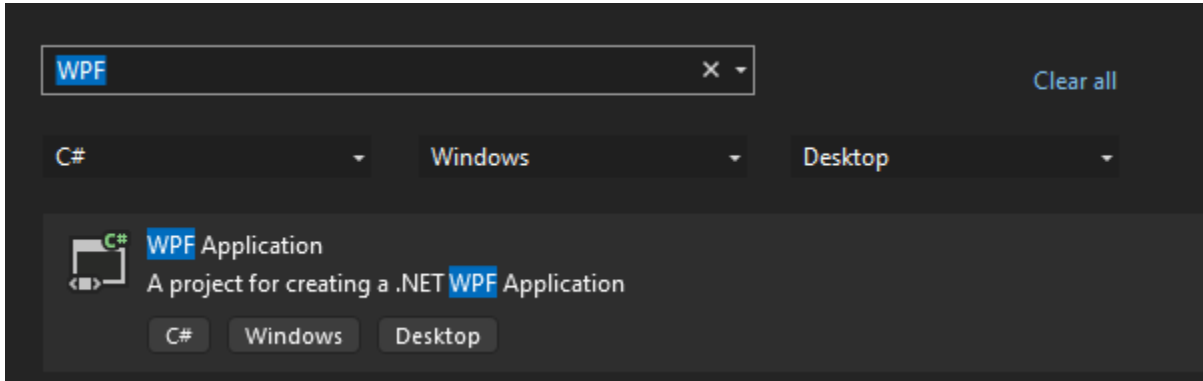
Example of screen during an Oximetry session



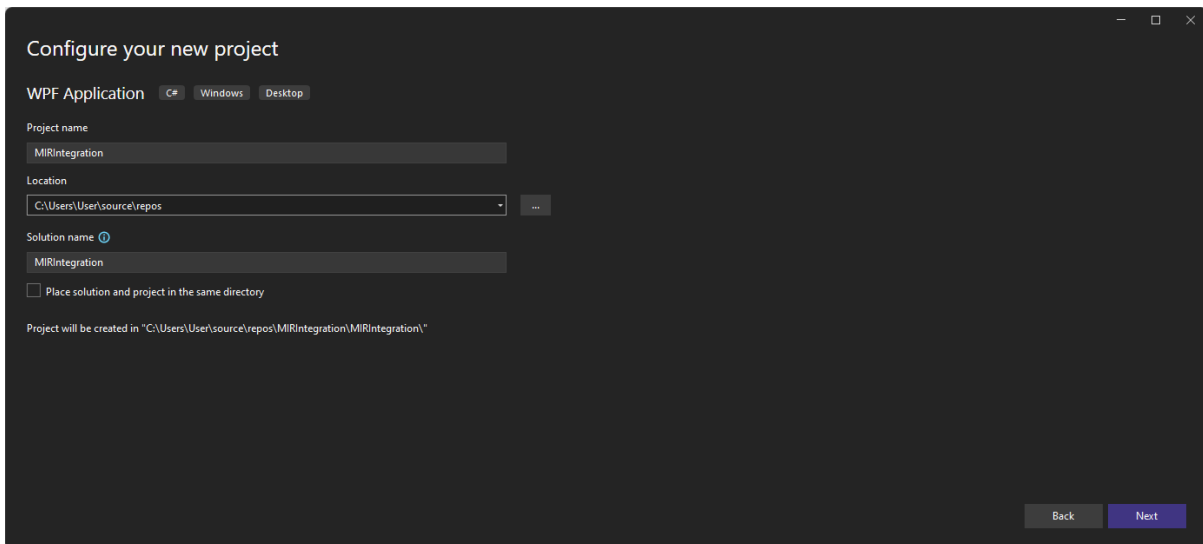
Integration into a Microsoft Visual Studio project

For this example, we'll create a blank WPF project with a simple button that will call the SDK to retrieve the file "MIRXFile".

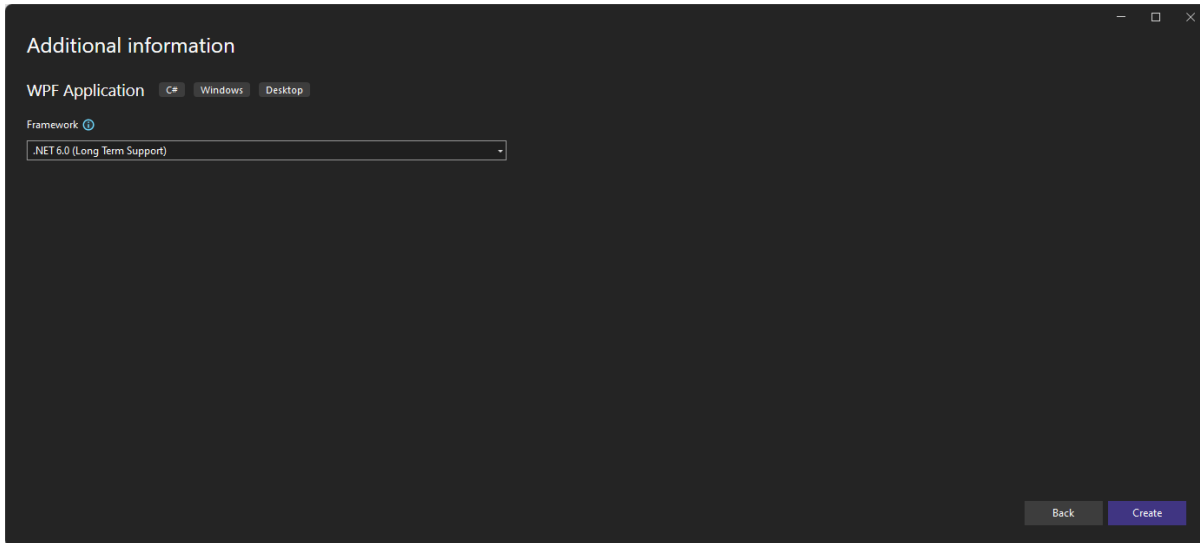
1) Open Microsoft Visual Studio:



2) Enter the project name and directory:

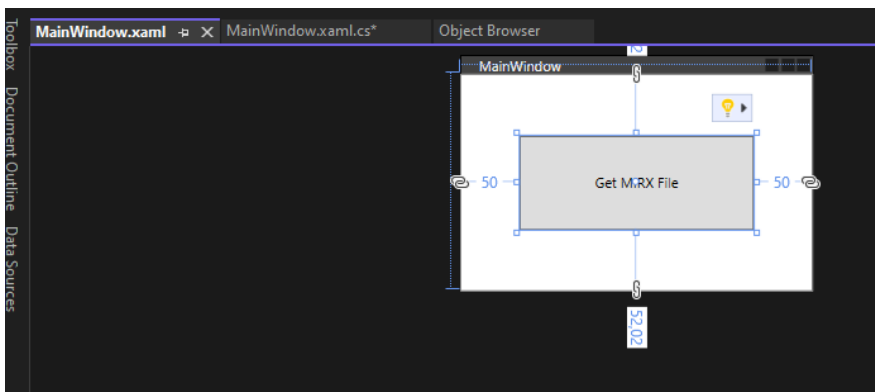


3) Select the .NET version (here version 6 - LTS):



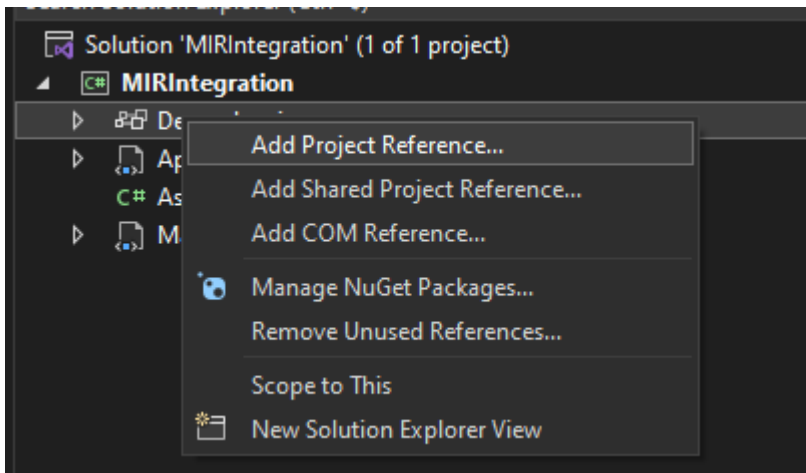
4) Open the MainWindow.xaml file and create a button with the following code:

```
<Grid>
  <Button
    x:Name="btnGetMirXFile"
    Width="200"
    Height="80"
    Click="btnGetMirXFile_Click">
    Get MIRX File
  </Button>
</Grid>
```

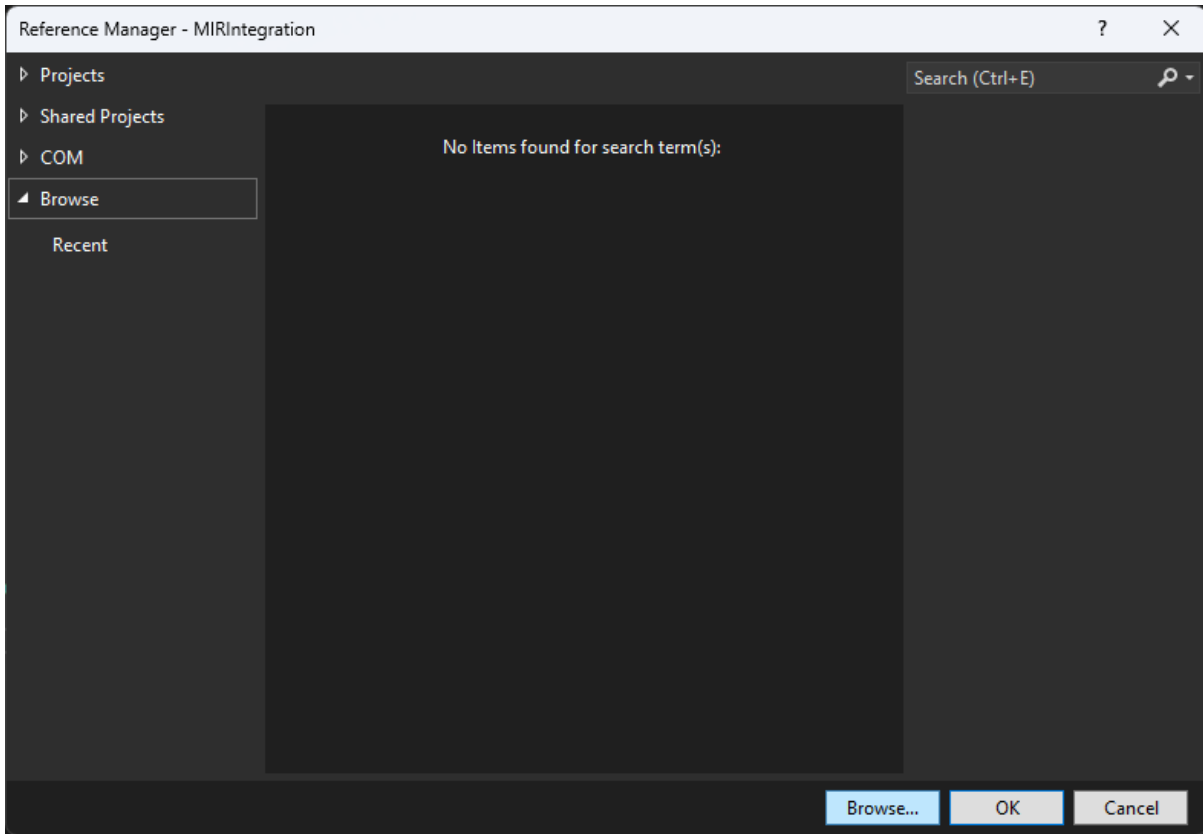


It is now necessary to add references to the SDK.

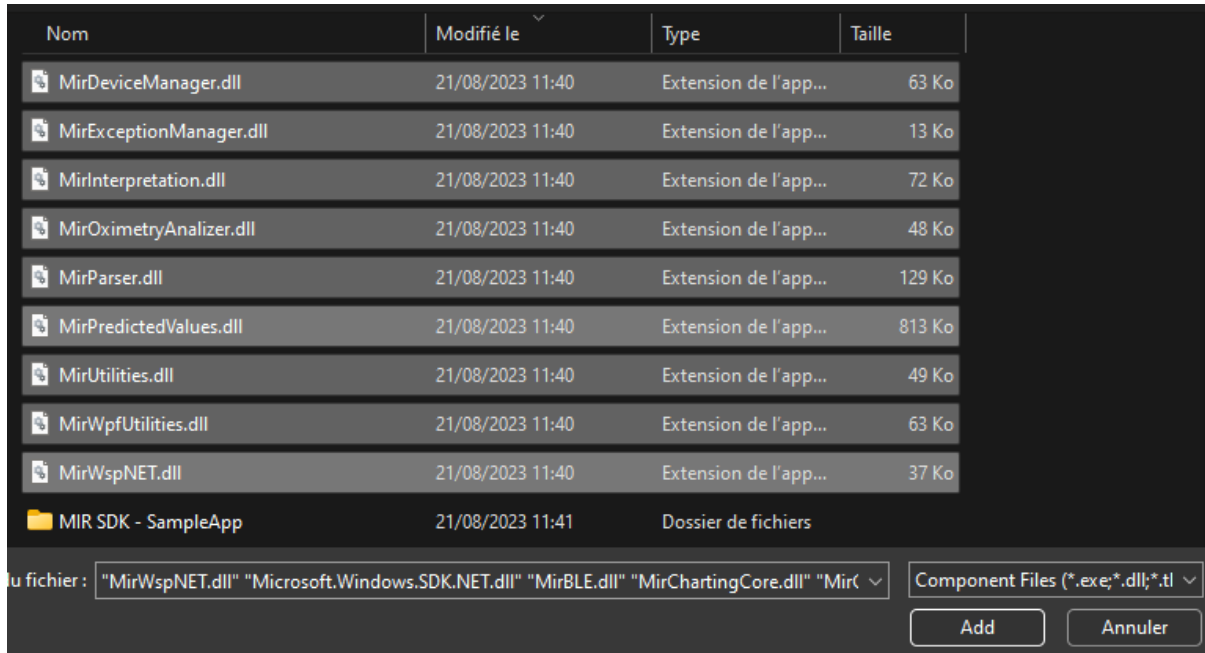
5) Click on the project name (top right in Visual Studio) and click on “Add Project Reference...”:



6) Press "Browse":



7) Select all .dll files provided in the "MIR SDK <version>" folder:



8) Click on "Add" and then "OK".

9) Now add the code to retrieve the archive:

```
using MirDeviceManager;
using MirInterpretation;
// ...

namespace MIRIntegration
{
    public partial class MainWindow : Window
    {
        private UsbManager USBManager;
        private MirDevice myDevice;

        public MainWindow()
        {
            InitializeComponent();

            USBManager = UsbManager.GetInstance();
        }

        private void btnGetMirXFile_Click(object sender, RoutedEventArgs e)
        {
            try
```

```

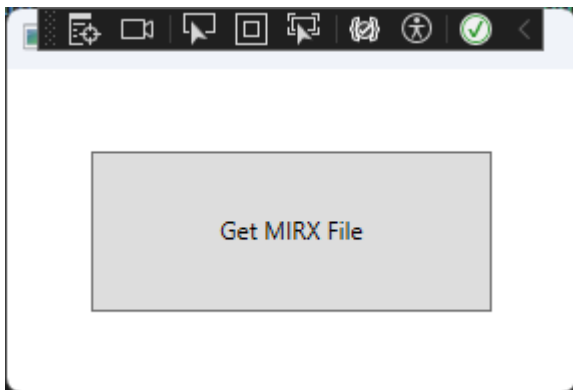
        {
            myDevice = USBManager.GetDeviceConnected();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
            return;
        }
        myDevice.OnGetMirXFileProgress += MyDevice_OnGetMirXFileProgress; ;
        myDevice.OnGetMirXFileComplete += MyDevice_OnGetMirXFileComplete; ;
        myDevice.GetMirXFile(MirInterpretationConfiguration.GenerateDefault());
    }

    private void MyDevice_OnGetMirXFileComplete(object? sender, GetMirXFileCompleteArgs
e)
    {
        throw new NotImplementedException();
    }

    private void MyDevice_OnGetMirXFileProgress(object? sender, GetMirXFileProgressArgs
e)
    {
        throw new NotImplementedException();
    }
}
}

```

10) Launch the application and conduct the test.



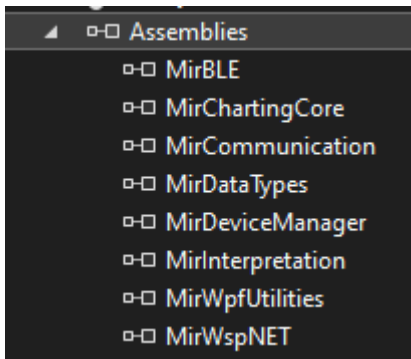
You can now use the full SDK in your project.

Main features

The list below provides an example for each of the main functions, namely:

- Connecting to the device.
- Conducting a test.
- Retrieving results.
- Retrieving and interpreting an archive.
- Updating the device.
- Save the device archive as a .mir or .mirx file.

The other available functions can be accessed from Microsoft Visual Studio:



Downloads the archive from a MIR device via USB or BLE (Bluetooth Low Energy)**1) Connect to the device**

The device supports 2 types of connections:

- **USB**: Requires a connection via USB and the Windows driver installed on your computer.
- **BLE**: Requires a computer that supports Bluetooth Low Energy (BLE) and a "BLE" device.

1.1) USB connection

a) Create an instance of the USBManager class to open the connection

```
USB = UsbManager.GetInstance();
```

b) Create an instance of the MirDevice class to connect the device

```
MirDeviceManager.MirDevice myDevice;  
myDevice = USB.GetDeviceConnected();
```

1.2) BLE connection

a) Create an instance of the BleManager class to open the connection:

```
BleManager = BluetoothLowEnergyManager.GetInstance()
```

b) Monitor events during discover operation:

Subscribe event DeviceDiscovered and DiscoverComplete:

```
BleManager.DeviceDiscovered += DeviceDiscovered;  
BleManager.DiscoverComplete += DiscoverComplete;
```

To get more information about BLE connection operation subscribe also to the following:

- onBleDeviceConnected
- onBleDeviceDisconnected
- onBluetoothStateOn
- onBluetoothStateOff
- onBleDeviceConnectionError

```
BleManager.RegisterToBleEvents(OnBleDeviceConnected, OnBleDeviceDisconnected,  
onBluetoothStateOn, onBluetoothStateOff, OnBleDeviceConnectionError)
```

Unregister as following:

```
BleManager.UnregisterToBleEvents(OnBleDeviceConnected, OnBleDeviceDisconnected,  
onBluetoothStateOn, onBluetoothStateOff, OnBleDeviceConnectionError);
```

c) Start discovery method:

```
BleManager.StartDiscovery()
```

To force discovery interruption call `BleManager.StopDiscovery()` method.

d) Connect to discovered device.

```
BleManager.Connect(MirCommunication.BLEDeviceAdvertisement.id)
```

2) Download the data from the connected device and generate the data Archive (sessions list)

a) Start downloading the tests from the device:

```
myDevice.GetArchive(MirInterpretationConfiguration.GenerateDefault(PredictedValuesProvider.Knudson_Knudson));
```

The GetArchive method features as an optional subject matter the “GenerateDefault” method of the namespace “MirInterpretationConfiguration”.

The “GenerateDefault” method allows you to establish which predicted value to use for interpreting the imported data.

The implemented predicted values are:

ATS/ERS:

- Knudson – Knudson
- ERS (ECCS) / Knudson
- Crapo & Bass / Knudson
- ERS (ECCS) / Zapletal
- Barcellona / Zapletal

GLI:

- Caucasian
- African Descendent
- North East Asian
- South East Asian
- Others

b) Monitor events during the operation for generating the sessions Archive

Downloading of data from device complete:

```
myDevice.OnEndDownloading += new EventHandler(OnArchiveDownloaded);  
void OnArchiveDownloaded(object sender, EventArgs e)  
{  
    label11.Text = "Start Archive .....";  
}
```


Interpretation of data downloaded from device complete:

```
myDevice.OnEndParsing += new EventHandler(OnArchiveParsed);
void OnArchiveParsed(object sender, EventArgs e)
{
    label11.Text = "Finalize Parsing .....;
}
```

Data loading status in sessions Archive:

```
myDevice.OnGetArchiveProgress += new
EventHandler<GetArchiveProgressArgs>(OnGetArchiveProgress);
```

(This event is normally used to manage the progress of the import)

```
void OnGetArchiveProgress (object sender, GetArchiveProgressArgs e)
{
    progressBar1.Value = e.Value;
}
```

Loading of the sessions Archive complete:

```
myDevice.OnGetArchiveComplete += new
EventHandler<GetArchiveCompleteArgs>(OnGetArchiveComplete);
void OnGetArchiveComplete (object sender, GetArchiveCompleteArgs e)
{
    label11.Content = "Parsing Complete"
}
```

3) Reading of the sessions archive when the operation has been completed

```
myDevice.OnGetArchiveComplete += new
EventHandler<GetArchiveCompleteArgs>(OnGetArchiveComplete);
void OnGetArchiveComplete (object sender, GetArchiveCompleteArgs e)
{
    int sessionsNumber = e.Archive.Sessions.Count
}
```

Conduct the FVC test, generate the graph during the test and receive the results

1) Connect to the device

1.1) USB

1. Create an instance of the USBManager class to open the connection:

```
USB = UsbManager.GetInstance();
```

2. Create an instance of the MirDevice class to connect the device:

```
MirDeviceManager.MirDevice myDevice;  
myDevice = USB.GetDeviceConnected();
```

1.2) BLE

1. Create an instance of the MirDevice class

```
MirDeviceManager.MirDevice myDevice;
```

2. Connect the device (see paragraph 1.2 BLE connection)

```
myDevice =BleManager.Connect(MirCommunication.BLEDeviceAdvertisement.id)
```

2) Conduct the FVC test

- a) Start the FVC test

```
myDevice.StartTest(TestType.FVC, TurbineType.Disposable);
```

- b) Receive the flow/volume data during the test.

- Sign the test execution event to receive the flow/volume data:

```
myDevice.OnFvcFlowVolume += MyDevice_OnFvcFlowVolume;
```

- Reading of data during test event:

```
void MyDevice_OnFvcFlowVolume(object sender, FlowAndVolumeArgs e)  
{  
    FVchart.AddPointToLine(e.Flow, e.Volume);  
}
```

- c) Receive the volume/time data during the test:

- Sign the test execution event to receive the volume/time data:

```
myDevice.OnFvcVolumeTime += MyDevice_OnFvcVolumeTime;
```

- Reading of data during test event:

```
MyDevice_OnFvcVolumeTime(object sender, VolumeAndTimeArgs e)
{
    VTchart.AddPointToLine(e.Volume, e.Time);
}
```

- d) Receive results at the end of the test:
- Sign the test completed event:

```
myDevice.OnFvcComplete += MyDevice_OnFvcComplete;
```

- Reading of data at the end of the test:

```
void MyDevice_OnFvcComplete(object sender, TrialSpiro e)
{
    lstCodLast.Items.Add(e.TrialType);
    lstCodLast.Items.Add(e.DateAndTime);
    lstCodLast.Items.Add(e.CurvePoints.Count);
    lstCodLast.Items.Add(e.Parameters.Count);
    lstCodLast.Items.Add(e.TrialSubType);
}
```

3) End the test

```
myDevice.StopTest()
```

Conduct the VC test, generate the graph during the test and receive the results

1) Connect to the device

1.1) USB

a) Create an instance of the USBManager class to open the connection

```
USB = UsbManager.GetInstance();
```

b) Create an instance of the MirDevice class to connect the device

```
MirDeviceManager.MirDevice myDevice;  
myDevice = USB.GetDeviceConnected();
```

1.2) BLE

a) Create an instance of the MirDevice class:

```
MirDeviceManager.MirDevice myDevice;
```

Connect the device (see paragraph 1.2 BLE connection)

```
myDevice = BleManager.Connect(MirCommunication.BLEDeviceAdvertisement.id)
```

2) Conduct the VC test

a) Start the FVC test:

```
myDevice.StartTest(TestType.VC, TurbineType.Disposable);
```

b) Receive the volume/time data during the test:

- Sign the test execution event to receive the volume/time data

```
myDevice.OnVcVolumeTime += MyDevice_OnVcVolumeTime;
```

- Reading of data during test event

```
private void MyDevice_OnVcVolumeTime(object sender, VolumeAndTimeArgs e)  
{  
    VTchart.AddPointToLine ((float)e.Volume, (float)e.Time);  
}
```

c) Start the breathing profile phase:

- Sign the start breathing profile event:

```
myDevice.OnVcVentilatoryProfile += MyDevice_OnVcVentilatoryProfile;  
private void MyDevice_OnVcVentilatoryProfile(object sender, EventArgs e)  
{  
    MessageBox.Show("Start VC test");  
}
```

- d) Receive results at the end of the test:
- Sign the test completed event:

```
myDevice.OnVcComplete += MyDevice_OnVcComplete;
```

Load the tests from the .mir or .mirx file

1) Select the .mir or .mirx file

- a) Create the subject OpenFileDialog:

```
Microsoft.Win32.OpenFileDialog dlg = new Microsoft.Win32.OpenFileDialog();
```

- b) Set the search filter for files with the extension .mir and .mirx:

```
dlg.Filter = "MIR Files | *.mir; *.mirx";
```

- c) View the file selection window through the ShowDialog function:

```
Nullable<bool> result = dlg.ShowDialog();
```

2) Interpret the content of the selected file

- a) Read the .mir or .mirx file:

```
string path = dlg.FileName;  
string extension = System.IO.Path.GetExtension(path);  
string filename = System.IO.Path.GetFileName(path);  
byte[] fileContent = System.IO.File.ReadAllBytes(path);
```

- b) Generate the virtual MIR device:

```
MirVirtualDevice myVirtualDevice;  
MirfileManager FileManager = new MirfileManager();  
myVirtualDevice = FileManager.GetVirtualDeviceFromMirFile(fileContent, filename,  
FileManager.GetFileType(extension));
```

3) Download the data from the connected device and generate the data Archive (sessions list)

a) Start downloading the tests from the virtual device

```
myVirtualDevice.GetArchive(MirInterpretationConfiguration.GenerateDefault());
```

b) Monitor events during the operation for generating the sessions Archive from the virtual device

- Data loading status in sessions Archive.

```
myVirtualDevice.OnGetArchiveProgress += new  
EventHandler<GetArchiveProgressArgs>(OnGetArchiveProgress);
```

(This event is normally used to manage the progress of the import)

```
void OnGetArchiveProgress (object sender, GetArchiveProgressArgs e)  
{  
    progressBar1.Value = e. Value;  
}
```

- Loading of the sessions Archive complete.

```
myVirtualDevice.OnGetArchiveComplete += new  
EventHandler<GetArchiveCompleteArgs>(OnGetArchiveComplete);
```

```
void OnGetArchiveComplete (object sender, GetArchiveCompleteArgs e)  
{  
    label11.Content = "Parsing Complete"  
}
```

4) Reading of the sessions archive when the operation has been completed

```
myVirtualDevice.OnGetArchiveComplete += new  
EventHandler<GetArchiveCompleteArgs>(OnGetArchiveComplete);
```

```
void OnGetArchiveComplete (object sender, GetArchiveCompleteArgs e)  
{  
    int sessionsNumber = e.Archive.Sessions.Count  
}
```

Generate the .mirx file downloading the data (not interpreted) by the device

1) Connect to the device

1.1) USB

a) Create an instance of the USBManager class to open the connection

```
USB = UsbManager.GetInstance();
```

b) Create an instance of the MirDevice class to connect the device

```
MirDeviceManager.MirDevice myDevice;  
myDevice = USB.GetDeviceConnected();
```

1.2) BLE

a) Create an instance of the MirDevice class

```
MirDeviceManager.MirDevice myDevice;
```

b) Connect the device (see paragraph 1.2 BLE connection)

```
myDevice =BleManager.Connect(MirCommunication.BLEDeviceAdvertisement.id)
```

2) Download the data (not interpreted) by the connected device and generate the .mirx file

a) Start downloading the tests from the device:

```
myDevice.GetMirXFile(MirInterpretationConfiguration.GenerateDefault());
```

b) Monitor the events during the operation for downloading data from the device

- Status of data download from device:

```
myDevice.OnGetMirXFileProgress += OnGetMirXFileProgress;
```

(This event is normally used to manage the progress of the data download)

```
void OnGetMirXFileProgress(object sender, GetMirXFileProgressArgs e)  
{  
    Application.Current.Dispatcher.Invoke(new Action(() =>  
    {  
        progressBar1.Value = e.Value;  
    }));  
}
```

- Data download from the device complete

```
myDevice.OnGetMirXFileComplete += OnGetMirXFileComplete;  
void OnGetMirXFileComplete(object sender, GetMirXFileCompleteArgs e)  
{  
    label1.Content = "Download Completed"  
}
```

- c) Generate the .mirx file:

```
void OnGetMirXFileComplete(object sender, GetMirXFileCompleteArgs e)  
{  
    Directory.CreateDirectory(strAppDataFolder);  
    string stringMirXfile = strAppDataFolder + "binary_archive" + ".mirx";  
    FileStream fs = new FileStream(stringMirXfile, FileMode.Create,  
    FileAccess.Write, FileShare.Write);  
    fs.Write(e.Archive, 0, e.Archive.Length);  
    fs.Close();  
}
```

Perform the Firmware upgrade of the MIR device

1) Connect to the device

1.1) USB

- a) Create an instance of the USBManager class to open the connection:

```
USB = UsbManager.GetInstance();
```

1.2) BLE

- a) Create an instance of the MirDevice class:

```
MirDeviceManager.MirDevice myDevice;
```

- b) Connect the device (see paragraph 1.2 BLE connection):

```
myDevice =BleManager.Connect(MirCommunication.BLEDeviceAdvertisement.id)
```

2) Select the .tsk file to upgrade the firmware of the MIR devices

- a) Select the .tsk file:

```
OpenFileDialog openFileDialog = new OpenFileDialog();  
openFileDialog.ShowDialog();
```


b) Read the content of the .tsk file

```
byte[] tsk = System.IO.File.ReadAllBytes(openFileDialog.FileName);
```

c) Monitor the progress of the upgrade process

- Sign the firmware upgrade status event

```
mirDevice.OnFirmwareUpgradeProgress += new  
EventHandler<UpgradeFirmwareArgs>(OnFirmwareUpgradeProgress);
```

(This event is normally used to manage the progress of the data download)

```
private void OnFirmwareUpgradeProgress(object sender, UpgradeFirmwareArgs e)  
{  
    progressBar1.Maximum = 100;  
    if (e.Progress > 100) return;  
    progressBar1.Value = (e.Progress);  
}
```

Additional Resources

- MIR Website: <https://www.spirometry.com>

Troubleshooting

1. The reference assemblies for framework ".NETFramework,Version=v4.X" were not found.

Solution: Ensure that .NET Framework 4.X is installed on your system and that the path to the assemblies is correct.

2. Error: The type or namespace name 'XYZ' could not be found.

Solution: Ensure that all necessary project references and NuGet packages are properly added to your project.

3. Could not load file or assembly 'MirABCD, Version=A.A.A.A, Culture=neutral, PublicKeyToken=abcd1234' or one of its dependencies. The system cannot find the file specified.

Solution: Ensure that you have correctly added the reference to the 'MirABCD' assembly in your project. Verify that the DLL file is present in the output and that all dependencies of this assembly are also available.

4. The application was unable to start correctly (0xc000007b).

Solution: This error may occur if you try to run the application built with a newer version of Microsoft Visual Studio on a system where the required .NET Framework version is not installed or is outdated. To resolve this issue, install or update to the necessary version of the .NET Framework.

Annex C. Instruction for Use - MIR SUPER SDK WEB (CLOUD/API)

Introduction

MIR API is a RESTful API used to interface a client application or information system with the platform following the HTTP protocol.

REST (Representational State Transfer) is an architectural style allowing to build applications (Web, Intranet, Web Service) by exploiting endpoints (endpoints urls) and referencing resources to be exploited according to the verbs of the HTTP protocol (GET, POST, PUT, DELETE etc. ...).

Environments

MIR API provides 2 URL for the integrations. The Staging is used to make your tests. The Production operations will be immediately applied on your production account.

Environment	URL	URL	Description
Staging		https://ssdk-api.staging.spirometry.com	This environment must be only for your test with fake data
Production		https://ssdk-api.spirometry.com	The production environment.

Endpoints

The endpoints are accessible with the following URLs:

Environment	URL
Staging	https://ssdk-api.staging.spirometry.com/{endpoint}
Production	https://ssdk-api.spirometry.com/{endpoint}

Vocabulary

Trial: A trial is a test performed (Spirometry or Pulse Oximetry) that contains all the data related to the test (i.e., date, time, test values)

Session: A session (Spirometry or Pulse Oximetry) can contain 1 to 8 trials. The session contains the date, time, and trials.

Parameter: A parameter is a "value" calculated (e.g., FVC for Spirometry or SpO2 Average for Pulse Oximetry). A parameter contains a unit and a value.

Patient: A patient is an object that contains all the information necessary to perform at least a Spirometry as well as their identification information (ID, first name, last name)

Interpretation: Each test can have an interpretation in the form of a numerical value and in the form of a string (for example: normal spirometry).

Predicted values: In the context of spirometry, each patient has "predicted values" for each parameter. These predicted values are different depending on the reference used (also called "author").

Authentication

Principles

Our API uses OpenID Connect (OIDC) to identify and authenticate clients. Any request to the API is verified thus must include a valid JWT (Json Web Token).

This must be sent in the Authorization header.

If the Authorization header is not completed and valid, the request will be considered as not authenticated (HTTP code 401).

If any alteration of the token happens (IP address change is one of them) you'll receive HTTP code 403.

More information on OpenID Connect and JWT on:

- <https://openid.net/connect/>
- <https://jwt.io/>

Credentials

Please contact us to get your API keys.

Permissions

To access our APIs, users must have specific permissions defined by "scopes". Each scope requires a distinct permission to access its associated resources.

If a user attempts to access a resource without the necessary permissions for the relevant scope, an HTTP 403 Forbidden response will be returned.

Should a user require additional access or permissions, they are encouraged to contact us for further assistance.

Scopes

data-types: Allows returning the description and translation (if applicable) of a parameter.

imports: Allows reading of MIR data and converting it into raw data.

convert: Allows converting data (either in file form or raw data) from one format to another.

interpretation: Allows obtaining the medical interpretation from a Session or a Trial.

oximeter-analysis: Allows performing a full analysis of an Oximetry session.

predicted-values: Allows calculating the predicted value for a given patient.

print: Allows generating the <format> report for the given session.

How to get a Bearer Token

To access the protected resources of our API, authentication is required. This authentication can be achieved in two ways:

1. **Simple Authentication:** By sending a username and password.
2. **Two-Factor Authentication:** A One-Time Password (OTP) might be requested after simple authentication to enhance security

Authentication Request

To initiate an authentication request, send a POST request with the following JSON in the request body:

```
{
  "username": "your_username",
  "password": "your_password"
}
```

Authentication Response

The API's response will be in the format:

```
{
  "status": int,
  "access_token": "eyJ...", // or null if "modes" != null
  "token_type": "Bearer", // or null if "modes" != null
  "scopes": ["parser", "import", ".."], // or null if "modes" != null
  "modes": null // or ["sms", "mail"],
  "session_id": "string", // or null if "modes" != null
}
```

Status Codes:

- 0: Authentication OK
- 1: Malformed request
- 2: Unknown application
- 3: Incorrect credentials
- 4: Account blacklisted
- 5: No callback channel defined
- 9: User_Info empty
- 10: Callback required

If the response contains a non-null modes key and contains a valid session_id, it indicates that two-factor authentication is required.

Two-Factor Authentication

For the following requests (which contain `/otp/{route}`), you must include in the header: `"X-Session-Id": "the value of the field session_id"`.

1. Choose a mode to receive the OTP:

Call the `/otp/mode` route with the payload:

```
{
  "mode": "sms" // or "mail"
}
```

2. Receive and send the OTP:

Once you've received the OTP (via SMS or email depending on your chosen mode), call the /otp/send route with the payload:

```
{
  "code": "123456" // Your OTP code
}
```

The response for this step will be:

```
{
  "status": 0,
  "access_token": "eyJ...",
  "token_type": "Bearer",
  "modes": null
}
```

Once you have obtained the Bearer Token, you must send this token in each of your requests by adding to the header "Authorization: Bearer <token>"

Example:

```
Content-Type: application/json;charset=utf-8
Accept: application/json;charset=utf-8
Authorization: Bearer ...

POST /the/route
{"the": "data"}
```

Rate limiting

The API limits the number of requests to 60 requests per minute per user.

If the incoming request exceeds the specified rate limit, a response with a 429 HTTP status code will automatically be returned.

Pagination

When retrieving a list of objects with a [GET] request, results are being paginated by the API.

```

"meta":{
  "current_page": 1,
  "from": 1,
  "last_page": 1,
  "path": "http://example.com/users",
  "per_page": 15,
  "to": 10,
  "total": 10
}

```

Pagination information will be presented in the meta object, available in the payload body and described below.

The meta object:

Field	Type	Description
current_page	integer	Index of the current page (first page: 1)
from	integer	Index of the first item available on the current page (for example: 20)
last_page	integer	Index of the last page
path	string	URL of the route with the current page
per_page	integer	Number of items per page (for example: 20)
to	integer	Index of the last item available on the current page (for example: 40)
total	integer	Total number of items that will be returned by the API.

You can send the query parameter `?page=<int>` to specify the page you wish to view. It's important to note that if the parameter is not specified, the default page is set to 1.

Errors

The API uses standard HTTP response codes to indicate the success or failure of an API request:

- 2xx codes indicate success.
- 4xx codes indicate an error that failed given the information provided.
- 5xx codes indicate an error with our servers.

An error and a troubleshoot key will be present in the response payload body. The troubleshoot key is a readable description of the error.

More detailed HTTP response codes will be provided in endpoints documentation.

Code descriptions:

Code	Description
200	OK - The request has succeeded.
201	Created - The request has been fulfilled and has resulted in one or more new resources being created.
204	No Content - Server has successfully fulfilled the request, no additional content sent in the response payload body.
400	Bad Request - Server cannot process the request due to something that is perceived to be a client error.
401	Unauthorized - Lack of valid authentication credentials for the target resource.
404	Not Found - Server did not find the target resource.
405	Method Not Allowed - The method is known by the server but not supported.
429	Rate limit exceeded
422	Unprocessable Entity - The server understands the content type of the request entity, and the syntax of the request entity is correct, but it was unable to process the contained instructions. It happens when the data sent in the POST/PUT or DELETE request is invalid.
500	Internal Server Error - Server encountered an unexpected condition.

Error example response:

```
{
  "code": 404, // The HTTP code
  "message": "...", // A description
}
```

Content-Type

Every POST, PUT and DELETE HTTP request sent to the API must specify the "Content-Type" and the "Accept" entities header to application/json;charset=utf-8.

Routes

The following list describes the available API routes with their description, input data, output data, and request type. For the input and output JSON payloads, please refer to the OpenAPI Swagger document attached to this documentation.

Data Types

[/v1/data-types/translate](#)

Scope: data-types
Request type: POST
Input data: Array of parameters to translate and/or to get the translation.
Output data: Array of the given parameters with their translations (if applicable) and their descriptions.
Description: Translate parameters to the specified language (for FVC = CVF in French (France)) and return a complete description of the parameter (with its Unit)

Imports

[/v1/imports/from-archive/device](#)

Scope: imports
Request type: POST
Input data: Base64 of the MIR archive
Output data: Array of sessions with the patient object for each session
Description: Parse the archive coming from a device (sent in base64) and return a JSON object (array of sessions)

[/v1/imports/from-archive/mir-file](#)

Scope: imports
Request type: POST
Input data: Base64 of the MIR file
Output data: Array of sessions
Description: Parse the archive coming from a .mir or .mirx file (sent in base64) and return a JSON object (array of sessions)

Convert

[/v1/convert/hl7/from/patient](#)

Scope: convert
Request type: POST
Input data: HL7 string of the patient
Output data: Patient object
Description: Convert a HL7 string of a patient to a JSON object

[/v1/convert/hl7/from/session](#)

Scope: convert
Request type: POST
Input data: HL7 string of the session
Output data: Session object
Description: Session object (with "patient" object "nested" if applicable)

[/v1/convert/hl7/from/spirometry](#)

Scope: convert
Request type: POST
Input data: HL7 string of the spirometry

Output **data:** Spirometry (trial) object
Description: Convert a HL7 string of a spirometry test (for e.g., sent from MIR Spiro Ipad) to a JSON object

[/v1/convert/hl7/from/oximetry](#)

Scope: convert
Request **type:** POST
Input **data:** HL7 string of the spirometry object
Output **data:** Oximetry (trial) object
Description: Convert a HL7 string of an oximetry test (for e.g., sent from MIR Spiro Ipad) to a JSON object

[/v1/convert/hl7/to/patient](#)

Scope: imports
Request **type:** POST
Input **data:** Patient object
Output **data:** HL7 string of the patient
Description: Create the HL7 string of the patient for the given JSON object

[/v1/convert/hl7/to/session](#)

Scope: convert
Request **type:** POST
Input **data:** Session object
Output **data:** HL7 string of the session
Description: Create the HL7 string of a full session for the given JSON object

[/v1/convert/hl7/to/spirometry](#)

Scope: convert
Request **type:** POST
Input **data:** Spirometry (trial) object
Output **data:** HL7 string of the spirometry (trial)
Description: Create the HL7 string of a spirometry test for the given JSON object

[/v1/convert/hl7/to/oximetry](#)

Scope: convert
Request **type:** POST
Input **data:** Oximetry (trial) object
Output **data:** HL7 string of the oximetry (trial)
Description: Create the HL7 string of an oximetry test for the given JSON object

[/v1/convert/gdt/from/spirometry](#)

Scope: convert
Request **type:** POST
Input **data:** Base64 of the GDT file content
Output **data:** Spirometry (trial) object
Description: Convert a GDT exported file of a spirometry (for e.g., sent from MIR Spiro Windows) to a JSON object

[/v1/convert/gdt/from/oximetry](#)

Scope: convert
Request **type:** POST
Input **data:** Base64 of the GDT file content
Output **data:** Oximetry (trial) object
Description: Convert a GDT exported file of a oximetry (for e.g. sent from MIR Spiro Windows) to a JSON object

Interpretation

[/v1/interpretation/spirometry/{author}/{elements?}](#)

Scope: interpretation
Request type: POST
Input data: Spirometry object
Output data: Interpretation object
Description: Return the interpretation of the given spirometry session with the given author and return only the given elements.

[/v1/interpretation/oximetry/{author}](#)

Scope: interpretation
Request type: POST
Input data: Oximetry object
Output data: Interpretation object
Description: Return the interpretation of the given oximetry session

Oximeter Analysis

[/v1/oximetry-analysis/holter](#)

Scope: oximetry-analysis
Request type: POST
Input data: Oximetry object
Output data: Oximetry object (converted to Holter analysis)
Description: Return the Holter analyze for the given oximetry session

[/v1/oximetry-analysis/sleep](#)

Scope: oximetry-analysis
Request type: POST
Input data: Oximetry object
Output data: Oximetry object (converted to Sleep analysis)
Description: Return the sleep analyze for the given oximetry session

[/v1/oximetry-analysis/walking-test](#)

Scope: oximetry-analysis
Request type: POST
Input data: Oximetry object
Output data: Oximetry object (converted to Walking Test analysis)
Description: Return the 6MWT analyze for the given oximetry session

Predicted Values

[/v1/predicted-values/{predictedAuthor}](#)

Scope: predicted-values
Request type: POST
Input data: Patient object
Output data: Predicted values object
Description: Return the predicted values for the given patient

Print

[/v1/print/{impressionFormat}](#)

Scope:

Request

Input

Output

Description: Return the PDF of the given session (in base64)

data:

Base64

of

the

type:

Session

PDF

file

print

POST

object

generated

[/v1/print/{impressionFormat}/dicom](#)

Scope:

Request

Input

Output

Description: Return the DICOM PDF of the given session (in base64)

data:

Base64

of

the

type:

Session

DICOM

file

print

POST

object

generated

Troubleshooting

1. Network Connectivity Issue Error: Application can't connect to the API.

Solution: Check your internet connection and firewall settings to ensure that the application can reach the external API. You may need to contact your IT department.

2. Incorrect API URL Error: HTTP 404 Not Found.

Solution: Double-check the API URL for the official documentation. Make sure you're pointing to the correct endpoint and including any necessary path parameters.

3. Invalid API Key or Token Error: HTTP 401 Unauthorized.

Solution: Verify that your API keys or tokens are correct. Regenerate a new key or token if necessary and update it in your application's configuration settings.

4. Incorrect HTTP Headers Error: HTTP 400 Bad Request.

Solution: Check your HTTP headers, such as Content-Type and Authorization. Make sure they are correctly formatted and include all the necessary information, as specified in the API documentation.

5. Wrong HTTP Status Codes Error: Receiving unexpected HTTP status codes.

Solution: Investigate the received HTTP status codes and consult the API documentation to understand what they mean. Take corrective action based on this information.

6. Rate Limit Exceeded Error: HTTP 429 Too Many Requests.

Solution: Review the rate limits in the API documentation and implement rate-limiting or throttling in your application to avoid hitting these limits.

7. Mismatched Response Format Error: Sending JSON payloads but not receiving JSON in return.

Solution: Ensure that you're setting the *Accept: application/json* header when making the API request. This tells the server that you expect a JSON-formatted response.

8. Persistent Issues Error: Issues remain despite checks.

Solution: Contact our team.

Annex D. Instruction for Use - MIR SUPER SDK MacOS

Introduction

The purpose of this guide is to facilitate use of the tool **SUPER SDK (Software Development Kit)** for the rapid development of applications running on MacOS for monitoring and storing patients, archives of spirometry and oximetry tests obtained through MIR devices.

MIR SSDK MacOS allows you to quickly perform all the necessary operations to use Spirometry or Oximetry.

It consists of various libraries as described below:

- **MirCharting**: Contains all the functions necessary for the creation and management of graphs.
- **MirCommunication**: Allows to manage communication in USB and BLE with all devices, regardless of the protocol.
- **MirDataTypes**: Contains all the properties (in the form of Enums and methods) for all the elements necessary for Spirometry or for the Oximetry (for e.g., spirometry parameters, ethnic groups codes, etc.)
- **MirDeviceManager**: Oversees the management of the communication of MIR devices.
- **MirInterpretation**: Analyze a session and return its interpretation.

Prerequisites

Before you begin integrating the MacOS SSDK into your application, make sure you have the following in place:

Xcode: Ensure you have a functional installation of Xcode on your system. The MacOS SSDK is primarily developed in Swift and need at least Swift version 4.2.

OS Version: Our SDK is compatible with MacOS 10.13 and iOS 11.0 and later versions. Using an older version does not guarantee the functionality of all features.

For MacOS, the SSDK needs at least 2 GB of RAM, but we recommend using 4 GB of RAM.

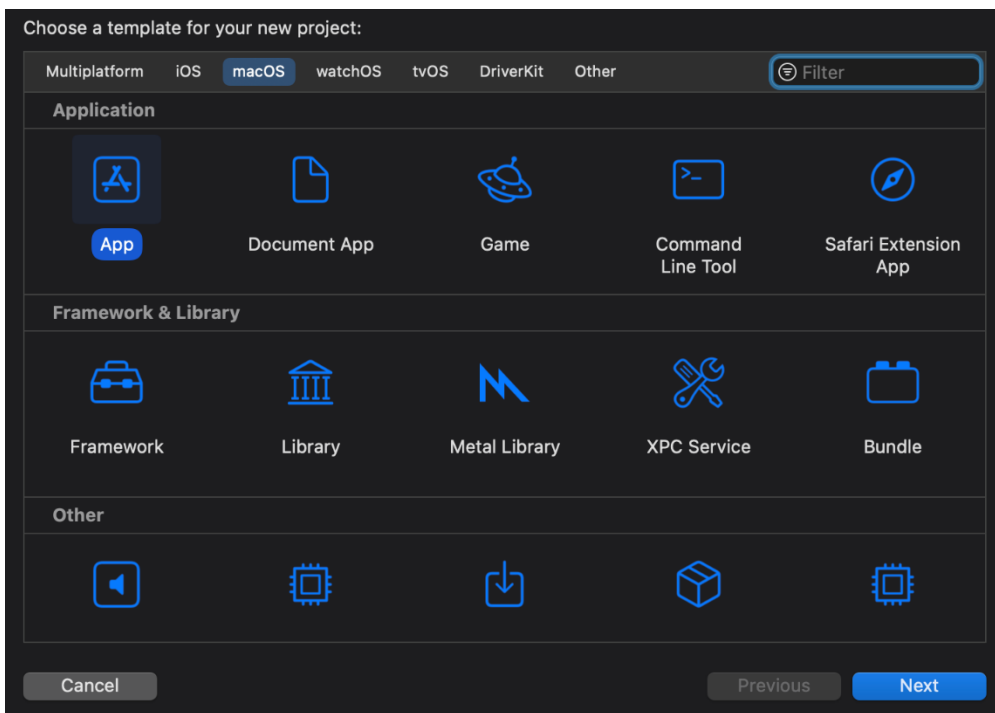
How to implement

To implement the SDK, you can follow the chapter "Integration into a Xcode project" to learn the procedure to use it in your project.

Integration into a XCode project

For this example, we'll create a blank MacOS project with a simple button that will call the SDK to retrieve a device archive.

- 1) Open Xcode and create new project, choose MacOS and App:



- 2) Enter the product name, the organization identifier, the interface and the language:

Choose options for your new project:

Product Name:

Team:

Organization Identifier:

Bundle Identifier:

Interface:

Language:

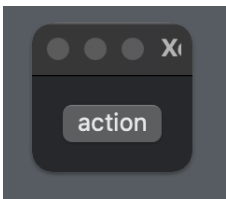
Storage:

Host in CloudKit

Include Tests

3) Now that you have chosen the path of your application, let's create a button to use a function of the SSDK, open the ContentView file and create a button with the following code:

```
struct ContentView: View {
    var body: some View {
        Button(action: action) {
            Text("action")
        }.padding()
    }
}
```



It is now necessary to add references to the SSDK, for this we'll use Cocoapods.

You can find how to install Cocoapods [here](#).

4) Start a terminal and go to the root of your project and initialize Cocoapods.

```
$ pod init
```

5) After this, you'll have a file named Podfile, this is where you'll put all your dependencies, for this example we'll use MirCommunication.

```
Pods ) Podfile ) No Selection
1 # Uncomment the next line to define a global platform for your project
2 # platform :ios, '9.0'
3
4 target 'mir.sampleapp' do
5     # Comment the next line if you don't want to use dynamic frameworks
6     use_frameworks!
7
8     # Pods for mir.sampleapp
9     pod 'MirCommunication', :path => 'MirCommunication'
10    pod 'MirDataTypes', :path => 'MirDataTypes'
11 end
12
```

MirCommunication depends on MirDataTypes so we need to add it too.

- 6) Move the folder of the dependencies to the path you specified in the Podfile, then in the terminal start the installation of the dependencies.

```
acker@MBPdeAckermann mir.sampleapp % pod install
Analyzing dependencies
Downloading dependencies
Generating Pods project
Integrating client project
Pod installation complete! There are 2 dependencies from the Podfile and 2 total pods installed.
```

- 7) Now add the code to retrieve the archive:

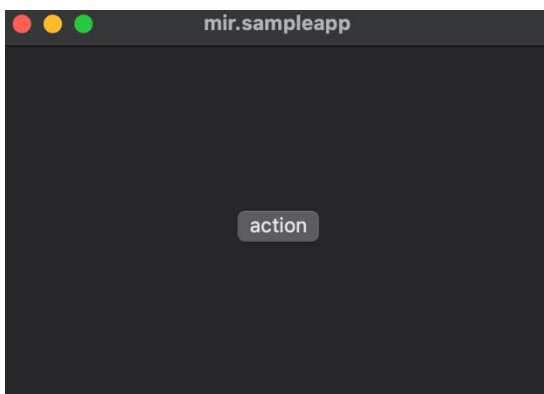
```
import SwiftUI
import MirCommunication

struct ContentView: View {
    var body: some View {
        Button(action: action) {
            Text("action")
        }.padding()
    }
}

#Preview {
    ContentView()
}

func action() {
    let shared = MIRCommUSB()
    shared.getDeviceArchive()
}
```

- 8) Launch the application and conduct the test.



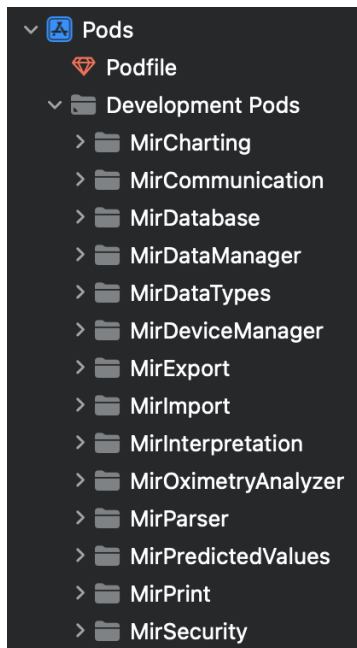
You can now use the full SDK in your project.

Main features

The list below provides an example for each of the main functions, namely:

- Connecting to the device.
- Conducting a test.
- Retrieving results.
- Retrieving an archive.
- Save the device archive as a .mir or .mirx file.
- Updating the device.

The other available functions can be accessed from Xcode Pods list:



Connecting to a device

The device supports 2 types of connections:

- **USB:** Requires a connection via USB.
- **BLE:** Requires a computer that supports Bluetooth Low Energy (BLE) and a "BLE" device.

1.1) USB connection

- a) First find the USB devices by calling the shared instance of MirDeviceManager

```
MirDeviceManager.shared.startDiscovering()
```

b) Here's an example on how to monitor events.

```
MirDeviceManager.shared.startDiscovering {
    (devices) in

    if let device = devices.last, device is MirUsbDevice
    {
        connectDevice(device: device)
    }

    NotificationUtility.post(.deviceDiscovered)
}
```

```
    } deviceConnected: {  
        (device) in  
  
        connectedDevice = device  
        NotificationUtility.post(.deviceConnected)  
  
    } deviceDisconnected: {  
        (device) in  
  
        NotificationUtility.post(.deviceDisconnected)  
    }  
}
```

c) You can check discovered devices in this variable.

```
MirDeviceManager.shared.UsbDiscoveredDevices
```

d) Then call the shared instance of MirDeviceManager and call the function connectUsbDevice with the selected device.

```
MirDeviceManager.shared.connectUsbDevice(device)
```

For more information for the parameters, please refer to the file MirDeviceManager in MirDeviceManager pod.

1.2) **BLE connection**

a) First find the Bluetooth devices by calling the shared instance of MirDeviceManager

```
MirDeviceManager.shared.startDiscoveryBluetooth()
```

b) Here's an example of how to monitor events.

```
var deviceDiscoveredObserver: NSObjectProtocol?  
  
MirDeviceManager.shared.startDiscoveryBluetooth()  
  
deviceDiscoveredObserver =  
NotificationUtility.addObserver(for: .deviceDiscovered) { notification in  
  
    let devices = MirDeviceManager.shared.bluetoothDiscoveredDevices.filter { d in  
        return d as? MirBluetoothDevice != nil  
    } as! [MirBluetoothDevice]  
  
}
```

c) You can check discovered devices in this variable.

```
MirDeviceManager.shared.bluetoothDiscoveredDevices
```

d) Then call the shared instance of MirDeviceManager and call the function connectBluetoothDevice with the selected device.

```
MirDeviceManager.shared.connectBluetoothDevice(device)
```

For more information for the parameters, please refer to the file MirDeviceManager in MirDeviceManager lib.

Conduct the FVC test, generate the graph during the test and receive the results

1) Connect to the device.

See [Connecting to a device](#).

2) Conduct the FVC test.

a) Once you have connected your device, you can get the connected device and start the FVC test.

```
DeviceManager.connectedDevice?.startFvcTrial(...)
```

b) You can get the trial result in the completion of the startFvcTrial

```
[...], completion: {  
    [weak self] (trial) in  
  
    if let trial = trial {  
        self?.trialReceived(trial)  
    }  
}
```

c) If you need a certain value from the trial, you can use the function *getParameterMeasuredValue* from MirDataTypes.

```
trial.getParameterMeasuredValue(code: MirParameterCode.FEV1)
```

The MirParameterCode can also be found in MirDataTypes.

3) End the test.

```
DeviceManager.connectedDevice?.stopTrial(...)
```

Conduct the VC test, generate the graph during the test and receive the results

1) Connect to the device.

See [Connecting to a device](#).

2) Conduct the VC test.

a) Once you have connected your device, you can get the connected device and start the VC test.

```
DeviceManager.connectedDevice?.startVcTrial(...)
```

b) You can get the trial result in the completion of the startVcTrial

```
[...], completion: {  
  [weak self] (trial) in  
  
  if let trial = trial {  
    self?.trialReceived(trial)  
  }  
}
```

c) If you need a certain value from the trial, you can use the function *getParameterMeasuredValue* from *MirDataTypes*.

```
trial.getParameterMeasuredValue(code: MirParameterCode.FEV1)
```

The *MirParameterCode* can also be found in *MirDataTypes*.

4) End the test.

```
DeviceManager.connectedDevice?.stopTrial(...)
```

Retrieving an archive and generating the .mirx file

1) Connect to the device.

See [Connecting to a device](#).

2) Download the data (not interpreted) by the connected device and generate the .mirx file

a) Start downloading the tests from the device:

```
DeviceManager.connectedDevice?.queryArchive(...)
```

You can monitor the download of the archive by using the parameter *progressBlock* of the function and *completion* when the file has finished the downloading.

b) Generate the .mirx file:

```
DeviceManager.connectedDevice.getMirXFile(progressBlock: {  
    [weak self] (progress) in  
  
    self?.updateProgressBar(withValue: progress)  
}, completion: {  
    [weak self] (rawDeviceInfo, rawArchive) in  
  
    MirFiles.createMirFileFromArchive(rawArchive!.rawData, rawDeviceInfo!.rawData)  
})
```

Perform the Firmware upgrade of the MIR device

1) Connect to the device.

See [Connecting to a device](#).

2) Select the .tsk file to upgrade the firmware of the MIR devices

a) Select the .tsk file and read his content:

```

let openPanel = NSOpenPanel()
openPanel.allowsMultipleSelection = false
openPanel.canChooseDirectories = false
openPanel.canCreateDirectories = false
openPanel.canChooseFiles = true
openPanel.allowedFileTypes = ["tsk"]

openPanel.beginSheetModal(for: window) {
    (result) -> Void in
    if result == NSApplication.ModalResponse.OK {
        if let validUrl = openPanel.url {
            self.resetDataSource()

            var title = "Are you sure to update the connected device with
$0?"
            title = title.replacingOccurrences(of: "$0", with:
validUrl.deletingLastPathComponent().lastPathComponent)
            if AlertUtility.showConfirmationAlert(withTitle: title) {
                let fw = MirFirmware()
                fw.data = try? Data(contentsOf: validUrl)

                DispatchQueue.main.async {
                    self.showUpdateFirmwareScreen(fw)
                }
            }
        }
    }
}

```

b) Once we have the firmware, we can start the upgrade:

```

firmwareUpdate = MIRCommUSBFirmwareUpdate()

let tskFromFile = firmware!.data
let byteArray = [UInt8](tskFromFile!)

firmwareUpdate?.updateFirmware(byteArray) {status, positionPercent

```

You can use *status* and *positionPercent* to keep track of the progress of the firmware upgrade.


```
if positionPercent >= 0 {  
    self.updateProgressBar(withValue: Int(positionPercent))  
}
```

```
switch status {  
    case .initializing:  
        self.labelStatus.text = "Initialization"  
        break  
    case .error:  
        [...]
```

Additional Resources

- MIR Website: <https://www.spirometry.com>

Annex E. Instruction for Use - SSDK Android

Introduction

The purpose of this guide is to facilitate use of the tool SUPER SDK (Software Development Kit) for the rapid development of applications running on Android for monitoring and storing patients, archives of spirometry and oximetry tests obtained through MIR Bluetooth devices.

MIR SSDK Android allows you to quickly perform all the necessary operations to use Spirometry or Oximetry.

The SSDK is only compatible with Spirobank Smart, Smart One and Spirobank II Smart devices.

Prerequisites

Before you begin integrating the Android SDK into your application, make sure you have the following in place:

- Minimum Android 4.3 (Spirobank II Smart) or 5.0 (Spirobank Smart/SmartOne) version
- Compatible Hardware for BLE Support
- Bluetooth Permissions (BLUETOOTH, BLUETOOTH_ADMIN, and ACCESS_FINE_LOCATION (for Android 6.0 and above)).
- 1GB RAM
- Free Space of the device
- Full access to internet Connection (with at least 1 Mbits/s)

Integration (How to implement?)

Spirobank Smart SDK Android Guide

Import and configure the SDK Module

To import the SDK Module inside your project just take the following steps:

1. Open your Android Studio Project and go to menu "File" -> "New" -> "New Module..."
2. Select "Import .JAR/.ARR Package" and click next
3. Select "spirobanksmartsdk.aar" file and click finish
4. Right click on your project and "Open Module Settings", go to "Dependencies" tab and add "Module dependency" "spirobanksmartsdk".

Starting up a Device Manager

The first step to take to use this framework is to get a DeviceManager object.

The DeviceManager has been implemented according to the singleton pattern so you would get a same instance of it to be used in every context of your app.

The DeviceManager class also exposes the [setDeviceManagerCallback](#) to set DeviceManagerCallback.

Perform a scan (discovery)

With an instance of deviceManager the client app may call the [startDiscovery](#) method.

The discovery will retrieve all the SPIROBANK SMART, SPIROBANK SMART OXI, SMART ONE and SMART ONE OXI devices in range. For each device discovered, the deviceManager call its callback method [deviceDiscovered](#).

Perform a “direct connection” to a Device

With an instance of deviceManager, the client app may call the [connect](#) method passing the DeviceInfo object.

When deviceManager calls its callback method [deviceConnected](#) it means that the device is connected and all its services and characteristics have been read.

If the same device is already connected the connect method will not perform any action.

If a different device is already connected, the connect method disconnect it before connecting that new device

Start a test in the “multitest mode” environment

IMPORTANT NOTE: For the scope of this SDK, “test” means a complete expiratory maneuver.

The current version of SDK supports the “multitest mode”. This means that different kind of tests can now be started.

At present the tests supported are:

- the Spirometry test (FVC test) (**only SPIROBANK SMART**): a forced expiratory maneuver that lasts 6 seconds (3 for kids)
- the Spirometry enhanced test (FVC PLUS test) (**only enabled SPIROBANK SMART with FW 3.1+**): a forced expiratory and inspiratory maneuver, with the possibility to do multiple loops of inspiration and expiration
- the PeakFlow/Fev1 test: a forced expiratory maneuver that lasts 1 second
- the Flow Monitoring Test (**only SPIROBANK SMART from firmware 3.0**)
- the Oximetry Test (**only SPIROBANK SMART OXI and SMART ONE OXI**)
- the Vital Capacity test (VC test) (**only enabled SPIROBANK SMART with FW 4.4+**)

Please

note:

1. SpirobankSmart devices equipped with firmware versions (<1.7) do not support the multitest mode, the only valid is FVC test. The command to start the PeakFlow/Fev1 test would be ignored.
2. SmartOne devices supports only PeakFlow/Fev1 test, the command to start the FVC test would be ignored.

To require a specific test to be started by the device, the SDK provides a method with a parameter.

With an instance of Device the [startTest\(Context context, TestType testType, TurbineType turbineType\)](#) method has to be invoked to start one of the supported test type. Pass to this method the parameter [TestType.Fvc](#) to start the FVC test, [TestType.FvcPlus](#) to start the FVC PLUS test, the parameter [TestType.PefFev1](#) to start the PeakFlow/Fev1 test, the parameter [TestType.FT_Monitor](#) to start the Flow Monitoring test, the parameter [TestType.Oxi](#) to start the Oximetry test or the parameter [TestType.Vc](#) to start the VC test.

During the spirometry test the Device object calls its delegate method [flowUpdated \(Device device, float flow,](#)

`int stepVolume`, `boolean isFirstPackage`) to pass the flow values measured.

For each flow point received the current volume can be get by adding the `stepVolume` to the current volume.

At the end of each expiration maneuver (test), Device object usually calls its delegate method (according to the specific TestType) to provide the test's results:

`resultsUpdated (ResultsPefFev1 resultsPefFev1)`

`resultsUpdated (ResultsFvc resultsFvc)`

`resultsUpdated (ResultsFvcPlus resultsFvcPlus)`

`resultsUpdated(ResultsVc resultsVc)`

*Note that if the test is performed very bad (too poor flows, or no expiration at all) the device is not able to calculate the results and therefore the delegate method `resultsUpdated` **IS NOT CALLED AT ALL**.*

During the oximetry test the Device object calls its delegate method `oximetryValuesUpdated(int signal, int spO2, int heartRate, Device.OximetryWarnings warning, boolean isValid)` to pass the values measured and `oximetryPlethysmographicValuesUpdated (double ppgSignal,int minY,int maxY)` to pass plethysmographic curve data.

At the end of Oximetry the device provide the test's results:

`resultsUpdated (ResultsOxy resultsOxi)`

Customize the End Of Test Time Out

You can customize the End Of Test Time Out from 15 sec. (default) to 120 sec. For the Oximetry test this parameter will be ignored.

The parameter can be passed to the method:

`startTest(Context context, TestType testType, TurbineType turbineType, byte endOfTestTimeout_sec)`

This function works from Spirobank Smart firmware version 2.4 and Smart One 3.5. The previous versions always use 15 sec. We recommend to use the minimum necessary value to preserve device battery life.

Specify turbine type

You can specify the turbine type `Device.TurbineType.disposable` or `Device.TurbineType.reusable` (default). The parameter can be passed to the method:

`startTest(Context context, TestType testType, TurbineType turbineType, byte endOfTestTimeout_sec)`

This function works from Spirobank Smart firmware version 2.7 and Smart One firmware version 3.6. The previous versions always use reusable turbine. This instruction does affect the reading made by the device because different algorithms are used for each turbine type

StartTest handshaking

When the client app sends the StartTest method (whatever is the overload used, and whatever is the kind of test requested) the SDK, from version 2.6.0 call a new call back: `testStarted` This method is invoked when the device is actually READY to take measurements. For this reason, this new method is the right place to raise a message that the user can START EXHALING (blowing).

The Results provided by the different test types are the following:

PARAMETER	PROVIDED BY
pef_cLs (Peakflow cL sec)	Fvc-PefFev1
fev1_cL (Forced Exp Vol at 1th sec in cL)	Fvc-PefFev1
quality (acceptability calculation)	Fvc-FvcPlus-PefFev1
fvc_cL (Forced Exp Capacity in cL)	Fvc
fev1_fvc_pcmt (Fev1% in percentage)	Fvc-FvcPlus
fev6_cL (Forced Exp Volume at 6th sec in cL)	Fvc
fef2575_cLs (Max mid-expiratory flow)	Fvc
tempCelsius (Room temperature)	FvcPlus
btps (BTPS)	FvcPlus
fvc_L (Forced Exp Capacity in L)	FvcPlus
fev1_L (Forced Exp Vol at 1th sec in L)	FvcPlus
pef_Ls (Peakflow L sec)	FvcPlus
fef75_Ls	FvcPlus
fef2575_Ls	FvcPlus
fet_s	FvcPlus
fev6_L (Forced Exp Volume at 6th sec in L)	FvcPlus
fev6perc	FvcPlus
fev6_fvc	FvcPlus
fef25_Ls	FvcPlus
fef25_Ls	FvcPlus
fef50_Ls	FvcPlus
fivc_L	FvcPlus
fiv1_L	FvcPlus
fiv1perc	FvcPlus
pif_Ls	FvcPlus
fev3_L	FvcPlus
fev3perc	FvcPlus
timeToPef_s	FvcPlus
fev05_L	FvcPlus
fev05perc	FvcPlus
fev075_L	FvcPlus
fev075perc	FvcPlus
fev2_L	FvcPlus
fev2perc	FvcPlus
fef7585_Ls	FvcPlus
fif25_Ls	FvcPlus
fif50_Ls	FvcPlus
fif75_Ls	FvcPlus
fvPoints (Array: all flow volume points)	FvcPlus
vtPoints (Array: all volume time points)	FvcPlus
hesitationTime_s	FvcPlus
eVol_mL (Extrapolated volume in mL)	PefFev1

vext_L (Extrapolated volume in L)	FvcPlus
pefTime_sec (time to reach Peakflow in sec)	PefFev1
spO2Mean (%)	Oximetry
spO2Max (%)	Oximetry
spO2Min (%)	Oximetry
HeartRateMean (bpm)	Oximetry
HeartRateMax (bpm)	Oximetry
HeartRateMin (bpm)	Oximetry
spo2Points (Array)	Oximetry
heartRatePoints (Array)	Oximetry
evc_L	VC
ivc_L	VC
ic_L	VC
setOrSit_s	VC
vtPoints (Array)	VC

Device Info

From device object you can get device info by calling `getDeviceInfo()` method.

Device Info provides the following information:

METHOD	RESPONSE DESCRIPTION
<code>getAdvertisementDataName()</code>	Bluetooth device name
<code>getAddress()</code>	Bluetooth device address
<code>getName()</code>	MIR product name
<code>getProtocol()</code>	MIR communication protocol
<code>getSerialNumber()</code>	MIR device serial number

How the test is stopped/restarted in the “multitest mode” environment

The test is stopped in two ways:

- 1) when the [stopTest](#) method is invoked
- 2) automatically, when the device/framework detects the EOT (**End Of Test**) criteria. See above the chapter **End Of Test Criteria**.

In the FVC-FVC PLUS test

When the test is stopped (automatically or not) the device always quits from “test mode” and a new command “startTest” needs to be sent to start a new test.

The sequence of the [Device](#) delegate methods called during an FCV test are the following:

[flowUpdated](#)

[testStopped](#) (always called, even if the test was stopped by the invocation of the stopTest method)

[resultsUpdated](#) (which might not be called in case there aren't the conditions to return the Results)

In the Peakflow/Fev1 test

There is a different behavior depending how the stop the test has occurred.

1) when the test is stopped:

- by the invocation of the stopTest method
- by the expiration of the timeouts

device quits from the “test mode” and a new command “startTest” needs to be sent to start a new test.

In this case the sequence of the [Device](#) delegate methods called are the following:

[flowUpdated](#)

[resultsUpdated](#) (which might not be called in case there aren't the conditions to return the Results)

[testStopped](#) (always called, even if the test was stopped by the invocation of the stopTest method)

It is strongly recommended to avoid to place the call of the [startTest](#) method into the [testStopped](#) delegate method because this might cause a loss of flows. If you need to give more time to the user, you can “Customize the End Of Test Time Out” in “startTest” command.

2) when the test is stopped:

- because the device has automatically detected the end of the expiratory maneuver. See above the chapter **End Of Test Criteria**.

device stops the test but it DOES NOT quit from the “test mode” and RESTART AUTOMATICALLY a new test (no need to call the startTest method to start a new test)

In this case the sequence of the delegate methods called are the following:

[flowUpdated](#)

[resultsUpdated](#) (which might not be called in case there aren't the conditions to return the Results)

[testRestarted](#) (always called. It means that a new test is started, the user can blow)

This behavior, called **AutomaticTestRestarting**, has been designed for the Peakflow test where the patient can perform the 3 tests, recommended for a valid session, without any rest interval because of the short duration of each maneuver (1 seconds).

Though the AutomaticTestRestarting approach is recommended, the developer can decide to handle the Peakflow/Fev1 test using the **Start&Stop** approach: with this approach the developer should invoke the [stopTest](#) method as soon as the delegate method [testRestarted](#) is called and then use the [startTest](#) method to start a new test.

See *Best Practices* section for more detailed info.

In the Flow Time Monitoring test

when the test is stopped (automatically or not) the device always quits from “test mode” and a new command “startTest” needs to be sent to start a new test

the sequence of the delegate methods called during a Flow Time Monitoring test are the following:

[flowFT_MonitorUpdated](#)

[stopTest](#) (always called, even if the test was stopped by the invocation of the stopTest method)

NO RESULTS ARE SENT WITH THIS KIND OF TEST

In the OXIMETRY test

When the test is stopped the device always quits from “test mode“ and a new command “startTest” needs to be sent to start a new test.

The sequence of the delegate methods called during an Oximetry test are the following:

[oximetryValuesUpdated](#)
[oximetryPlethysmographicValuesUpdated](#)
[resultsUpdated](#)
[testStopped](#)

In the VC test

When the test is stopped (automatically or not) the device always quits from “test mode” and a new command “startTest” needs to be sent to start a new test.

The sequence of the [Device](#) delegate methods called during a VC test are the following:

[volumeUpdated](#)
[testStopped](#) (always called, even if the test was stopped by the invocation of the stopTest method)
[resultsUpdated](#) (which might not be called in case there aren't the conditions to return the Results)

Real Time Animation and Quality Report in the FVC test

The Patient class can be instantiated to get some important information during the test to be used to display the animated feedback of the user's expiration.

The model of animation proposed by Patient, is based on the concept of the Predicted Area (calculated from user's personal data according the specific TestType). Two graphic objects have to move inside the Predicted Area: One graphic object (target object) is moved by the user's expired volume with a preset speed (based on the predicted flow). The other graphic object (user object) is moved according to the user's expired volume and at the speed of the user's flow (measured flow)

The method [actualPercentageOfTargetWithFlow](#) retrieves the percentage of the Predicted Area which has been covered by the “user object”.

This percentage value can be asked to Patient for each flow retrieved by the method [flowUpdated](#)

The method [predictedPercentageOfTargetWithFlow](#) retrieves the percentage of the Predicted “AREA” which has been covered by the “target object”.

This percentage value can be asked to Patient object for each flow retrieved by the method [flowUpdated](#).

The class Patient also provides information about the acceptability of each single maneuver via the [getQualityReport](#) method, calculated according to the specific TestType mode.

With Spirobank Smart and Spirobank Smart Oxi devices, since firmware version 4.4, the [QualityReport](#) generated by the method is ATS 2019 compliant. It contains a different [AcceptabilityStatus](#) for FVC and FEV1, and a list of [QualityReport.Indication](#). An indication contains both a quality message, and an [AcceptabilityInstruction](#), to better evaluate the quality of the blow.

With firmwares below v. 4.4, the QualityReport will only contain a [AcceptabilityStatus](#) (trialAcceptability), and a single [QualityReport.Indication](#), as per ATS 2015 guidelines.

Real Time Animation in the Flow Time Monitoring test

During a Flow Time Monitoring test the Flow Time loop (expired and inspired points) can be plotted. The method [flowFT_MonitorUpdated](#) provide Flow points at a constant time of 10 milliseconds. The value of flow is an int type that is positive for expiration and negative for expiration. It is provided in cL/s.

Real Time Animation in the Oximetry test

During an Oximetry test the following curves can be plotted:

- the plethysmographic curve, using [oximetryPlethysmographicValuesUpdated](#)
- the sPO2 curve and/or the Pulse Rate curve, using [oximetryValuesUpdated](#)

SpO2 range is 70 → 99

HeartRate range is 30 → 300

the above method can also be used for displaying other info to the user such as signal (range 0 to 8)

warnings (OximetryWarnings)

The following values can be assigned to the parameter warning

- NoWarning
- DefectiveSensor
- BatteryLow
- NoFinger
- PulseSearching
- PulseSearchingTooLong
- LossOfPulse
- LowSignalQuality
- LowPerfusion
- ArtifactDetected

CAUTION: when the warning parameter = BatteryLow, the device will stop the test and the delegate method TestStopped is called.

Session quality grade

To evaluate the quality grade of the whole session, the SDK provides some functions from the [SpirometryInterpretation](#) class.

The functions [SpirometryInterpretation.getAts2019SessionQualityGrade](#) and [SpirometryInterpretation.getAts2015SessionQualityGrade](#) both return a GradingReport, which contains:

- ATS standard used
- Quality grade for FVC and FEV1, in case of ATS 2019 standard
- Trial grade, in case of ATS 2015 standard

[getAts2019SessionQualityGrade](#) requires, as parameters:

- Age of the patient, expressed as a double (obtainable through the [age](#) property of the [Patient](#) class)
- Number of acceptable tests for the FVC parameter (obtainable through the [fvcAcceptability](#) property of

- the [QualityReport](#) class for each test: must be [AcceptabilityStatus.ACCEPTABLE](#) to count as an acceptable test)
- Number of usable trials for the FVC parameter (obtainable through the [fvcAcceptability](#) property of the [QualityReport](#) class for each test: must be [AcceptabilityStatus.ACCEPTABLE](#) or [AcceptabilityStatus.NOT_ACCEPTABLE_BUT_USABLE](#) to count as a usable test
 - Number of acceptable tests for the FEV1 parameter (obtainable through the [fev1Acceptability](#) property of the [QualityReport](#) class for each test: must be [AcceptabilityStatus.ACCEPTABLE](#) to count as an acceptable test)
 - Number of usable trials for the FEV1 parameter (obtainable through the [fev1Acceptability](#) property of the [QualityReport](#) class for each test: must be [AcceptabilityStatus.ACCEPTABLE](#) or [AcceptabilityStatus.NOT_ACCEPTABLE_BUT_USABLE](#) to count as a usable test
 - Largest and second largest FVC measured value, in liters, for the whole session
 - Largest and second largest FEV1 measured value, in liters, for the whole session

[getAts2015SessionQualityGrade](#) requires, as parameters:

- Age of the patient, expressed as a double (obtainable through the [age](#) property of the [Patient](#) class)
- Number of acceptable trials (obtainable through the [trialAcceptability](#) property of the [QualityReport](#) class for each test: must be [AcceptabilityStatus.ACCEPTABLE](#) to count as an acceptable test)
- Largest and second largest FVC measured value, in liters, for the whole session
- Largest and second largest FEV6 measured value, in liters, for the whole session

Both functions return a [GradingReport](#), which contains the ATS standard used, and:

- In case of ATS 2019 standard, [fvcGrade](#) and [fev1Grade](#) may contain a value between A and U (will never be [NOT_APPLICABLE](#)), while [trialGrade](#) will always be [NOT_APPLICABLE](#)
- In case of ATS 2015 standard, [fvcGrade](#) and [fev1Grade](#) will always be [NOT_APPLICABLE](#), while [trialGrade](#) may contain a value between A and F (will never be [NOT_APPLICABLE](#) or U)

Get the FVC curve points at the highest resolution

From SDK version 3.1.5, when connected to a Spirobank smart supporting this feature, you can get the expired curve points of the last FVC PLUS test performed at the resolution of 100Hz (one point each 10 milliseconds).

Note that the high resolution curve points ARE NOT automatically retrieved with the [FvcPlusResults](#) object passed by the [resultsUpdated](#) call back .

To get the high resolution curve points the method [GetHighResolutionFvcPlusCurvePoint](#) of the class [Device](#) must be called. It must be called AFTER receiving the [resultsUpdated](#) call back.

After the [GetHighResolutionFvcPlusCurvePoint](#) is called, the high resolution curve points are provided by the call back [highResolutionFvcPlusCurvePointsUpdated](#) ([Device](#) class)

The above callback retrieves a collection of objects [HrCurvePoint](#) having has a properties Flow, Volume and Time.

From SDK version 3.1.7 the *GetHighResolutionFvcPlusCurvePointWithInspiration* is available that extends the functionality of the previous method providing in addition also the inspiration part of the curve. The callback and the object used to provide high resolution points are the same.

Update device internal software

Currently, the firmware update is only available for Spirobank Smart devices with firmware version < 4.0, or Smart One with firmware version < 4.0:

<i>Device supported</i>	<i>Firmware supported</i>
Smart One	< 4.0
Spirobank Smart	< 4.0

With an instance of connected Device call [startSoftwareUpdateProcedure\(Context context, byte\[\] updateData\)](#). UpdateData it's a bin file provided by MIR. During the update SDK calls [softwareUpdateProgress](#) callback with the following parameters: progress, status, and error. Progress starts from 0 to 100, status can be: InProgress, Complete or Error. Parameter error eventually describes what happened.

The error descriptions can be:

- "Update start timed out" when the time it takes to start the firmware loading procedure exceeds the timeout
- "Communication timed out" when the time it takes to load one of the "packets" (of firmware) exceeds the timeout this message can also appear after 100% if the firmware doesn't match the device.

Run the Project

ATTENTION: the projects with the SpirobankSmart SDK Sample embedded can only be compiled and run in an Android device. **Simulator is not supported.**

Best Practices

The best practice to handle Mir Spirometer and avoid instability and malfunctioning is the following:

PEAK-FLOW / FEV1 TEST (AutomaticTestRestarting approach)

1. Get an instance of the [DeviceManager](#) Class
2. Perform a scan and connect OR perform a "direct connection" to a spirometer
3. Add your [PefFev1DeviceCallback](#) listener to your [Device](#) instance, via the [addDeviceCallback](#) method
4. Start the test (user must start blowing within 15 sec)
[testStopped](#) delegate method is called if user doesn't blow within 15 sec (in this case give to the user

the ability to restart the test manually)

It is strongly NOT recommended to call the Start test on the testStopped delegate method as a workaround to contrast the 15 sec timeout effect. If you need to give more time to the user, you can “Customize the End Of Test Time Out” in “startTest” command.

5. Use a visual feedback to prevent that user start blowing before the app has received the delegate method [testStarted](#)
6. Use [flowUpdated](#) delegate method to show the animated feedback (also in connection with the Patient class’s dedicated methods)
7. Use [resultsUpdated](#) to show the result (this method might not be called if the device was not able to calculate the results)
8. Use [testRestarted](#) delegate method to detect that the current expiratory maneuver is ended and advice the user to start blowing and perform a new expiratory maneuver
9. Repeat from step 4 (3 times at least)
10. send the [stopTest](#) command to quit from test and wait for the [testStopped](#) delegate method to be called.
11. Remove your [PefFev1DeviceCallback](#) listener via the [removeDeviceCallback](#) method of your [Device](#) instance to avoid memory leaks and possible bugs

It is strongly NOT recommended to disconnect the device ([DeviceManager disconnect](#)) at the end of each session or test. The bluetooth disconnection of the device should be called only if no more spirometry tests need to be performed in a short time (less than 1 minute). Even better is to disconnect when the app became inactive

PEAK-FLOW / FEV1 TEST (Start&Stop approach)

1. Get an instance of the [DeviceManager](#) Class
2. Perform a scan an connect OR perform a “direct connection” to a spirometer
3. Add your [PefFev1DeviceCallback](#) listener to your [Device](#) instance, via the [addDeviceCallback](#) method
4. Start the test (user must start blowing within 15 sec)
[testStopped](#) delegate method is called if user doesn’t blow within 15 sec (in this case give to the user the ability to restart the test manually)

It is strongly NOT recommended to call the Start test on the testStopped delegate method as a workaround to contrast the 15 sec timeout effect. If you need to give more time to the user, you can “Customize the End Of Test Time Out” in “startTest” command.

5. Use a visual feedback to prevent that user start blowing before the app has received the delegate method [testStarted](#)
6. Use [flowUpdated](#) delegate method to show the animated feedback (also in connection with the Patient class’s dedicated methods)
7. Use [resultsUpdated](#) to show the result (this method might not be called if the device was not able to calculate the results)
8. Use [testRestarted](#) delegate method to detect that the current expiratory maneuver is ended and use the [stopTest](#) command to quit from test (then wait for the [testStopped](#) delegate method to be called).
9. Repeat from step 3 (3 times at least)

- Remove your [PefFev1DeviceCallback](#) listener via the [removeDeviceCallback](#) method of your [Device](#) instance to avoid memory leaks and possible bugs

It is strongly NOT recommended to disconnect the device ([DeviceManager disconnect](#)) at the end of each session or test. The bluetooth disconnection of the device should be called only if no more spirometry tests need to be performed in a short time (less than 1 minute). Even better is to disconnect when the app became inactive

FVC-FVC PLUS TEST (Spirobank Smart only)

- Get an instance of the [DeviceManager](#) Class
- Perform a scan and connect OR perform a “direct connection” to a spirometer
- Add your [FvcDeviceCallback](#) listener to your [Device](#) instance, via the [addDeviceCallback](#) method
- Start the test (user must start blowing within 15 sec) [testStopped](#) delegate method is called if user does not blow within 15 sec (give to the user the ability to restart test manually)

It is strongly NOT recommended to call the Start test on the testStopped delegate method as a workaround to contrast the 15 sec timeout effect. If you need to give more time to the user, you can “Customize the End Of Test Time Out” in “startTest” command.

5. Use a visual feedback to prevent that user start blowing before the app has received the delegate method [testStarted](#)

- Use [flowUpdated](#) delegate method to show the animated feedback (also in connection with the [SOPatient](#) class’s dedicated methods)
- If the [receivedEndOfForcedExpirationIndicator](#) delegate method is called, the user can be advised to stop blowing, since a plateau or the end of expiratory time was reached
- Use [resultsUpdated](#) to show the result (this method might not be called if the device was not able to calculate the results)
- [testStopped](#) delegate method is called: the test is over
- Remove your [FvcDeviceCallback](#) listener via the [removeDeviceCallback](#) method of your [Device](#) instance to avoid memory leaks and possible bugs

It is strongly NOT recommended to disconnect the device ([DeviceManager disconnect](#)) at the end of each session or test. The bluetooth disconnection of the device should be called only if no more spirometry tests need to be performed in a short time (less than 1 minute). Even better is to disconnect when the app became inactive

VC TEST (Spirobank Smart only from firmware version 4.4+)

- Get an instance of the [DeviceManager](#) Class
- Perform a scan and connect OR perform a “direct connection” to a spirometer
- Add your [VcDeviceCallback](#) listener to your [Device](#) instance, via the [addDeviceCallback](#) method
- Start the test (user must start blowing within 15 sec) [testStopped](#) delegate method is called if user does not blow within 15 sec (give to the user the ability to restart test manually)

It is strongly NOT recommended to call the Start test on the testStopped delegate method as a workaround to contrast the 15 sec timeout effect. If you need to give more time to the user, you can “Customize the End Of Test Time Out” in “startTest” command.

5. Use a visual feedback to prevent that user start blowing before the app has received the delegate method [testStarted](#)

6. Use [volumeUpdated](#) delegate method to show the animated feedback

7. When the [ventilatoryProfilePerformed](#) callback is called, patient’s ventilatory profile is acquired

8. Use [resultsUpdated](#) to show the result (this method might not be called if the device was not able to calculate the results)

9. [testStopped](#) delegate method is called: the test is over

10. Remove your [VcDeviceCallback](#) listener via the [removeDeviceCallback](#) method of your [Device](#) instance to avoid memory leaks and possible bugs

It is strongly NOT recommended to disconnect the device ([DeviceManager disconnect](#)) at the end of each session or test. The bluetooth disconnection of the device should be called only if no more spirometry tests need to be performed in a short time (less than 1 minute). Even better is to disconnect when the app became inactive

End Of Test Criteria

During a spirometry maneuver the End of Test is detected by the spirometer (and thus propagated by the SDK) according to the following criteria:

FVC TEST

The FVC maneuver AUTOMATICALLY ends:

1. when an expiratory PLATEAU has been reached. The expiratory plateau is detected, by the [spirobankSmart](#), when no significant volumes (< 20mL) have been measured within a timeframe of 3 seconds
2. when a significant inhaled volume is detected AND an exhalation has been performed.
3. when a timeout is expired
 - a. when the user has not blown at all for 15-120 (according to the [endOfTestTimeout_sec](#) parameter passed to the “startTest” command) seconds since the test was started
 - b. when the user stops blowing for 3 sec
 - c. when the user keeps on blowing for 60 seconds and no plateau has been reached

According to ATS/ERS guidelines, to have an acceptable FVC maneuver, the exhalation must last no less than 6 seconds (3 in case of child) but it is also acceptable a maneuver that lasts less than 6 seconds (or 3) if it meets the criteria 1 (an expiratory plateau has been reached)

FVC PLUS TEST

Since the FVC PLUS test is a multiloop test, there are no conditions for it to end automatically, apart from timeouts:

- a. when the user has not blown at all for 15-120 (according to the [EndOfTestTimeOut](#) parameter passed to the "startTest" command) seconds since the test was started
- b. when the user stops blowing for 3 sec
- c. when the user keeps on blowing for 60 seconds and no plateau has been reached

PEAKFLOW/FEV1 test

The PEAKFLOW/FEV1 maneuver AUTOMATICALLY ends:

1. when the device detects a volume < 200mL AND a flow < 300mL/s within a timeframe of 2 seconds (here the test is AUTOMATICALLY RESTARTED)
2. when a significant inhaled volume is detected (here the test is AUTOMATICALLY RESTARTED)
3. when a timeout is expired
 - a. when the user has never been blowing for 15-120 (according to the [EndOfTestTimeOut](#) parameter passed to the "startTest" command) seconds since the test was started
 - b. when the user stop blowing for 3 sec (here the test is AUTOMATICALLY RESTARTED)
 - c. when the user keeps on blowing for 60 seconds AND none of the previous condition has been met

To have an acceptable PEAKFLOW/FEV1 maneuver, the exhalation must last no less than 1 seconds

VC TEST

The VC test:

- a. when the user has not blown at all for 15-120 (according to the [EndOfTestTimeOut](#) parameter passed to the "startTest" command) seconds since the test was started
- b. when the user stops blowing for 3 sec

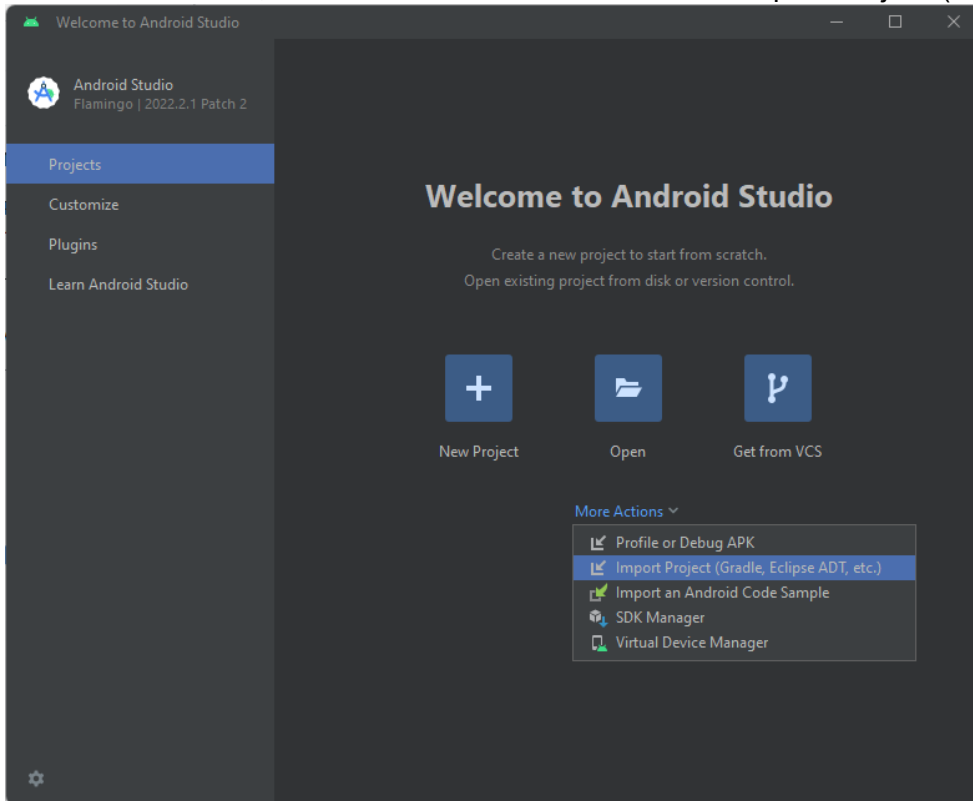
Sample Demo Application

The archive contains a subfolder named " SpirobankSmartSDKSample".

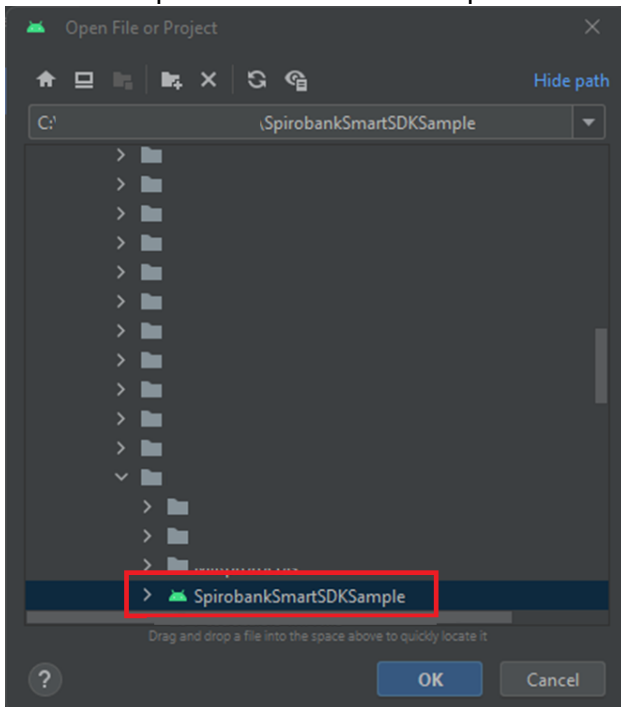
This folder contains an Android studio project.

To use it, open Android Studio and follow the instructions below:

a) On the main screen, select “More Actions” and select “Import Project (Gradle, Eclipse ADT, etc.)”

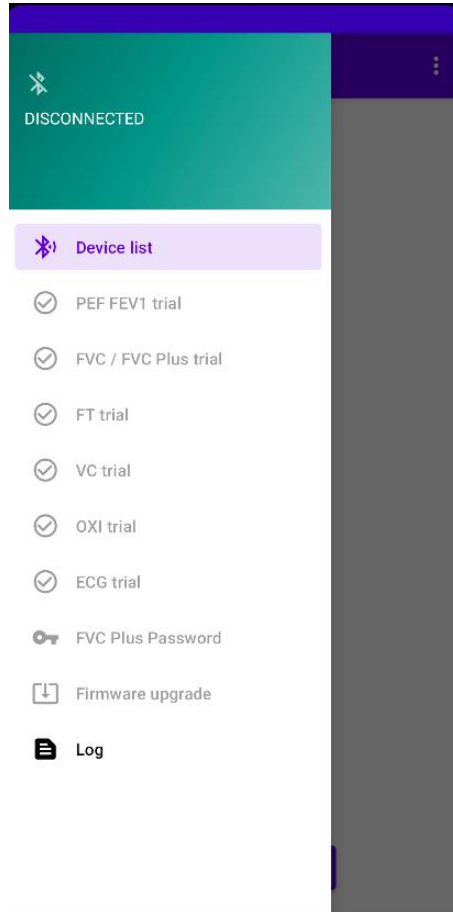
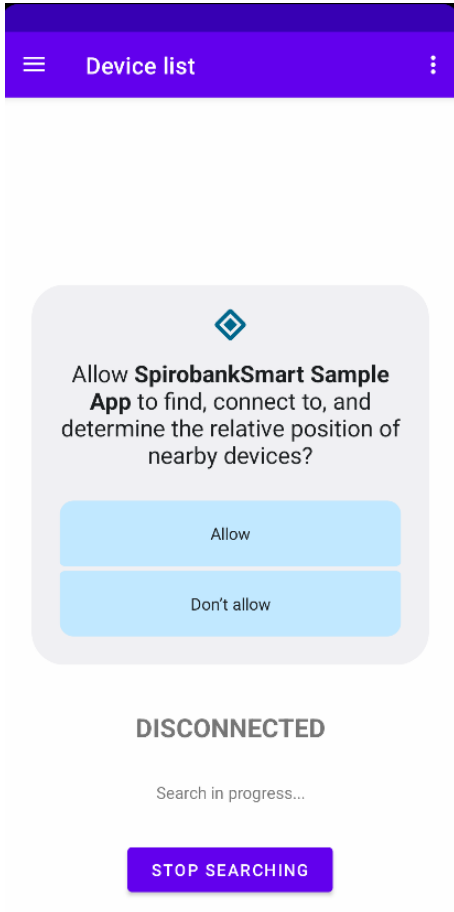


b) Select the “SpirobankSmartSDKSample” directory which should be symbolized with the Android icon.



c) Build the App and start it on a device.

The App allows you to scan the MIR Bluetooth Low Energy devices and to perform different types of tests:



Additional Resources

- MIR Website: <https://www.spirometry.com>

Troubleshooting

1. Could not find method compile() for arguments Gradle

Solution: `compile` method has been deprecated and removed in Gradle 7.0.

Open the file `build.gradle` and replace `compile` by `implementation` and `testCompile` by `testImplementation`.

2. Error: Could not initialize class com.android.sdklib.repository.AndroidSdkHandler

Solution: Make sure your `build.gradle` file is using Gradle 7.0+:

```
dependencies {  
    classpath 'com.android.tools.build:gradle:7.0.0'  
}
```

3. Gradle Sync Failed: No installed build tools found. Install the Android build tools version 19.1.0 or higher.

Solution: Install or update the Android Build Tools via the SDK Manager in Android Studio.

4. Error: JAVA_HOME is not set and no 'java' command could be found in your PATH.

Solution: Ensure the JDK is installed and set the `JAVA_HOME` environment variable to point to the JDK installation directory.

5. Unable to start the daemon process: could not reserve enough space for object heap.

Solution: Increase the memory allocated to Gradle by modifying or adding the line `org.gradle.jvmargs=-Xmx2048m` in the `gradle.properties` file.

6. INSTALL_FAILED_VERSION_DOWNGRADE

Solution: uninstall the existing application from the device or emulator and try installing again.

7. Emulator: PANIC: Missing emulator engine program for 'x86' CPU.

Solution: Ensure the necessary Android Emulator components are properly installed via the SDK Manager.

8. AAPT2 error: check logs for details.

Solution: Check the logs for specific errors. Often, this is due to issues in resource files, such as misnamed images or errors in XML files.

9. Failed to resolve: com.android.support:appcompat-v7:26.1.0

Solution: Ensure the correct repositories are added in the `build.gradle` file and that the library version is available.